МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Літня школа з програмування

(Ужгород, 31 липня – 7 серпня 2016 року)

Матеріали лекцій, умови та розбір задач

Ужгород
2017

УДК 004.42(042.3)
М-70

У книзі вміщено матеріали лекцій, умови та розбір задач Літньої школи з програмування, яка відбулася в Ужгороді з 31 липня до 07 серпня 2016 року.

Збірник буде корисний для всіх, хто цікавиться програмуванням, готується до олімпіад, хоче вдосконалити знання зі складання алгоритмів і розв'язування задач.

Літня школа з програмування (Ужгород, 31 липня – 7 серпня 2016 року) : Матеріали лекцій, умови та розбір задач / За ред. Олександра Міци, Сергія Оришича. – Ужгород: Видавництво «ФОП Сабов А.М.», 2017. – 164 с.

ISBN 978-617-7344-46-8

ЗМІСТ

День 1. Контест Василя Білецького

Задачі та розв'язання

A. Літня школа

Одного разу до Ужгорода на літню школу з алгоритмічного програмування приїхали N студентів, що сформували K команд. Відомо, що кожна команда складається з одного, двох або трьох студентів.

Вам необхідно визначити, скільки студентів було в кожній із команд.

Формат вхідних даних

У єдиному рядку записані два цілі числа N та K через пробіл ($1 \leq N \leq 1000$, $1 \leq K \leq 100$).

Формат вихідних даних

Виведіть K цілих чисел $A_1, \dots, A_k$ через пробіл ($1 \leq A_j \leq 3$). Тут A_j – кількість студентів у j-ій команді. Якщо існує більше одного розв'язку, – виведіть будь-який. Якщо розв'язку не існує – виведіть, «`Impossible`» (без лапок).

Приклади

stdin	stdout
7 4	1 3 1 2

Пояснення до прикладів

Якщо на літню школу приїхали 7 студентів, а разом було 4 команди, то можливі такі дві конфігурації:

- одна команда з 3 студентів, одна команда з 2 студентів та дві команд з 1 студента;
- три команд з 2 студентів та одна команда з 1 студента.

Розв'язання задачі «A. Літня школа»

Очевидно, якщо $N < K$ або $3K < N$, то розвязку не існує.

Інакше є багато способів побудови одного з можливих розв'язків.

Наприклад, перші $(N-K)/2$ команд складаються з трьох студентів, а всі інші з одного, але якщо число $(N-K)$ – непарне, то остання команда складається з двох студентів.

Легко бачити, що сума кількостей студентів щодо всіх команд рівна N.

B. Тест Джона

Малий Джон займається тестуванням чисел. Він має два цілих числа A та B. Нехай Джон розглядає певне ціле число X. Якщо $A \leq X \leq B$, то число X успішно проходить тест Джона. В іншому випадку Джон будує ціле число Y, використовуючи цифри десяткового запису числа X. Спочатку він бере першу цифру (найбільш значущу) числа X, потім – останню цифру X, потім – другу, далі – передостанню і так далі, поки не будуть використані всі цифри числа X. Наприклад, якщо $X = 1234567$, то $Y = 1726354$, а якщо $X = 1020$, то $Y = 1002$. Після побудови числа Y, якщо $A \leq Y \leq B$, то число X проходить тест Джона, інакше X не проходить тест.

Вам необхідно порахувати всі цілі додатні числа, що проходять тест Джона.

Формат вхідних даних

У єдиному рядку записані два цілих числа А та B через пробіл ($1 \leq A \leq B \leq 10^{18}$).

Формат вихідних даних

Виведіть єдине число – кількість цілих додатних чисел, що проходять тест.

Приклади

stdin	stdout
98 102	7
123456789 12345678	2

Пояснення до прикладів

У першому прикладі числа 98, 99, 100, 101, 102, 110 та 120 проходять тест Джона.

Розв'язання задачі «B. Тест Джона»

Розв'яжемо цю задачу, використовуючи методику динамічного програмування.

Розглянемо ціле число X та $Y = F(X)$. Тут F – функція перетворення, що описана в умові задачі.

Будемо будувати Y цифра за цифрою, починаючи з найбільш значущої.

Нехай у певному стані ми маємо побудований префікс числа Y (позначимо його $prefY$).

Оскільки $Y = F(X)$, ми також будемо мати побудованими певний префікс та певний суфікс числа X (позначимо їх $prefX$ та $suffX$, відповідно).

Тепер порівняємо $prefY$ із відповідними префіксами чисел A та B.

Також порівняємо $prefX$ із відповідними префіксами A та B, а $suffX$ із відповідними суфіксами A та B.

Зауважимо, що для нас не важливі самі значення $prefY$, $prefX$ та $suffX$, а лише їхні відношення (менше ніж, дорівнює, більше ніж) до відповідних префіксів та суфіксів A та B.

Тож для кожної з можливих довжин числа Y побудуємо динаміку, що буде мати такі виміри:

- довжина $prefY$,
- відношення $prefY$ до відповідного префікса числа A,
- відношення $prefY$ до відповідного префікса числа B,
- відношення $prefX$ до відповідного префікса числа A,
- відношення $prefX$ до відповідного префікса числа B,
- відношення $suffX$ до відповідного суфікса числа A,
- відношення $suffX$ до відповідного суфікса числа B.

Значення – кількість різних $prefY$.

Для кожного стану перебираємо 10 можливих варіантів наступної цифри, що додаємо до $prefY$.

Загальну кількість станів динаміки можна оцінити як $\log^2 B \cdot 3^6$.

Знайшовши значення динаміки, ми можемо легко знайти результат, врахувавши тільки ті стани, де хоча б одне з чисел X та Y знаходиться в інтервалі $[A, B]$.

C. Монети на дошці

Ви маєте шахову дошку розміру N на M. Рядки дошки пронумеровані від 1 до N, а стовпці – від 1 до M. Деякі клітинки дошки містять монети. За одну дію Ви можете обрати дві порожні клітинки дошки (r_1, c_1) та (r_2, c_2) такі, що $r - r_2 = DR$, $c_1 - c_2 = DC$ й поставити по одній монеті у кожну з них. Ви можете повторити таку дію довільну кількість разів.

Вам необхідно знайти максимальну кількість монет, що Ви можете поставити на дошку (без урахування тих, що вже були на дошці).

Формат вхідних даних

У першому рядку записані п'ять цілих чисел N, M, K, DR та DC через пробіл ($1 \le N, M \le 10^9$, $1 \le DR, DC \le 10^9$, $0 \le K \le 10^5$). Тут K – кількість клітинок дошки, що початково містять монети. Кожен із наступних K рядків містить два цілих числа r_j та c_j через пробіл ($1 \le r_j \le N$, $1 \le c_j \le M$) – номер рядка та стовпця j-ї клітинки відповідно. Всі клітинки (r_j, c_j) попарно різні.

Формат вихідних даних

Виведіть єдине число – максимальну кількість монет.

Приклади

stdin	stdout
3 3 0 1 1	6
5 4 2 2 1 3 2 3 3	10

Пояснення до прикладів

У першому прикладі Ви маєте порожню дошку 3 на 3, а $DR = DC = 1$.

У першій дії Ви можете обрати пару клітинок $(1, 1)$ та $(2, 2)$, у другій дії – $(1, 2)$ та $(2, 3)$, а у третій – $(2, 1)$ та $(3, 2)$.

Розв'язання задачі «C. Монети на дошці»

Будемо називати ланцюгом послідовність порожніх комірок дошки (r, c), $(r + DR, c + DC)$, $(r + 2 \cdot DR, c + 2 \cdot DC)$ і так далі.

Для розв'язання цієї задачі достатньо знайти кількість ланцюгів непарної довжини.

Спочатку розв'яжемо задачу для порожньої дошки.

Якщо $F(N, M)$ – кількість ланцюгів непарної довжини для порожньої дошки розміру N на M, то можна показати, що $F(N + 2 \cdot DR, M + 2 \cdot DC) = F(N, M) + 2 \cdot DR \cdot DC$. Тому ми можемо легко знайти це значення для порожньої заданої дошки.

Далі групуємо дані монети за ланцюгами порожньої дошки, до яких вони належать.

Для кожного такого ланцюга розіб'ємо його на менші ланцюги відповідно до монет, що йому належать. Так ми знайдемо загальну кількість ланцюгів непарної довжини.

Відповідь до задачі рівна $N \cdot M - K -$ (кількість ланцюгів непарної довжини).

D. Гра на дошці

Двоє гравців влаштували просту гру на шаховій дошці розміру N на M. Рядки дошки пронумеровані від 1 до N, а стовпці – від 1 до M. Перший гравець має одного білого пішака, що на початку гри розташований у клітинці (RW, CW). Другий гравець має одного чорного пішака, що розміщений у клітинці (RB, CB). Гравці ходять по черзі, розпочинає перший гравець.

Роблячи хід, гравець обирає один із чотирьох діагональних напрямків (північно-східний, північно-західний, південно-східний чи південно-західний) та рухає свого пішака на довільну додатну кількість клітинок у цьому напрямку. Гравця, після ходу якого обидва пішаки знаходяться в одній клітинці, вважають переможцем, а гру – завершеною.

Обидва гравці керуються оптимальною ігровою стратегією. Якщо гравець може перемогти, він використовуватиме стратегію, що мінімізує загальну кількість ходів гри. Якщо ж гравець не може перемогти, то він буде керуватися стратегією, що максимізує загальну кількість ходів.

Вам необхідно визначити результат гри.

Формат вхідних даних

У першому рядку записані два цілих числа N та M через пробіл ($2 \leq N, M \leq 10^9$). У наступному рядку чотири цілих числа RW, CW, RB та CB через пробіл ($1 \leq RW, RB \leq N$, $1 \leq CW, CB \leq M$). Клітинки (RW, CW) та (RB, CB) різні.

Формат вихідних даних

Якщо переможе перший гравець, виведіть «`White` X» (без лапок), якщо ж переможе другий гравець, виведіть «`Black` X» (без лапок), інакше виведіть «`Draw`» (без лапок). Тут X – загальна кількість ходів у грі.

Приклади

stdin	stdout
47 74 4 7 7 4	White 1
2 5 1 2 1 4	Black 4
7 4 1 3 7 2	Draw

Пояснення до прикладів

У другому прикладі перший гравець має два варіанти першого ходу. Якщо він поставить білого пішака у клітинку $(2, 3)$, то програє на наступному ході, тому відповідно до оптимальної стратегії він рухає свого пішака у клітинку $(2, 1)$. Після цього другий гравець ходить у $(2, 3)$, далі перший повертається у $(2, 1)$. Наступним ходом другий гравець завершує гру.

Розв'язання задачі «D. Гра на дошці»

Якщо пішаки знаходяться на клітинках різного кольору, то гра закінчиться у нічию. Інакше якщо пішаки знаходяться на одній діагоналі, то перший гравець виграє гру своїм першим ходом. Далі вважатимемо, що на початку гри пішаки знаходяться на клітинках одного кольору, але не на одній діагоналі.

Доведемо такий факт: якщо пішак знаходиться на діагоналі, що містить принаймні три клітинки, то він не програє гру.

Доведення: у кожен момент гри пішак перебуває на одній із клітинок діагоналі й не більше ніж одна клітинка знаходиться під боєм пішака суперника. Тому пішак завжди має принаймні один варіант ходу, що не призведе до програшу та залишить його на цій же діагоналі.

Наслідок: зауважимо, що якщо і N, і M більші ніж 3, то кожна клітинка знаходиться принаймні на одній діагоналі, що має принаймні три клітинки. Тому гра у випадку $N, M > 3$ завжди завершиться нічиєю.

Якщо хоча б одне з чисел N та M рівне 2, то гра еквівалентна одновимірній аналогічній грі, де гравець ходить рівно на одну клітинку. Тоді залежно від парності початкової відстані між пішаками легко визначити результат гри разом із загальною кількістю ходів.

Розглянемо останній випадок, коли одне з чисел N та M рівне 3. Не втрачаючи загальності, нехай $N = 3$, $M \geq 3$.

Далі якщо $M = 3$, то або обидва пішаки знаходяться на діагоналях довжини 3 і гра є нічийною, або другий гравець виграє гру на другому ході гри.

Якщо ж $M > 3$, то є тільки дві клінинки, що не знаходяться на діагоналі довжини принаймні 3, - $(2, 1)$ та $(2, M)$. Назвемо їх критичними.

Якщо жоден гравець не знаходиться на критичній клітинці, то гра є нічийною.

Якщо перший гравець знаходиться на критичній клітинці, то він програє на другому ході тільки якщо другий гравець розташований за дві клітинки від нього по горизонталі. А саме: перший - $(2, 1)$, другий - $(2, 3)$, або перший - $(2, M)$, другий - $(2, M - 2)$.

Якщо другий гравець знаходиться на критичній клітинці, то він програє на третьому ході тільки якщо перший гравець за перший хід зможе перемістити свого пішака на клітинку, що розташована на відстані 2 від другого гравця по горизонталі. Наприклад, програшною для другого гравця є конфігурація: перший - $(1, 4)$, другий $(2, 1)$.

E. Пасьянс

Розв'язавши всі задачі на змаганнях, Ви вирішили розкласти пасьянс на комп'ютері. У Вас є N гральних карт, що пронумеровані від 1 до N, а на ігровому полі – M комірок, що пронумеровані від 1 до M зліва направо. Вам необхідно розмістити кожну карту у певну комірку, враховуючи такі вимоги:

- карта 1 повинна бути розміщена рівно D_1 комірок лівіше або рівно D_1 комірок правіше від карти 2;
- карта 2 повинна бути розміщена рівно D_2 комірок лівіше або рівно D_2 комірок правіше від карти 3;
- …
- карта N повинна бути розміщена рівно D_N комірок лівіше або рівно D_N комірок правіше від карти 1.

Вам необхідно знайти кількість різних способів, якими Ви можете розкласти пасьянс. Зауважте, що в одній комірці можна розмістити довільну кількість карт.

Формат вхідних даних

У першому рядку записані два цілих числа N та M через пробіл ($2 \le N \le 36$, $1 \le M \le 10^6$). У наступному рядку N цілих чисел D_1, D_2,…, D_N через пробіл ($1 \le D_j \le 10^4$).

Формат вихідних даних

Виведіть кількість способів розкладу пасьянсу.

Приклади

stdin	stdout
5 5 3 1 3 1 1	8
2 47 4 7	0

Пояснення до прикладів

У першому прикладі можливі такі розклади:

```
{1}-{5}-{4}-{2}-{3}          {3}-{2}-{4}-{5}-{1}
{5}-{1, 4}-{}-{3}-{2}        {2}-{3}-{}-{1, 4}-{5}
{1, 4}-{5}-{3}-{2}-{}        {}-{2}-{3}-{5}-{1, 4}
{}-{1, 4}-{5}-{3}-{2}        {2}-{3}-{5}-{1, 4}-{}
```

Розв’язання задачі «E. Пасьянс»

Для розв’язання задачі використаємо методику «meet in the middle».

Нехай ми маємо нескінченну кількість комірок, кожній із яких відповідає ціле число (включно з від’ємними). Зафіксуємо карту 1 у комірці 0 та згенеруємо $2^{N/2}$ варіантів розставити карти $2,\ 3, \ldots, N/2+1$ відносно неї (для кожної наступної карти ми маємо два варіанти – ліворуч або праворуч від попередньої).

Також згенеруємо $2^{N-N/2}$ варіантів розставити карти $N,\ N-1, \ldots, N/2+1$ відносно карти 1 у зворотному порядку (будемо називати їх зворотними варіантами).

Для кожного згенерованого варіанту нас цікавлять такі речі:

- позиція найлівішої карти,
- позиція найправішої карти,
- позиція карти $N/2 + 1$.

Далі нам необхідно поєднати прямі та зворотні варіанти. Причому позиція карти $N/2 + 1$ має збігатися. Згрупуємо всі варіанти за позицією карти $N/2 + 1$. Для кожної групи будемо робити наступне: кожен варіант визначає неперервний діапазон комірок, у кожну з яких ми можемо поставити карту 1 так, щоб найлівіша та найправіша карти були на прийнятних позиціях.

Отже, для кожної комірки (від 1 до M) нас цікавить кількість діапазонів прямих варіантів, що містять її, та кількість діапазонів зворотних варіантів, що також містять її. Добуток цих кількостей додаємо до результату.

Такі кількості зручно рахувати, ввівши два лічильники (прямий та зворотній) та визначивши для кожного варіанту дві події:

- збільшити відповідний лічильник на 1 (на початку діапазону),
- зменшити відповідний лічильник на 1 (після кінця діапазону).

Далі сортуємо всі події за координатами, проходимося по них, оновлюючи лічильники та результат.

F. Більярдні кулі

У Вас є N більярдних кульок та K коробок. Кулі пронумеровані від 1 до N, а коробки – від 1 до K. На кожній більярдній кулі написано певне ціле число. Вам необхідно розкласти кулі в коробки так, щоб у кожній коробці була принаймні одна куля. Після цього для кожної коробки можна порахувати її значення як *AND* суму чисел більярдних кульок, що знаходяться в коробці. Тут *AND* – операція побітової кон'юнкції.

Вам необхідно визначити, яку максимальну суму (звичайну) значень коробок Ви можете отримати та остачу від ділення на 1000000007 кількості способів, якими можна отримати максимальну суму. Два способи вважаються різними, якщо відповідно до них принаймні одна кулька знаходиться в різних коробках. Зауважте, що кулі з однаковими числами вважаються різними.

Формат вхідних даних

У першому рядку записані два цілих числа N та K через пробіл ($1 \leq K \leq N \leq 10^5$). У наступному рядку N цілих чисел A_1, A_2,…, A_N через пробіл ($0 \leq A_j \leq 10^9$). Тут A_j – ціле число, що написано на j-й кульці.

Формат вихідних даних

Виведіть два цілих числа через пробіл – максимальну суму значень коробок та остачу від ділення на 1000000007 кількості способів, якими можна її отримати.

Приклади

stdin	stdout
3 2 4 7 5	11 2
4 1 44 47 74 77	8 1
4 4 44 47 74 77	242 24

Пояснення до прикладів

У першому прикладі максимальна сума рівна $11 = 7 + (5\ AND\ 4)$. Її можна отримати такими способами:

- $\{7\} - \{4, 5\}$,
- $\{4, 5\} - \{7\}$.

Розв'язання задачі «F. Більярдні кулі»

Нехай $f(bit, K, a[1], a[2], \ldots, a[N])$ – функція, що повертає максимальне значення суми для K коробок та N куль ($K \leq N$) з цілими невід'ємними числами $a[1], a[2], \ldots, a[N]$. Причому всі числа менші ніж $2 \cdot 2^{bit}$.

Упорядкуємо числа за незростанням: $2 \cdot 2^{bit} > a[1] \geq a[2] \geq \ldots \geq a[N] \geq 0$.

Нехай $f(bit, K, a[1], a[2], \ldots, a[N]) = value$.

Розглянемо декілька випадків:

1. $a[1] = 0$, тоді $value = 0$. Усі числа рівні нулю.

2. $a[1] < 2^{bit}$, тоді $value = f(bit-1, K, a[1], a[2], \ldots, a[N])$. Усі числа у двійковій системі числення мають 0 у позиції bit.

3. $a[N] \geq 2^{bit}$, тоді $value = f(bit-1, K, a[1]-2^{bit}, a[2]-2^{bit}, \ldots, a[N]-2^{bit}) + K \cdot 2^{bit}$. Усі числа мають 1 у позиції bit, отже, незалежно від розстановки по коробках до результату додаємо $K \cdot 2^{bit}$ і переходимо до наступної позиції.

Нехай P – така позиція, що $a[P] \geq 2^{bit}$, але $a[P+1] < 2^{bit}$.

4. $P < K$, тоді $value = f(bit-1, K-P, a[P+1], \ldots, a[N]) + a[1] + \ldots + a[P]$. Менше ніж K чисел мають 1 у позиції bit. Не складно бачити, що завжди вигідно обирати для найбільших P чисел окремі коробки.

Доведення: від супротивного – якщо в оптимальній конфігурації це не так, то, переставивши кулі, отримаємо кращий результат.

5. $P \geq K$, тоді $value = f(bit-1, K, a[1]-2^{bit}, \ldots, a[P]-2^{bit}, a[P+1]\ AND\ a[P+2]\ AND \ldots AND\ a[N]) + (K-1) \cdot 2^{bit}$. Не менше ніж K чисел мають 1 у позиції bit. Не складно бачити, що завжди вигідно згрупувати останні $N-P$ чисел і розглядати їх як одне ціле.

Доведення: від супротивного.

Тоді незалежно від розстановки тільки одна коробка буде мати 0 на позиції bit.

Для обрахунку кількості способів отримати максимальну суму, зауважимо, що вибір коробок для куль здійснюється у випадках 1 та 4.

Нехай $\mathrm{C}(n,k)$ – число комбінацій з n елементів по k.

У випадку 4 нам необхідно домножити результат на кількість способів обрати P різних коробок, що рівна $C(K,P) \cdot (P!)$.

У випадку 1 нам потрібно знайти кількість способів розмістити N куль в K коробках так, щоб у кожній коробці була принаймні одна куля.

Згідно з принципом включення-виключення ця кількість рівна $K^N \cdot C(K,K) - (K-1)^N \cdot C(K,K-1) + (K-2)^N \cdot C(K,K-2) - \ldots$.

G. Щаслива сума

Відомо, що щасливим числом є те, десятковий запис якого містить тільки четвірки та сімки. Наприклад, щасливими є числа 4, 7, 47, 7777 та 4744474.

Нехай S – множина щасливих чисел не менших ніж A та не більших ніж B:

$$S = \{n: A \leq n \leq B, n - \text{щасливе число}\}.$$

Ваше завдання – обчислити остачу від ділення на 1234567891 такої суми:

$$\sum_{n \in S} n^n.$$

Формат вхідних даних

У першому рядку записані два цілих числа A та B через пробіл ($1 \leq A \leq B \leq 10^{18}$).

Формат вихідних даних

Виведіть остачу від ділення щасливої суми на 1234567891.

Приклади

stdin	stdout
1 10	823799

Пояснення до прикладів

$4^4 + 7^7 = 823799$.

Розв'язання задачі «G. Щаслива сума»

Оцінимо загальну кількість щасливих чисел на проміжку $[A, B]$.

Очевидно, що кількість щасливих чисел довжини k рівна 2^k. Отже, загальна кількість не більша ніж $2^1 + 2^2 + 2^3 + \ldots + 2^{18} = 2^{19} - 2$. Тому ми можемо згенерувати всі щасливі числа довжини не більше ніж 18 та обрати ті, які належать проміжку $[A, B]$.

Для кожного обраного щасливого числа X необхідно обчислити $X^X \; mod \; 1234567891$. Це можна зробити за допомогою двійкового піднесення до степеня.

День 2. Контест UzhNU_Brainiacs (Роман Рубаненко, Роман Мельник, Сергій Оришич)

Теоретичний матеріал. Графи

I. ДЕРЕВА

1.1. Діаметр Дерева

Діаметр дерева – максимальна довжина (в ребрах) найкоротшого шляху в дереві між будь-якими двома вершинами.

Нехай є граф $G = (V, E)$. Тоді діаметром d називається $\max_{u,v \in V} dist(v, u)$, де $dist$ – довжина найкоротшого шляху між вершинами.

Алгоритм пошуку діаметру дерева

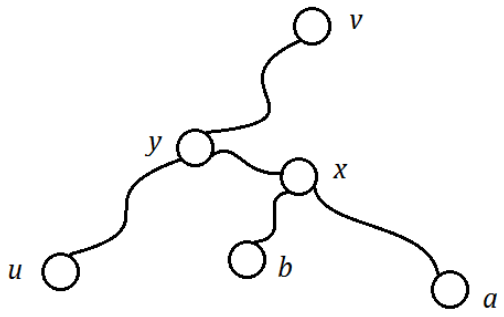
- Візьмемо будь-яку вершину $v \in V$ та знайдемо відстань до всіх інших вершин. $d[i] = \min_{u,i \in V} dist(u, i)$.
- Візьмемо вершину $u \in V$ таку, що $d[u] \geq d[t]$ для будь-якого t. Знову знайдемо відстань від u до всіх інших вершин. Нехай найбільша відстань до вершини w, тоді $dist(u, w)$ – діаметр графу.
- Вищеперераховані операції можна здійснити за допомогою двох запусків BFS. Складність $O(|V| + |E|)$.

Коректність алгоритму пошуку діаметру дерева

Будемо користуватися властивістю, що в будь-якому дереві більше одного листка. Виняток – дерево з однієї вершини, проте очевидно, що алгоритм працюватиме правильно і в цьому випадку.

Припустимо, що алгоритм знайдені дві вершини u та w не утворюють шлях, довжина якого рівна діаметру дерева, а правильний діаметр утворюють вершини a та b. Нехай вершина $x \in a \leftrightsquigarrow b$ та є найближчою до вершини u,

тоді припустимо, що $dist(x,a) \geq dist(x,b)$. Вершина $x \in u \leftrightsquigarrow v$, інакше ж $dist(u,a) > dist(b,a)$:



$$dist(u,a) > dist(a,b)$$

Далі можна стверджувати, що $dist(u,x) \geq dist(x,a)$, інакше ж $dist(v,a) > dist(v,u)$, що суперечить тому, що вершина u – найвіддаленіша від вершини v. Отже, $dist(u,a) = dist(u,x) + dist(x,a) \geq dist(b,x) + dist(x,a) = d(a,b)$, що доводить те, що вершина u є початком хоча б одного з найдовших шляхів. Відповідно, найвіддаленіша від вершини u – вершина w, буде кінцем найдовшого шляху в дереві.

1.2. Центр дерева

Ексцентриситет вершини $e(v)$ - $\max_{u,v\in} dist(v,u)$, де V - множина вершин зв'язного графа G.

Радіус $r(G)$ – найменший з ексцентриситетів вершин графу G.

Центральна вершина – вершина графа G така, що $e(v) = r(G)$.

Центр графу G – множина всіх центральних вершин графу G.

Алгоритм пошуку центру графу

Теорема. Кожне дерево має центр, що складається з однієї або двох суміжних вершин.

Доведення. Твердження очевидне для дерева з однією або двома вершинами. Продемонструємо, що у будь-якого дерева T ті ж центральні вершини, що і у дерева T', отриманого з T видаленням його висячих вершин. Відстань від даної вершини дерева u до будь-якої іншої вершини v досягає

найбільшого значення, коли v - висяча вершина. Таким чином, ексцентриситет кожної вершини дерева T' точно на одиницю менше ексцентриситету цієї ж вершини в дереві T, відповідно, центри цих дерев збігаються. Продовживши процес видалення і отримаємо необхідне.

Отже, алгоритм знаходження центру:

- Обійдемо дерево обходом углиб та помітимо всі висячі вершини числом 0.
- Видалимо помічені вершини.
- Новостворені листки помітимо числом 1 і також видалимо.
- Будемо повторювати, поки на поточній глибині не опиниться не більше двох листків, і при цьому в дереві буде також не більше двох вершин.
- Листки, що залишилися, є центром дерева.

Для того, щоб алгоритм працював за $O(n)$, необхідно обробляти листки по одному, підтримуючи в черзі два послідовні за глибиною шари.

1.3. Динаміка на деревах

Параметром стану динаміки на деревах зазвичай є вершина, що позначає піддерево, в якому ця вершина – корінь. Для отримання поточного стану необхідно знати результати всіх своїх дітей. У більшості випадків, це роблять звичайним пошуком вглиб з кореня дерева.

Нехай є кореневе дерево T з n вершинами. Підвісимо T за якусь вершину r. Позначимо через $Ch(x)$ множину синів вершини x. А через $lca(u,v)$ - найдальшого від кореня спільного батька вершин u та v.

Задача про суму довжин всіх шляхів у дереві

Нехай P шлях у дереві між вершинами u та v. Є два випадки:

1) $lca(u,v) \in \{u,v\}$

2) $lca(u,v) \notin \{u,v\}$, тоді P можна розбити на 2 частини, кожна з яких буде мати перший вигляд.

Позначимо через

- s_u - розмір піддерева u;
- d_u – суму довжин всіх шляхів між u та вершинами в піддереві u;
- r_u – суму довжин всіх шляхів між вершинами a та b, для яких $lca(a,b) = u$.

Якщо вершина u листок, то $s_u = 1$, $d_u = r_u = 0$, інакше:

$$s_u = 1 + \sum_{v \in Ch(u)} s_v.$$

$$d_u = \sum_{v \in Ch(u)} (d_v + s_u) = s_u - 1 + \sum_{v \in Ch(u)} d_v.$$

$$r_u = d_u + \sum_{\substack{v,w \in Ch(u),\\ v \neq w}} \big(s_v(d_w + s_w)\big) = d_u + \sum_{v \in Ch(u)} s_v\big(d_u - (d_v + s_v)\big) =$$

$$= d_u s_u - \sum_{v \in Ch(u)} s_v(d_v + s_v).$$

Відповіддю буде $\sum_{u=1}^{n} r_u$. Час роботи алгоритму $O(\sum_{u=1}^{n} |Ch(u)|) = O(n)$.

Альтернативне рішення: для кожного ребра можемо порахувати, скільки разів воно ввійде до шуканої суми. Це кількість вершин зліва, помножена на кількість вершин справа.

Задача про вершинне покриття дерева

Вершинне покриття дерева неорієнтованого графа $G = (V, E)$ - це множина його вершин S , така що, в кожного ребра графу хоча б один з кінців належить до S.

Знайдемо таке вершинне покриття дерева S, що $|S|$ - мінімально можливе серед усіх можливих S. Рішення полягає у використанні динаміки за піддеревами.

Нехай

- $a_{u,0}$ – розмір мінімального вершинного покриття піддерева з коренем у вершині u серед усіх покриттів, що містять вершину u;

- $a_{u,1}$ – розмір мінімального вершинного покриття піддерева з коренем у вершині u серед всіх покриттів, що не містять вершину u.

Якщо вершина u листок, то $a_{u,0} = 0$ і $a_{u,1} = 1$, інакше:

$$a_{u,0} = \sum_{v \in Ch(u)} a_{v,1}$$

$$a_{u,1} = 1 + \sum_{v \in Ch(u)} \min(a_{v,0}, a_{v,1})$$

Відповіддю буде значення, що рівне $\min(a_{r,0}, a_{r,1})$. Час роботи $O(\sum_{u=1}^{n} |Ch(u)|) = O(n)$.

Задача про паросполучення максимальної ваги в дереві

Нехай є зважене дерево, з вагами, які позначенні як $w_{i,j}$, де i та j - вершини дерева, що з'єднанні ребром. Задача: скласти таке паросполучення, щоб сумарна вага всіх ребер, що входять до нього, була максимальна.

Для розв'язання цієї задачі існує кілька алгоритмів. Наприклад, алгоритм Куна, який має верхню оцінку порядку $O(n^3)$. Але оскільки ми маємо дерево, то є можливість використання динамічного програмування, час роботи алгоритму з яким покращується до $O(n)$.

Позначимо через

- a_u – паросполучення максимальної ваги у піддреві з коренем у вершині u, при чому вершина u з'єднана ребром, що входить до паросполучення, з вершиною, що входить у піддерево i-ої вершини;
- b_u – паросполучення максимальної ваги в піддереві з коренем у вершині u, але при чому вершина u з'єднана ребром, що входить у паросполучення, з вершиною, що не входить у піддерево i-ої вершини;
- $c_u = \max(a_u, b_u)$.

Отже, відповідь задачі буде зберігатися в c_r.

Якщо вершина u – листок, то $a_u = b_u = 0$, інакше:

$$a_u = \max_{x \in Ch(u)} (b_x + w_{u,x} + \sum_{\substack{y \neq x \\ y \in Ch(u)}} \max(a_y, b_y))$$

$$b_u = \sum_{x \in Ch(u)} \max(a_x, b_x)$$

З урахуванням того, що $c_u = \max(a_u, b_u)$, вищеописані формули можна переписати так:

$$a_u = \max\left(\sum_{x \in Ch(u)} (b_x + w_{u,x} - c_x\right) + b_u$$

$$b_u = \sum_{x \in Ch(u)} c_x$$

Тепер оцінимо кількість операцій, необхідних для знаходження c_r. Оскільки $c_u = \max(a_u, b_u)$, то для обчислення c_r необхідно порахувати a_r та b_r. Для обчислення й того, й іншого необхідно час порядку $O(\sum_{u=1}^{n} |Ch(u)|) = O(n)$.

1.4. Запити на деревах

Перш за все навчімося знаходити LCA (Least Common Ancestor) двох довільних вершин у дереві. Зафіксуємо довільну вершину як корінь дерева, тоді LCA двох вершин – це найвіддаленіша від кореня вершина, що одночасно є предком обох цих вершин. Є кілька підходів швидкого знаходження LCA:

- Метод двійкового підйому [http://e-maxx.ru/algo/lca_simpler].
- Алгоритм Тарьяна [http://e-maxx.ru/algo/lca_linear_offline].
- Онлайн-версія пошуку LCA.

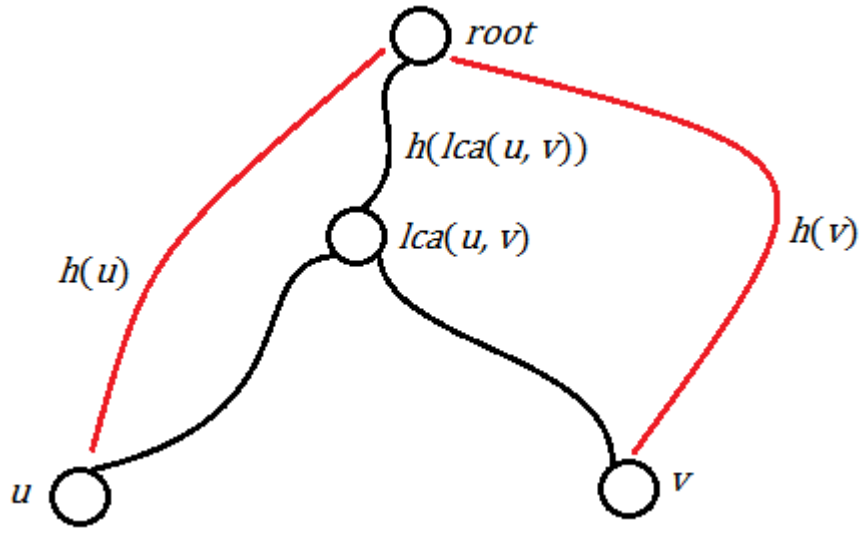
Більшість задач про запити на деревах можна поділити на такі групи:

- Запити на шляхах
- Запити на піддеревах
- Запити в радіусі

Запити на шляхах

Довжина шляху (довжина ребер статична)

Є дерево та запити, для кожного з яких необхідно знайти $dist(u, v)$ - відстань між заданими вершинами u та v цього дерева. Нехай $h(u)$ - це відстань від кореня дерева до вершини u. Тоді можна стверджувати, що $dist(u, v) = h(u) + h(v) - 2h\big(lca(u, v)\big)$, оскільки в сумі $h(u) + h(v)$ двічі враховується «зайва» відстань $h\big(lca(u, v)\big)$ - відстань від кореня до $lca(u, v)$. Отже, задача зводиться до знаходження $h(u)$ для всіх вершин дерева та ефективного пошуку $lca(u, v)$. Якщо ваги ребер не змінюються, то $h(u)$ можна порахувати на початку за допомогою будь-якого алгоритму обходу графа.



$$dist(u, v) = h(u) + h(v) - 2h\big(lca(u, v)\big)$$

Довжина шляху (довжина ребер динамічно змінюється)

У випадку, якщо, крім запитів на знаходження довжини шляху між деякими вершинами, є ще запити зміни ваги ребер, слід використовувати певні структури даних для підтримки актуальних на даний момент часу значень $h(u)$. Приклад розв’язання такої задачі описаний у статті «Покраска рёбер дерева» [http://e-maxx.ru/algo/tree_painting].

Максимальне (мінімальне) ребро на шляху

Розглянемо випадок, якщо необхідно визначати максимальне (мінімальне) ребро на шляху від $\boldsymbol{u}$ та $\boldsymbol{v}$. Очевидно, що за такої умови рівність на зразок $\boldsymbol{dist}(\boldsymbol{u},\boldsymbol{v}) = \boldsymbol{h}(\boldsymbol{u}) + \boldsymbol{h}(\boldsymbol{v}) - \boldsymbol{2h}\big(\boldsymbol{lca}(\boldsymbol{u},\boldsymbol{v})\big)$ не правильна. Тож позначимо через $\boldsymbol{maxEdge}(\boldsymbol{u},\boldsymbol{v})$ - максимальне ребро на шляху від $\boldsymbol{u}$ до $\boldsymbol{v}$. Тоді правильно, що $\boldsymbol{maxEdge}(\boldsymbol{u},\boldsymbol{v}) = \max\big(\boldsymbol{maxEdge}(\boldsymbol{lca}(\boldsymbol{u},\boldsymbol{v}),\boldsymbol{u}), \boldsymbol{maxEdge}(\boldsymbol{lca}(\boldsymbol{u},\boldsymbol{v}),\boldsymbol{v})\big)$, тобто ми розбили запит на два подібні, які мають вид «вершина пращур, вершина нащадок». Якщо вага ребер не змінюється з часом, то достатньо зберігати максимум (мінімум) на відповідному інтервалі (зручно використовувати метод двійкового підйому, зберігаючи пращура на певній висоті та максимальне (мінімальне) ребро на шляху до нього). Якщо ж вага ребер динамічно змінюється, то слід застосовувати інші структури даних.

Запити на піддереві

Тепер розглянемо запити, що стосуються піддерева певної вершини $\boldsymbol{u}$. Та для початку розглянемо *дерево ейлерового обходу* – спосіб представлення підвішеного неорієнтованого дерева масивом чисел. Побудувати це представлення можна кількома способами, кожен із яких має свої переваги та недоліки:

```
      1
     / \
    2   5
   / \
  3   4
```

1. Виписувати всі ребра дерева (орієнтовані) в порядку обходу DFS: `(1, 2),(2, 3),(3, 2),(2, 4),(4, 2),(2, 1),(1, 5),(5, 1).`

2. Зберігати вершини. Кожна вершина додається в масив двічі: при вході та при виході з цієї вершини. Кожному листку (крім, можливо, кореня)

при такому збережені відповідають два послідовні входження: `1, 2, 3, 3, 4, 4, 2, 5, 5, 1.`

3. Зберігати вершини. Але кожна вершина додається до масиву кожен раз, коли ми відвідуємо її (тобто коли йдемо від батька до цієї вершини, а також кожного разу, коли повертаємося від дитини): `1, 2, 3, 2, 4, 2, 1, 5, 1.`

Можна помітити, що будь-якому піддереву цього дерева завжди відповідає неперервний відрізок послідовних чисел. Отже, запити на піддеревах зводяться до запитів на відрізках.

Наприклад, якщо кожна вершина дерева має вагу, то необхідно опрацьовувати запити двох типів:

1) $add(v, x)$ - збільшити вагу вершини v на величину x;

2) $getSum(v)$ - знайти суму ваг вершин у піддереві вершини v.

Для розв'язання задачі можна побудувати дерево Фенвіка чи дерево відрізків на обході 2-го типу. Позначимо через $first(v)$ - позицію першої зустрічі вершини v, $last(v)$ - позицію останньої зустрічі вершини v. Тоді при появі запиту типу $add(v, x)$ необхідно додати до елементу з індексом $first(v)$ величину x, при запиті $getSum(v)$ - повернути значення суми на відрізку $[first(v), last(v)]$.

За допомогою певної модифікації можна рахувати суму ваг вершин на шляху від кореня до певної вершини. При запиті на зміну ваги $add(v, x)$ необхідно елемент з індексом $first(v)$ додати величину x, а елемент з індексом $last(v)$ зменшити на величину x. Тоді відповіді на запит пошуку суму ваг на шляху від кореня $root$ до вершини v буде рівна сумі на відрізку $[root, first(v)]$.

Також для знаходження LCA двох вершин може бути використаний третій тип представлення дерева. Зберігатимемо масив висот h, де $h[i] = height(v[i])$. Тоді $lca(u, v) = v\left[argmin\left(first(u), first(v)\right) \text{ у } h\right]$, рівність є правильною, оскільки $lca(u, v)$ – це найвища вершина між u та v в порядку обходу пошуку вглиб.

II. НАЙКОРОТШІ ШЛЯХИ В ГРАФАХ

2.1. Найкоротші шляхи з однієї вершини

У задачі про найкоротший шлях (shortest-paths problem) задається зважений орієнтований граф $G = (V, E)$ з ваговою функцією $w: E \to R$, що відображає ребра на їхню вагу, значення яких виражаються дійсними числами. ***Вага*** (weight) шляху $p = \langle v_0, v_1, \dots, v_k \rangle$ рівна сумарній вазі ребер, що входять до нього:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

Вага найкоротшого шляху (shortest-path weight) $\delta(u, v)$ з вершини u до вершини v визначається співвідношенням

$$\delta(u, v) = \begin{cases} \min\left\{w(p): u \xrightarrow{p} v\right\}, & \text{якщо існує шлях з } u \text{ до } v, \\ \infty, & \text{в іншому випадку} \end{cases}$$

Тоді за визначенням ***найкоротший шлях*** (shortest path) з вершини u до вершини v - це будь-який шлях, вага якого задовольняє співвідношення $w(p) = \delta(u, v)$.

Нижче буде описано методи розв'язання ***задачі про найкоротші шляхи з однієї вершини*** (single-source shortest-paths problem), у якій для заданого графу $G = (V, E)$ необхідно знайти найкоротші шляхи, які починаються у визначеній початковій вершині (source vertex) $s \in V$ та закінчуються в кожній з вершин $v \in V$. Призначений для розв'язання цієї задачі алгоритм дозволяє розв'язувати багато інших задач, у тому числі перераховані нижче.

Варіанти:

- **Задача про найкоротші шляхи в одну вершину.** Необхідно знайти найкоротші шляхи до заданої ***цільової вершини*** (destination vertex) t, які починаються з кожної із вершини v. Змінивши напрямок кожного ребра заданого графу, задачу можна звести до задачі з однією визначеною початковою вершиною.

- **Задача про найкоротші шляхи між заданою парою вершин.** Необхідно знайти найкоротший шлях із заданої вершини u до заданої вершини v. Якщо розв'язана задача пошуку найкоротших шляхів із заданої початкової вершини u, то ця задача теж розв'язується. Більше того, всі відомі для розв'язання даної задачі алгоритми мають такий же час роботи в гіршому випадку, що і найкращі алгоритми пошуку найкоротших відстаней з однієї вершини.
- **Задача про найкоротші шляхи між усіма вершинами.** Необхідно знайти найкоротший шлях із кожної вершини u до кожної вершини v. Цю задачу також можна розв'язати за допомогою алгоритму, призначеного для розв'язання задачі про одну визначену початкову вершину, однак зазвичай вона розв'язується швидше.

Ребра з від'ємною вагою

У деяких екземплярах задачі про найкоротший шлях із фіксованої початкової вершини ваги ребер можуть набувати від'ємного значення. Якщо граф $G = (V, E)$ не містить циклів із від'ємною вагою, що досяжні з визначеної початкової вершини s, то вага найкоротшого шляху $\delta(s, v)$ залишається цілком визначеною величиною для кожної вершини $v \in V$, навіть якщо він приймає від'ємне значення. Якщо ж такий цикл досяжний із початкової вершини s, ваги найкоротших шляхів перестають бути цілком визначеними величинами. У цій ситуації жоден шлях із початкової вершини s у будь-яку вершину циклу не може бути найкоротшим, тому що завжди можна знайти шлях із меншою вагою, який проходить по запропонованому «найкоротшому» шляху, а потім обходить цикл із від'ємною вагою. Якщо на певному шляху з вершини s до вершини v зустрічається цикл із від'ємною вагою, ми визначаємо $\delta(s, v) = -\infty$.

Цикли

Чи може найкоротший шлях містити цикл? Щойно ми переконалися в тому, що він не може містити цикл із від'ємною вагою. У нього також не може входити цикл із додатною вагою, оскільки в результаті видалення цього циклу

зі шляху вийде шлях, який виходить із тієї ж початкової вершини та закінчується в тій же вершині, але має меншу вагу. Тобто якщо $p = \langle v_0, v_1, \dots, v_k \rangle$ - шлях, а $c = \langle v_i, v_{i+1}, \dots, v_j \rangle$ - цикл із додатною вагою на цьому шляху ($v_i = v_j$ та $w(c) > 0$), то шлях $p' = \langle v_0, v_1, \dots, v_i, v_{j+1}, v_{j+2}, \dots, v_k \rangle$ має вагу $w(p') = w(p) - w(c) < w(p)$, отже, p не може бути найкоротшим шляхом з v_0 до v_k.

Залишаються тільки цикли з нульовою вагою. Проте зі шляху можна видалити цикл із нульовою вагою, у результаті чого вийде інший шлях із такою ж вагою. Отже, якщо існує найкоротший шлях із початкової вершини s до кінцевої вершини v, що містить цикл із нульовою вагою, то існує й інший найкоротший шлях між цими вершинами, у якому цикл не міститься. Тепер вважатимемо, що ми будемо знаходити найкоротші шляхи, і вони не матимуть циклів. Оскільки в будь-який ациклічний шлях у графі $G = (V, E)$ входить не більше $|V|$ різних вершин, у ньому міститься не більше $|V| - 1$ ребер. Отже, можна обмежитися тільки найкоротшими шляхами, що складаються не більше ніж з $|V| - 1$ ребер.

Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда (Bellman-Ford algorithm) розв'язує задачу про найкоротший шлях із однієї вершини в загальному випадку, коли вага кожного з ребер може бути від'ємною. Для заданого зваженого орієнтованого графу $G = (V, E)$ з початковою вершиною s та ваговою функцією $w: E \to R$ алгоритм Беллмана-Форда повертає логічне значення, що вказує, чи міститься в графі цикл із від'ємною вагою, що досяжний із початкової вершини s. Якщо такий цикл існує, алгоритм вказує, що рішення не існує. Якщо ж таких циклів немає, алгоритм повертає найкоротші шляхи та їх вагу для кожного з них.

У цьому алгоритмі використовуються послаблення, у результаті якого величина $v.d$, що є оцінкою ваги найкоротшого шляху від початкової вершини до кожної з вершин $v \in V$, поступово зменшується, аж поки не стане рівною фактичній вазі найкоротшого шляху $\delta(s, v)$. Значення TRUE повертається

алгоритмом тоді й тільки тоді, коли граф не містить циклів із від'ємною вагою, що досяжні із визначеної початкової вершини.

Вважатимемо, що для довільного дійсного числа $a \neq \infty$ виконуються співвідношення $a + \infty = \infty + a = \infty$. Крім того, щоб наші доведення зберігали силу при наявності циклів від'ємної ваги, будемо вважати, що для довільного дійсного числа $a \neq \infty$ виконуються співвідношення $a + (-\infty) = (-\infty) + a = -\infty$.

```
Bellman-Ford(G,w,s)
Initialize-Single-Source(G,s)
for i = 1 to |G.V| − 1
    for кожного ребра (u,v) ∈ G.E
        Relax(u,v,w)
for кожного ребра (u,v) ∈ G.E
    if v.d > u.d + w(u,v)
        return FALSE
return TRUE
```

```
Initialize-Single-Source(G,s)
for кожної вершини v ∈ G.V
    v.d = ∞
s.d = 0
```

```
Relax(u,v,w)
if v.d > u.d + w(u,v)
    v.d = u.d + w(u,v)
```

Час роботи алгоритму Беллмана-Форда складає $O(VE)$.

Отже, алгоритм Беллмана-Форда дозволяє розв'язати задачу про найкоротший шлях із фіксованої початкової вершини в загальному випадку, коли вага будь-якого ребра може бути від'ємною. Цей алгоритм вирізняється своєю простотою. До його переваг також відносять те, що він визначає, чи міститься в графі цикл із від'ємною вагою, що досяжний із початкової вершини.

Найкоротші шляхи з однієї вершини в орієнтованих ациклічних графах

Послаблюючи ребра зваженого орієнтованого ациклічного графа $G = (V, E)$ в порядку, визначеному топологічним сортування його вершин, найкоротші відстані з однієї вершини можна знайти за час $O(V + E)$. В орієнтованому ациклічному графі найкоротші шляхи завжди цілком визначені, оскільки, навіть якщо вага деяких ребер від'ємна, циклів із від'ємною вагою не існує.

Робота алгоритму починається з топологічного сортування орієнтованого ациклічного графу, яке встановить лінійне впорядкування вершин. Якщо шлях з вершини u до вершини v існує, то в топологічному сортуванні вершина u йде раніше вершини v. По вершинах, розташованих у топологічному порядку, прохід здійснюється лише один раз. При обробці кожної вершини виконується послаблення всіх ребер, що виходять з цієї вершини.

```
Dag-Shortest-Paths(G,w,s)
Топологічне сортування вершин графу G
Initialize-Single-Source(G,s)
for кожної вершини u в порядку топологічного сортування
    for кожної вершини v ∈ G.Adj[u]
        Relax(u,v,)
```

Алгоритм Дейкстри

Алгоритм Дейкстри розв'язує задачу пошуку найкоротших шляхів з однієї вершини у зваженому орієнтованому графі $G = (V, E)$ у випадку, коли вага кожного з ребер невід'ємна. Тому при подальшому описі припускається, що $w(u, v) \geq 0$. За правильної реалізації час роботи алгоритму Дейкстри менше часу роботи алгоритму Беллмана-Форда.

В алгоритмі Дейкстри підтримується множина вершин S, для кожної з вершин якої вже порахована кінцева вага найкоротшого шляху від вершини s. У цьому алгоритмі почергово вибирається вершина $u \in V - S$, якій на поточному етапі відповідає мінімальна оцінка найкоротшого шляху. Після додавання цієї вершини u до множини S проводиться послаблення всіх ребер,

що виходять із неї. У наведеній нижче реалізації використовується неспадна черга з пріоритетами Q, що складається з вершин, у ролі ключових атрибутів яких виступають значення їх атрибутів d.

```
Dijkstra(G,w,s)
1 Initialize-Single-Source(G,s)
2 S = ∅
3 Q = G.V
4 while Q ≠ ∅
5     u = Extract-Min(Q)
6     S = S ∪ {u}
7     for кожної вершини v ∈ G.adj[u]
8         Relax(u,v,w)
```

Коректність алгоритму Дейкстри

Після опрацювання алгоритмом Дейкстри зваженого орієнтованого графу $G = (V, E)$ з невід'ємною ваговою функцією w та фіксованою початковою вершиною s для всіх вершин $u \in V$ виконується рівність $u.d = \delta(s, u)$.

Доведення. Скористаємося наступним інваріантом циклу:

На початку кожної ітерації циклу **while** у рядках 4-8 для кожної вершини $v \in S$ виконується рівність $v.d = \delta(s, v)$.

Достатньо показати, що для кожної вершини $u \in V$ рівність $u.d = \delta(s, u)$ виконується в момент її додавання до множини S. Після того, як буде продемонстрована справедливість рівності $u.d = \delta(s, u)$, на основі властивості верхньої межі покажемо, що ця рівність продовжує виконуватися і в подальшому.

Ініціалізація. Спочатку $S = \emptyset$, тому інваріант виконується.

Збереження. Потрібно показати, що при кожній ітерації рівність $u.d = \delta(s, u)$ виконується для кожної вершини, що додана до множини S. Скористаємося методом «від супротивного». Щоб отримати протиріччя, припустимо, що u - перша додана до множини S вершина, для якої $u.d \neq \delta(s, u)$. Зосередимо увагу на ситуації, що склалася на початку тієї ітерації циклу

while, в якій вершина u додається до множини S. Проаналізувавши найкоротший шлях з вершини s до вершини u, можна буде отримати протиріччя, яке полягає в тому, що в той момент справедлива рівність $u.d = \delta(s,u)$. Повинна виконуватися умова $u \neq s$, оскільки s - перша вершина, що додана до множини S, і в момент її додавання виконується рівність $s.d = \delta(s,s) = 0$. З умови $u \neq s$ випливає також те, що безпосередньо перед додаванням вершини u до множини S вона не є порожньою. З вершини s до вершини u повинен йти якийсь шлях, оскільки в протилежному випадку відповідно до умови відсутності шляху виконується співвідношення $u.d = \delta(s,u) = \infty$, яке заперечує справедливість припущення, що $u.d \neq \delta(s,u)$. Оскільки хоча б один шлях існує, повинен існувати й найкоротший шлях p з вершини s до вершини u. Перед додаванням вершини u до множини S шлях p з'єднує вершину з множини S (а саме – вершину s) з вершиною з множини $V - S$ (а саме – з вершиною u). Розглянемо вершину y на шляху p, що належить множині $V - S$, а також припустимо, що на цьому шляху їй передує вершина $x \in S$. Тоді шлях p можна розкласти на складові: p_1 - шлях від s до x, $x \to y$, p_2 - шлях від y до u (p_1 або p_2 можуть не містити жодного ребра).

Стверджується, що $y.d \neq \delta(s,y)$ при додаванні u до S. Для доведення цього твердження помітимо, що $x \in S$. Потім, оскільки ми обираємо u як першу вершину, для якої $u.d \neq \delta(s,u)$, при її додавання до S. При додаванні x в S маємо $x.d \neq \delta(s,x)$. Ребро (x,y) в цей момент було послаблено, і наше твердження випливає з властивості збіжності.

Тепер можна отримати протиріччя, яке дозволить довести, що $u.d = \delta(s,u)$. Оскільки на найкоротшому шляху з вершини s до вершини u вершина y знаходиться перед вершиною u й вага кожного з ребер виражається невід'ємним значенням (це особливо важливо для ребер, з яких складається шлях p_2), виконується нерівність $\delta(s,y) \leq \delta(s,u)$, тому

$y.d = \delta(s,y) \leq \delta(s,u) \leq u,d$ (згідно з властивістю верхньої межі).

Проте оскільки і вершина u, і вершина y під час вибору вершини u в рядку 5 знаходилися в множині $V - S$, виконується нерівність $u.d \leq y.d$. Отже, обидві нерівності фактично є рівностями, тобто

$y.d = \delta(s, y) = \delta(s, u) = u.d.$

Тоді $u.d = \delta(s, u)$, що суперечить нашому вибору вершини u. Отже, приходимо до висновку, що під час додавання вершини u до множини S виконується рівність $u.d = \delta(s, u)$, а відповідно, вона виконується і в подальшому.

Завершення. По закінченні алгоритму $Q = \emptyset$. З цього факту та інваріанту $Q = V - S$ випливає, що $S = V$. Отже, для всіх вершин $u \in V$ виконується рівність $u.d = \delta(s, u)$.

Час роботи алгоритму Дейкстри

Наскільки швидко працює алгоритм Дейкстри? У ньому підтримується неспадна черга з пріоритетами Q та трьома операціями, характерними для черги з пріоритетами: `Insert` (явно викликається в рядку 3), `Extract-Min` (рядок 5) і `Decrease-Key` (неявно є в процедурі `Relax`, яка викликається в рядку 8). Процедура `Insert`, як і процедура `Extract-Min`, викликається по одному разу для кожної вершини. Оскільки кожна вершина $u \in V$ додається до множини S рівно по одному разу, кожне ребро у списку суміжності $Adj[u]$ обробляється в циклі **`for`** у рядках 7 та 8 рівно по одному разу протягом алгоритму. Оскільки загальна кількість ребер у всіх списках суміжності рівна $|E|$, всього виконується $|E|$ ітерацій цього циклу **`for`**, а отже, не більше $|E|$ викликів `Decrease-Key`.

Час роботи алгоритму Дейкстри залежить від реалізації неспадної черги з пріоритетами. Спочатку розглянемо випадок, коли неспадна черга з пріоритетами підтримується за рахунок того, що всі вершини пронумеровані від 1 до $|V|$. Атрибут $v.d$ просто вміщується в елемент масиву з індексом v. Кожна операція `Insert` і `Decrease-Key` займають час $O(1)$, а кожна

операція `Extract-Min` – час $O(V)$ (оскільки в ній виконується пошук по всьому масиву); в результаті повний час роботи алгоритму рівний $O(V^2 + E) = O(V^2)$.

Якщо граф достатньо розріджений, зокрема, якщо кількість вершин і ребер у ньому зв'язана співвідношенням $E = o(V^2/lgV)$, алгоритм можна зробити більш ефективним шляхом реалізації неспадної черги з пріоритетами за допомогою бінарної неспадної піраміди. Кожна операція `Extract-Min` займає час $O(lgV)$. Як і раніше, всього таких операцій - $|V|$. Час, що необхідний для побудови неспадної піраміди, рівний $O(V)$. Кожна операція `Decrease-Key` виконується за час $O(lgV)$, а всього таких операцій виконається не більше $|E|$. Тому повний час роботи алгоритму складає $O((V + E)lgV)$, що рівно $O(ElgV)$, якщо всі вершини досяжні з початкової вершини. Цей час роботи є кращим у порівнянні з часом роботи прямої реалізації $O(V^2)$, якщо $E = o(V^2/lgV)$.

Фактично можна досягти часу роботи алгоритму $O(VlgV + E)$, якщо неспадна черга з пріоритетами реалізується за допомогою фібоначевої піраміди. Амортизована вартість кожної з $|V|$ операцій `Extract-Min` рівна $O(lgV)$, а кожен виклик процедури `Decrease-Key` (всього їх не більше $|E|$) потребує лише $O(1)$ амортизованого часу.

Алгоритм Дейкстри має певну схожість як із пошуком вшир, так і з алгоритмом Пріма для побудови мінімального кістякового дерева.

Приклади задач на алгоритм Дейкстри

Класичні приклади задач, які можна розв'язати за допомогою алгоритму Дейкстри, – це найкоротший шлях із урахуванням заторів на дорогах, найкоротший маршрут із урахуванням розкладу громадського транспорту.

Розв'яжемо задачу знаходження найкоротшого шляху між двома вершинами s і t, якщо вага шляху визначатиметься як сума двох максимальних ребер на цьому шляху. Цю задачу неможливо розв'язати за допомогою

алгоритму Дейкстри у явному вигляді. Проте можна запустити алгоритм Дейсктри із двох кінців: з початкової вершини шляху та з кінцевої вершини шляху по транспонованому графу, з ваговою функцією – максимальне ребро на шляху. Тоді ми порахуємо $d[v]$ - найкоротший шлях з вершини s до v та $e[v]$ - найкоротший шлях від v до t. Виконавши такий попередній перерахунок, можна розв'язати задачу знаходження найкоротшого шляху в графі з вершини s до вершини v з ваговою функцією – сума двох максимальних ребер, де максимальним ребром шляху є ребро з вершини u до вершини v. Якщо вага цього ребра w більша або рівна $max(d[u], e[v])$, то такий шлях існує і його вага рівна $w + max(d[u], e[v])$. У протилежному випадку такого шляху не існує. Відповіддю на задачу буде мінімум по всім існуючим шляхам.

2.2. Найкоротші шляхи між усіма парами вершин

Є орієнтований або неорієнтований зважений граф $G = (V, E)$, у якому $|V| = n$. Необхідно знайти значення всіх величин $d_{i,j}$ - довжини найкоротшого шляху з вершини i до вершини j.

Припускається, що граф не містить циклів від'ємної ваги (тоді відповідь між певною парою вершин може просто не існувати – вона буде нескінченно малою).

Алгоритм Флойда-Воршалла

Основна ідея алгоритму – розбиття процесу пошуку найкоротших шляхів на ***фази***.

Перед k-ою фазою ($k = 1 \dots n$) вважається, що в матриці відстаней $d[][]$ збережені довжини таких найкоротших шляхів, які містять в якості проміжних вершин тільки вершини з множини $\{1,2, \dots, k-1\}$ (вершини графу нумеруються, починаючи з одиниці).

Іншими словами, перед k-ою фазою величина $d[i][j]$ рівна довжині найкоротшого шляху з вершини i до вершини j, якщо цьому шляху

дозволяється заходити тільки у вершини з номерами, що менші за k (початок та кінець шляху не враховуються).

Не складно переконатися, для того, щоб ця властивість виконувалася для першої фази, достатньо у матриці відстаней $d[][]$ записати матрицю суміжності графа: $d[i][j] = g[i][j]$ – вага ребра з вершини i до вершини j. При цьому, якщо між якимись вершинами ребра не існує, то слід записати ∞. З вершини в саму себе завжди слід записувати величину 0, це критично для алгоритму.

Нехай тепер ми знаходимося на k-ій фазі, і хочемо перерахувати матрицю $d[][]$ так, щоб вона відповідала вимогам уже для $k+1$-ї фази. Зафіксуємо вершини i та j. Виникає два принципово різних випадки:

- Найкоротший шлях з вершини i до вершини j, якому дозволено додатково проходити через вершини $\{1,2,\dots,k\}$, ***збігається*** з найкоротшим шляхом, якому дозволено проходити через вершини множини $\{1,2,\dots,k\}$. В цьому випадку величина $d[i][j]$ не зміниться при переході від k-ї до $k+1$-ї фази.
- «Новий» найкоротший шлях став ***кращим*** за «старий» шлях. Це означає, що «новий» найкоротший шлях проходить через вершину k. Можна помітити, якщо розбити «новий» шлях вершиною k на дві частини (перша – шлях від i до k, інша – шлях від k до j), то кожна з цих частин уже не заходить до вершини k. Отже, виходить, що довжини кожної з цих частин були пораховані на $k-1$ фазі або ще раніше, і достатньо взяти суму $d[i][k] + d[k][j]$, ця сума й буде довжиною «нового» найкоротшого шляху.

Об'єднуючи ці два випадки, отримуємо, що на k-ій фазі необхідно перерахувати довжини шляхів між усіма парами вершин i та j наступним чином:

```
new_d[i][j] = min(d[i][j], d[i][k] + d[k][j])
```

Отже, вся робота, яку необхідно виконати на k-ій фазі, – це перебрати всі пари вершин та перерахувати довжину найкоротшого шляху між ними. В результаті виконання n фаз у матриці відстаней $d[i][j]$ буде рівне довжині

найкоротшого шляху між вершинами i та j, або ∞, якщо шляху між цими вершинами не існує.

Одним зауваженням до описаного алгоритму буде те, що можна не створювати окрему матрицю $new\_d[][]$ для тимчасової матриці найкоротших шляхів на k-ій фазі: всі зміни можна виконувати одразу в матриці $d[][]$. Насправді, якщо ми покращили (зменшили) якесь значення в матриці відстаней, ми не могли погіршити тим самим довжину найкоротшого шляху для якихось інших пар вершин, що будуть опрацьовані пізніше.

Асимптотика алгоритму складає $O(n^3)$.

Знаходження циклу від'ємної ваги

Оскільки алгоритм Флойда-Воршалла послідовно послаблює відстань між усіма парами вершин (i, j), в тому числі і тими, у яких $i = j$, а початкова відстань між парою (i, i) рівна нулю, то послаблення може виконатися лише за наявності вершини k такої, що $d[i][k] + d[k][i] < 0$, що еквівалентно наявності від'ємного циклу, що проходить через вершину i.

Для пошуку циклу від'ємної ваги необхідно після завершення роботи алгоритму знайти вершину i, для якої $d[i][i] < 0$, та вивести найкоротший шлях між парою вершин (i, i). При цьому слід враховувати, що за наявності від'ємного циклу відстані можуть зменшуватися експоненціально. Для запобігання переповнення всі обчислення варто обмежувати знизу величиною $-\infty$, або перевіряти наявність від'ємних чисел на головній діагоналі під час перерахунку матриці.

III. ПАРОСПОЛУЧЕННЯ В ГРАФАХ

Паросполученням або ***незалежною множиною ребер*** у простому графі $G = (V, E)$ називають множину ребер, у якій ніякі два ребра не суміжні. Паросполучення графа G називають ***максимальним***, якщо воно не міститься в жодному паросполученні з більшою кількістю ребер, і ***найбільшим***, якщо кількість ребер у ньому найбільша серед усіх паросполучень графа G.

3.1. Найбільше паросполучення у дводольних графах

Дводольним графом (також ***біграфом***, ***двочастковим графом***) називається граф, множина вершин якого може бути розбита на дві підмножини так, що кожне ребро графа має одну вершину з першої підмножини й одну з другої.

Задача полягає в знаходженні найбільшого паросполучення у заданому дводольному графі.

Введемо такі означення:

- *ланцюгом* назвемо певний простий шлях;
- *доповнюючим ланцюгом* відносно певного паросполучення назвемо такий ланцюг парної довжини, початкова і кінцева вершини якого не належать цьому паросполученню. Вершина належить паросполученню, якщо існує ребро, яке виходить із цієї вершини і належить паросполученню.

Згідно з теоремою Бержа паросполучення є максимальним тоді й тільки тоді, коли не існує жодного доповнюючого ланцюга.

На цій теоремі ґрунтується алгоритм Куна. На кожній ітерації відшукується доповнюючий ланцюг. Якщо ланцюг не знайдено, то алгоритм завершується. Доповнюючий ланцюг шукається за допомогою пошуку вглиб так: перебирається перша вершина ланцюга, яка не належить паросполученню. Далі розглядаються всі ребра, суміжні з поточною вершиною. Нехай поточне ребро - (u, v). Якщо вершина v не належить паросполученню, то ми отримали шуканий доповнюючий ланцюг. Інакше існує ребро (v, t), яке належить паросполученню. Спробуємо продовжити пошук доповнюючого ланцюга з вершини t відповідним викликом пошуку вглиб.

Нехай знайдений маршрут є послідовністю вершин $v_{p_1}, v_{p_2}, \ldots, v_{p_k}$, вершини v_{p_1}та v_{p_k} не належать паросполученню, решта належать. Вилучимо з паросполучення всі ребра, які належать цьому шляху - ребра (v_{p_2}, v_{p_3}), (v_{p_4}, v_{p_5}), ..., $(v_{p_{k-2}}, v_{p_{k-1}})$. Їхня кількість – $(k-2)/2$. Додамо в паросполучення ребра (v_{p_1}, v_{p_2}), (v_{p_3}, v_{p_4}), ..., $(v_{p_{k-1}}, v_{p_k})$. Їх кількість – $k/2$. Отже, кількість

ребер у паросполученні збільшується. Складність алгоритму Куна можна оцінити як $O(nm)$ або $O(n^3)$, де $n = |V|$, а $m = |E|$.

IV. ПОТОКИ В ГРАФАХ

Мережу можна представити як систему, яка транспортує певний продукт з однієї точки в іншу. Цим продуктом можуть бути люди, електроенергія, природний газ, нафта та багато іншого.

Використовуючи таке представлення, розглянемо транспортну мережу як орієнтований граф $G(V, E)$, де кожному ребру $e = (v, u)$ відповідає додатне дійсне число $c(e)$, яке називається пропускною спроможністю ребра e. Якщо між вершинами немає ребра, то пропускна спроможність дорівнює нулю.

Потоком із джерела s у стік t назвемо дійсну функцію: $f: V \times V \to R$, яка задовольняє такі властивості:

- обмеження пропускної здатності: $f(u, v) \leq c(v, u)$. Потік уздовж ребра не може перевищувати його пропускної здатності;
- антисиметричність: $f(u, v) = -f(v, u)$. Потік з вершини u в v має бути протилежним до потоку з v в u;
- збереження потоку: $\sum_{v \in V} f(u, v) = 0, \forall u \in V, u \neq s, u \neq t$. Жодна вершина, крім джерела та стоку, не поглинає і не генерує потік.

Величиною потоку називається число $|f| = \sum_{v \in V} f(s, v)$.

У теорії оптимізації та теорії графів задача про максимальний потік полягає у знаходженні такого потоку, величина якого максимальна.

Відомим є алгоритм Форда-Фалкерсона для знаходження максимального потоку. Алгоритм полягає в ітеративному відшуканні шляху з джерела в стік по ненасичених ребрах. Ребро називається насиченим, якщо величина потоку, який проходить через цю вершину, рівна пропускній здатності цієї вершини. Умовою припинення є відсутність шляху з джерела в стік.

Існують такі модифікації цього алгоритму:

- Алгоритм Едмондса-Карпа. На кожній ітерації знаходимо найкоротший шлях з джерела в стік. Складність алгоритму - $O(|V||E^2|)$.

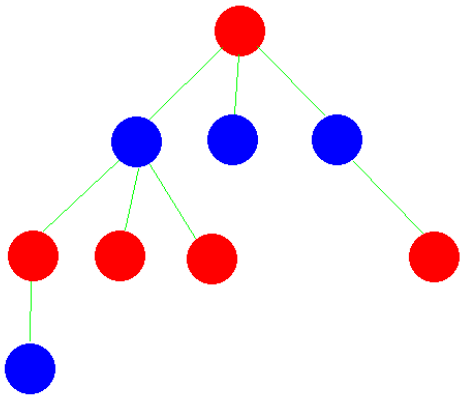
- Алгоритм Дініца. На кожній ітерації, використовуючи пошук ушир, визначаємо відстані від джерела до всіх вершин у залишковій мережі. Будуємо граф, який містить лише такі ребра залишкової мережі, на яких ця відстань зростає на 1. Виключаємо з графа усі тупикові вершини з інцидентними їм ребрами, поки всі вершини стануть не тупиковими. (Тупиковою називається вершина, в яку не входить і з якої не виходить жодне ребро, крім джерела та стоку.) На отриманому графі відшукуємо найкоротший шлях, що збільшується (ним буде будь-який шлях з s в t). Виключаємо із залишкової мережі ребро з мінімальною пропускною здатністю, знову виключаємо тупикові вершини, і так поки ще існують шляхи, що збільшуються. Складність алгоритму – $O(|V^2||E|)$. У випадку одиничних пропускних здатностей ребер – $O(|V||E|)$.

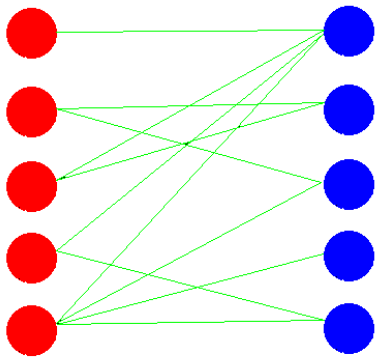
V. NP-ПОВНІ ЗАДАЧІ

У теорії графів можна виділити окремий клас задач - NP. Особливістю задач цього класу є те, що для знаходження оптимального розв'язку не відомо алгоритмів із поліноміальною складністю. Тобто для знаходження розв'язку на довільному графі необхідно перебирати всі можливі варіанти. Тому якщо при розв’язуванні задачі вона звелася до певної відомої NP задачі, пошук оптимального розв'язку можна припинити і єдиним варіантом є перебір розв’язків. Не зважаючи на це, якщо граф має специфічну структуру (дерево, дводольний граф), для деяких NP задач існують швидкі алгоритми відшукання розв'язку.

5.1. Задача про розфарбування вершин графа

Розглянемо задачу розфарбовування вершин графа. В цій задачі необхідно знайти мінімальну кількість кольорів, що існує спосіб "розфарбувати" вершини таким чином, щоб жодна пара з'єднаних ребром вершин не мала однакового кольору. В загальному випадку розв'язок цієї задачі має експоненційну складність. Але якщо граф є дводольним або представляє собою дерево, то необхідно всього два кольори.





5.2. Задача про вершинне покриття графа

Розглянемо іншу задачу - задача вершинного покриття графа. Її можна сформулювати так: знайти мінімальну за розміром підмножину $S \subseteq V$, що для довільного ребра хоча б одна з інцидентних йому вершин належить до S. В загальному випадку для знаходження оптимального розв'язку необхідно перебирати всі можливі варіанти. Розглянемо випадок, коли граф представляє собою дерево. Тоді задачу можна ефективно розв'язати жадібним алгоритмом або застосувавши динамічне програмування.

Позначимо через

- a_u – розв'язок на піддереві вершини u, такий, що вершина u входить до мінімального покриття;
- b_u - розв'язок, коли вершина u не входить до покриття.

Для знаходження значень a_u, b_u необхідно спочатку розв'язати цю ж задачу для всіх дітей вершини u. Якщо u – листок, то $a_u = 1$, $b_u = 0$, інакше перерахунок динаміки буде таким:

$$a_u = \sum_{v \in Ch(u)} \min(a_v, b_v)$$

$$b_u = \sum_{v \in Ch(u)} a_v$$

Якщо ж заданий граф є дводольним, розв'язок буде будуватися за допомогою найбільшого паросполучення. Для того, щоб показати оптимальність цього розв'язку, необхідно довести, що

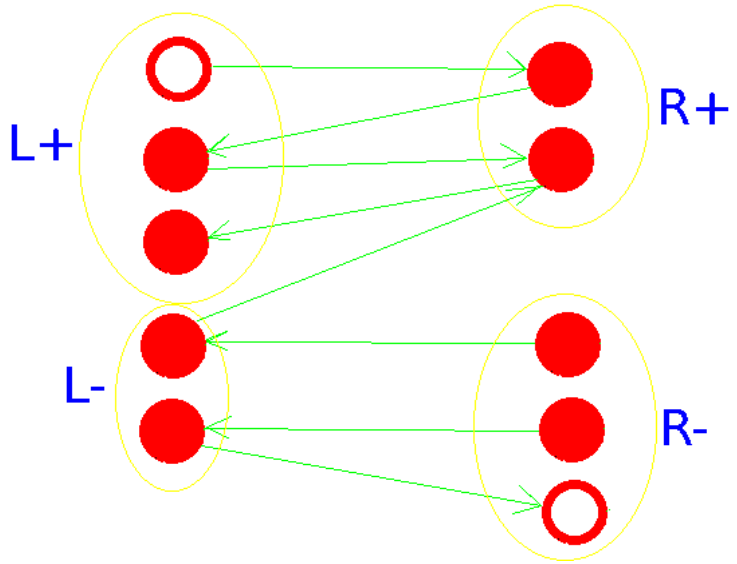
1. Величина мінімального покриття не менша за розмір найбільшого паросполучення.

2. Із максимального паросполучення завжди можна отримати коректне покриття такого ж розміру.

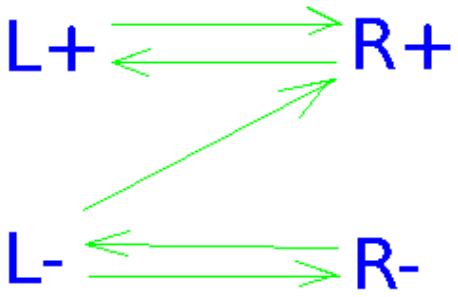
Доведемо попередні твердження:

1. Розглянемо лише ребра, які належать максимальному паросполученню. Усі вершини, які належать максимальному паросполученню, різні. І для кожного ребра має забезпечуватися умова, що хоча б одна з його вершин належить покриттю. Це означає, що кількість вершин у покритті повинна бути не меншою, ніж кількість ребер у максимальному паросполученні.

2. Для побудови покриття з паросполучення змінимо наш граф так: орієнтуємо ребра, що входять у паросполучення, з правої долі в ліву, а ті, що не входять, – з лівої у праву. Запустимо пошук углиб з усіх вершин, які не входять у пароутворення. Позначимо множини вершин, відвіданих процедурою через L^+ та R^+ для лівої та правої долі відповідно. Невідвідані вершини позначимо аналогічно через L^- та R^-.



Тоді граф матиме такий вигляд:



У якості мінімального покриття візьмемо множини L^- та R^+. Потужність цієї множини буде збігатися з кількістю ребер у найбільшому пароутворенні. З рисунка видно, що будь-яке ребро буде інцидентним одній із вершин цієї множини. Таким чином ми довели, що це і буде розв'язком задачі.

Задачі та ідеї розв'язання

A. Evacuation

Сталося жахливе – вороже налаштовані особи захопили секретну лабораторію в країні Ужляндія. Лабораторія складається з n кімнат, деякі з яких з'єднані коридорами. Усього в лабораторії $n-1$ коридор, і відомо, що з кожної кімнати можна дістатися до будь-якої іншої. Інакше кажучи, лабораторія представляє собою дерево з n вершинами. Для кожного коридору відомий час l_i, необхідний для того, щоб дістатися з одного його кінця в інший. По коридорах можна рухатися в обох напрямках.

У кожній кімнаті лабораторії знаходиться по одній людині. Вороже налаштовані особи збираються вбити всіх цих людей, проте вони залишили їм шанс на порятунок. Лабораторія влаштована так, що в листках цього дерева знаходяться виходи. Листком у дереві називатимемо вершину, степінь якої 0 або 1. Якщо людина дістанеться до виходу – вона врятована. У кожному коридорі вороже налаштовані особи встановили детонатор. Для кожного з них відомий момент часу t_i, коли він вибухне. Після того, як детонатор вибухає, по коридору, у якому він знаходився, рухатися більше неможливо, і люди, які знаходилися в ньому в цей момент, гинуть. Якщо в момент вибуху людина добігає до кінця коридору, вона виживає.

Сміливі захисники Ужляндії не встигають прийти на допомогу цим бідолахам, тому їм потрібно рятуватися самотужки. Від Вас же вимагається дізнатися – яка максимальна кількість людей зможе врятуватися.

Формат вхідних даних

У першому рядку записане одне число n ($1 \le n \le 10^5$) – кількість кімнат у лабораторії. У кожному наступному $n-1$ рядку знаходяться чотири числа a_i, b_i, l_i та t_i ($1 \le a_i, b_i \le n$; $1 \le l_i \le 10^4$; $1 \le t_i \le 10^9$) – номера кімнат, які з'єднує цей коридор, час, необхідний для того, щоб дістатися від одного його кінця до іншого, та момент часу, у який у цьому коридорі спрацює детонатор.

Гарантується, що лабораторія представляє собою дерево.

Формат вихідних даних

Виведіть одне ціле число – максимальну кількість людей, які зможуть врятуватися.

Приклади

stdin	stdout
5 3 5 2 3 1 3 3 2 4 5 2 4 2 5 1 2	5
7 1 6 4 5 3 4 3 3 5 1 4 4 7 4 2 4 2 4 4 6 3 1 3 4	6

Пояснення до прикладу

У першому прикладі людина, яка знаходиться в кімнаті номер 5, біжить до кімнати 4, людина з кімнати 3 також рухається в цю ж кімнату, зверніть увагу, що вона досягає туди в момент вибуху, але встигає врятуватися.

У другому прикладі людина з 4-ої кімнати рухається в кімнату номер 2, людина з кімнати 1 – до кімнати 5, а людина, що знаходиться в 3-й кімнаті, не встигає врятуватися.

Розв’язання задачі «A. Evacuation»

Розв’язок будемо шукати динамікою по дереву. Підвісимо дерево. Нехай dp_v – максимальний момент часу, у який ми можемо почати бігти вниз по дереву, щоб урятуватися, якщо ми можемо це зробити, або -1, якщо це неможливо. Для листків поставимо $dp_v = inf$. Тоді перерахунок буде таким: $dp_v = \max(-1, \max_u(\min(dp_u, t_{v,u}) - l_{v,u}))$, де u – дочірня вершина. Але крім того, щоб іти вниз, ми також можемо піднятися до предка, і звідти шукати

вихід. Тому необхідний другий запуск пошуку вглиб із перерахунком $dp_u = \max(dp_u, \min(dp_v, t_{v,u}) - l_{v,u})$. Таким чином ми врахуємо випадок, коли необхідно піднятися з вершини u до її предка, а вже звідти шукати вихід.

B. Gift

Андрію подарували DAG (Directed Acyclic Graph – орієнтований ациклічний граф), але він хоче максимально його прокачати, а саме: додати в нього якомога більше ребер таким чином, щоб граф все ще залишався DAG-ом. Допоможіть Андрію – знайдіть максимальну кількість ребер, які можна додати, та виведіть ці ребра.

Формат вхідних даних

Перший рядок містить число N $(1 \leq N \leq 1500)$ – кількість вершин. Далі йдуть N рядків по N символів кожен, які задаються матрицею суміжности графа, який подарували Андрію. Якщо в графі є ребро із вершини x в y, то у стовпчику номер y рядка номер x стоїть 1, інакше 0. Вершини нумеруються з 1.

Формат вихідних даних

У першому рядку виведіть максимальну кількість ребер, а далі виведіть самі ребра, які необхідно додати. Якщо є декілька варіантів додати таку кількість ребер, що задовольняє умову задачі, то виведіть лексикографічно мінімальну послідовність ребер. Одна послідовність ребер лексикографічно менша іншої, якщо перше ребро, яке в них не збігається, менше у першій послідовності, ніж у другій. Ребра порівнюються як послідовності двох чисел.

Приклади

stdin	stdout
3 010 000 000	2 1 3 2 3

Розв'язання задачі «B. Gift»

Важливо зрозуміти, що максимальна кількість ребер, які можуть бути в графі, а він при цьому залишався ациклічним, рівна C_n^2 (з кожної вершини v можна провести ребро у вершину u, якщо у топологічному сортуванні вершина v знаходиться лівіше вершини u). Очевидно, що додавати вже існуючі ребра не потрібно.

Тепер залишається побудувати лексикографічно мінімальну послідовність ребер. Робитимемо це таким чином. На кожному кроці вибиратимемо вершину з максимальним номером та кількістю ребер, що виходять із неї, рівною 0. Проведемо з усіх інших вершин графу ребра в цю вершину та видалимо її з графу. Продовжуватимемо виконувати це, поки граф не залишиться порожнім.

C. Cubes

Батьки подарували Сергієві набір дитячих кубиків. Оскільки Сергій скоро піде в школу, вони купили йому кубики з буквами. На кожній із шести граней кожного кубика написана буква.

Тепер Сергій хоче похвалитися перед старшою сестрою, що навчився читати. Для цього він хоче скласти з кубиків її ім'я. Але це виявилося досить складно зробити, адже різні літери можуть перебувати на одному й тому ж кубику, і тоді Сергій не зможе використовувати обидві літери в слові. Правда, одна і та ж буква може зустрічатися на різних кубиках. Допоможіть Сергію!

Дано набір кубиків і ім'я сестри. З'ясуйте, чи можна викласти її ім'я за допомогою цих кубиків, і якщо так, то в якому порядку слід викласти кубики.

Формат вхідних даних

У першому рядку число N $(1 \leq N \leq 100)$ – кількість кубиків у наборі Сергія. У другому рядку записано ім'я сестри – слово, що складається тільки з великих латинських літер, не довше 100 символів. Наступні N рядків містять по 6 літер (тільки великі латинські літери), які записані на відповідному кубику.

Формат вихідних даних

У першому рядку виведіть "YES", якщо можливо викласти ім'я сестри Сергія, "NO" в протилежному випадку.

Якщо відповідь "YES", у другому рядку виведіть M різних цілих чесел із діапазону $1 \ldots N$, де M – кількість букв в імені сестри Сергія. i-е число має бути номером кубика, який необхідно покласти на i-е місце при складанні імені. Кубики нумеруються з 1, у тому порядку, в якому вони задані у вхідному файлі. Якщо рішень декілька, виведіть будь-яке.

Приклади

stdin	stdout
4 ANN ANNNNN BCDEFG HIJKLM NOPQRS	NO
5 HELEN ABCDEF GHIJKL MNOPQL STUVWN EIUOZK	YES 2 1 3 5 4

Розв'язання задачі «C. Cubes»

Ця задача зводиться до знаходження максимального паросполучення в дводольному графі. Лівою долею будуть кубики, правою – букви імені. Проведемо ребро від i-ої вершини лівої долі до j-ої вершини правої долі, якщо i-ий кубик містить j-у літеру. Якщо кількість ребер у максимальному паросполученні менша за кількість літер в імені, то розв'язку немає. Інакше для відновлення відповіді необхідно знайти, з якою вершиною лівої долі з'єднана кожна з вершин правої долі.

D. Segments

Дано N $(1 \leq N \leq 250)$ відрізків у 2D-площині, кожен із яких або вертикальний, або горизонтальний, та заданий двома своїми кінцями – точками $(x1_i, y1_i)$ та $(x2_i, y2_i)$ $(1 \leq x1_i, y1_i, x2_i, y2_i \leq 10^9)$.

Вам необхідно обрати якомога більше відрізків із цієї множини так, щоб ніякі два з них не перетиналися.

Формат вхідних даних

Перший рядок містить єдине число N. Далі йдуть N рядків, кожен із яких складаться з чотирьох чисел: $x1$, $y1$, $x2$, $y2$ – кінці відповідного відрізку.

Формат вихідних даних

Виведіть одне число – максимальну кількість відрізків, яку можна обрати, щоб жодні два з них не перетиналися.

Приклади

stdin	stdout
3 4 5 10 5 6 2 6 12 8 3 8 5	2

Розв’язання задачі «D. Segments»

Розглянемо граф, у якому вершинами будуть задані відрізки. Проведемо ребро для кожної пари вершин, якщо відповідні їм відрізки перетинаються. Оскільки вертикальні (і горизонтальні) відрізки між собою не перетинаються, то такий граф буде дводольним. Тоді задача зводиться до відшукання максимальної за кількістю вершин множини, такої, що жодна пара вершин із цієї множини не з’єднана ребром.

Сформулюємо наступне твердження: кількість вершин у такій множині не перевищує $n - k$, де n – кількість вершин у графі, k – величина максимального паросполучення в графі. Розглянемо вершини, що належать максимальному паросполученню. Оскільки для кожної пари вершин, з’єднаних ребром, лише

одна з вершин може належати до нашої множини, пари не перетинаються, а кількість пар – k, то серед $2 \cdot k$ вершин, що належать максимальному паросполученню, лише k можна включити до нашої множини. А це і доводить наше твердження.

Використовуючи позначення, введені в лекційній частині, в якості відповіді ми можемо обрати вершини L^+ R^-. Очевидно, потужність такої множини рівна $n - k$ і жодна пара вершин не з'єднана ребром.

E. Paths

Заданий неорієнтований зв'язний граф з n вершин та $n - 1$ ребра. Необхідно для кожного ребра порахувати сумарну довжину простих шляхів, що проходять через це ребро. Довжина шляху – кількість ребер у ньому.

Формат вхідних даних

У першому рядку записане ціле число n ($2 \leq n \leq 3 \cdot 10^5$). Наступні $n - 1$ рядків містять пари чисел від 1 до n – ребра графа.

Формат вихідних даних

Виведіть $n - 1$ рядок. i-й рядок повинен містити ціле число – відповідь для i-го ребра.

Приклади

stdin	stdout
5 1 2 2 3 2 4 5 1	13 8 8 9

Розв’язання задачі «E. Paths»

Підвісимо дерево за першу вершину. Першим пошуком углиб порахуємо для кожної вершини v величину $s_1[v]$ – кількість вершин у піддереві вершини v, та $s_2[v]$ – сумарну довжину всіх шляхів від вершини v вниз. Другим пошуком углиб по дереву для кожного ребра порахуємо відповідь. Кожне ребро має два кінці – нижній кінець a та верхній кінець b. Кількість вершин знизу – $s_1[a]$, кількість вершин зверху $t_1[b] = n - s_1[a]$, сумарна довжина шляхів знизу – $s_2[a]$, сумарну довжину шляхів згори $t_2[a]$ можна обчислювати при спуску вниз: $t_2[a] = t_2[b] + t1[b] + s_2[b] - (s_2[a] + s_1[a])$, $t_2[1] = 0$.

Відповідь для ребра (a, b) обчислюється за виразом: $s_1[a] \cdot t_2[b] + s_2[a] \cdot t_1[b] + s_1[a] \cdot t_1[b]$.

F. Didove

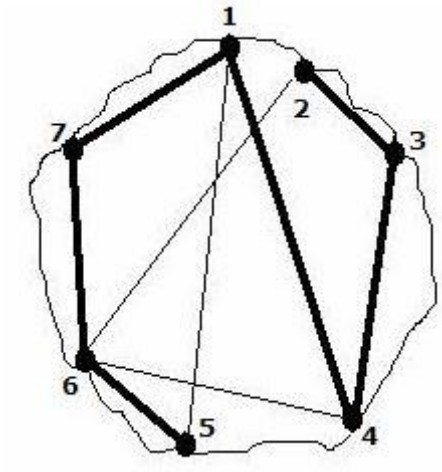
Дідове розташоване в прекрасній долині, відомій як Долина Дідове, на місці якої розташоване озеро. Близько 1300 року ужляндські релігійні лідери випустили указ про те, що центр озера повинен бути засипаний, щоб побудувати столицю їх імперії.

Навколо озера були розташовані N прибережних міст. Деякі з цих міст уклали між собою комерційні угоди. Між містами, що встановили комерційні угоди, по озеру на човнах перевозилися різні товари. Будь-яку пару міст можна було з'єднати відрізком прямої, що повністю проходить через озеро.

У певний момент королі міст вирішили впорядкувати товароперевезення. Вони розробили маршрут товароперевезень, який з'єднує всі міста навколо озера. Маршрут відповідає таким умовам:

- Він починається в будь-якому місті, проходить через кожне прибережне місто і закінчується в місті, відмінному від того, в якому він почався.
- Маршрут проходить через кожне місто рівно один раз.
- Будь-які два послідовно відвідувані міста маршруту зобов'язані мати між собою комерційну угоду.

- Маршрут складається з відрізків прямих, кожен із яких з'єднує два послідовно відвідувані міста маршруту.
- Щоб уникнути зіткнення човнів, маршрут не повинен мати самоперетинів.



На малюнку показано озеро і міста навколо нього. Тонкі й жирні лінії відрізків позначають комерційні угоди між містами. Жирні лінії показують маршрут вантажоперевезень, що починається в місті 2 і закінчується в місті 5.

Цей маршрут не має самоперетинів. Але якщо побудувати маршрут, що йде з міста 2 в місто 6, потім в місто 5, а потім в місто 1, то він буде неправильним, оскільки має самоперетин. Міста нумеруються цілими числами від 1 до N за годинниковою стрілкою.

Напишіть програму, яка за заданою кількістю міст і списком комерційних угод між містами знайде маршрут товароперевезень, що відповідає зазначеним вище умовам.

Формат вхідних даних

Перший рядок містить ціле число N $(3 \leq N \leq 1000)$. У другому рядку міститься ціле число M – кількість угод між містами. Кожен із наступних M рядків містить по два цілі числа, розділені пробілами, які відповідають номерам міст, що уклали між собою комерційну угоду.

Формат вихідних даних

Якщо можливо побудувати маршрут товароперевезень, виведіть N рядків, у кожному з яких записано ціле число. Ці числа представляють собою порядок відвідування міст по маршруту товароперевезень. Якщо неможливо побудувати маршрут товароперевезень, що задовольняє всі зазначені вимоги, виведіть один рядок, що містить число -1.

Приклади

stdin	stdout
7 9 1 4 5 1 1 7 5 6 2 3 3 4 2 6 4 6 6 7	5 6 7 1 4 3 2

Розв'язання задачі «F. Didove»

Припустимо, що торговельний маршрут починається в місті з номером x. Він може продовжитися тільки до одного з сусідів - лівого або правого, тому що будь-який інший варіант розіб'є озеро на два регіони з невідвіданими містами в кожному, роблячи побудову маршруту без самоперетинів неможливою. Те ж стосується і кожного наступного продовження маршруту - кожне наступне місто має бути сусідом одного з уже відвіданих міст. Таким чином, на кожному кроці є два кандидати на продовження маршруту.

Варто зауважити, що якщо є певна множина відвіданих міст з кінцем у місті y, нам неважливий порядок з'єднання міст. Для прикладу, нехай ми маємо маршрут, який проходить по містах 2, 3, 4 і 5 та закінчується в місті 5. Нам не важливо, чи це був маршрут $(2-3-4-5)$, $(3-2-4-5)$, чи $(4-3-2-5)$. Кандидатами на наступну вершину всеодно будуть 1 та 6.

Для розв'язування задачі можемо використати обхід у глибину з трьома параметрами: перші два параметри l, r – «лівий» і «правий» кінець маршруту та змінна булевого типу, яка буде визначати, де закінчився маршрут – *false*, якщо в лівій вершині, та *true*, якщо у правій. Тоді за умови наявності відповідних ребер – угод між містами можливі переходи в стани $(l-1, r, false)$ та $(l, r+1, true)$. Потрібно окремо обробляти випадки коли $l = 1$ або $r = N$.

Також необхідно зробити запам'ятовування – якщо на певному етапі ми маємо ті ж параметри, що вже колись були, очевидно, розв'язку ми знову не знайдемо.

Таким чином для одержання розв'язку необхідно запустити обхід у глибину з параметрами $(v, v, false)$, $v = 1 \ldots n$.

G. Tree

Дано неорієнтоване дерево, у якому з самого початку лише одна вершина – вершина номер один. Усі інші вершини від другої до N додаються ітераційно, в порядку збільшення номерів, при чому кожна нова вершина v з'єднується ребром із уже доданою вершиною p $(1 \leq p < v)$. Після кожного додавання вам необхідно знайти діаметр поточного дерева – найдовший за кількістю ребер простий шлях між двома вершинами.

Формат вхідних даних

Перший рядок містить число t $(1 \leq t \leq 15)$ – кількість тестів. Кожен тест починається з числа N $(2 \leq N \leq 10^5)$ – кількості вершин. Далі йдуть $N-1$ число для кожного v $(2 \leq v \leq N)$ вказано вершину p, з якою v з'єднується ребром. Сума всіх N в одному файлі не перевищує $2 \cdot 10^5$.

Формат вихідних даних

Обробляти тести потрібно в тому порядку, у якому вони задані у вхідному файлі. В окремому рядку виведіть діаметр дерева після кожного додавання нової вершини у дерево.

Приклади

stdin	stdout
2 3 1 1 5 1 2 3 3	1 2 1 2 3 3

Розв'язання задачі «G. Tree»

Нехай вершини v_1 та v_2 – найвіддаленіші та утворюють діаметр поточного дерева. Якщо після додавання нової вершини w діаметр збільшується, то він буде утворений парою вершин (v_1, w) або (v_2, w).

Отже, для вдалого розв'язання задачі слід підтримувати пару найвіддаленіших вершини на кожному кроці. При додаванні нової вершини потрібно за потреби оновити пару найвіддаленіших вершин. Алгоритм пошуку відстані між парою вершин у дереві був розглянутий у лекції.

H. Cities

У країні Ужляндія є N міст. Деякі пари міст з'єднані дорогами. З метою економії, «зайвих» доріг немає (з будь-якого міста в будь-яке інше можна дістатися, використовуючи наявні дороги, лише одним способом).

Відомо, що Ужляндія знаходиться в сейсмічно активній зоні. Тому президент країни захотів дізнатися, яку шкоду може завдати його державі землетрус. Зокрема, він хоче знати, яку *мінімальну* кількість доріг треба зруйнувати, щоб утворилася ізольована від інших група рівно з P міст така, що з будь-якого міста цієї групи до будь-якого іншого міста з цієї групи, як і раніше, можна було дістатися по вцілілих дорогах (група *ізольована* від інших, якщо ніяка незруйнована дорога не з'єднує місто з цієї групи з містом не з цієї групи).

Вам необхідно написати програму, яка допоможе президенту в цьому.

Формат вхідних даних

Перший рядок містить два натуральні числа: N та P $(1 \leq P \leq N \leq 150)$. Всі інші рядки містять опис доріг, по одній у кожному рядку: опис дороги складається з двох номерів міст (від 1 до N), які ці дорога з'єднує.

Формат вихідних даних

Виведіть єдине ціле число – шукану кількість доріг.

Приклади

stdin	stdout
3 2 1 2 3 2	1
11 6 1 2 1 3 1 4 1 5 2 6 2 7 2 8 4 9 4 10 4 11	2

Пояснення до прикладу

У другому прикладі група міст $(1, 2, 3, 6, 7, 8)$ опиниться ізольованою від інших, якщо будуть зруйновані дороги $1 - 4$ та $1 - 5$.

Розв'язання задачі «H. Cities»

Підвісимо дерево. Для кожної вершини v порахуємо $f_{v,k}$ мінімальну кількість ребер, яку необхідно розрізати, щоб компонента зв'язності з коренем у вершині v мала розмір рівно k. Для розв'язання задачі будемо рекурсивно застосовувати пошук у глибину та перераховувати динаміку для вершини v, поступово додаючи кожного з її дочірніх вершин таким чином:

```
count_v = 1;
f_{v,0} = f_{v,1} = 1;
for u ∈ children(v)
{
        temp_s = min(f_{v,a} + f_{u,b} | a + b = s, 1 ≤ a ≤ count_v, 1 ≤ b ≤ count_u)
        count_v += count_u
        f_{v,s} = temp_s
}
```

Тоді відповідь рівна $ans = min\left(f_{v,p} | count_v \geq p, 1 \leq v \leq n\right)$.

I. Friends

Після завершення Міжнародної олімпіади з інформатики Ілля вирішив зайнятися бізнесом. Він знайшов соціальну мережу, яка називається “TheScorpyBook.com”. На цей момент там зареєстровано N користувачів. Як і в будь-якій соціальній мережі, два користувачі можуть бути друзями. Ілля хоче порекомендувати дружбу деяким парам користувачів. Він буде рекомендувати дружбу користувачам u і v, якщо вони ще не друзі, але є користувач w, який є їх спільним другом. Зауважте, що u, v та w – різні користувачі. Ілля зараз дуже зайнятий Міжнародною олімпіадою з фізики, тому він просить вас порахувати, скільки рекомендацій друзів він має відправити в його соціальній мережі.

Формат вхідних даних

Перший рядок містить одне число N $(1 \leq N \leq 2000)$ – кількість користувачів у мережі. Наступні N рядків містять по N символів: j-ий символ i-ого рядка рівний одиниці, якщо користувачі i та j є друзями, і нулю в протилежному випадку. Їх відносини симетричні, тобто якщо користувач i є другом користувача j, то j є другом i.

Формат вихідних даних

Виведіть єдине число – кількість рекомендацій друзів, які потрібно надіслати Іллі.

Приклади

stdin	stdout
4 0111 1000 1000 1000	6

Розв'язання задачі «I. Friends»

Розглянемо тривіальний алгоритм розв'язання задачі. Перебираючи кожну пару вершин u та v, переберемо вершину w та перевіримо, чи є вона їх спільним другом (вершини u та v з'єднані ребром з вершиною w). Складність алгоритму $O(n^3)$.

Можна помітити, що для кожної вершини u слід зберігати n бітів, i-ий біт рівний 1, якщо u з'єднана з вершиною i, та 0 в іншому випадку. Для зменшення об'єму використаної пам'яті можемо об'єднати послідовності 0 та 1 в групи по 32 біти та зберігати кожну з них як одне число. Всього таких груп буде $n/32$.

Тепер, перебираючи дві вершини u та v, потрібно для кожної групи перевірити факт наявності спільних бітів, що можна зробити, перевіривши побітове I двох чисел, що представляють ці групи. Якщо побітове I більше 0, то числа мають спільні одиничні біти.

J. Tree again

Дано дерево із N вершин. Вершини нумеруються від одиниці до N, коренем є перша вершина. У кожній вершині записано додатне число W[i]. Поглянемо на наступний код:

```
Integer sum:=0;
Array of Boolean marked:={false, false, false,…, false};
Procedure Dfs(Integer v)
    Begin
        sum:=sum+W_v
        marked[sum]:=true;
        For each node u that v is a parent of u do
            Begin
```

```
                    Dfs(u);
                End;
    End;

Dfs(1)
```

Можна помітити, що стан масиву $marked$ залежить від порядку обробки вершин u в циклі. Вам необхідно для всіх цілих чисел s від 1 до суми всіх $W[i]$, перевірити, чи існує такий порядок обробки вершин u у циклі, що $marked[s]$ дорівнює $true$. Зверніть увагу, що sum та $marked$ – це глобальні, початково ініціалізовані змінні, та що $Dfs(1)$ викликається із зовнішньої програми лише раз.

Формат вхідних даних

Перший рядок містить єдине натуральне число N $(1 \leq N \leq 500)$ – кількість вершин. У наступному рядку записані N чисел $W[i]$. Сума всіх $W[i]$ не перевищує 10^5. Далі для кожної вершини i від 2 до N в окремому рядку записана її батьківська вершина p $(p < i)$.

Формат вихідних даних

Нехай сума всіх $W[i]$ дорівнює S. Вам необхідно вивести бінарний рядок довжиною S, і-ий символ якого рівний 1, якщо можливо отримати $marked[i] == true$, та 0 в іншому випадку. Символи в цьому рядку нумеруються з одиниці.

Приклади

stdin	stdout
5 1 7 7 2 4 1 1 2 4	100000010100011010001

Розв'язання задачі «J. Tree again»

Коли ми обираємо наступну вершину, ми маємо можливість обирати будь-якого сина поточної вершини. Цей вибір вплине на стан масиву $marked[]$.

Якщо на певному кроці ми опинилися у вершині v, можливі наступні випадки:

- ми вперше прийшли до вершини v по прямому шляху з кореня;
- ми відвідали деяких синів піддерева вершини v. Помітимо, що в цьому випадку не буде жодного піддерева вершини v, яке ми частково відвідали.

Як це використати? Будемо зберігати булевий масив для глобального розв'язку. Розмір масиву рівний сумі всіх ваг вершин. Запустимо DFS з кореня та будемо зберігати рюкзак, який показує, які ваги ми можемо отримати в момент відвідання поточної вершини. Коли ми знаходимося в деякій вершині, додамо предмети з вагами, рівними розмірами піддерев всіх синів поточної вершини. Під розміром піддерева будемо розуміти суму всіх ваг вершин, що входять до нього. Коли ми хочемо обробити одного з синів, ми маємо видалити з рюкзака розмір його піддерева, перейти вниз, повернутися та додати величину назад до рюкзака.

K. Dreaming

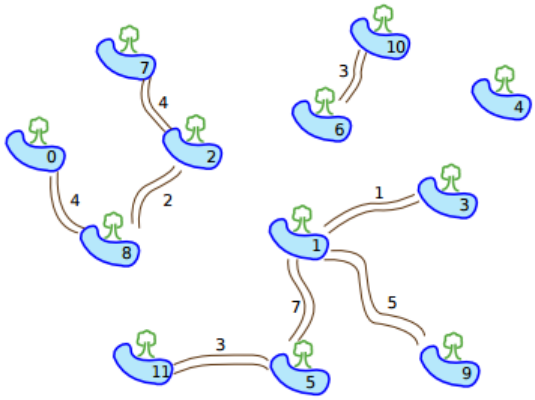
У країні Ужляндія високо в горах живе Ведмідь. В горах є N озер, що пронумеровані від 0 до $N-1$. Крім того, є M двосторонніх стежок, що з'єднують деякі пари озер. Ведмідь може переміщуватися по цих стежках. Кожна пара озер з'єднана (напряму або через проміжні озера) не більше, ніж однією послідовністю початково заданих стежок, проте деякі пари озер можуть бути не з'єднані взагалі (тобто $M \leq N-1$). Для кожної початково заданої стежки відома кількість днів, які витрачає Ведмідь на її подолання.

Друг Змія, Олень, хоче прокласти $N-M-1$ нову стежку так, щоб Ведмідь зміг подорожувати між будь-якою парою озер. Олень може створювати

нову стежку між будь-якою парою озер, на подолання кожної з таких новостворених стежок Ведмедю потрібно буде витратити L днів.

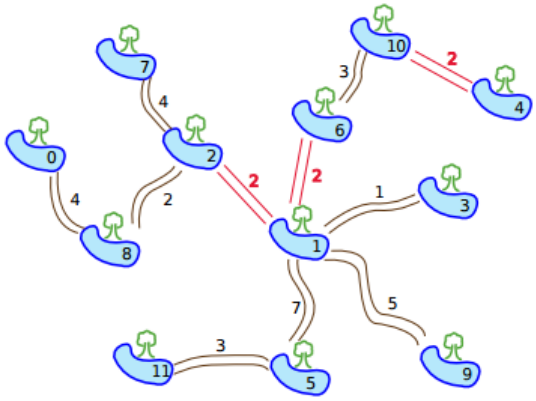
Олень хоче зробити подорож Ведмедя якомога швидшою. З цією метою Олень збирається прокласти нові стежки так, щоб максимальний час подорожі між двома озерами був якомога меншим. Допоможіть Оленю та Ведмедю визначити максимальний час подорожі між двома озерами після того, як олень прокладе нові стежки вказаним способом.

Розглянемо приклад:



На рисунку вище показані $N = 12$ озер та $M = 8$ початково заданих стежок. Припустимо, що $L = 2$, тобто час переміщення по кожній з новостворених стежок займає 2 дні. Тоді Олень може прокласти такі три нові стежки:

- між озерами 1 та 2;
- між озерами 1 та 6;
- між озерами 4 та 10.



На рисунку вище показаний кінцевий набір стежок. Максимальний час подорожі – 18 днів, він досягається при переміщені між озерами 0 та 11. Це найменший можливий максимальний час подорожі: як би Олень не будував нові стежки, завжди знайдеться якась пара озер, для подорожі між якими знадобиться 18 днів або більше.

Формат вхідних даних

Перший рядок містить три цілих числа N, M та L ($1 \leq N \leq 10^5$; $0 \leq M \leq N-1$; $1 \leq L \leq 10^4$) – кількість озер, кількість початкових стежок та час у днях, який потрібний Ведмедю на подолання будь-якої новоствореної стежки.

Кожен з наступних $M-1$ рядків містить опис початкових стежок, що задається трьома числами a_i, b_i та t_i – номери озер, що з'єднує ця стежка, та час у днях, потрібний для подорожі по ній.

Формат вихідних даних

Виведіть єдине ціле число – максимальний час у днях, який буде потрібний для подорожі від одного озера до іншого, з розрахунком, що Олень додав $(N-M-1)$ нову стежку так, що всі озера з'єднані між собою, і цей максимальний час є мінімально можливим.

Приклади

stdin	stdout
12 8 2 0 8 4 8 2 2 2 7 4 5 11 3 5 1 7 1 3 1 1 9 5 10 6 3	6

Розв'язання задачі «K. Dreaming»

За умовою задачі на початку заданий неорієнтований граф, компоненти зв'язності якого є деревами. Граф містить N вершин та M ребер. Необхідно

зробити початковий граф зв'язним, побудувавши $N - M - 1$ ребро вартості L, мінімізувавши при цьому діаметр кінцевого графу.

Для початку розглянемо кожну компоненту-дерево окремо. Знайдемо в кожній з них діаметр, радіус та центр. Впорядкуємо всі компоненти по не зростанню діаметру та об'єднуватимемо їх в такому порядку, при цьому слід навчитися перераховувати новий діаметр, центр та радіус.

Нехай дані першої компоненти – $diameter_1$, $center_1$, $radius_1$, дані другої компоненти – $diameter_2$, $center_2$, $radius_2$. Тоді, якщо позначити через $diameter_M$, $center_M$, $radius_M$ дані компоненти, що є їх об'єднанням, стверджується наступне:

- $diameter_M = radius_1 + radius_2 + L$;
- $center_M$ - або $center_1$ або $center_2$ залежно від радіусів, що отримуються.

Якщо ми оберемо $center_1$ як $center_M$, тоді $radius_M = \max(radius_1, radius_2 + L)$, якщо ж оберемо $center_2$, то $radius_M = \max(radius_1 + L, radius_2)$. Оскільки $radius_M$ потрібно мінімізувати, то оберемо варіант із меншим значенням.

Слід також врахувати варіант, коли $diameter_1$ або $diameter_2$ більший, ніж $radius_1 + radius_2 + L$, його необхідно опрацювати окремо.

День 3. Контест Сергія Нагіна

Теоретичний матеріал. Замітаюча пряма

1. Вступ

В олімпіадних задачах досить часто виникає проблема, коли в певній метриці задані об'єкти та треба знайти значення певної функції, аргументами якої є задані об'єкти. У багатьох задачах прямий підхід до розв'язання призводить до дуже повільних розв'язків, що не зможуть вкластися в обмеження за часом. Також кожна задача може мати свої специфічні властивості, що дозволяють використовувати різні підходи. Але ми розглянемо загальний підхід, що базується на розбитті даних на події та їх послідовній обробці. Далі ми розглянемо схему алгоритму та приклади задач, що можуть бути розв'язані методом замітаючої прямої.

2. Загальна схема алгоритму

У загальному вигляді важко описати алгоритм, оскільки для кожної задачі аналіз даних буде проводитися по-різному. У цій схемі буде описаний алгоритм для більш вузького класу задач, але ця схема може бути масштабована для інших задач.

Припустимо, що на вході об'єктами є деякі фігури на площині, а функція - певна величина, що описує властивість розташування деяких об'єктів. Прикладами функції може бути кількість площини, що покрита об'єктами, наявність попарного перетину між об'єктами тощо. При такому формулюванні загальна схема буде виглядати так:

a. Виділення основних подій, коли функція може змінюватися.

b. Розбиття об'єктів на виділені події.

c. Обхід подій у певному порядку.

d. Обчислення функції.

Зрозуміло, що наведена вище схема є формальною, тому розглянемо приклади задач та їх розв'язання.

3. Задача про перетин та об'єднання відрізків

Розглянемо таку задачу: на координатній осі задано N відрізків координатами початку та кінця. Знайти довжину перетину та довжину об'єднання відрізків.

Зробимо аналіз за кроками:

a. Які події нас можуть цікавити? Очевидно, що в цій задачі це два типи подій: початок відрізка, кінець відрізка. Ми вибираємо саме ці події, бо в них може змінитися розв'язок задачі.

b. Розбиття зробити дуже просто. Для кожного відрізка з координатами $[l, r]$ створимо дві події $(l, 0)$ та $(r, 1)$.

c. Події будемо обходити у порядку сортування за першою компонентою в події, а при рівності першої компоненти – за другою компонентою.

d. На цьому етапі ми вже робимо обхід події, припустимо, що зараз ми обробляємо подію (x, y). Якщо y дорівнює 0, то ми повинні збільшити значення певного глобального лічильника. Далі можливі декілька випадків:

I. Для задачі перетину, якщо значення лічильника стало дорівнювати загальній кількості відрізків, то змінити положення загальної змінної $left$ на x.

II. Для задачі об'єднання, якщо значення лічильника стало дорівнювати одиниці, то змінити положення загальної змінної $left$ на x.

Якщо ж y дорівнює 1, то ми повинні зробити дії залежно від випадків:

I. Для задачі перетину, якщо значення лічильника дорівнює загальній кількості відрізків, то потрібно додати до відповіді значення $x - left$.

II. Для задачі об'єднання, якщо значення лічильника дорівнює одиниці, то потрібно додати до відповіді значення $x - left$.

Після цього значення глобального лічильника потрібно зменшити на одиницю.

У результаті ми маємо значення функції. За рахунок сортування складність цього рішення становить $O(N \log N)$, що не є оптимальним для задачі перетину, але є оптимальним для задачі об'єднання.

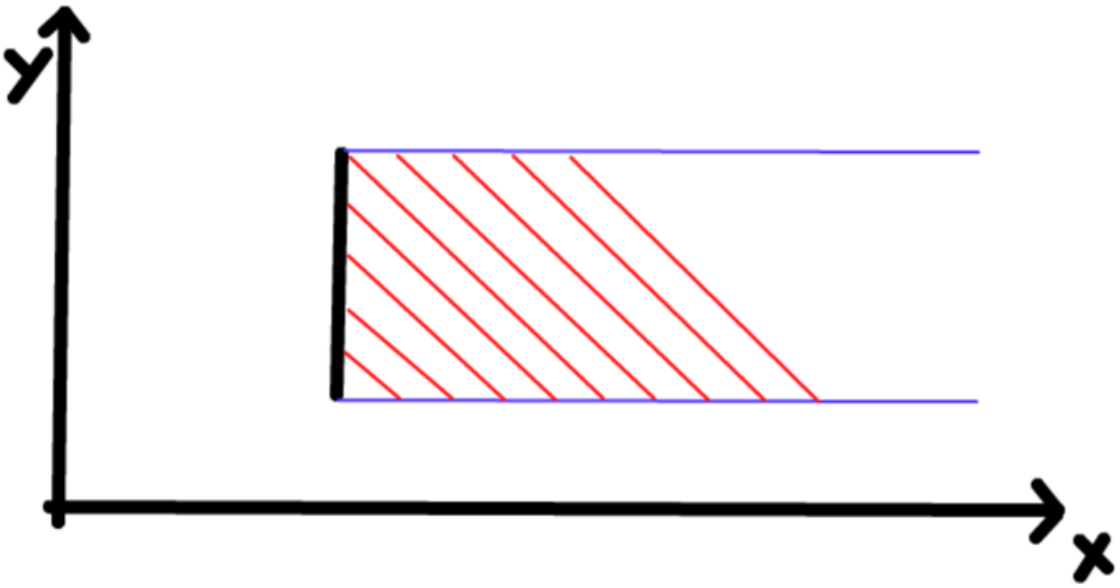
Варто зауважити, що для задачі об'єднання є оптимізація, що може працювати у випадку невеликих координат, у такому разі складність рішення може становити $O(N + MaxN)$.

4. Задача про прямокутники

Розглянемо приклад ще однієї простої задачі, що легко може бути розв'язана методом замітаючої прямої. На декартовій площині задано N прямокутників із цілими координатами, сторони цих прямокутників паралельні координатним осям. Треба порахувати площу поверхні, яку покривають задані прямокутники. Якщо деяка ділянка поверхні покрита більш ніж одним прямокутником, то у відповіді вона повинна бути врахована лише один раз.

По-перше, для розв'язання задачі зрозуміємо, які події існують при "скануванні" площини зліва направо та як вони можуть впливати на відповідь.

a. Зустрічаємо ліву сторону прямокутника з координатами (x, y_1, y_2), що означає, що кожна клітинка праворуч від відрізка буде покрита ще одним прямокутником

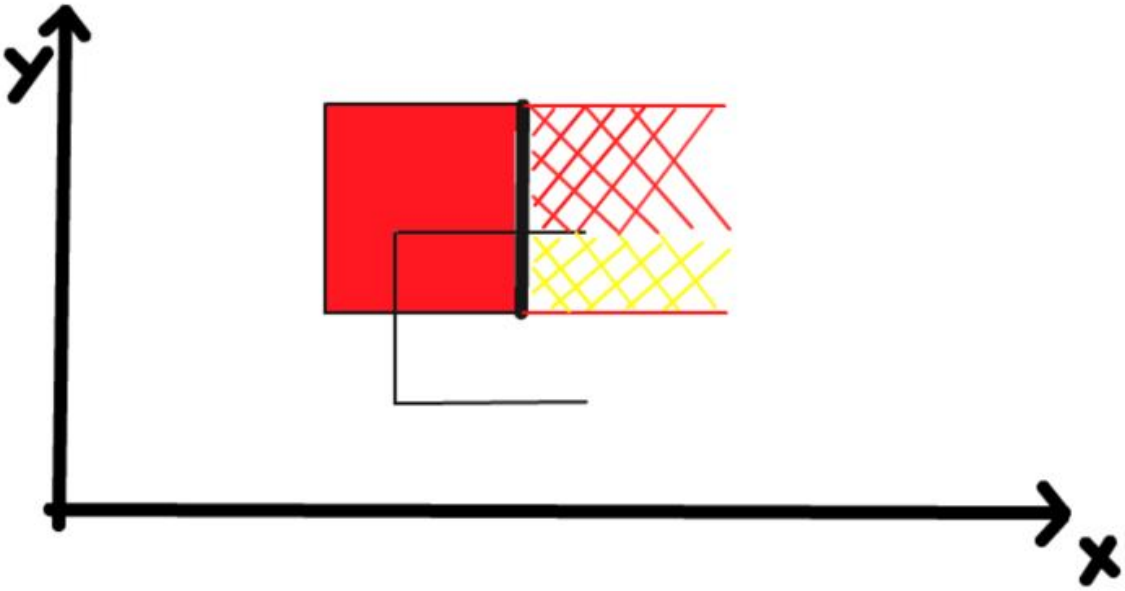


Зображення площі, що покриває відрізок

b. Зустрічаємо праву сторону прямокутника, у цьому випадку маємо дві дії

b.1. потрібно додати все, що знаходиться ліворуч від відрізка й не додано до відповіді. На малюнку - зафарбована ділянка

b.2. потрібно відзначити, що праворуч від відрізка площина покрита на один прямокутник менше. На малюнку - заштрихована ділянка.



Ілюстрація “правого” відрізка, що позначений широкою лінією.

Жовтим позначено ділянку, що на цьому етапі зберігається як покрита.

Реалізувати алгоритм, що буде підтримувати описані події, можна так:

1. Будемо мати дерево відрізків, що підтримує дві операції: додати/відняти одиницю на відрізку, сказати кількість чисел, що більше нуля.

2. Будемо групувати події за координатою x.

3. Опрацюємо кожний x у порядку зростання

3.1. Опрацюємо всі “праві” сторони та віднімемо одиницю в дереві на кожному такому відрізку.

3.2. Порахуємо кількість додатніх чисел у дереві. Помножимо знайдену кількість на різницю нашого x та попереднього x, таким чином додамо до відповіді декілька прямокутників.

3.3. Опрацюємо всі “ліві” сторони й додамо одиницю в дереві на кожному такому відрізку.

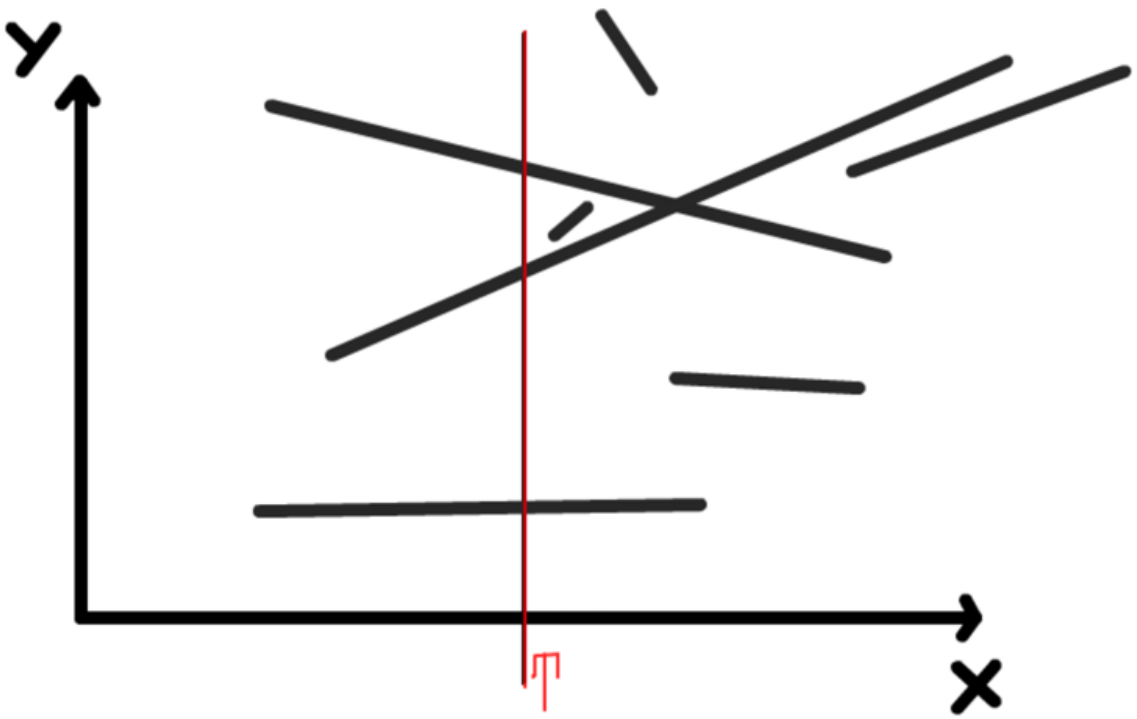
4. Маємо розв’язок задачі.

Складність такого алгоритму становить $O(N\log(N) + N\log(maxN))$, бо нам потрібно відсортувати події за першою координатою, одна операція додавання в дерево займає $O(\log(maxX))$ часу.

5. Задача про перетин відрізків

Тепер розглянемо більш складну задачу: на площині задано N відрізків, потрібно знайти пару відрізків, що перетинаються. Одразу зауважимо, що задача про знаходження всіх перетинів відрізків не може бути розв’язана швидше ніж за $O(N^2)$, бо в гіршому випадку кожна пара відрізів буде перетинатися.

Для полегшення розв’язання будемо вважати, що вертикальні відрізки відсутні або просто повернемо простір на певний дуже малий кут.



Розглянемо зображену на малюнку пряму $x = t$. Нехай ми будемо рухати її зліва направо та підтримувати список відрізків, що вона перетинає. Відрізки мають бути впорядковані по Y координаті точки перетину. Зрозуміло, що відрізки, що не перетинаються, завжди будуть знаходитися в одному й тому ж відносному порядку в списку. Отже, відрізки, що перетинаються, будуть знаходитися поряд у списку при певному t.

Тепер покажемо події, що можуть нас цікавити:

1. Зустрічаємо лівий край відрізка.
2. Зустрічаємо правий край відрізка.

Припустимо, що ми додаємо точки в порядку сортування по x. Розглянемо кожен випадок детальніше:

1. Якщо ми додаємо початок деякого відрізка, то потрібно знайти його лівого та правого сусіда у списку. Тільки вони можуть перетинати відрізок на цей момент. Після цього точку потрібно додати до списку між сусідами.
2. Якщо ми зустрічаємо правий кінець, то відрізок потрібно видалити зі списку. Але після видалення ми отримаємо пару, можливо, нових сусідів, які також можуть перетинатися. Тому треба організувати ще одну перевірку

Отже, ми маємо алгоритм розв'язання. Для зберігання списку можна використовувати бінарне дерево пошуку. Зауважимо, що ми знаходимо лише один перетин, для наступних перетинів алгоритм буде працювати некоректно, тому його буде потрібно модернізувати на обробку третього виду подій - перетин відрізків, при якому ми повинні переставити елементи в списку. У нашому випадку при знаходженні першого перетину весь алгоритм треба перервати та повернути відповідь.

6. Висновки

Ми розглянули метод замітаючої прямої та приклади задач, де він може бути використаний. Зрозуміло, що цей алгоритм може бути використаний не лише у двовимірному просторі

Задачі та ідеї розв'язання

Задача A

Задано два відрізки. Єгор вирішив намалювати задані відрізки на площині. Єгор не любить дійсні числа, тому він малює відрізки таким чином:

1. Кожну точку (позначимо як (x, y)), що належить відрізку, Єгор перетворює у точку (x', y'), де $x' = f(x)$, $y' = f(y)$.
2. Єгор наносить отримані точки на площину.

Для першого відрізка $f(t) = \lfloor t \rfloor$, для другого $f(t) = \lceil t \rceil$. Тепер Єгора цікавить питання: чи є спільна точка в намальованих відрізків?

Формат вхідних даних

Перший рядок містить ціле число T, що означає кількість тестів. Далі йде опис T тестів. Кожен тест описується вісьма числами x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, де перші чотири – координати першого відрізка, чотири наступні – координати другого відрізка.

Формат вихідних даних

Для кожного тесту виведіть 1, якщо намальовані відрізки перетинаються, та 0 – інакше.

Обмеження

1. $1 \leq T \leq 10^5$.
2. Координати – невід'ємні цілі числа, що не перевищують 10^3.

Приклади

stdin	stdout
3 1 1 2 2 2 2 3 3 1 1 2 2 3 3 4 4 0 2 2 0 0 1 1 0	1 0 1

Розв'язання задачі «А»

Перше, що потрібно помітити: кількість різних точок множин невелика. Пройдемося по точках відрізка з певною дискретністю та додамо точки в множини. Якщо множини перетинаються, - то відповідь 1, інакше 0.

Важливий момент реалізації: обході точок не слід впустити точки з цілими координатами.

Найкращий варіант обходу. Окремо розібрати випадки відрізків, що паралельні осям координат. Перебрати всі можливі X із деякою дискретністю та порахувати відповідний Y. Перебрати всі можливі Y з деякою дискретністю та порахувати відповідний X.

Задача B

Задано N відрізків на осі OX координатами їх кінців. Порахуйте:

1. довжину частини прямої, що покрита кожним із заданих N відрізків;
2. довжину прямої, що покрита хоча б одним із заданих N відрізків.

Формат вхідних даних

Перший рядок містить ціле число N. Наступні N рядків містять опис відрізків, кожний рядок містить два числа x_1, x_2, що позначають кінці відрізків.

Формат вихідних даних

Виведіть два числа через пропуск: довжину перетину та довжину об'єднання.

Обмеження

1. $1 \leq n \leq 10^5$.
2. Для всіх відрізків виконується $1 \leq x_1, x_2 \leq 10^9$.

Приклади

stdin	stdout
4 1 5 1 3 2 5	1 5
2 1 2 10 11	0 2

Задача C

Задано велике коло з радіусом R. Чи можна розмістити n кіл радіусом r усередині великого кола, якщо:

1. Кожне коло дає хоча б одну точку дотику з великим.
2. Кожна точка кожного кола знаходиться на великому колі або всередині великого кола.
3. Жодна пара кіл не перетинаються.

Формат вхідних даних

Перший рядок містить ціле число T, що означає кількість тестів. Далі йде опис T тестів. У єдиному рядку задано три цілих числа n, R і r.

Формат вихідних даних

Для кожного тесту виведіть 1, якщо можна розташувати кола потрібним чином, та 0 – інакше.

Обмеження

1. $1 \leq n \leq 100$.
2. $1 \leq R, r \leq 1000$.
3. $1 \leq T \leq 100$.

Приклади

stdin	stdout
4 1 10 10 40 10 4 10 10 9 1 9 10	1 1 0 0

Розв'язання задачі «C»

Окремо розглянемо випадок, коли внутрішній радіус більше зовнішнього. Для правильного розташування внутрішніх кіл потрібно, щоб їхні центри лежали на колі радіуса $R - r$. Тепер потрібно перевірити, чи можна розмістити N точок у вершинах правильного багатокутника, уписаного в коло центрів, що його сторона буде більше за $2r$. Формула для підрахунку довжини сторони: $a = 2(R - r)\sin\frac{\pi}{N}$.

Задача D

У Єгора є електронна таблиця розміру $N \times N$, кожна клітина якої може бути виключена або включена. Він хоче, щоб на полі були включені не менше ніж C клітин, коли ця умова буде виконана, – Єгор стане щасливим.

Будемо вважати, що рядки таблиці пронумеровані зверху вниз від 1 до N, а стовпці – зліва направо від 1 до N. Спочатку включена рівно одна клітина з координатами (x, y) (x – номер рядка, y – номер стовпчика), а всі інші клітини знаходяться у вимкненому стані. Далі кожну секунду відбувається включення виключених клітин, у яких є включені сусідні за стороною клітини.

Для клітини з координатами (x, y) сусідніми за стороною є клітини з координатами $(x - 1, y)$, $(x + 1, y)$, $(x, y - 1)$, $(x, y + 1)$.

Через скільки секунд Єгор стане щасливим?

Формат вхідних даних

Перший рядок містить ціле число T, що означає кількість тестів. Далі йде опис T тестів. У єдиному рядку містяться чотири цілі числа n, x, y, c.

Формат вихідних даних

Для кожного тесту виведіть час, за який Єгор стане щасливим.

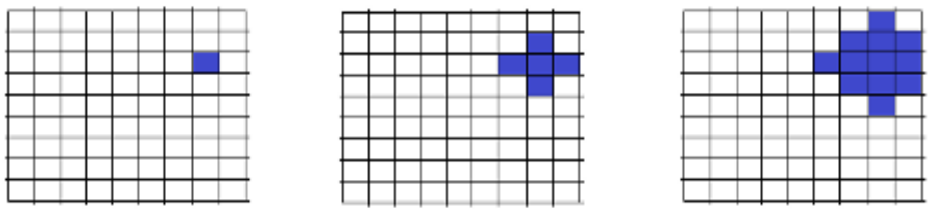
Обмеження

1. $1 \leq n, c \leq 10^9$.
2. $1 \leq x, y \leq n$.
3. $c \leq n^2$.
4. $1 \leq T \leq 100$.

Приклади

stdin	stdout
2 9 3 8 10 6 4 3 1	2 0

Ілюстрація до першого прикладу:



Розв'язання задачі «D»

Відповідь будемо з'ясовувати бінарним пошуком. Протягом певного часу нам потрібно порахувати перетин ромбу клітин з квадратом клітин. Для підрахунку визначимо:

1. площу ромба,
2. площу ромба, що перетинає ліву сторону,
3. площу ромба, що перетинає праву сторону,
4. площу ромба, що перетинає верхню сторону,
5. площу ромба, що перетинає нижню сторону,

6. сумарну площу ромба, що перетинає дві сторони, тобто лежить за деяким кутом.

Відповіддю буде число (1) плюс число (6) мінус числа (2), (3), (4), (5).

Задача E

Задано N точок з цілими координатами. Порахуйте кількість пар точок (i, j) $(1 \leq i < j \leq N)$ таких, що відстань між точками виражається цілим числом.

Формат вхідних даних

У першому рядку задано число N. Наступні N рядків містять опис точок, кожна точка задана парою координат.

Формат вихідних даних

В єдиному рядку виведіть кількість пар точок, між якими відстань між точками виражається цілим числом.

Обмеження

1. $1 \leq N \leq 10^5$.
2. Координати – невід'ємні цілі числа, що не перевищують 100.

Приклади

stdin	stdout
3 1 1 2 2 1 2	2
3 1 1 1 1 1 1	3

Розв'язання задачі «E»

Занесемо кожну задану точку (x, y) у клітинку масиву $A_{x,y}$. Тоді $A_{i,j}$ означає кількість точок, що знаходяться в (i, j). Переберемо всі пари (dx, dy),

що існує ціле число d, $d^2 = dx^2 + dy^2$. Для кожної пари підрахуємо суму всіх $A_{i,j} \cdot A_{i+dx,j+dy}$. Складність такого рішення $O(size^4)$.

Задача F

Задано N відрізків координатами їх кінців. Чи є серед заданих відрізків пара таких, що перетинаються?

Формат вхідних даних

У першому рядку задано число N. Наступні N рядків містять опис відрізків, кожен відрізок заданий чотирма числами: перші два – координати одного кінця, останні два – координати другого кінця.

Формат вихідних даних

У єдиний рядок виведіть 1, якщо існує пара відрізків, що перетинаються, інакше ж виведіть число 0.

Обмеження

1. $1 \leq N \leq 10^5$.
2. Координати – невід'ємні цілі числа, що не перевищують 10^9.

Приклади

stdin	stdout
3 1 1 2 2 3 3 4 4 2 1 1 2	1
3 1 1 2 2 3 3 4 4 5 5 6 6	0

Задача G

Задано N різних точок із цілими координатами. Також задано певне ціле число C ($0 \leq C \leq 100$). Визначте, чи можна провести пряму таким чином:

1. На прямій не буде заданих точок.
2. Кількість точок по одну сторону від прямої утворює строго C відсотків від загальної кількості точок.

Формат вхідних даних

У першому рядку задано число N та число C. Наступні N рядків містять опис точок, кожна точка задана парою координат.

Формат вихідних даних

У єдиний рядок виведіть 1, якщо можна провести потрібну пряму, інакше ж виведіть число 0.

Обмеження

1. $1 \leq N \leq 10^5$.
2. Координати – невід'ємні цілі числа, що не перевищують 10^9.

Приклади

stdin	stdout
2 50 1 1 2 2	1
3 33 1 1 2 2 1 2	0

Розв'язання задачі «G»

Оскільки всі точки мають різні координати, то можна поділити точки у будь-якому відношенні. Отже, пряму можна провести тоді і тільки тоді, коли $\frac{N \cdot C}{100}$ – ціле число або $N \cdot C \; mod \; 100 = 0$.

Задача H

Єгор намалював коло радіусом R у центрі координатної системи. Також Єгор задав точку, яка початково має координати (x_p, y_p) та обертається навколо центра координатної системи проти годинникової стрілки зі швидкістю v_p.

Також є точка, що початково має координати (x, y). Точка може рухатись у будь-якому напрямі зі швидкістю v. Точка не може перетинати Єгорове коло.

За який мінімальний час дві точки зможуть мати однакові координати?

Формат вхідних даних

Перший рядок містить ціле число T, що означає кількість тестів. Далі йде опис T тестів. У першому рядку задано цілі числа x_p, y_p та v_p. У другому рядку задано цілі числа x, y, v та R.

Формат вихідних даних

Для кожного тесту виведіть мінімальний час із точністю до 10^{-6}, що описаний в умові.

Обмеження

1. $-10^4 \le x_p, y_p \le 10^4$, $1 \le v_p < 10^4$.
2. $-10^4 \le x, y \le 10^4$, $1 < v \le 10^4$.
3. $1 \le R \le 10^4$.
4. $R^2 < x^2 + y^2$, $R^2 < x_p^2 + y_p^2$, $v_p < v$.
5. $1 \le T \le 100$.

Приклади

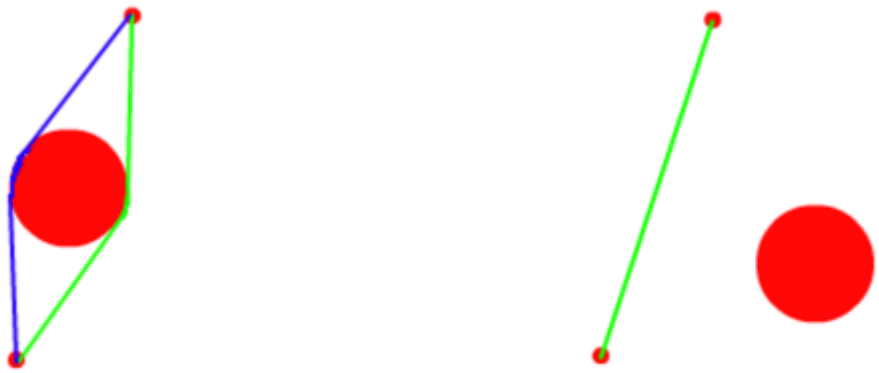
stdin	stdout
2 10 0 1 -10 0 2 8 50 60 10 50 60 20 40	9.584544103 0

Розв'язання задачі «Н»

Оскільки швидкість точки, яку ми рухаємо, перевищує швидкість точки, що рухається по колу, – ми можемо застосувати бінарний пошук за відповіддю. Припустимо, що ми перевіряємо час t, тоді ми можемо сказати координату, де буде знаходиться точка, що рухається по колу. Тепер єдине, що залишилося знайти, – відстань між двома точками, коли перешкодою є коло. Залежно від можливості пройти відстань за час t змінимо межі бінарного пошуку.

Для знаходження відстані треба розглянути декілька випадків. Якщо коло не перетинає наш відрізок, то відповідь - відстань між двома точками. Інакше знайдемо дотичні до кола, що виходять із наших точок (таких буде 4). Візьмемо також найкращу пару дотичних, тобто таких, що сума довжин дотичних та відстань по колу буде мінімальною.

Шляхи зображені на рисунку нижче.



Задача I

Єгор вважає, що набір точок має центр симетрії P, якщо для кожної точки знайдеться пара, що симетрична своїй парі відносно точки P, єдина точка може знаходитись у центрі симетрії та бути парою самій собі. У Єгора була певна кількість точок із цілими координатами, усі точки різні. Сергій стер не більше ніж K точок, залишилось N точок.

Тепер Єгора цікавить, які точки, могли бути центром симетрії в оригінальному наборі. Порахуйте кількість різних цікавих точок.

Формат вхідних даних

У першому рядку задано цілі числа N та K. Наступні N рядків містять координати точок, що зашилися не стертими. Усі задані точки – різні.

Формат вихідних даних

У єдиний рядок виведіть кількість різних центрів симетрії в оригінальному наборі. Якщо таких точок безліч, то виведіть -1.

Обмеження

1. $1 \leq N \leq 2 \cdot 10^5$, $0 \leq K \leq 10$.
2. Координати точок – цілі числа, що за модулем не перевищують 10^9.

Приклади

stdin	stdout
4 2 0 0 0 1 1 0 1 1	5
4 4 0 0 0 1 1 0 1 1	-1

Розв'язання задачі «I»

Для початку зрозуміємо, як перевірити точку на те, що вона є центром симетрії. Якщо ми візьмемо найлівішу точку та знайдемо їй пару, то пара буде - найправіша. Відкинемо ці дві точки та продовжимо процес. Якщо на певному кроці найлівіша та найправіша не симетричні відносно заданої точки, то точка не є центром симетрії. Тому відсортуємо всі точки за x, а при рівності за y. Тепер будемо перебирати точку з перших $k+1$ та точку з останніх $k+1$ у списку. Очевидно, що хоча б одна пара повинна мати центр симетрії на

середині відрізка, що сполучає точки. Для перевірки просто порахуємо кількість точок, що мають симетричну пару. Якщо кількість таких точок більша або дорівнює ніж $N - k$, то точку симетрії потрібно додати до відповіді. Потрібно не забувати, що точки треба складати в *set*, бо на деяких кроках є можливість отримати рівні точки. Складність рішення $O(N \cdot k^2 \cdot \log n)$.

Задача J

Заданий набір цілих чисел a_1, a_2, …, a_N. Чи існує спосіб вибрати два набори по q та w чисел, якщо кожен із наборів описує довжини сторін певного невиродженого багатокутника? Кожне число може бути використане лише в одному наборі.

Формат вхідних даних

У першому рядку задано цілі числа N, q та w. Наступний рядок містить N чисел a_1, a_2, …, a_N.

Формат вихідних даних

У єдиний рядок виведіть 1, якщо існують два потрібні набори, та 0 – в іншому випадку.

Обмеження

1. $1 \le N \le 10^5$, $3 \le q, w \le 10$.
2. $1 \le a_i \le 10^9$.

Приклади

stdin	stdout
6 3 3 2 3 4 10 11 12	1
6 3 4 10 11 12 13 14 1000000000	0

Розв'язання задачі «J»

По-перше, нехай є сторони a_1, a_2, …, a_p, та a_p - найдовша сторона. Тоді багатокутник можна побудувати, якщо $a_1 + a_2 + \cdots + a_p > a_p \cdot 2$.

По-друге, якщо $N > 70$, то відповідь завжди існує. Для доведення побудуємо найгірший випадок, та побачимо, що сторони ростуть експоненційно.

Тепер відсортуємо всі відрізки. Для одного набору, якщо відповідь існує, всі відрізки будуть знаходитися підряд у відсортованому списку. Для двох наборів є два випадки розташування:

1. Як два набори відрізків, що стоять підряд у відсортованому списку.
2. Або як один набір відрізків, що стоять підряд у відсортованому списку.

Перший випадок розглянути досить просто. Для другого випадку ми будемо перебирати перший відрізок, а потім будемо перебирати розбиття, що також не складно реалізувати.

Задача K

Нехай A – масив цілих чисел A_1, A_2, …, A_N ($0 \le A_i \le 1$, $A_1 = 1$, $A_N = 1$).

Нехай $F(A)$ – кількість різних способів добратися від першої клітинки масиву A до останньої клітинки, коли можна рухатися на одну та на дві клітинки вперед, а на клітинки зі значеннями 0 ставати не можна.

Вам задано два цілі числа N та K. Чи існує масив A довжиною N, що $F(A)$ дорівнює K?

Формат вхідних даних

У єдиному рядку задано цілі числа N та K. Наступний рядок містить N чисел a_1, a_2, …, a_N.

Формат вихідних даних

У єдиний рядок виведіть 1, якщо існує потрібний масив, та 0 – в іншому випадку.

Обмеження

1. $1 \leq N \leq 1000$.
2. $0 \leq K \leq 10^{18}$.

Приклади

stdin	stdout
3 1	1
566 239	0

Розв'язання задачі «K»

У задачі потрібно представити число K у вигляді добутку чисел Фібоначчі F_{k_1}, F_{k_2}, …, F_{k_l}, що $k_1 + k_2 + \cdots + k_l + l + 1 = N$. Помітимо, що дільників N, що є числами Фібоначчі, або числа, що діляться на числа Фібоначчі, невелика кількість. Тому можемо написати динамічне програмування за всіма станами та рахувати можливість представлення певного числа з певною сумою номерів чисел Фібоначчі.

Цікавий випадок: $K = 0$, для нього при $N < 4$ відповідь 0, інакше 1.

День 4. Контест LNU_Penguins (Віталій Герасимів, Роман Білий)

Задачі та ідеї розв'язання

A. Злови їх усіх!

Андрій просто фанатіє від нової гри про покемонів. Та оскільки він та його команда не мають часу на такі забавки, все не так просто, як здавалося б.

На вулиці, де буває Андрій, усього є N різних покемонів. Для простоти будемо вважати, що вулиця – це пряма лінія, а i-го покемона можна зловити, знаходячись у будь-які точці, яка є в межах $[L_i; R_i]$ (границі включно).

Андрій вирішив, що половить покемонів на наступному командному тренуванні. Він може домовитися з хлопцями, щоб це тренування відбулося в одній із M заданих точок $X_1, \ldots, X_M$.

Ваше завдання – допомогти знайти для Андрія кількість можливих непустих множин покемонів, які він може зловити та провести тренування одночасно. Оскільки відповідь може бути дуже великою, виведіть її за модулем 1000000007.

Формат вхідних даних

У першому рядку задано два цілих числа N та M – кількість покемонів та можливих місць для тренування відповідно. В наступних N рядках задано пари цілих чисел L_i та R_i, які описують, де можна зловити відповідного покемона. В останньому рядку через пробіл задано M цілих чисел X_i – місця, одне з яких Андрій може вибрати для тренування.

Формат вихідних даних

В єдиному рядку виведіть одне ціле число – відповідь до задачі за модулем 1000000007.

Обмеження

3. $1 \leq N, M \leq 10^5$,
4. $1 \leq L_i \leq R_i \leq 10^9$,
5. $1 \leq X_i \leq 10^9$.

Приклади

stdin	stdout
3 2 7 11 1 5 3 8 4 7	5

Розв'язання задачі «A. Злови їх усіх!»

Очевидно, що задача зводиться до того, щоб знайти, скільки є непустих множин проміжків – таких, що їх перетин містить хоча б одну задачу точку.

Розв'яжемо спочатку задачу повільно – зі складністю $O(N^2)$. Для цього переберемо точку x, а також позначимо через y найближчу справа від x точку, у якій можна проводити тренування. Тоді нам потрібно знайти кількість підмножин, для яких виконуються дві умови:

1. ліва границя рівна x,
2. права границя не лівіше за y.

Для того, щоб знайти цю кількість, потрібно знати два числа:

1. a – кількість проміжків, для яких $L_i = x$, $R_i \geq y$,
2. b – кількість проміжків, для яких $L_i < x$, $R_i \geq y$.

Неважко побачити, що тоді шукана кількість множин буде рівною $(2a - 1) \cdot 2b$ (-1 в першому множнику гарантує, що ліва границя перетину буде саме x). Додавши знайдені значення для всеможливих x, отримаємо відповідь до задачі. Оскільки значення a та b можна легко знайти одним проходом по всіх проміжках, складність алгоритму – $O(N^2)$.

Для того, щоб пришвидшити розв'язок, потрібно вміти підтримувати значення a та b під час перебору точки x зліва направо. Для цього достатньо

потрібні проміжки підтримувати в структурі set і оновлювати множину при зміні значення x. Складність розв'язку в такому випадку – $O(N \log N)$.

B. Назва для покемонів

Тарас знайшов новий вид покемонів, і тепер йому залишилось тільки придумати назву для нього. Ваше завдання – допомогти йому в цьому.

Наразі у Тараса є стрічка T, яка складається з символів англійського алфавіту, а також знаків запитання. Для того, щоб завершити процес придумування назви, потрібно кожен знак запитання замінити на довільний символ.

Крім цього, відомо, що новий вид споріднений з іншим, давно відкритим видом під назвою S. Оскільки Тарас хоче, щоб назва нового виду відображала цю спорідненість, вона має містити S як підстрічку.

Визначте, чи може Тарас замінити знаки запитання в назві T на символи таким чином, що відобразити спорідненість двох покемонів в назві.

Формат вхідних даних

У першому та другому рядках задано дві стрічки T та S відповідно.

Формат вихідних даних

В єдиному рядку виведіть "`YES`", якщо Тарас може придумати назву для нового покемона, або "`NO`" в протилежному випадку (без лапок).

Обмеження

1. $1 \leq |T|, |S| \leq 1000$.

Приклади

stdin	stdout
PI??PIP?KA???X PIKACHU	YES

Розв'язання задачі «B. Назва для покемонів»

Оскільки обмеження в задачі невеликі, можна перебрати індекс початку входження S в T і явно перевірити, чи може таке входження існувати. Для того, щоб перевірити, достатньо пройтись по відповідних символах і перевірити, чи немає неоднакових символів у тих позиціях, де в T є не знак запитання.

Складність розв'язку – $O(N^2)$.

C. Хто швидше

Як відомо, зловити покемона – не задачу розв'язати, потрібно бути швидшим за своїх суперників. Тож Володя зрозумів, що велосипед – не найкращий засіб пересування для ловлі покемонів, і вирішив використовувати телепорт.

Спочатку Володя знаходиться в точці 0 нескінченної прямої. Покемон знаходиться в точці X. Телепорт налаштований так, що Володя може за одну хвилину дістатись з точки A в точку B тільки якщо $|A - B|$ є степенем двійки, тобто $|A - B| = 2^k$, для певного цілого $k \geq 0$. Зауважте, що Володя може також потрапити в точку з від'ємною координатою. Та не все так просто, адже також є N монстрів, яких Володя поки боїться, бо покемона він ще не зловив. i-ий монстр розташований у точці A_i. Тож Володя аж ніяк не хоче заходити в будь-яку точку, де знаходиться монстр.

Допоможіть Володі та знайдіть, за скільки хвилин він найшвидше зможе зловити покемона.

Формат вхідних даних

У першому рядку задано два цілих числа N та X – кількість монстрів та координата покемона. У наступному рядку задано N цілих чисел, розділених пробілами – координати монстрів. Гарантується, що у точці X немає жодного монстра.

Формат вихідних даних

В єдиному рядку виведіть одне ціле число – відповідь до задачі.

Обмеження

1. $1 \leq N \leq 10^5$,
2. $0 < X \leq 10^8$,
3. $0 < A_i \leq 10^9$,
4. $A_i \neq X$.

Приклади

stdin	stdout
4 3 1 2 4 7	2

Пояснення до прикладів

Маршрут Володі $0 \to -1 \to 3$.

Розв'язання задачі «C. Хто швидше»

Нехай спершу немає монстрів. Нам необхідно за найменшу кількість хвилин перейти з точки 0 в точку X. Скажемо, що навпаки потрібно перейти з X в 0. Розкладемо X у двійкову систему числення. Для кожного i можна зробити максимум один хід 2^i, адже два однакові ходи можна замінити на один хід 2^{i+1}. Будемо розглядати число X від меншого до більшого біту. Візьмемо найменший ненульовий біт k. Тоді потрібно зробити хід $+2^k$ або -2^k.

1) -2^k означає, що ми міняємо k-ий біт на 0.
2) $+2^k$ означає, що ми міняємо k-ий біт і всі наступні одиничні біти на 0, а перший нульовий біт на 1.

З цього можна отримати наступний алгоритм.

Зробимо -2^k, якщо (k + 1)-ий біт 0, $i + 2^k$ навпаки.

Чому це так? Якщо (k+1)-ий біт 0, то друга операція просто перенесе одиницю з k-го на (k + 1)-ий біт, а перша повністю забере цю одиницю. Тому перша операція вигідніша.

Якщо (k + 1)-ий біт 1, то друга операція замінить, як мінімум, дві одиниці на нулі. Якщо вигідно зробити першу операцію, то вигідно також робити першу

операцію для $(k + 1)$-го біту. Це означає, що ми зробимо $-2^k - 2^{k+1} = 2^k - 2^{k+2}$. Тобто можливо отримати той же результат, якщо спершу зробити другу операцію.

Нехай мінімальна кількість операцій без монстрів d.

Утворимо вектор, де i-те значення буде -1, 0 або 1 – коефіцієнт при 2^i.

Надалі скористаємось нерівністю $2^i > 2^{i-1} + 2^{i-2} + \cdots + 2^0$. Старший ненульовий коефіцієнт рівний -1, інакше сума всіх ходів – додатна, і перейти з X в 0 неможливо. Скористаємось рівністю $-2^i = -2^{i+1} + 2^i$. Тобто ми один хід розбили на 2.

Тепер монстри вже є. Зробимо хід -2^{i+1}. Сума всіх решти ходів додатна (за тією ж нерівністю), тому зараз ми зайшли у від'ємну координату. Всі монстри знаходяться у додатних координатах, тому цей хід валідний. Надалі можливо зробити всі ходи першого типу, потім всі ходи другого типу. Так ми перейдемо в точку 0, проходячи тільки по точках з від'ємними координатами. Тож ми побудували шлях довжини $d + 1$.

Потрібно ще перевірити, чи існує шлях довжини d. Скажемо, що точка з координатою t – хороша, якщо вона може теоретично лежати на найкоротшому шляху з 0 в X. Зробимо пошук у глибину з точки 0 в точку X, заходячи тільки в хороші точки. Виявляється, що найбільша кількість відвіданих станів можлива при $X = 1100110011\ldots_2$. При цих обмеженнях їх може бути близько 10^6. Нехай кількість відвіданих станів рівна M, тоді складність розв'язку $O(M \log^2 X)$.

D. Дерево

Щоб хоч трішки відволіктися від покемонів, Андрій вирішив порозв'язувати алгоритмічні задачки. Недавно йому трапилася така проблема: за заданим кореневим деревом знайти мінімальну кількість вершин, які потрібно видалити з нього так, щоб воно стало двійковим деревом із тим самим коренем.

Двійковим називається дерево, у якому в кожної вершини не більше, ніж два сини. Зауважте, що після видалення вершин повинна залишитися рівно одна компонента зв'язності, і вона повинна містити корінь дерева.

Допоможіть Андрію розв'язати описану задачу.

Формат вхідних даних

У першому рядку задано одне ціле число N – кількість вершин у дереві. У наступних $N-1$ рядках задано пари цілих чисел A_i та B_i, які описують напрямлені ребра дерева. Вершини дерева пронумеровані від 1 до N включно, а вершина з номером 1 є коренем.

Формат вихідних даних

У єдиному рядку виведіть одне ціле число – відповідь до задачі.

Обмеження

1. $1 \leq N \leq 10^5$.

Приклади

stdin	stdout
4 1 2 1 3 1 4	1

Розв'язання задачі «D. Дерево»

Розв'яжемо задачу методом динамічного програмування. Нехай R_v – мінімальна кількість вершин, які потрібно видалити, щоб піддерево з коренем у вершині v стало двійковим. Очевидно, що для довільного листка дерева R_v рівне 0.

Нехай нам потрібно порахувати значення R_v для довільної вершини, знаючи значення для всіх його синів. Очевидно, що ми можемо видалити всі вершини, окрім v, і дерево буде двійковим. Але ми також можемо залишити не більше, як дві вершини. Для того, щоб визначити, які саме вершини вигідно

залишити, посортуємо всіх синів за значенням $R_v - S_v$ (де S_v – кількість вершин в піддереві вершини v) і виберемо дві найменші.

E. Кольорові відрізки

«Досить із мене тих покемонів», – подумав Тарас і відкрив цікаву гру-головоломку.

Згідно з правилами цієї гри, гравцю задається набір із N різних точок на прямій $Y = 4$, пронумерованих зліва направо від 1 до N включно, а також M різних точок на прямій $Y = 7$, аналогічно пронумерованих від 1 до M включно. Деякі пари цих точок (із різних прямих) з'єднано відрізком, пофарбованим в один із двох кольорів – червоний або зелений. Завданням гравця є вибрати найбільшу множину відрізків таким чином, щоб виконувалися наступні дві умови:

- Кожна із $N + M$ точок належить не більше, ніж одному відрізку.
- Не існує пари відрізків різного кольору, які перетинаються.

Допоможіть Тарасу знайти найбільшу кількість відрізків, які він може вибрати.

Формат вхідних даних

У першому рядку задано три цілих числа N, M та K. У наступних K рядках описуються відрізки у форматі A_i B_i C_i, де A_i та B_i описують індекси точок, які з'єднує відповідний відрізок, а C_i може бути або R або G (червоний та зелений колір відповідно).

Формат вихідних даних

У єдиному рядку виведіть одне ціле число – відповідь до задачі.

Обмеження

1. $1 \le N, M \le 44$,
2. $1 \le A_i \le N$,
3. $1 \le B_i \le M$.

Приклади

stdin	stdout
2 2 2 1 2 R 2 1 G	1

Розв'язання задачі «E. Кольорові відрізки»

Розв'яжемо задачу методом динамічного програмування. Нехай стан динаміки $R[a][b]$ – максимльна кількість відрізків, які ми можемо вибрати, якщо ми розглядаємо тільки вершини з номерами від 1 до a включно (з першої долі) і від 1 до b включно (з другої долі). Оскільки перетинатися можуть вершини тільки якогось одного кольору, переберемо, який саме це колір, а також пару індексів c та d ($c > a,\ d > b$), і розглянемо тільки ті ребра вибраного кольору, які мають кінці в проміжках $[a+1 \ldots c]$ та $[b+1 \ldots d]$. Оскільки всі ці ребра одного кольору, задача знаходження найбільшої кількості ребер, що можна вибрати, зводиться до задачі максимальної паросполуки у дводольному графі, яку можна розв'язати алгоритмом Куна. Після цього спробуємо покращити стан $R[c][d]$ новим значенням. Такий підхід гарантує, що жодні два відрізки різного кольору не перетнуться, а також що в кінці значення $R[N][M]$ буде містити правильну відповідь до задачі.

Складність такого алгоритму – $O(N^7)$. Для пришвидшення розв'язку до $O(N^6)$ будемо спершу перебирати значення c, а потім d, запускаючи алгоритм Куна з кожної вершини по черзі.

F. Суперзакляття

Щоб зловити покемона, необхідно спершу сказати чарівне закляття. Володя знає одне закляття, але він вважає, що його можна зробити ще сильнішим, так, щоб усі покемони самі бігли до нього, коли він його промовить.

Як відомо, Володя – програміст, тому саме закляття складається з цифр. Звісно ж, зі щасливих цифр. Нагадаємо, що щасливими називаються цифри 4 та 7.

Володя вважає, що це закляття буде суперзакляттям, якщо всі цифри у ньому будуть посортованими, тобто спершу будуть іти всі четвірки, потім усі сімки. За одну хвилину Володя може взяти будь-яку підстрічку закляття та повернути її. Нехай, закляття Володі рівне 4774447, за одну хвилину він може отримати наступні закляття (жирним позначено підстрічку, яку перевернув Володя): 4**477**447, 47**4447**7, 4774**74**, **774**4447 та інші.

Допоможіть Володі та знайдіть, за яку мінімальну кількість часу Володя зможе отримати суперзакляття.

Формат вхідних даних

У першому рядку задано єдине ціле число N – довжина закляття Володі. У наступному рядку знаходиться саме закляття.

Формат вихідних даних

У єдиному рядку виведіть одне ціле число – відповідь до задачі.

Обмеження

1. $1 \leq N \leq 10^5$.

Приклади

stdin	stdout
10 4447477744	2

Пояснення до прикладів

Один із можливих варіантів, як Володі отримати суперзакляття за 2 хвилини:

1) 44474**44777**,
2) 444**4447**777.

Розв'язання задачі «F. Суперзакляття»

Додамо до нашої стрічки першим символом 4. Наприклад, зі стрічки 4774 утвориться стрічка 44774. Відповідь після цього зменшитися не могла.

Порахуємо cnt - кількість груп однакових символів, що йдуть підряд, cnt_4 – кількість груп четвірок, cnt_7 – кількість груп сімок. Наприклад, для стрічки 447447774: $cnt = 5$, $cnt_4 = 3$, $cnt_7 = 2$.

Укінці має залишитися лише одна група четвірок, а за кожну операцію можна зменшити кількість груп четвірок лише на 1. Тому відповідь не може бути меншою за $cnt_4 - 1$.

Покажемо, що за таку кількість операцій можна посортувати стрічку. Алгоритм: поки є хоча б 2 групи четвірок, візьмемо другу групу четвірок, першу групу сімок та перевернемо. Отримаємо на одну групу четвірок менше.

Наприклад, для стрічки 447447774:

1) 44**447**774,
2) 4444**4777**.

Складність такого алгоритму $O(N)$.

G. Секрет

Володя розгадав секрет покемонів!

Виявляється, кожного покемона можна задати певною матрицею чисел, причому розміри цієї матриці задають вид покемона, а самі значення – сила та інші параметри. Також Володя знає, що якщо існує покемон, який задається матрицею A, та покемон, який задається матрицею B, то точно існує покемон, який задається матрицею $A \cdot B$ (множення матриць). Нагадаємо, що якщо помножити матрицю розміром $n \times m$ та матрицю з розміром $m \times k$, то утвориться матриця розміром $n \times k$.

Володя знає N покемонів. i-го покемона можна задати матрицею розміром $a_i \times b_i$. Йому стало цікаво, скільки всього видів покемонів існує точно. Володя вважає, що вид точно існує, якщо даних про N покемонів достатньо для того, щоб гарантувати, що є покемон цього виду.

Допоможіть Володі та знайдіть, скільки ж видів покемонів є.

Формат вхідних даних

У першому рядку задано єдине ціле число N – кількість покемонів, яких знає Володя. У наступних N рядках задано по парі цілих чисел, розділених пробілами a_i та b_i – розміри матриці, яка задає i-го покемона.

Формат вихідних даних

У єдиному рядку виведіть одне ціле число – відповідь до задачі.

Обмеження

1. $1 \leq N \leq 5000$,
2. $0 < a_i, b_i \leq 10^5$.

Приклади

stdin	stdout
4 1 4 47 74 4 7 1 4	4

Пояснення до прикладів

Існують покемони видів 1×4, 4×7, 47×74 та 1×7. Покемон 1×7 точно існує тому, що є покемони 1×4 та 4×7.

Розв'язання задачі «G. Секрет»

Утворимо граф, де вершинами є всі можливі розміри. Для кожної матриці $a \times b$ проведемо напрямлене ребро з вершини a в вершину b.

Тепер можливо утворити матрицю $X \times Y$, якщо існує шлях ненульової довжини від вершини X у вершину Y. Доведемо це.

Якщо існує шлях із вершини X у вершину Y, то існує послідовність матриць $X \times A_1$, $A_1 \times A_2$, $A_2 \times A_3$, …, $A_k \times Y$. Перемноживши їх, отримаємо матрицю $X \times Y$.

Аналогічно і в іншу сторону, якщо можна утворити матрицю $X \times Y$, то існує шлях від вершини X до вершини Y.

Для того, щоб знайти кількість матриць вигляду $A \times Y$, потрібно знайти кількість вершин, досяжних із вершини A. Це можна зробити пошуком уширину або вглибину. Також матрицю $A \times A$ можна утворити, тільки якщо існує цикл через вершину A.

Зауважимо, що запускати пошук потрібно тільки з тих вершин, для яких існує хоча б одна матриця з таким розміром. Складність такого алгоритму $O(N^2)$.

День 5. Контест Богдана Прищенка

Теоретичний матеріал. Хешування

```
      Hashing in strings based problems.
      This code compares substrings using two hashes (one
      uses 2^64 as a modulo, another 10^9 + 7)
      Based on problem C from here: http://codeforces.ru/gym/100133
  ***********************************************************************
#include <iostream>
#include <fstream>
#include <cmath>
#include <algorithm>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <cstdlib>
#include <cstdio>
#include <string>
#include <cstring>
#include <cassert>
#include <utility>
#include <iomanip>

using namespace std;

const int MAXN = 105000;
const int mod = (int) 1e9 + 7;
const int p = 53;

string s;
int n, m;
long long h1[MAXN], h2[MAXN];
long long pp1[MAXN], pp2[MAXN];

long long getHash1(int l, int r) {
    l--; r--;
    long long h = h1[r];
    if (l > 0)
        h -= h1[l - 1];
    h *= pp1[n - 1 - r];
    return h;
}

long long getHash2(int l, int r) {
    l--; r--;
```

```
    long long h = h2[r];
    if (l > 0)
        h = (h - h2[l - 1] + mod) % mod;
    h = (h * pp2[n - 1 - r]) % mod;
    return h;
}

int main() {
    assert(freopen("substrcmp.in","r",stdin));
    assert(freopen("substrcmp.out","w",stdout));

    getline(cin, s);
    n = (int) s.length();

    pp1[0] = 1; pp2[0] = 1;
    for (int i = 1; i <= n; i++) {
        pp1[i] = pp1[i - 1] * p;
        pp2[i] = (pp2[i - 1] * p) % mod;
    }

    h1[0] = s[0] - 'a' + 1;
    h2[0] = (s[0] - 'a' + 1) % mod;
    for (int i = 1; i < n; i++)
{
h1[i] = h1[i - 1] + pp1[i] * (s[i] - 'a' + 1);
h2[i] = (h2[i - 1] + pp2[i] * (s[i]-'a'+ 1))%mod;
}

    scanf("%d", &m);

    for (int i = 1; i <= m; i++) {
        int a, b, c, d;
        scanf("%d %d %d %d", &a, &b, &c, &d);
        if (getHash1(a, b) == getHash1(c, d) && getHash2(a, b) ==
getHash2(c, d))
            puts("Yes");
        else
            puts("No");
    }
    return 0;
}
```

Задачі та ідеї розв’язання

A. Болотний лікар

Під час останнього походу до центру Зони сталкер Штир потрапив у невідому раніше аномалію. Справа невтішна – у Штиря почали відростати ікла і почервоніли очі. Тепер від перетворення на кровососа Штиря зможе врятувати тільки болотний лікар...

Штир не знає адреси електронної пошти лікаря, але припускає, що він є в КПК загиблого сталкера Семецького, який Штир недавно знайшов. Біда в тому, що хитрий Семецький не використав адресну книгу, а ховав адреси електронної пошти у вмісті великого текстового файлу. Незважаючи на те, що Штир може прочитати весь цей файл, таємницю того, який фрагмент файлу є адресою лікаря, Семецький забрав із собою в могилу.

Штир вирішив перебрати всі фрагменти тексту, які можуть бути адресами, відправляючи по одному листу на кожну з адрес. Скільки ж листів йому доведеться відправити?

Адреса електронної пошти в околицях Зони складається з імені користувача і домену, розділених символом «@». Ім'я користувача і домен – непусті рядки, що складаються з малих англійських літер і крапок. При цьому вони не можуть починатися з крапки, закінчуватися крапкою і містити дві крапки поспіль.

Формат вхідних даних

У вхідних даних записано вміст файлу на КПК Семецького. У файлі можуть траплятися тільки символи з ASCII-кодами від 32 до 126 і переклади рядків. Обсяг вхідних даних не перевищує 106 байт.

Формат вихідних даних

Виведіть, скільки різних електронних адрес містить файл Семецького.

Приклади

stdin	stdout
a.b@c.d b@c.de. @qq q@.q 2 5 1 2	5

Пояснення до прикладів

У файлі трапляються такі адреси: «a.b@c», «a.b@c.d», «b @ c», «b@c.d», «b@c.de».

Розв'язання задачі «A. Болотний лікар»

Джерело: Чемпіонат Уралу 2011; Timus 1835

Коротко про розв'язок: акуратна реалізація через LCP і суфіксний масив.

Розв'язок: згадаємо ідею підрахунку кількості різних підрядків – будуємо суфіксний масив і для кожного суфікса додаємо до відповідь всі його префікси, котрі довші за найдовший спільний префікс з попереднім елементом суфіксного масиву. Розвинемо цю ідею для того, аби отримати розв'язок нашої задачі. Ми мали одне обмеження знизу – довжина спільного префікса, і одне обмеження зверху - довжина всього суфікса. Акуратно розглянемо всі правила з умови, аби отримати додаткові обмеження – дві точки поспіль дають обмеження зверху, перший символ @ дає обмеження знизу, другий символ @ дає обмеження зверху і т.д.. Якщо пропарсити вхідні дані та прекалькнути перші справа "цікаві" позиції кожного типу (@, пара точок поспіль), то можна отримати ці додаткові обмеження за $O(1)$. Припускаючи, що ми вибрали валідну позицію початку підрядка, нам потрібно підрахувати кількість валідних кінцевих позицій в певному вікні; для цього можна прекалькулювати часткові суми кількостей валідних позицій на префіксах, і відповідати на запит суми на відрізку через різницю сум на двох префіксах за $O(1)$. Ми описали правильний розв'язок з асимптотикою $N \log N$, але він є доволі повільним для заданих обмежень. Отримати AC, використовуючи для всіх операцій хеші, видається неймовірно складним завданням, і навіть більшість стандартних реалізацій ризикують отримати TL. Для знаходження довжини спільного префікса використаємо алгоритм Касаї, аби отримати лінійну асимптотику в цій частині розв'язку. Побудову суфіксного масиву також можна проводити за лінійний час, але йдеться про складнішу реалізацію і значну приховану константу; можна отримати розв'язок зі стандартним алгоритмом побудови суфіксного масиву через двійкові підйоми за $N \log N$, якщо реалізувати його оптимально – без зайвих стрибків по пам'яті, взяттів за модулем і т.д.; також корисною буде така оптимізація – як тільки певна ітерація подвоєння не збільшила кількість

класів еквівалентності в суфіксному масиві, припинити його побудову; можна показати, що всі наступні ітерації також нічого не змінять. Розв'язок з такими оптимізаціями отримує AC з двократним запасом.

B. Однакові квадрати

На розборі задач одного з контестів петрозаводських зборів Вова і Саша посперечалися, хто з них зможе знайти за 300 хвилин у матриці розміром $N \times M$, що складається з малих латинських букв, пару однакових квадратів найбільшого розміру. Квадрати можуть накладатися один на одного, але не можуть збігатися. Хто знайшов пару більшого розміру, той і виграв. Поруч проходив Петя, подивився на матрицю, сказав, що оптимальна пара квадратів має сторону K, і пішов далі. Вова і Саша досі намагаються знайти цю відповідь. Можливо, ви скажете, яку пару квадратів мав на увазі Петя?

Формат вхідних даних

У першому рядку через пробіл дано два цілі числа N і M. $1 \leq N, M \leq 500$. У наступних N рядках по M символів у кожному рядку наведена матриця з малих латинських букв.

Формат вихідних даних

У першому рядку виведіть ціле число K, про яке сказав Петя. У наступних двох рядках виведіть координати лівої верхньої клітинки кожного з квадратів. Якщо існує більше однієї пари однакових квадратів найбільшого розміру, то можна вивести будь-яку з них. Ви можете вважати, що ліва верхня клітинка матриці має координати $(1, 1)$, права нижня - координати (N, M). Якщо Петя сказав, що в матриці не існує пари однакових квадратів, виведіть єдине число 0.

Приклади

stdin	stdout
5 10 ljkfghdfas isdfjksiye pgljkijlgp eyisdafdsi lnpglkfkjl	3 1 1 3 3

Розв'язання задачі «B. Однакові квадрати»

Джерело: Чемпіонат УрФУ 2006; Timus 1486

Коротко про розв'язок: бінпошук по відповіді + хешування в 2D.

Розв'язок: функція існування пари рівних квадратів є монотонною - якщо існує шукана пара квадратів для довжини сторони К, то вона існує також і для $K-1$; якщо таких пар нема для сторони K, то їх нема і для сторони $K+1$. Для перевірки певного значення K можна згенерувати хеші всі квадратів з такою стороною (їх є $O(N^2)$), та перевірити отриманий набір на наявність пари рівних хешів, скориставшись сортуванням. Для заданих обмежень не можна тривіальним чином скласти тест проти хешування за модулем вбудованих типів даних, це дозволяє використовувати складні структури хешів без ризику отримати TL. Позбувшись від повільної операції взяття за модулем, можна використати одночасно три чи навіть чотири хеші. При правильній реалізації розв'язок з двома хешами також отримує AC.

C. Орган Дейві Джонса

Гіббс: Тихо. У тутешніх водах стільки піратів... Хочете біду на нас накликати?

Джеймс Норрінгтон: Містер Гіббс, досить.

Гіббс: Вона співала про піратів. Не на добро такі пісні, коли корабель оповитий туманом. Згадайте моє слово!

Одразу дві мелодії не виходять із голови Дейві Джонса. То одна, то інша періодично зринає у його свідомості. Щоби позбутися від набридливих мелодій, Дейві Джонс вирішив зіграти їх на своєму органі. Спочатку він хоче

послідовно виконати першу і другу мелодії. А потім Дейві Джонс збирається зіграти ті ж ноти, що і в перший раз, але у зворотному порядку.

Якщо ці дві композиції прозвучать абсолютно однаково, то, за задумом Дейві Джонса, його підсвідомість перестане розрізняти мелодії і, нарешті, вони залишать його в спокої.

Джонс записав ноти обох мелодій. У тривалості мелодій він упевнений, а от у тому, з якого місця починається кожна, – ні, адже у підсвідомості вони крутяться циклічно, і в кожній можна вибрати будь-яке місце як початок.

Формат вхідних даних

У першому рядку записано n малих латинських букв – запис першої мелодії за нотами. У другому рядку – за допомогою m букв аналогічно задана друга мелодія ($1 \leq m < n \leq 10^5$). Порядок нот у мелодії відповідає порядку їх відтворення з точністю до вибору початку композиції.

Формат вихідних даних

Якщо зіграти композиції задуманим Дейві Джонсом чином неможливо, виведіть «`No`». Інакше в першому рядку виведіть «`Yes`», а в другому – пару цілих чисел через пробіл – номери нот у першій і другій мелодіях відповідно, які варто вибирати як початок. Нумерація нот відповідає запису мелодій у вхідних даних і починається з одиниці. Якщо існує кілька можливих рішень, то виведіть будь-яке.

Приклади

stdin	stdout
cdedab bac	Yes 5 3
aaaa bbb	No

Розв’язання задачі «С. Орган Дейві Джонса»

Джерело: Чемпіонат УрФУ 2012; Timus 1937

Коротко про розв'язок: реалізація перебору з використанням суфіксного масиву або хешів.

Розв'язок: нехай ми знаємо, на якій позиції розпочинається запис першого рядка. Оскільки перший рядок є довшим за другий, і результат конкатенації повинен бути паліндромом, то це однозначно задає нам те, що потрібно отримати з другого рядка циклічним зсувом. Наївна реалізація працює за N^2 – переберемо позиції старту в другому рядку та перевіримо, чи якась з них дає потрібний рядок. Можна побудувати суфіксний масив над вхідними рядками та їх реверсами (а точніше - над подвоєними копіями, щоб коректно врахувати циклічність) і працювати з ним для прискорення розв'язку, або ж скористатись ідеєю хешування. Розв'язок з хешуванням видається більш інтуїтивним - для кожного циклічного зсуву другого рядка обчислимо його хеш; для кожного циклічного зсуву першого рядка обчислимо хеш рядка, котрий потрібно приписати для отримання паліндрома (як хеш реверсу певного префікса) і перевіримо, чи серед хешів другого рядка є потрібне значення. Асимптотика розв'язку - $N \log N$.

D. Ключові підрядки

Незважаючи на те, що програмний комітет працює досить злагоджено, не обходиться і без запеклих суперечок. Наприклад, про те, який клієнт системи контролю версій зручніший у використанні: програма з графічним інтерфейсом чи консольна утиліта.

Розглянемо будь-яку команду консольної утиліти. Підрядок цієї команди, яка не є підрядком ніякої іншої команди утиліти, можна назвати ключовим, оскільки він однозначно ідентифікує команду. В останніх версіях утиліти можна не вводити команду повністю, достатньо лише ввести будь-який ключовий підрядок цієї команди.

Прихильник консольної утиліти хоче переконати весь інший програмний комітет використовувати саме її. Щоби показати, наскільки швидко і зручно з

нею працювати, він хоче знайти для кожної команди ключовий підрядок мінімальної довжини. Допоможіть йому в цьому.

Формат вхідних даних

У першому рядку записано ціле число n ($2 \leq n \leq 1000$) - кількість команд консольної утиліти. У наступних n рядках записані ці команди – непусті рядки, що складаються з малих латинських букв. Довжина кожної команди не перевищує 100. Ніяка команда не є підрядком іншої команди.

Формат вихідних даних

Виведіть n рядків. В i-му рядку повинен бути записаний будь-який із найкоротших ключових підрядків i-ї команди (команди пронумеровані в тому порядку, у якому вони дані на вході).

Приклади

stdin	stdout
3 abcm acm bcd	ab ac d

Розв'язання задачі «D. Ключові підрядки»

Джерело: NEERC 2009 Eastern Subregional; Timus 1713

Коротко про розв'язок: пошук всіх хороших підрядків хешуванням.

Розв'язок: скористаємось хешуванням. Спочатку для кожного рядка вхідних даних виділимо всі наявні там підрядки. Після цього злиємо всі отримані набори та виберемо ті підрядки, котрі зустрічаються тільки один раз - про такі підрядки ми знаємо, що вони входять тільки в одне слово з вхідного набору (один або декілька разів), отже кожен такий підрядок може бути ключовим. Тепер можна ще раз розглянути всі підрядки всіх рядків і для кожного з них перевірити, чи зустрічається він в списку "хороших" підрядків. Серед всіх таких підрядків вибираємо найкоротший (будь-який з них). Обмеження у задачі виставлені достатньо жорстко; типова помилка -

використовувати можливості STL, map з хешів в рядки і т.д.; такі розв'язки зазвичай отримують TL – у той час як оптимальна реалізація описаного розв'язку вкладається в обмеження часу з трикратним запасом. Для прискорення розв'язку варто працювати з векторами хешів, після сортування такого вектору можна легко визначити кратність кожного значення одним лінійним проходом, а пошук входження виконується *lower_bound*'ом за логарифмічний час з малою константою. Для розв'язування такої задачі можна також скористатись бором, аби позбутись логарифмічного множника в асимптотиці, але такий розв'язок є вимогливим до пам'яті і при заданих обмеженнях повинен отримувати ML.

E. Книга Сандро

Минуло вже чимало років відтоді, коли Ліч Сандро пішов на заслужений відпочинок. Іноді вечорами, коли йому стає зовсім сумно, він бере в руки книгу, яку йому подарували вихованці-маги з нагоди виходу на пенсію.

Ось і зараз великий маг узяв із полиці книгу і занурився в читання. В одному з розділів розповідалося про знамените відкриття Сандро - багато років тому він придумав універсальне заклинання. Виявилося, що будь-який його підрядок (послідовність букв, що йдуть підряд) теж є заклинанням, а сила будь-якого заклинання дорівнює кількості раз, яку це заклинання трапляється в універсальному (наприклад, рядок «ue» трапляється в рядку «queue» двічі, а рядок «aba» в рядку «abababa» - тричі).

Зараз у Сандро багато вільного часу, і він вирішив знайти найсильніше заклинання. Допоможіть йому в цьому.

Формат вхідних даних

Єдиний рядок містить універсальне заклинання, яке відкрив Сандро. Заклинання - непорожній рядок з малих латинських букв довжиною не більше 50.

Формат вихідних даних

Виведіть будь-яке із заклинань, що мають, на думку Сандро, найбільшу силу.

Приклади

stdin	stdout
tebidohtebidoh	tebidoh

Розв'язання задачі «E. Книга Сандро»

Джерело: УРКОП 2009; Timus 1723

Коротко про розв'язок: необхідно знайти букву, котра зустрічається в рядку найбільшу кількість разів.

Розв'язок: нехай якийсь підрядок є відповіддю для нашої задачі; в такому випадку всі його підрядки також підходять, оскільки вони входять не меншу кількість разів, бо кожне входження рядка дає нам також і входження всіх його підрядків. Звідси очевидно, що завжди існує розв'язок довжини 1, тобто достатньо серед всіх рядків довжини 1 вибрати той, котрий зустрічаєтся найбільшу кількість разів. Такий розв'язок має лінійну асимптотику; обмеження в задачі дуже низькі, тому будь-який наївний розв'язок також проходить – можна просто перебрати всі підрядки і для кожного підрахувати кількість входжень.

F. Паліндроми

У штаб-квартиру 'Ю.С.Роботс енд Меканікл Мен Інкорпорейтед' надійшла тривожна анонімка. У ній було сказано, що конкуруюча компанія 'Роботс Анлімітед' упровадила свого співробітника агентом в 'Ю.С. Роботс '. Усе, здавалося б, керівництву 'Ю.С. Роботс' доведеться почати довгу, кропітку і неймовірно секретну роботу з виявлення шпигуна. Але, на щастя, укінці анонімки обмовлено, як відбувається передача отриманих відомостей. Агент друкує певну статтю в альманасі 'Соляріс' (звичайно, у завуальованій формі), а потім у компанії 'Роботс Анлімітед' цю газета проглядає спеціальний пристрій (надсекретна розробка 'Роботс Анлімітед', модель NPRx8086).

Прочитавши записку, генеральний директор 'Ю.С. Роботс' згадав, що недавно до них на роботу був прийнятий колишній співробітник 'Роботс Анлімітед' Вася Сидоров. Ні, не може бути, що саме ця людина і є агент. Навпаки, директор чітко знав, що Вася був буквально видворений із 'Анлімітед' за два дні перед тим, як начальству компанії була представлена модель NPRx8086, і, природно, був сильно розлючений на керівництво. "Звичайні внутрішньоорганізаційні розборки, хтось захотів привласнити собі винахід і усунув небажаного конкурента", - правильно оцінив цю ситуацію директор 'Ю.С. Роботс'. - Але це й на краще, тепер у нас буде людина, що відмінно знає багато розробок 'Анлімітед', треба би їй доручити створення програми, яка б і знаходила повідомлення агента в 'Солярісі' ".

Коли Вася Сидоров отримав завдання, він трохи розгубився: "Як у такому величезному альманасі, як 'Солярис', можна знайти відносно коротке повідомлення шпигуна. Це ж - голка в стозі сіна". Але тут до нього прийшла геніальна думка. У той день, коли його вигнали з 'Анлімітед', він доробляв останній модуль пристрою NPRx8086, який повинен був визначати напрямок, у якому в апарат був завантажений текст. А оскільки Васю вигнали, коли він ще не встиг довести роботу до кінця, то з великою ймовірністю можна припускати, що цей модуль так і не був ніким дороблений, а пристрій у такому вигляді і було представлено керівництву. І якщо це так (а ми-то знаємо, що це дійсно так), то NPRx8086 буде вибирати абсолютно випадковий порядок прочитання тексту, а значить, абсолютно правильно сприйматиме лише повідомлення, які читаються однаково в обох напрямках. Залишилося тільки знайти максимальне за довжиною повідомлення, яке відповідає цій властивості (звичайно, агент буде прагнути передати настільки великий текст, наскільки він зможе).

Ваше завдання допомогти Васі написати цю програму. Слід врахувати, що пристрій NPRx8086 читає, пропускаючи всі пробіли і розділові знаки. Однак Вася вже написав ту процедуру, яка видаляє з тексту всі пропуски і розділові знаки.

Формат вхідних даних

Містить усього один рядок (не довший за 1000 символів), що включає лише літери латинського алфавіту (усі пробіли і розділові знаки з рядка Вася вже видалив).

Формат вихідних даних

Максимальний за довжиною підрядок, що читається однаково в обох напрямках. Якщо максимальних за довжиною підрядків більше одного, вивести найлівіший із них.

Приклади

stdin	stdout
ThesampletextthatcouldbereadedthesameinbothordersArozaupalanalapuazorA	ArozaupalanalapuazorA

Розв'язання задачі «F. Паліндроми»

Джерело: Весняна першість школярів 2004; Timus 1297

Коротко про розв'язок: розв'яжемо задачу для кожного центра паліндрома, виберемо найкращий результат.

Розв'язок: наївний розв'язок має асимптотику $O(N^3)$ – для кожного з $O(N^2)$ рядків за $O(N)$ перевіримо, чи він є паліндромом. За замовчуванням цей розв'язок не є достатньо швидким для отримання AC. Прихована константа не є великою, і її можна зменшити ще більше, порівнюючи одразу по декілька символів - для цього можна розглянути рядок та його реверс, і сказати, що рядок є паліндромом тоді і тільки тоді, якщо відповідні входження в початковий рядок та в його реверс співпадають посимвольно – а порівнювати масиви чарів ми можемо одразу по 4 чи 8 за одну операцію, порівнюючи цілі машинні слова; також можна впорядкувати рядки, котрі ми перевірятимемо, за компаратором з умови - таким чином, що перший знайдений паліндром точно є відповіддю на задачу. З цими евристиками задачу вже можна здати, але надійнішим є розв'язок за $O(N^2)$ – зафіксуємо центр паліндрома і розширимо його настільки, наскільки це можливо, дописуючи по одному символу зліва і

справа доти, доки це не зіпсує паліндром. Серед всіх паліндромів для заданого центра нас цікавить тільки найбільший; він і є кандидатом на відповідь. Зауважимо, що задачу також можна розв'язати за $O(N)$, оскільки аналогічні обчислення (знаходження найбільшого паліндрома для кожного центру) виконуються за лінійний час з використанням алгоритму Манакера. В даній задачі в цьому нема потреби, оскільки квадратичний розв'язок без будь-яких проблем вкладається в обмеження.

G. Дивний діалог

Одна сутність на ім'я "one" розмовляє зі своїм другом, сутністю "puton", і нас цікавить їх розмова. "One" може говорити слова "out" і "output", крім того, він може називати свого друга на ім'я. "Puton" може говорити слова "in", "input" і "one". Вони прекрасно розуміють один одного і навіть пишуть діалоги в рядки без пробілів між словами.

Дано N рядків. Визначте, які з них є діалогами.

Формат вхідних даних

У першому рядку введення міститься одне невід'ємне ціле число $N \leq 1000$. Наступні N рядків містять непусті послідовності малих латинських букв. Загальна довжина всіх рядків не перевищує 10^7 символів.

Формат вихідних даних

Вивід складається з N рядків. Рядок містить слово "`YES`", якщо відповідний рядок вводу є певним діалогом сутностей "one" і "puton", в іншому випадку рядок містить "`NO`".

Приклади

stdin	stdout
6 puton inonputin oneputonininputoutoutput oneininputwooutoutput outpu utput	YES NO YES NO NO NO

Розв'язання задачі «G. Дивний діалог»

Джерело: Tetrahedron Team Online Contest May 2001; Timus 1102

Коротко про розв'язок: динамічне програмування "чи є даний префікс валідним діалогом?".

Розв'язок: В цій задачі можна використати ідею динамічного програмування. Нехай ми знаємо, що весь рядок є валідним діалогом. Якщо цей рядок має ненульову довжину - виділимо в ньому останнє слово діалогу, відкинемо його – і новий рядок також має бути валідним діалогом. Оскільки всі валідні слова мають малу довжину, і кількість цих слів також мала, можна порахувати значення динамічного програмування для всіх префіксів в порядку зростання довжини, а при перевірці чергового рядка-префікса - перебрати всі можливі слова, на які він може завершуватись; при цьому суфікс поточного рядка повинен точно співпасти зі словом-зразком, а для префікса повинна виконуватись умова валідності (оскільки він коротший за увесь рядок, то цю умову ми вже перевірили раніше). Такий розв'язок можна оцінити як $O(N)$ з великою прихованою константою, або ж $O(N \cdot SZ)$ загалом, де SZ – це сумарний розмір всіх валідних слів заданої мови.

Альтернативний розв'язок - побудувати автомат для заданої граматики, і перевірити чи цей автомат приймає вхідне слово.

H. Старенька Nokia

У Михайла є старенька Nokia і багато друзів. Так багато, що Мишко постійно доводиться витрачати купу часу на пошук потрібного номера.

Імена друзів у телефонному довіднику впорядковані за алфавітом. Від поточного імені в списку можна перейти до наступного за допомогою кнопки «вниз», а до попереднього - за допомогою кнопки «вверх». Крім того, список зациклений, тобто натискання кнопки «вверх» для найпершого імені в довіднику переводить на останнє ім'я в ньому, а натискання кнопки «вниз» для останнього імені переводить на перше ім'я.

Коли Мишко відкриває довідник, у ньому видно імена всіх друзів, а поточним є перше за алфавітом ім'я. Якщо почати набирати якесь слово на клавіатурі телефону, то в довіднику будуть відображатися тільки імена, що починаються на вже введену послідовність літер. Поточним у цьому випадку також стане ім'я, що йде раніше за інші в алфавітному порядку. Якщо після цього стерти останню введену букву, позиція в довіднику не зміниться, а доступними стануть усі імена, що починаються на коротший рядок. Якщо стерти всі букви, то доступними стануть усі записи в довіднику, а поточне ім'я не зміниться. Якщо після певного натискання повинна з'явитися послідовність літер, на яку не починається жодне ім'я, то телефон видасть неприємний звук і остання введена буква не з'явиться на екрані.

На одне натискання кнопки у Мишка йде рівно одна секунда. Перша буква на кнопці набирається за одне натискання, друга за два і т.д. Клавіатура телефону виглядає так:

	abc	def
ghi	jkl	mno
pqrs	tuv	wxyz

Даний список імен друзів Мишка. Для кожного імені обчисліть мінімальний час, за який Мишко зможе вибрати це ім'я в списку.

Формат вхідних даних

У першому рядку дано ціле число n - кількість записів у телефонному довіднику Мишка ($1 \leq n \leq 10^5$). Далі в n рядках записано по одному непорожньому слову з малих латинських букв. Слова перераховані в алфавітному порядку. Усі записи в довіднику різні. Сумарна довжина всіх слів не перевищує 10^5.

Формат вихідних даних

Виведіть n чисел через пропуск. i-е число повинно дорівнювати мінімальному часу знаходження i-го імені в довіднику.

Приклади

stdin	stdout
5 a aaa aab b d	0 1 2 2 1

Розв'язання задачі «H. Старенька Nokia»

Джерело: УРКОП 2011; Timus 1882

Коротко про розв'язок: суфіксні структури / хешування + алгоритм Дейкстри

Розв'язок: розглянемо граф, в котрому вершина - позиція в довіднику та довжина набраного префіксу. Його розмір $O(N)$ - щонайбільше 100 тисяч вершин для непорожніх префіксів та 100 тисяч вершин для порожніх префіксів. Припустимо, що нам відомі всі переходи в ньому – з кожної вершини для кожної дії (набрати букву, видалити букву, переміститись вверх або вниз) ми потрапимо в якусь іншу вершину, і знаходження найкоротших шляхів від стартової вершини до всіх інших дасть нам відповідь на задачу; для певного слова відповіддю є мінімальна з відстаней до всіх вершин, котрі відповідають позиції курсора на цьому слові. Маємо граф розміру менше $60N$, знайти в

ньому найкоротші шляхи можна алгоритмом Дейкстри чи Левіта або модифікаціями k-bfs (оскільки ваги на ребрах є малими числами). Як побудувати цей граф? Можна помітити, що нам достатньо навчитись відповідати на запити такого формату: взяти певний префікс одного з вхідних слів (або префікс з дописаною до нього однією додатковою буквою) і сказати підмасив вхідних слів, для слів з якого цей рядок є префіксом. Це можна робити, наприклад, використовуючи суфіксний масив; враховуючи тему лекції, доречно акцентувати увагу на розв'язку, котрий використовує хешування. Обчислимо значення хеша для кожного префіксу кожного рядка вхідних даних, та знайдемо для кожного хеша позицію першого та останнього входження. Тепер для відповіді на запит достатньо обчислити хеш нового рядка, що можна робити за $O(1)$, та дістати перше і останнє входження цього рядка в наш масив хешів - за допомогою lower_bound на відсортованому масиві чи використавши map для зберігання максимуму і мінімуму. Отримали розв'язок за $(N + M)\log N$, де M не перевищує $30N$.

I. Підпаліндроми

Дано слово і запити двох типів:

- замінити i-у букву у слові на букву a;
- перевірити, чи є підслово $s_j \ldots s_k$ паліндромом.

Формат вхідних даних

У першому рядку записано слово з n малих латинських букв. У другому рядку записано ціле число m - кількість запитів ($5 \le n, m \le 10^5$). Наступні m рядків містять запити.

Кожен запит має вигляд «`change` i a» або «`palindrome?` j k», де i, j, k - цілі числа ($1 \le i \le n$; $1 \le j \le k \le n$), а символ a – мала латинська буква.

Формат вихідних даних

На всі запити другого типу виведіть «`Yes`», якщо підслово $s_j \ldots s_k$ є паліндромом, і «`No`» в іншому випадку.

Приклади

stdin	stdout
abcda 5 palindrome? 1 5 palindrome? 1 1 change 4 b palindrome? 1 5 palindrome? 2 4	No Yes Yes Yes

Розв'язання задачі «I. Підпаліндроми»

Джерело: Чемпіонат УрФУ 2013; Timus 1989

Коротко про розв'язок: будемо підтримувати хеші для рядка та його реверсу і порівнювати хеші відповідних підрядків для відповіді на запит.

Розв'язок: скористаємось спостереженням – рядок є паліндромом тоді і тільки тоді, коли він співпадає зі своїм реверсом. Тоді ми можемо видозмінити запит рівності, звівши його до порівняння певного підрядка вхідного рядка та певного підрядка рядка-реверсу. Для порівняння цих рядків будемо обчислювати нормовані хеші та порівнювати їх – у припущенні, що рівність хешів майже точно означає рівність рядків. Ми не можемо скористатись ідеєю часткових сум для обчислення хеша за $O(1)$, оскільки в задачі присутні запити оновлення значень. Заміна одного символа вплине на значення префікс-сум для $O(N)$ префіксів, і ми не можемо робити це наївно. Є декілька можливих способів оптимізувати такий розв'язок. Перший підхід – помітити, що апдейт в точці виглядає як додавання константи до певного суфікса часткових сум, і можна підтримувати всі зміни в дереві відрізків, щоб обчислювати хеш відрізка за $O(\log N)$ через два запити на часткову суму. Другий підхід – побудувати дерево відрізків, котре підтримуватиме у вершині хеш відповідного підрядка; в цьому випадку потрібно також навчитись комбінувати пару вершин, для чого

потрібно підтримувати інформацію про довжину рядка в префіксі, яка дозволить зсунути новий підрядок на правильну кількість позицій.

J. Шифр Бекона

Програмістові Васі не пощастило - замість відпустки його послали у відрядження, на наукову конференцію. Треба підвищувати рівень знань, сказав начальник, важлива конференція з криптографії, проводиться у Франції – а там шифрували ще за часів Рішельє і зламували чужі шифри ще за часів Вієта.

Вася швидко з'ясував, що всі луврські картини він уже десь бачив, вигляд Ейфелевої вежі приївся йому ще раніше, ніж мишка стерла його з килимка, а такі скляні піраміди у нас роблять над усякими кіосками та сумнівними забігайлівками. Одним словом, дивитися в Парижі виявилося просто ні на що, рибу половити ніде, тому Васі довелося відвідувати доповіді на конференції.

Один із доповідачів, у черговий раз намагаючись розгадати шифри Бекона, висунув гіпотезу, що ключ до таємниць Бекона можна підібрати, проаналізувавши всі можливі підрядки творів Бекона.

«Але їх же занадто багато!» - уголос здивувався Вася.

«Ні, не так уже й багато!» - закричав доповідач - «підрахуйте - і ви самі переконаєтеся!».

Того ж вечора Вася знайшов в інтернеті повне зібрання творів Бекона. Він написав програму, яка переробила тексти в однин довгий рядок, викинувши з текстів усі пробіли й розділові знаки. І ось тепер Вася вельми спантеличений - а як же підрахувати кількість різних підрядків цього рядка?

Формат вхідних даних

На вході даний непорожній рядок, отриманий Васею. Рядок складається лише з малих латинських символів. Його довжина не перевищує 5000 символів.

Формат вихідних даних

Виведіть кількість різних підрядків цього рядка.

Приклади

stdin	stdout
aaba	8

Розв'язання задачі «J. Шифр Бекона»

Джерело: NEERC 2007 Eastern Subregional; Timus 1590

Коротко про розв'язок: суфіксний масив + LCP

Розв'язок: оскільки дана задача була використана в тематичному контесті про хешування, напрошується ідея обчислити хеші всіх підрядків даного рядка та підрахувати кількість різних серед них. Обмеження було підібрано так, що цей розв'язок дуже складно реалізувати достатньо швидко для уникнення TL. У задачі є багато різноманітних розв'язків з різною асимптотикою – наприклад, лінійний розв'язок з використанням суфіксного автомата. Розглянемо розв'язок з використанням суфіксного масива. Ідея – для кожного суфікса ми можемо сказати, що він породжує кількість нових унікальних підрядків, котра дорівнює його довжині мінус довжина спільного префікса з попереднім елементом суфіксного масиву. Дійсно, всі префікси нашого суфікса, котрі не довші за спільний префікс, вже були пораховані раніше. Тут можна скористатись хешуванням - побудований на хешах суфіксний масив за $N \log^2 N$ та пошук спільного префікса за $\log N$ дадуть нам достатньо швидкий розв'язок. Насправді ж навіть наївний розв'язок є достатньо швидким; наївна побудова суфіксного масиву має асимптотику $N^2 \log N$, наївний пошук LCP загалом працюватиме N^2, що дає нам $N^2 \log N$. Асимптотика співпадає з асимптотикою розв'язку з хешуванням всіх підрядків, але прихована константа є набагато нижчою – компаратор в сортуванні порівнює рядки, навіть в наївній реалізації йдеться про швидкі операції, при цьому розв'язок можна прискорити, порівнюючи рядки одразу цілими блоками по декілька символів - навіть на 32-бітній архітектурі ми можемо помістити в одне машинне слово одразу 4 символи, а для 64-бітної системи це число виросте до 8.

K. Чим вищі гори

Зима в Єкатеринбурзі – найдовша пора року. І кожен коротає довгі зимові вечори по-своєму. Катя регулярно виїжджає зі своїми друзями на найближчий гірськолижний курорт – гору Шпільну – покататися на сноуборді. Щойно там відкрили нову трасу. Але коли Катя вперше проїхала по цій трасі, у неї було відчуття дежавю, ніби вона до цього вже каталася по такому ж схилу. Ну, або, можливо, по трохи більш крутому або пологому. Це настільки її схвилювало, що замість того, щоби продовжити кататися, вона вирішила перевірити, чи не було на якомусь із схилів, на яких вона каталася раніше, ділянки, яка була би схожа на схил, по якому вона щойно з'їхала. Повернувшись у свою машину і діставши ноутбук, Катя знайшла в інтернеті детальну інформацію про всі схили, по яких вона їздила, включно з новим схилом гори Шпільної, - карту висот із кроком в один метр. Катя вважає схожими дві ділянки різних схилів однакової довжини, якщо для висот першої ділянки x_0, x_1, ..., x_n, висот другої ділянки y_0, y_1, ..., y_n і деяких чисел a і b виконується рівняння $x_i - y_i = a \cdot i + b$.

Формат вхідних даних

У першому рядку дано ціле число n - кількість схилів, на яких Катя каталася раніше ($1 \le n \le 10^5$). У другому рядку через пробіл дано ціле число m і цілі числа x_0 ... x_m, де m – довжина схилу, по якому Катя проїхала щойно, а x_i - висота над рівнем моря точки, що знаходиться за і метрів від початку схилу ($1 \le m \le 10^5$; $-10^9 \le x_i \le 10^9$). У наступних n рядках в такому ж форматі описані всі схили, на яких Катя каталася раніше. Гарантується, що сума їх довжин не перевищує 10^5.

Формат вихідних даних

Якщо існують такі числа i та j, що схил, по якому щойно проїхала Катя, схожий на ділянку, що починається з j-го метра від початку i-го схилу з тих, на яких вона каталася раніше, то виведіть числа i та j через пробіл. Схили

нумеруються цілими числами від 1 до n у тому порядку, у якому вони перераховані у вхідних даних. Якщо є декілька відповідних пар i та j, виведіть ту з них, при якій значення модуля параметра a з критерію схожості мінімальне. Якщо таких все ще кілька, виведіть будь-яке з них. Якщо таких чисел не існує, виведіть -1.

Приклади

stdin	stdout
2 2 3 2 1 5 21 15 10 6 3 2	2 1

Розв'язання задачі «K. Чим вищі гори»

Джерело: Індивідуальна першість УрФУ 2014; Timus 2097

Коротко про розв'язок: перехід до часткових різниць + хеші, або ж подвійний перехід до часткових різниць + хеші чи точний пошук підрядка.

Розв'язок: описана в умові функція подібності виглядає "складно". Спробуємо її спростити. Розглянемо замість висот в сусідніх точках - прирости висот між ними. Переписавши формулу, отримаємо, що для приростів виконується $Y[i] = X[i] + a$, тобто ми вважаємо два масиви подібними, якщо один з них відрізняється від другого на певну додану константу. Якщо навчитись виконувати перевірку двох підмасивів на рівність за $O(1)$, то це вже дозволить нам розв'язувати задачу шляхом перебору всіх можливих позицій входження масиву-зразка. Можна повторити прийом з різницями ще раз, розглянувши різниці між різницями між сусідніми елементами; розписавши такий вираз, отримаємо, що ці масиви повинні точно співпадати - таку задачу можна розв'язати звичайним алгоритмом пошуку входжень рядка в текст через КМП або хешування. Хешуванням також можна розв'язати і проміжну версію, отриману розглядом різниць між сусідніми елементами. Як працювати з константою a? Ми можемо знайти її значення, розглянувши різницю між першими двома елементами, а після цього – обчислити хеш модифікованого масиву, додавши до нього хеш масиву з усіх одиниць потрібної нам довжини,

помножений на a. Таким чином ми навчились порівнювати два масиви на рівність з точністю до поелементного додавання константи за $O(1)$.

L. Абревіатури

При введені нових термінів, що складаються з декількох слів, буває доцільно використовувати абревіатури. **Абревіатура** — це слово, що складається з перших букв декількох послідовних слів.

Абревіатура називається однозначною, якщо наступні дві умови виконуються:

- вона відповідає рівно одній послідовності слів у даному тексті (при цьому дана послідовність може зустрічатись у тексті більше одного разу);
- вона не з'являється в тексті як окреме слово.

Для прикладу, у тексті “A recursive acronym KINA means "KINA is not abbreviation"”, рядки “ARA” і “K” є однозначними абревіатурами, рядки “A” і “KINA” є неоднозначними абревіатурами, а рядки “RAA” та “KNA” не є абревіатурами.

Для введення абревіатури в текст її записують у дужках одразу після послідовності слів, котрій вона відповідає. Наступні входження цієї послідовності в текст можуть бути замінені на абревіатуру. У прикладі вище, використання абревіатури “K” дасть такий текст: “A recursive acronym KINA (K) means "K is not abbreviation"”.

Якщо два входження послідовності слів перетинаються, тільки одне з них можна замінити абревіатурою. Слова у послідовності розділені одним або декількома символами, що не є буквами. Порівняння слів є чутливим до реєстру. Наприклад, “i18n” є входженням послідовності слів “I n”.

Ефективністю абревіатури називають те, на скільки зменшилась кількість букв у тексті після введення абревіатури. Зверніть увагу, що враховуються лише букви; пробіли, дужки та всі інші символи, котрі не є буквами, не враховуються.

Для заданого тексту знайдіть однозначну абревіатуру з максимальною ефективністю.

Формат вхідних даних

Вхідний файл містить текст довжиною не більше 4000 символів. Текст складається лише з символів з ASCII кодами від 32 (пробіл) до 126 (“~”), 13 (повернення каретки), та 10 (переведення рядка).

Формат вихідних даних

Якщо для заданого тексту відсутні однозначні абревіатури з додатною ефективністю, виведіть один рядок з числом 0.

У протилежному випадку перший рядок вихідних даних повинен містити ефективність оптимальної абревіатури. Другий рядок повинен містити цю абревіатуру. Якщо є декілька можливих абревіатур з однаковою ефективністю, виведіть будь-яку з них.

Приклади

stdin	stdout
This problem name is "KINA is not abbreviation". Once again: KINA is not abbreviation.	11 NA
To be or not to be: that is the question.	0
Here is the chorus of the song "Jingle Bells": Jingle bells, jingle bells, Jingle all the way; Oh what fun it is to ride In a one-horse open sleigh.	16 JB

Пояснення до прикладів

У першому прикладі оптимальними абревіатурами є “NA” та“INA”.

У третьому прикладі оптимальними абревіатурами є “JB” та “BJ”.

Розв’язання задачі «L. Абревіатури»

Джерело: NEERC 2008; Live Archive 4381

Коротко про розв’язок: динамічне програмування + бор + хешування

Розв'язок: побудувавши бор по всіх можливих абревіатурах із тексту, ми можемо для кожної з них визначити, чи вона є однозачною. Для цього при побудові абревіатури потрібно підтримувати хеш послідовності слів, котра її формує. Якщо у вершині з абревіатурою вже є інший хеш - це автоматично означає, що абревіатура неоднознача. Також для абревіатури можна підтримувати її довжину в символах та довжину в символах послідовності слів, котру ми описуємо цією абревіатурою. Знаючи, скільки разів ми замінимо абревіатуру в тексті, за цією інформацією ми можемо порахувати зекономлену кількість символів. Також можна підтримувати хеш абревіатури як рядка, або ж робити паралельний спуск по ще одному бору, побудованому за словами зі вхідних даних - щоб визначити, чи абревіатура не зустрічається в тексті як окреме слово. Залишилось розібратися з умовою про входження, котрі не перетинаються, - адже ми не можемо просто порахувати кількість входжень абревіатури в текст. Тут працює жадібність - зрозуміло, що вигідно кожного разу брати найлівіше входження, котре не перетинається з раніше вибраним. Тому можна додавати абревіатури в бор зліва направо і підтримувати у вершині також позицію останнього входження, котре ми взяли. Тепер, якщо нове входження не перекривається з ним, оновлюємо дані та робимо у вершині +1 до кількості взятих входжень.

M. Хак хеш-функції

Відповідно до документації стандарту бібліотеки Java, хеш від String обчислюється як

$$s[0] \cdot 31^{n-1} + s[1] \cdot 31^{n-2} + \cdots + s[n-1]$$

Тут $s[i]$ це i-ий символ рядка, n це довжина рядка, а ^ позначає піднесення до степеня. Для обчислень використовують знаковий 32-бітовий тип із доповнювальним кодом.

Хізер планує хакнути сервери Not Entirely Evil Recording Company (NEERC). Для здійснення атаки їй необхідні k різних рядків-запитів з однаковими значеннями хеш-функції. На жаль, сервери NEERC приймають

лише рядки-запити, котрі складаються з малих та великих літер англійського алфавіту.

Хізер найняла вас для написання програми, котра генеруватиме для неї такі рядки-запити.

Формат вхідних даних

В одному рядку вхідного файлу міститься ціле число k — кількість рядків-запитів, котрі необхідно згенерувати ($2 \leq k \leq 1000$).

Формат вихідних даних

Виведіть k рядків. Кожен рядок повинен містити один запит. Кожен рядок-запит повинен бути непустим та не довшим за 1000 символів. Рядки-запити повинні містити лише малі та великі літери англійського алфавіту. Усі рядки-запити повинні бути різними та мати рівні значення хеш-функції.

Приклади

stdin	stdout
4	edHs mENAGeS fEHs edIT

Розв'язання задачі «М. Хак хеш-функції»

Джерело: NEERC 2015 Northern Subregion

Коротко про розв'язок: побудова відповіді на основі прикладу з умови.

Розв'язок: у цій задачі використовується ідея про типову помилку в хеш-функції, описана під час лекції. Конструктивна ідея: розглянемо хеш-функцію як поліном від числа 31, для отримання пари рівних хешів в одному розряді одиницю віднімемо, а в іншому, на одиницю молодшому, – додамо 31 так, щоб при цьому не вийти за межі діапазону валідних значень. Ідея на основі семплу: виберемо пару рядків, котрі збігаються у двох символах із чотирьох; оскільки їх хеші рівні, то ми знаємо, що завжди можна заміняти одну пару з двох символів іншою парою (ці дві пари - символи, у котрих рядки відрізняються). Побудуємо

певний рядок, повторюючи оригінальну пару символів N разів поспіль; для отримання нових рядків будемо заміняти певні входження оригінальної пари на нову пару. Очевидно, є 2^N способів вибрати позиції, на яких ми робитимемо заміни, а для рядка довжиною 1000 маємо $N = 500$.

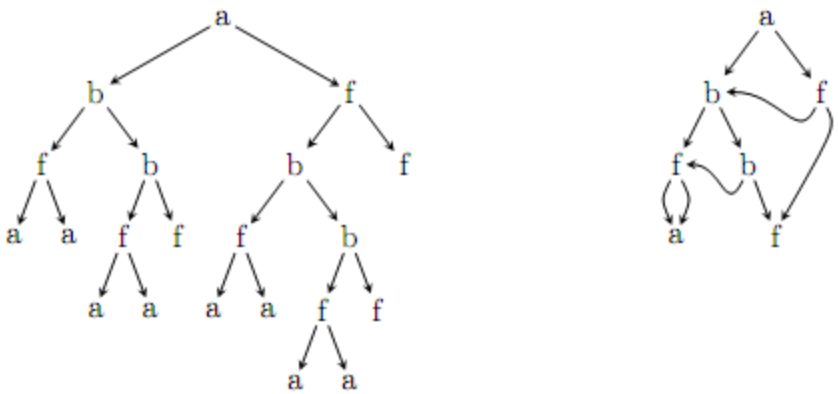
N. Видалення спільних підвиразів

Нехай множина Σ складається з усіх слів з 1–4 малих букв англійського алфавіту — наприклад, “a”, “b”, “f”, “aa”, “fun”, “kvqf”. Розглянемо вирази відповідно до граматики з двома правилами

$$E \rightarrow f$$

$$E \rightarrow f\ (E, E)$$

для кожного символу $f \in \Sigma$. Будь-який вираз можна подати у вигляді дерева відповідно до його синтаксису. Для прикладу, вираз “a(b(f(a,a),b(f(a,a),f)),f(b(f(a,a),b(f(a,a),f)),f))” записується деревом зліва на зображенні нижче:



Минулої ночі вам приснився неймовірний винахід, котрий значно зменшує розмір запису — використати довільний граф замість дерева, щоб не дублювати спільні підвирази. Наприклад, вираз вище може бути записаний як граф на зображенні справа. Оригінальне дерево містило 21 вершину, у новому графі для запису достатньо 7 вершин.

Оскільки дерево зліва також є графом, подання з використанням графів не обов'язково єдине. За заданим виразом, знайдіть граф для його позначення, кількість вершин у якому мінімальна.

Формат вхідних даних

Перший рядок вхідних даних містить число c $(1 \leq c \leq 200)$, котре позначає кількість виразів. Кожен із наступних c рядків містить вираз, записаний згідно з поданим синтаксисом, без пробілів. Дерево, котре описує цей вираз, має не більше 50000 вершин.

Формат вихідних даних

Для кожного виразу, виведіть єдиний рядок із поданням виразу графом із мінімальною кількістю вершин.

Подання у вигляді графа записується як рядок із заміною відповідних підвиразів на числа. Кожне число вказує на кореневу вершину підвиразу, котрий потрібно вставити на даній позиції. Вершини нумеруються послідовно, починаючи з 1. Ця нумерація враховує лише вершини графа (але не входження, котрі були замінені номерами). Числа повинні вказувати на вершини, записані раніше (а не на вершини, котрі ще не було додано). Для нашого прикладу, результатом буде “a(b(f(a,4),b(3,f)),f(2,6))”.

Приклади

stdin
3 this(is(a,tiny),tree) a(b(f(a,a),b(f(a,a),f)),f(b(f(a,a),b(f(a,a),f)),f)) z(zz(zzzz(zz,z),zzzz(zz,z)),zzzz(zz(zzzz(zz,z),zzzz(zz,z)),z))
stdout
this(is(a,tiny),tree) a(b(f(a,4),b(3,f)),f(2,6)) z(zz(zzzz(zz,z),3),zzzz(2,5))

Розв’язання задачі «N. Видалення спільних підвиразів»

Джерело: NWERC 2009; Live Archive 4610

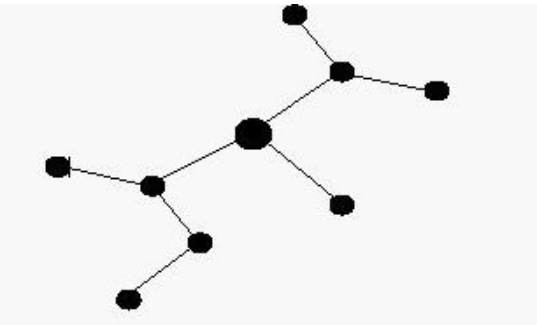
Коротко про розв’язок: хешування піддерев.

Розв’язок: перш за все потрібно реалізувати парсер, котрий отримає структуру дерева за заданим рядком. Після цього перейдемо до алгоритмічної частини розв’язку, зрозумівши, що вигідно завжди діяти жадібно – будемо обходити дерево і кожного разу зустрічаючи піддерево, яке вже було раніше -

використовувати ребро в перше входження цього піддерева, уникаючи дублювання вершин. Залишилося навчитися швидко перевіряти, чи зустрічалося таке піддерево раніше. Для цього скористаємося хешуванням; заведемо map, котрий для всіх наявних піддерев зберігатиме номер вершини-кореня, і будемо кожне нове піддерево додавати в цей map, а для кожного повторного входження - заміняти його вершиною першого входження. Такий розв'язок має асимптотику $N \log N$ через використання асоціативної структури, і формально може бути покращений до лінійного (з великою константою) шляхом переходу на *hashmap*.

O. Дерево метрополітену

У деяких великих містах система метрополітену має структуру дерева, тобто між будь-якою парю міст є рівно один шлях. Більше того, більшість таких міст мають єдину центральну станцію. Уявіть, що ви турист в одному з таких міст, і ви хочете дослідити всю його систему метрополітену. Ви розпочинаєте з центральної станції, вибираєте довільну лінію метро і вирушаєте цією лінією. Кожного разу після прибуття на станцію ви вибираєте одну з ліній, якими ви ще не подорожували. Якщо таких ліній для цієї станції не залишилось, ви обираєте лінію, котрою ви вперше добралися на станцію, і так доки ви не скористаєтесь



кожною лінією двічі (по разу в кожному з напрямів). У цей момент ваш маршрут завершиться — на центральній станції, де він і розпочинався. Усе, що ви запам'ятали про свою подорож, — в якому напрямі ви рухались у той чи інший момент: до центральної станції чи від неї. Формально ви закодували подорож як двійковий рядок, у котрому 0 означає рух по лінії метрополітену від центральної станції, а 1 означає рух у напрямі до центральної станції.

```
0010011101001011
0100011011001011
0100101100100111
...
```

Зліва: дерево метрополітену. Круг більшого розміру позначає центральну станцію.

Справа: деякі з можливих кодувань для цього дерева (залежно від обраного порядку подорожі).

Формат вхідних даних

У першому рядку вхідних даних — одне додатне число n, котре вказує на кількість тестів у файлі. Кожен тест складається з двох рядків, що складаються з символів '0' і '1' та мають довжину не більше 3000. Обидва рядки описують валідну подорож певною системою метрополітену.

Формат вихідних даних

Для кожного тесту виведіть рядок зі словом “`same`”, якщо два рядки з вхідних даних можуть описувати подорожі однією системою метрополітену, або ж “`different`”, якщо два рядки не можуть описувати одну й ту ж систему метрополітену.

Приклади

stdin	stdout
2 0010011101001011 0100011011001011 0100101100100111 0011000111010101	same different

Розв’язання задачі «O. Дерево метрополітену»

Джерело: NWERC 2003; Live Archive 2935

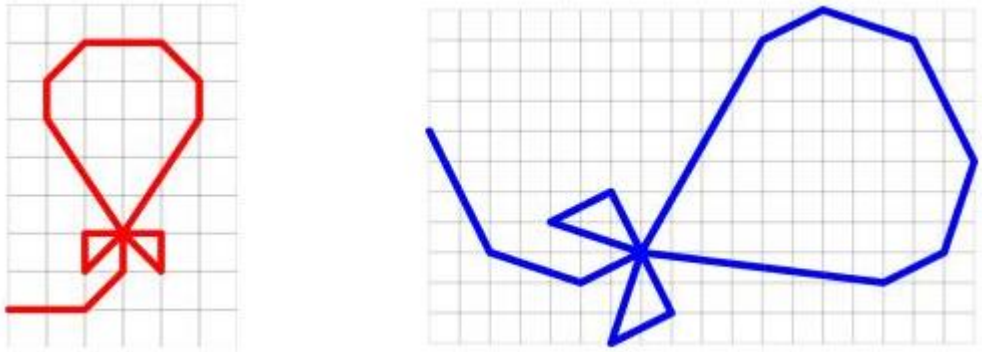
Коротко про розв’язок: приведення запису дерева в нормований формат.

Розв'язок: розв'язок цієї задачі є аналогічним до розв'язку задачі N, але при цьому низькі обмеження дозволяють уникнути хешування або індексного кодування піддерев, і замість цього, наприклад, для кожної вершини сортувати її список синів у лексикографічному порядку – оскільки кожен 0/1 символ задає зміну глибини, то неоднозначностей при цьому не виникне, бо при розгляді певного дерева ми можемо однозначно сказати, де завершується опис одного піддерева і починається опис наступного. Такий розв'язок можна легко оцінити зверху як $N^2 \log N$, оскільки ми N разів сортуємо масив розміру N.

P. Схожі зображення

Стеганографія — це особливий вид захисту повідомлень; замість того, щоб кодувати зміст повідомлення, саме повідомлення якось приховують. Ця техніка була популярною в минулому, але в наш час більш популярним є шифрування, особливо ті його види, котрі базуються на використанні ключів. Тим не менш, інколи техніка стеганографії використовується і в наш час. Один зі способів, що використовує цю техніку для цифрових даних — ховати інформацію у зображенні. Можна вносити невеликі зміни в зображення, так що вони будуть непомітні людському оку, але інформація в цих змінах буде легко читатись комп'ютерами. Ваше завдання — порівняти два зображення і знайти ці відмінності.

У цій задачі, ми будемо працювати з векторними зображеннями. Ваша програма повинна визначити, чи задані дві картинки містять одне й те ж зображення. Говорячи в термінах геометрії, визначте чи задана пара зображень є подібними, тобто чи можна перетворити одне з них в інше, використовуючи для цього тільки паралельне перенесення, поворот та масштабування (і не використовуючи віддзеркалення).



Приклад пари подібних зображень

Формат вхідних даних

Вхідний файл складається з одного чи декількох тестів. Кожен тест складається з опису двох зображень. Кожен зображення описується послідовністю інструкцій для його одержання, по одній інструкції в рядку. Кожна інструкція розпочинається з великої літери, після якої знаходиться пробіл, далі два цілих числа, також розділених пробілом. Велика літера - “`L`” (провести лінію) або “`M`” (переміститись без додавання лінії). Два цілих числа задають координати місця, до якого потрібно провести лінію (або переміститись). Усі координати задаються відносно кінцевої позиції попередньої інструкції; координати для першої інструкції задаються відносно довільної початкової точки.

Після останньої інструкції опису зображення міститься рядок із літерою “`E`” та ще один порожній рядок. Після останнього тесту розміщений рядок із літерою “`Q`”.

Кількість інструкцій для кожного зображення становить від 0 до 1000. Для жодної інструкції обидві координати не є рівними 0 одночасно. Усі (відносні) координати не перевищують 1000 за абсолютним значенням.

Формат вихідних даних

Для кожного тесту виведіть рядок зі словом “`YES`” (якщо два зображення подібні) або “`NO`” (якщо два зображення не є подібними).

Приклади

stdin	stdout
L 31 L 31 M 30 M 01 L -3 -1 E L10 L -1 0 E L10 L01 E L10 L -1 -1 E L 20 L 11 L 01 L 23 L 01 L -1 1 L -2 0 L -1 -1 L 0 -1 L 2 -3 M 10 L -2 0 M 20 L 0 -1 L -1 1 L -1 -1 L 01 E L 2 -4 L 3 -1 L 21 L 1 -2	YES NO YES

L -2 -1 L 13 L -3 1 L 21 L 1 -2 L 8 -1 L 21 L 13 L -2 4 L -3 1 L -2 -1 L -4 -7 E Q	

Розв'язання задачі «P. Схожі зображення»

Джерело: CERC 2011; Live Archive 5887

Коротко про розв'язок: переберемо всі валідні перетворення.

Розв'язок: перш за все необхідно нормалізувати вхідні дані – перейдемо до довільного стандартизованого мінімального задання зображення відрізками; для цього викинемо повтори відрізків, з'єднаємо відрізки, котрі перекриваються або такі, серед яких один є продовженням іншого. Тепер можна спробувати певні перетворення і визначити, чи якесь із них підходить. При заданих перетвореннях відрізок переходить у відрізок, тому можна перебрати, у який із відрізків другого зображення переходить один із відрізків першого зображення. Кожен такий варіант дозволяє однозначно відтворити перетворення – встановити кут повороту та вектор зсуву. Після цього ми можемо перевірити перетворення – для кожного відрізка першої фігури перевіримо, чи він переходить у якийсь із відрізків другої. Такий розв'язок у наївній реалізації має асимптотику $O(N^3)$, що може отримати AC після низки оптимізацій і відсікань, але за замовчуванням є надто повільним. Для того, аби його прискорити, скористаємося певною асоціативною структурою даних для збереження всіх відрізків другого зображення, або ж просто захешуємо їх, щоб перевіряти існування потрібного нам відрізка за логарифмічний час замість лінійного.

Варто звернути увагу, що ми маємо справу з операціями з дійсними числами, а прості реалізації хешування є абсолютно чутливими до похибок. Цього можна уникнути, перейшовши неявним чином до цілих чисел – округлимо всі координати до певного знака після коми, або ж скористаємось тим фактом, що вхідні дані є цілими числами (і будемо відкидати відрізки, котрі переходять у щось сильно віддалене від цілих чисел, а всі інші округлювати одразу до цілих), а після такої дискретизації всі порівняння будуть точними. Описаний розв'язок повинен вкладатися в обмеження без будь-яких проблем. Теоретично його можна прискорювати цілою низкою оптимізацій; наприклад, звернути увагу на те, що найдовший відрізок переходить у найдовший відрізок, а найбільш віддалена від центру мас точка - у найбільш віддалену від центру мас точку. Це не дає нам однозначного вибору варіанту переходу, бо ми можемо мати декілька рівновіддалених точок чи декілька відрізків однакової довжини, але зменшує кількість варіантів перебору до $O(1)$ для випадкових даних та більшості структурованих тестів (що дуже вигідно у випадку роботи зі змішаним мультитестом).

Q. Пошук файлів

Операційна система вашого комп'ютера індексує файли на вашому жорсткому диску відповідно до їхнього вмісту, і надає можливість текстового пошуку в них. Вміст кожного файлу — непорожній рядок, що складається з малих літер англійського алфавіту. Для здійснення пошуку ви задаєте ключ, котрий також є непорожнім рядком з малих літер англійського алфавіту. Результатом пошуку є список усіх файлів, котрі містять заданий ключ як підрядок.

Вам відомий вміст кожного файлу на вашому жорсткому диску; ви зацікавились, які з підмножин файлів можуть бути отримані в результаті пошуку. Підмножина файлів може бути отримана, якщо існує хоч один можливий ключ пошуку, результатом для якого є ця підмножина.

Знаючи вміст кожного файлу, підрахуйте кількість таких непорожніх підмножин.

Формат вхідних даних

Вхідний файл містить один або декілька тестів. Кожен тест описується декількома рядками. Перший рядок тесту містить ціле число F — кількість файлів на вашому жорсткому диску. Кожен із наступних F рядків описує вміст одного з файлів. Вміст файлу є непорожнім рядком, що складається з не більш як 10^4 малих літер англійського алфавіту.

Після останнього тесту йде рядок із числом 0, що є індикатором завершення вхідного файлу.

Формат вихідних даних

Для кожного тесту виведіть в окремому рядку кількість непорожніх підмножин, які можуть бути результатом пошуку.

Приклади

stdin	stdout
6 form formal malformed for man remake 3 cool cool old 0	11 3

Розв'язання задачі «Q. Пошук файлів»

Джерело: Latin American Regional Contests 2011; Live Archive 5794

Коротко про розв'язок: суфіксний масив + вказівники

Розв'язок: побудуємо суфіксний масив для вхідних даних. Як виглядає набір файлів, котрий ми отримуємо в результаті певного запиту? Знайдемо всі

суфікси, для котрих цей запит є префіксом – вони формують певний послідовний блок у суфіксному масиві; відповіддю на запит є об'єднання всіх індексів рядків, котрим належать суфікси з цього блоку. Для зручності можна описувати набір одним 8-байтовим числом - будемо задавати сет як бітову маску, і зберігатимемо цю маску прямо в числі 8-байтового типу. Розв'язок, котрий перебирає всі префікси всіх суфіксів, очевидно, є надто повільним. Як його прискорити? Нехай у нас зафіксований певний суфікс. Якщо ми візьмемо лише один символ з нього, то під такий запит пошуку будуть підходити всі рядки, котрі містять цей символ; якщо ми додамо другий символ - можливо, деякі рядки відпадуть; а в термінах суфіксного масиву – блок суфмасиву, котрий нас цікавить, буде звужуватися. Коли цей блок звужується після додавання певного символу, – можливо, з нього випадають усі входження суфіксів певного рядка, і цей рядок перестає підходити під наш пошуковий запит. Знайдемо для нашої позиції в суфмасиві перше входження суфіксів кожного з рядків зверху і знизу від нашої позиції. Очевидно, що ці крайні входження випадуть із блоку останніми, і рядок перестає нам підходити тоді, коли обидва входження випадають (або коли одне входження випадає, а з іншого боку входжень не було – такий розбір випадків можна обійти, ввівши фіктивні суфікси на позиціях $\pm INF$). Для кожного такого крайнього входження можна знайти LCP цього входження і нашого фіксованого суфікса, і отримати події вигляду "після додавання *i*-го символу ми втрачаємо останнє входження *j*-го рядка". Тепер залишилося відсортувати ці події і просимулювати їх, додаючи кожного разу маску в набір можливих результатів пошуку. Варто не забути про те, що маску можна додавати тільки після розгляду всіх подій одного блоку, і про те, що одна маска може бути знайдена декілька разів, тому треба використати set або ж зберегти всі результати у векторі і відсортувати його.

День 6. Контест Євгена Задорожнього

Теоретичний матеріал. Дерево відрізків

1. Код простого дерева відрізків на суму. Функції побудови, суми на відрізку, апдейту одиничного елемента.

```
const int maxN = 100000;
int tr[maxN * 4 + 1];
int a[maxN];

// cur - номер вершини,
// [l, r] - інтервал, за який відповідає вершина.
void build(int cur, int l, int r) {
    if (l == r) {
        tr[cur] = a[l];
    } else {
        int m = (l + r) / 2;
        int dcur = cur + cur;
        build(dcur, l , m);
        duild(dcur + 1, m + 1, r);
        tr[cur] = tr[dcur] + tr[dcur + 1];
    }
}

// cur - номер вершини,
// [l, r] - інтервал, за який відповідає вершина,
// [x, y] - інтервал запиту
int getSum(int cur, int l, int r, int x, int y) {
    if (l == x && r == y) {
        return tr[cur];
    } else {
        inf m = (l + r) / 2;
        int dcur = cur + cur;
        if (y <= m) {
            return getSum(dcur, l, m, x, y);
        } else if (x > m) {
            return getSum(dcur + 1, m + 1, r, x, y);
        } else {
            return  getSum(dcur,  l,  m,  x,  m)  +
getSum(dcur + 1, m + 1, r, m + 1, y);
        }
```

```
36      }
37  }
38
39  void update(int cur, int l, int r, int pos, int val) {
40      if (l == r) {
41          tr[cut] = val;
42      } else {
43          int m = (l + r) / 2;
44          int dcur = cur + cur;
45          if (pos <= m) {
46              update(dcur, m, pos, val);
47          } else {
48              update(dcur + 1, m + 1, r, pos, val);
49          }
50      }
51  }
52
53  int main() {
54      int n;
55      // ...
56      build(1, 1, n);
57      int x, y;
58      // ...
59      cout << getSum(1, 1, n, x, y) << endl;
60      return 0;
61  }
```

2. Код неявного (або розрідженого) дерева відрізків. Головна ідея - вершини створюються лише тоді, коли в них уперше записується якесь значення. Ця реалізація написана на індексах, що зберігаються в різних масивах. У деяких випадках це може бути повільніше, ніж робота зі структурою та вказівниками.

```
1   const int maxN = 100000;
2   int tr[maxN * 30];
3   int a[maxN];
4   int lp[maxN * 30];
5   int rp[maxN * 30];
6   int lastNode = 1;
7
8   // cur -номер вершини,
9   // [l, r] - інтервал, за який відповідає вершина,
10  // [x, y] - інтервал запиту
```

```
11 int getSum(int cur, int l, int r, int x, int y) {
12     // Якщо прийшли в неіснуючу вершину,
13     // повертаємо нульове значення
14     if (cur == 0) return 0;
15
16     if (l == x && r == y) {
17         return tr[cur];
18     } else {
19         int m = (l + r) / 2;
20         if (y <= m) {
21             return getSm(lp[cur], l, m, xm y);
22         } else if (x > m) {
23             return getsum(rp[cur], m + 1, r, x, y);
24         } else {
25             return getSum(lp[cur], l, m, x, m) +
26                    getSum(rp[cur], m + 1, r, m + 1,
27 y);
28         }
29     }
30 }
31
32 void update(int& cur, int l, int r, int pos, int val)
33 {
34     // Якщо вершини не існує, ми її створюємо
35     if (cur == 0) {
36         cur = lastNode++;
37     }
38
39     if (l == r) {
40         tr[cur] = val;
41     } else {
42         int m = (l + r) / 2;
43         if (pos <= m) {
44             update(lp[cur], l, m, pos, val);
45         } else {
46             update(rp[cur], m + 1, r, pos, val);
47         }
48     }
49 }
50
51 int root = 0;
52 int main() {
53     const int N = 1000000000;
54     update(root, 1, N, 100, 234);
55     update(root, 1, n, 456, 999);
```

```
56      cout << getSum(root, 1, N, 50, 500) << endl;
57      return 0;
58  }
```

3. Код персистентного дерева відрізків на структурі та вказівниках.

```
1   const int maxN = 100000;
2   struct Node {
3       Node *lp, *rp;
4       int val;
5   };
6   Node nodes[maxN * 30];
7   Node* lastNode = nodes + 1;
8
9   int getSum(Node* cur, int l, int r, int x, int y) {
10      // Якщо прийшли в неіснуючу вершину,
11      // повертаємо нульове значення
12      if (cur == nullptr) return 0;
13
14      if (l == x && r == y) {
15          return cur->val;
16      } else {
17          int m = (l = r) / 2;
18          if (y <= m) {
19              return getSum(cur->lp, l, m, x, y);
20          } else if (x > m) {
21              return getSum(cur->rp, m + 1, r, x, y);
22          } else {
23              return getSum(cur->lp, l, m, x, m) +
24                     getSum(cur->rp, m + 1, r, m + 1,
25  y);
26          }
27      }
28  }
29
30  void update(Node*& cur, int l, int r, int pos, int
31  val) {
32      Node* old = cur;
33      cur = lastNode++; // вершина копіюється в будь-
34  якому випадку
35      if (old != nullptr) {
36          *cur = *old;
37      }
38      if (l == r) {
39          cur->val = val;
```

```
40      } else {
41          int m = (l + r) / 2;
42          if (pos <= m) {
43              update(cur->lp, l, m, pos, val);
44          } else {
45              update(cur->rp, m + 1, r, pos, val);
46          }
47      }
48  }
49
50  const int N = 1000000000;
51  Node* roots[maxN + 1];
52  int a[maxN];
53  int main() {
54      int n;
55      // ...
56      for (int i = 1; i <= n; ++i) {
57          roots[i] = roots[i - 1];
58          update(roots[i], 1, N, i, a[i]);
59      }
60      return 0;
61  }
```

4. Ідеї та задачі.

4.1. Ідея: запити (L, R) в задачі можна сортувати за правою границею. Дерево відрізків або іншу структуру будуємо по одному елементу, коли додали елемент у позиції R, то треба відповісти на всі запити, що мають праву границю в позиції R. Завдяки цьому маємо ситуацію, коли всі запити є суфіксами поточного масиву. Ряд задач спрощується завдяки цьому. Можна досягти такого ж ефекту в онлайні завдяки персистентному ДВ.

4.2. Ідея: будувати дерево відрізків не на вихідному масиві, а на ряді натуральних чисел. Наприклад, для числа X у листі X дерева відрізків зберігати позицію останнього входження числа X в масив.

4.3. Ідея: стиснення координат. Беремо всі можливі вхідні дані, сортуємо, замінюємо елементи на індекси входження елементу у відсортованому списку.

4.4. Ідея: замість стиснення координат, будуємо неявне дерево відрізків. Недоліки – повільніше працює, більше пам'яті використовує. Переваги – легше у використанні.

4.5. Використання дерева відрізків як множини (set). Операція додавання елемента X => операція $+1$ у позиції X. Це дає можливість розв'язувати такі задачі, як пошук K-го елемента у множині, визначення позиції елемента у множині.

4.5.1. Визначення позиції елемента у ДВ. Позиція (в 0-індексації) = кількості елементів ліворуч того, що визначаємо. Шукаємо суму на префіксі, отримуємо позицію елемента.

4.5.2. Пошук K-го елемента у множині. Рекурсивний спуск по дереву. Стоїмо у розгалуженні, вирішуємо, куди йти. Якщо в лівому сині достатня кількість елементів (сума $\geq K$), то йдемо у лівого сина, не змінюючи K. Інакше рухаємося у правого сина та від K віднімаємо суму у лівому сині. Коли дійшли до листа, – отримали відповідь.

4.6. Пошук K-го елемента на підвідрізку масива. Необхідно звести до пункту 4.5.2. побудуємо персистентне ДВ => зможемо звернутися до його версії на будь-якому префіксі. ДВ будуватимемо як множину – додаємо елмент – робимо в його позиції $+1$. Нехай $[L, R]$ - запит, що нас цікавить. Візьмемо версії R та $L - 1$. Уявімо «різницю» цих двох дерев. Під різницею розуміємо віднімання значень у однакових вершинах. Результат такої операції відповідатиме дереву, що зберігає всі числа з інтервалу $[L, R]$. Дерево різниці будувати не треба, просто рухаємося одночасно по обох версіях та дивимося на різницю значень у відповідних вершинах (замість одного значення, як у пункті 4.5.2).

4.7. Пошук кількості різних елементів на підвідрізку масива.

4.7.1. Ідея розв'язання перша. Запишемо в кожну позицію ДВ найбільший індекс входження цього ж елемента ліворуч. Тоді відповідь на задачу – кількість числе $< L$ на інтервалі $[L, R]$. Це $\log^2$, повільно.

4.7.2. Ідея друга. Використаємо ідею з 4.1. Звели до задачі на суфіксі. Давайте в останнє входження кожного елемента запишемо 1, в усі інші позиції 0. Тоді відповідь на задачу – сума на суфіксі.

4.8. Задача. Маємо відрізки (X, Y). Запит – $L\ R$, скільки відрізків ми покриваємо повністю, тобто скільки таких, до $L \leq X$, $Y \leq R$. Ідея розв'язання – посортуємо відрізки і запити за правою координатою. Тоді при відповіді на запит ми вже матимемо лише ті відрізки, у яких $Y \leq R$. Тому залишилося пошукати кількість $L \leq X$. Рішення – коли додаємо відрізки в позиції R, необхідно у відповідних позиціях X поставити $+1$. Відповідь – сума на інтервалі.

5. Задачі на графах.

5.1. Ідея – будуємо Ейлерів обхід. У масив додаємо значення при вході у вершину, або при вході та при виході. Приклад задач: сума на піддереві, довжина шляху з модифікаціями, K-те значення у піддереві тощо.

5.2. У ДВ можна підтримувати відстань до всіх вершин при ДФСі. Перехід по ребру – множинна модифікація у ДВ.

6. Корисні матеріали:

- http://e-maxx.ru/algo/segment_tree - код та теорія.
- http://codeforces.com/blog/entry/18051 - нерекурсивна реалізація ДВ. Дуже швидка.
- http://www.e-olymp.com/ru/contests/5649 - контест із простих задач на стандартні модифікації ДВ.
- http://codeforces.com/gym/100093, http://codeforces.com/gym/100094 - ще два контести на ДВ.

Задачі та ідеї розв'язання

A. Підсилювачі

Вчений Андрій потребує напруження в N разів більше, ніж стандартна напруга в електричній розетці, для живлення своєї машини часу. Стандартна напруга дорівнює одному Бервольту. Андрійко вирішив використовувати підсилювачі напруги. У сусідньому магазині він знайшов підсилювачі двох типів, перший тип створює напругу $2X - 1$ Бервольт із X Бервольт, другий створює напругу $2X + 1$ Бервольт від X Бервольт. Кількість підсилювачів у магазині не обмежена. Андрійко хоче побудувати послідовність підсилювачів для живлення машини часу. Звичайно, він хоче звести до мінімуму кількість підсилювачів. Допоможіть йому в цьому.

Формат вхідних даних

Єдине ціле число $1 \leq N \leq 10^{10}$ – необхідна напруга.

Формат вихідних даних

Якщо можна зробити таку схему, виведіть у першому рядку мінімально можливу кількість підсилювачів. Другий рядок у цьому випадку повинен містити послідовність підсилювачів від розетки до машини часу. Використовуйте число 1 для підсилювачів першого типу і число 2 для підсилювачів другого типу. Якщо немає жодного рішення, виведіть -1.

Приклади

stdin	stdout
5	2 2 1
2	-1

Розв'язання задачі «A. Підсилювачі»

Перше спостереження – після застосування підсилювача ми можемо отримати лише непарну напругу. А тому, якщо на вхід подається парне число, одразу виводимо -1.

Далі найпростішим варіантом є розв'язок із кінця. Тобто давайте розглянемо, із яких чисел ми могли отримати число N. Це могло бути або число $(N+1)/2$, або число $(N-1)/2$. Оскільки це два сусідні натуральні числа, то одне з них є парним, і його ми не могли отримати. Таким чином, попереднє число для кожного числа N і відповідно вся послідовність дій встановлюються однозначно. Асимптотика розв'язку $O(\log N)$.

B. Морозиво

У цій задачі вам доведеться спробувати себе у ролі продавця морозива. Упродовж дня до вас надходить новий товар та приходять покупці по морозиво. Оскільки ви – програмісти, люди розумні та ділові, то ви вирішили розширити бізнес на оптову торгівлю та оптимізувати весь цей процес. Сьогодні вашезавдання – розробити автоматичну систему торгівлі. Ця система буде отримувати запити двох типів. Запити першого типу задаються у вигляді $A\ x\ y$ і означають, що на ваш склад поступило x упаковок морозива вартістю y гр. од. за кожну з них. Запити другого типу задаються у вигляді $B\ x\ y$, і означають, що до вас прийшов покупець, що бажає придбати x упаковок морозива, заплативши в цілому не більше, ніж y гр. од. Покупці у вас поки що жадібні, тому купляти хочуть завжди найдешевше морозиво із наявного. Для кожного такого запиту вам необхідно визначити, чи зможете ви повністю задовольнити замовлення цього покупця. Якщо так – покупець забирає товар, інакше – йде ні з чим. Усі запити варто оброблювати у прямому хронологічному порядку. Вважайте, що на початку роботи програми ви маєте порожній склад.

Формат вхідних даних

У першому рядку задано ціле число $1 \leq Q \leq 2 \cdot 10^5$ – кількість запитів. У кожному наступному з Q рядків задано по одному запиту. Запити першого типу задаються у вигляді $A\ x\ y$, де $1 \leq x \leq 10^4$, $1 \leq y \leq 10^9$, x – кількість упаковок морозива, що надійшли на склад, y – вартість кожної. Запити другого типу задаються у вигляді $B\ x\ y$, де $1 \leq x \leq 10^9$, $1 \leq y \leq 10^{18}$, x – кількість упаковок морозива, що хоче придбати покупець, y – максимальна кількість грошових одиниць, що даний покупець може заплатити.

Формат вихідних даних

Для кожного запиту другого типу в окремому рядку виведіть `YES`, якщо ви можете задовільнити замовлення, і `NO` - в іншому випадку.

Приклади

stdin	stdout
5 A 1 1 A 10 100 B 5 450 B 5 450 B 5 500	YES NO YES

Розв’язання задачі «B. Морозиво»

Розв’язок цієї задачі полягає у послідовному виконанні всіх запитів. Якщо робити це наївно, а саме – зберігати пари (вартість, кількість) у відсортовій множині (std::multiset у C++) та на кожен запит другого типу ітеративно набирати найдешевше морозиво до заданої кількості, то це буде працювати у гіршому випадку за квадратичний час.

Щоб пришвидшити цей розв’язок, необхідно використати структуру даних - дерево відрізків. Давайте по вартості, як по індексу, зберігати кількість упаковок такої вартості, а в кожній вершині дерева відрізків тримати сумарну кількість упаковок на відрізку та їх сумарну вартість. Тоді для кожного запиту на покупку заданої кількості упаковок необхідно знайти префікс із сумарною

кількістю, рівною кількістю із запиту, і знайти сумарну вартість на цьому префіксі. Такий запит виконується як класична операція “Пошук K-ого елемента у дереві”.

Якщо сумарна вартість задавільняє умову із запиту, необхідно видалити елементи з того префіксу. Це можна зробити за допомогою операції “Пофарбування на відрізку” або звичайним пошуком у глибину на цьому дереві, ігноруючи при цьому пусті вершини. Це буде працювати сумарно за той самий час, адже кожен елемент із дерева відрізків буде видалено не більше одного разу.

Враховуючи те, що значення вартості морозива можуть бути досить великими, необхідно або використати техніку “стиснення координат”, або будувати неявне (розріджене) дерево відрізків.

Асимптотика розв’язку $O(Q \log Q)$ або $O(Q \log MaxCost)$.

C. Скопіюйте масив

У вас є масив. Спочатку він зовсім малесенький – всього з одного числа. Та з кожним запитом він ставатиме все більшим і більшим. Чи впораєтеся ви з цим масивом і запитами на ньому?

Формальніше, вам треба обробити такі запити:

COPY - скопіювати масив та дописати його копію вкінець.

CHANGE $pos\ val$ - змінити значення у позиції pos масиву на val.

SUM $left\ right$ - вивести суму чисел цього масиву на підвідрізку від $left$ до $right$ за модулем $10^9 + 7$.

Формат вхідних даних

У першому рядку задано єдине число $1 \le A_1 \le 10^9$ – перший і єдиний елемент масиву. У другому рядку задано число $1 \le Q \le 2 \cdot 10^5$ – кількість запитів. У кожному з наступних рядків задано по одному запиту у таких форматах:

COPY – запит першого типу,

CHANGE $pos\ val$ – запит другого типу, $1 \le pos \le 10^{18}$, $1 \le val \le 10^9$,

SUM $left\ right$ – запит третього типу, $1 \le left \le right \le 10^{18}$.

Гарантується, що числа pos, $left$ та $right$ із другого і третього запитів не перевищують поточний розмір масиву.

Формат вихідних даних

Для кожного запиту третього типу виведіть в окремому рядку єдине число $\left(\sum_{i=left}^{right} A_i\right) mod\ (10^9 + 7)$.

Приклади

stdin	stdout
1 7 COPY SUM 1 2 CHANGE 2 4 COPY SUM 2 3 CHANGE 3 10 SUM 1 4	2 5 19

Розв'язання задачі «C. Скопіюйте масив»

Для розв'язку цієї задачі необхідно використати персистенте дерево відрізків на суму. З цією структурою даних операція копіювання масиву робиться так: створимо новий корінь для дерева та обидва його вказівники кинемо у старий корінь. Операції зміни елемента та суми на підвідрізку виконуються як у звичайному персистентному дереву відрізку. Завдяки персистентності операції зміни в одній половині ніяк не впливатимуть на іншу, і дерево буде завжди описувати валідний масив.

Асимптотика розв'язку $O(Q \log MaxIdx)$.

D. Кілоґґер

Кейлогер або Кілоґґер - програма, що відловлює та зберігає натиснення на клавіші клавіатури комп'ютера. Вам дано результат роботи цієї програми під

час введення користувачем цієї програми паролю. Цей лог являє собою єдиний рядок із маленьких латинських символів, а також символів L та R, котрі позначають зсув каретки вводу вліво та вправо відповідно. Ваша задача за заданим логом відновити пароль, що був уведений користувачем. Відомо, що цей пароль складається лише з маленьких латинських символів. Врахуйте, що відповідні зсуви каретки, коли вона вже знаходиться на початку або вкінці рядка, ігноруються.

Формат вхідних даних

Єдиний рядок S - лог із вводу пароля. $1 \leq |S| \leq 10^6$.

Формат вихідних даних

У єдиному рядку виведіть розшифрований пароль.

Приклади

stdin	stdout
gorLLLuzhRRRRRRRRod	uzhgorod

Розв'язання задачі «D. Кілоґґер»

Найпростіший розв'язок цієї задачі - використати двозв'язний список. Тоді зсуви каретки - це просто перехід до сусіднього елемента, а операція вставки букви виконується як звичайна вставка елемента у список.

Також цю задачу можна розв'язати за допомогою двох стеків - в одному зберігати символи, що знаходяться ліворуч від каретки, в іншому – ті, що праворуч.

E. Ка Те та Ре Бро

Друзі Ка Те та Ре Бро мають дерево. Та не звичайне дерево, а зв'язний ациклічний зважений неорієнтований граф.

Вони люблять грати в гру із цим деревом. Спочатку Ка Те називає пару номерів вершин x та y, потім Ре Бро називає натуральне число k. Після цього Ка Те має назвати k-те за величиною ребро на шляху між вершинами x та y. Та

їхнє дерево стало вже дуже великим, Ка Те довго вираховує шукане ребро, і через це Ре Бро починає сумувати. Ваше завдання – допомогти друзям у цій грі, щоб ніхто не сумував.

Формат вхідних даних

У першому рядку задано натуральне число $2 \leq N \leq 2 \cdot 10^5$ – кількість вершин у графі. У наступних $N-1$ рядках задано опис графа. i-ий рядок містить два цілих числа p та q - номер батьківської вершини для вершини i та значення на ребрі, що їх з'єднує. $1 \leq p \leq i, 1 \leq q \leq 10^9$. Гарантується, що на всіх ребрах записані різні числа.

У наступному рядку задано число Q – кількість запитів від друзів Ка Те та Ре Бро.

Кожен із наступних Q рядків містить по три цілих числа x y k - опис запитів. $1 \leq x, y \leq N$, $1 \leq k \leq 10^9$.

Формат вихідних даних

Виведіть Q цілих чисел по одному в рядку - віповідь на запити. Для кожного запиту ви маєте вивести значення, що записане на k-тому за величиною ребрі на шляху від вершини x до вершини y у заданому дереві. Якщо ж на шляху між цими вершинами менше ніж k ребер, виведіть -1.

Приклади

stdin	stdout
6 1 1 2 3 1 4 4 2 4 5 5 3 5 2 3 5 3 5 6 2 1 4 7 2 6 1	2 3 5 -1 1

Розв'язання задачі «E. Ка Те та Ре Бро»

Розв'язок цієї задачі дуже схожий на розв'язок задачі пошуку К-ого елемента на підвідрізку масива. Давайте обійдемо граф ДФС-ом та побудуємо персистентне розріджене дерево відрізків на величинах ребер, що знаходяться на стеку ДФСу. Індекс - величина, що записана на ребрі, значення - кількість таких ребер. Операція над деревом - сума. У кожній вершині дерева буде зберігатись окрема версія персистентного дерева відрізків.

Тоді для відповіді на запит необхідно взяти версії у вершинах x, y та $lca(x, y)$ і виконати пошук K-ого елемента в дереві, яке можна представити як комбінацію: $tree(x) + tree(y) - 2 \cdot tree(lca(x, y))$. Асимптотика розв'язку - $O((N + Q)\ \log N)$.

F. Mex на підмасиві

Як ви пам'ятаєте, в Теорії ігор часто використовується функція *mex* при розрахунку функції Шпрага-Гранді для окремого стану гри. Та зараз нам не до ігор. Ваша задача – знаходити *mex* на підвідрізку масива.

Якщо ви все ж таки забулися, нагадаємо, що функція *mex* від множини чисел повертає найменше невід'ємне ціле число, що не міститься в цій множині.

Формат вхідних даних

У першому рядку знаходиться число $1 \leq N \leq 2 \cdot 10^5$ – розмір масиву.

У другому рядку задано N цілих чисел A_1, A_2, ..., A_N - елементи масиву, $0 \leq A_i \leq 2 \cdot 10^5$.

У третьому рядку задано число $1 \leq Q \leq 3 \cdot 10^5$ – кількість запитів. У кожному наступному рядку знаходяться по два цілих числа L та R, $1 \leq L \leq R \leq N$, що задають границі підмасиву.

Формат вихідних даних

Виведіть Q невід'ємних цілих чисел, по одному в кожному рядку – відповідь на запити.

Приклади

stdin	stdout
7 0 1 0 2 0 1 0 3 1 7 3 5 6 6	3 1 0

Розв'язання задачі «F. Мех на підмасиві»

Давайте розв'яжемо цю задачу для запитів на суфіксу масиву. Для цього побудуємо дерево відрізків із операцією мінімуму над масивом, де в позиції P зберігається індекс останнього входження елементу P у вихідний масив (або -1 якщо входжень немає). Тоді відповідь на запит про відрізок, що починається в позиції L та закінчується у кінці масиву, є найменше число, індекс останнього входження якого менший за L. Це робиться однією операцією спуску по дереву: у поточній вершині перевіряємо лівого сина, якщо записане там значення менше за L, йдемо туди, інакше йдемо в правого сина. Для розв'язання вихідної задачі необхідно використати персистентне дерево відрізків над масивом – тоді для запиту (L, R) необхідно звертатися до R-ої версії дерева, або ж розв'язати задачу в офлайні за допомогою сортування запитів по правій границі. В обох випадках асимптотика розв'язку $O((N + Q)\ \log N)$.

G. Женя та чудовий код

Одного дня Женя написав чудовий (як йому здалося) код. Це була проста функція, що приймала ціле число і повертала ціле число. Та чомусь, ця функція працювала дуже і дуже повільно, із того часу Женя ще не вирахував усі значення цієї функції, які хотів. Враховуючи те, що гуртом це можна зробити швидше, Женя просить вас допомогти йому порахувати значення чудової

функції для певних вхідних даних. Сама функція зображена на псевдокоді нижче:

```
1   def foo(a, b):
2       if a < b:
3           return a == 0
4       else
5           return foo(a - b, b)
6
7   def awesome_function(n):
8       r = 0
9       for i = 1 .. n:
10          for j = 1 .. n:
11              v = []          # empty array
12              x = y = 0
13              while x < i:
14                  x += 1
15                  if foo(i, x):
16                      v.append(x)     # add element to array
17              while y < j:
18                  y += 1
19                  if foo(j, y):
20                      v.append(y)
21              t = 0
22              for a = 0 .. size(v) - 1:
23                  for b = 0 .. a - 1:
24                      if v[a] == v[b]:
25                          t = v[b]
26              if t == 1:
27                  r += x * y
28      return r
29
```

Формат вхідних даних

Єдине ціле число $1 \leq N \leq 10^7$.

Формат вихідних даних

Виведіть значення чудової функції від числа N за модулем $10^9 + 7$.

Розв'язання задачі «G. Женя та чудовий код»

Щоб розв'язати цю задачу, спочатку необхідно зрозуміти, що робить наведений код. Так рекурсивна функція $foo(a, b)$ перевіряє, чи ділиться число a на b. Цикли, що використовують цю функцію, заносять у масив v дільники чисел i та j. Наступний подвійний цикл, що перебирає елементи масиву v,

шукає у ньому дублікати та записує у змінну t значення останнього із повторюваних чисел. Оскільки в цьому масиві знаходяться дільники двох чисел, то дублікати – це їх спільні дільники, а останнє таке число – їх найбільший спільний дільник. Отже, весь цей код можна переписати так:

```
r = 0
for i = 1 .. n
  for j = 1 .. n
    if gcd(i, j) == 1:
      r += i * j
return r
```

Цього достатньо, щоб розв'язати задачу в обмеженнях для другого дивізіону.

Щоб розв'язати цю задачу для більших обмежень, модифікуємо код так:

```
r = 0
for i = 1 .. n
  s = 0
  for j = 1 .. i
   if gcd(i, j) == 1:
     s += j
  r += s * i
return r * 2
```

Із цього коду коду видно, що вкладений цикл по j шукає суму чисел взаємно простих з i та менших за i. Можна знати або вивести, що ця сума дорівнює $\frac{phi(i)\cdot i}{2}$, де phi – функція Ейлера, або кількість чисел менших за i, що взаємно прості з i. Вивести цю формулу можна з таких суджень, що, якщо число a $(a < b)$ взаємно просте з b, то число $b - a$ також взаємно просте. А тому всі такі числа розбиваються на пари, сума кожної з яких дорівнює числу a. Отже, розв'язок задачі зводиться до такого коду:

```
for i = 1 .. n
```

```
    r += phi(i) * i * i
```

Залишилося порахувати значення функції Ейлера для всіх чисел від 1 до n. Це досить стандартна задача, яку можна розв'язати за лінійний час за допомогою лінійного решета Ератосфену, або дещо простіше і за аналогією зі звичайним решетом за час $O(N \log\log N)$:

```
for i = 1 .. n
  phi[i] = i
for i = 2 .. n
  if phi[i] == i: // i - просте число
    for (j = i; j <= n; j += i)
      phi[j] = phi[j] / j * (j - 1);
```

Коректність цього коду випливає безпосередньо з вираження функції Ейлера у вигляді так званого добутку Ейлера:

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right), n > 1.$$

H. Ряд натуральних чисел

Чи не задумувалися ви над тим, що ряд натуральних чисел, яким ми всі його знаємо: 1, 2, 3, 4, 5 ... – не досконалий? Можливо, буде краще щось змінити у ньому?

Давайте це перевіримо. Однією з нудних особливостей натурального ряду є те, що він неперервно зростає. Та це можна просто змінити, задавши певні позиції, з яких необхідно змінювати напрям відліку. Вашезавдання – написати програму, яка буде міняти напрям ряду цілих чисел у деяких позиціях, вираховувати суму чисел на заданих позиціях та знаходити найбільше число на інтервалі позицій колишнього натурального ряду.

Більш формально, вам необхідно оброблювати запити:

TURN x – цей запит означає, що з цього момента необхідно змінювати напрям відліку в позиції x на протилежний. Наприклад, команда TURN 6

перетворить ряд натуральних чисел на ряд: 1 2 3 4 5 4 3 2 ... Команда TURN 3, виконана після цього, задасть наступний ряд: 1 2 1 0 -1 0 1 2 3 4... Як бачимо, після цих двох команд у позиціях 3 та 6 змінюється напрям відліку. Гарантується, що всі такі запити будуть мати різний x.

MAX x y – знайти найбільше число, що стоїть зараз на одній із позицій x, $x+1$, ..., y.

SUM x y – вирахувати суму всіх чисел, що стоять зараз у ряді на позиціях x, $x+1$, ..., y.

Перед першим запитом всі натуральні числа знаходяться на своїх місцях.

Формат вхідних даних

У першому рядку задано одне ціле число $1 \le Q \le 2 \cdot 10^5$ – кількість запитів. Кожен із наступних рядків описує один із трьох запитів:

TURN x, $2 \le x \le 10^9$,

MAX x y, $1 \le x \le y \le 10^9$,

SUM x y, $1 \le x \le y \le 10^9$.

Формат вихідних даних

На кожен запит другого і третього типів в окремому рядку виведіть відповідь на цей запит.

Приклади

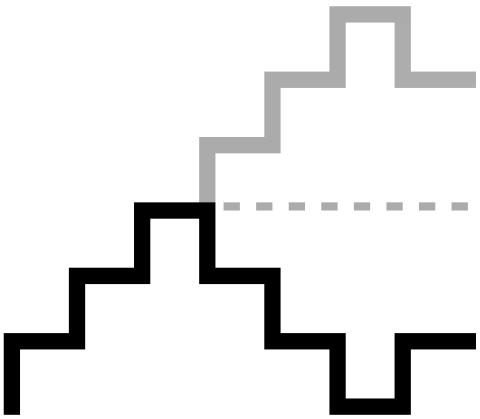
stdin	stdout
8 TURN 4 SUM 4 7 MAX 2 7 TURN 7 MAX 1 10 TURN 2 SUM 1 10 MAX 1 10	2 3 4 1 2

Розв'язання задачі «H. Ряд натуральних чисел»

Уявімо наш ряд натуральних чисел як послідовність сходинок. Тоді запит повороту у позицій X є розворотом ділянки сходів, що починається з позиції X.

Давайте побудуємо дерево відрізків, кожен із відрізків якого описуватиме ділянку сходів у відношенні до попередньої ділянки, тобто вважатимемо, що остання сходинка попередньої ділянки має висоту 0.

У вершині дерева зберігатимемо такі значення: максимальна висота сходинки (Min), мінімальна висота сходинок (Max), висота останньої (H), сума висот усіх сходинок на ділянці (S). Усі ці значення вираховуються із врахуванням того, що висота останньої сходинки попередньої ділянки – 0, або ж, можна сказати, що всі ці значення є відносними до висоти останньої сходинки попередньої ділянки. Операція повороту буде виконуватись із проштовхуванням як операція пофарбування на дереві відрізків. Значення висоти останньої і суми при цьому домножуватимуться на -1, значення мінімуму і максимуму також домножуватимуться на -1 і мінятимуться між собою.



Перерахунок цих значеннь у вершині із значень її дітей буде таким:

$$Min = min(Min\_L, Min\_R + H\_L)$$

$$Max = max(Max\_L, Max\_R + H\_R)$$

$$H = H\_L + H\_R$$

$$S = S\_L + S\_R + H\_L * Cnt\_R,$$

(тут Cnt_R – кількість сходинок у правого сина, явно не зберігаємо, вираховується під час проходу по дереву відрізків).

Для відповіді на запити необхідно повертати структуру з цими значеннями та комбінувати їх відповідним чином.

Асимптотика розв'язку $O(N \log MaxVal)$. .

I. Проста сума

Знайдіть усі пари простих чисел (A, B), таких що $A \leq B$ та їх сума є також простим числом і не перевищує N.

Формат вхідних даних

Єдине ціле число $1 \leq N \leq 10^7$.

Формат вихідних даних

Єдине ціле число – кількість пар, що задовольняють умову.

Приклади

stdin	stdout
4	0

Розв'язання задачі «I. Проста сума»

Усі прості числа, крім двійки, є непарними. Сума двох непарних простих чисел є парною і більша за два, а тому не може бути простим числом. А тому один із доданків – число 2. Таким чином необхідно знайти кількість чисел P таких, що числа P і $P + 2$ – прості. Зробити це можна звичайним решетом Ератосфена.

J. Скільки разів?

Вам дано код звичайнісінького запиту на суму в дереві відрізків, яке побудовано над масивом розміру N. Порахуйте і скажіть, яке значення буде мати змінна *counter* після виклику цього запиту для кожного можливого підвідрізка вихідного масиву. Тобто вам необхідно знайти, скільки сумарно разів буде братись якесь значення із дерева при виклику всіх можливих коректних запитів.

```
1
2   int counter = 0;
3   int sum(int cur, int l, int r, int x, int y) {
4       if (l == x && r == y) {
5           ++counter;
6           return tr[cur];
7       } else {
8           int m = (l + r) / 2;
9           int dcur = cur + cur;
10          if (y <= m) {
11              return sum(dcur, l, m, x, y);
12          } else if (x > m) {
13              return sum(dcur + 1, m + 1, r, x, y);
14          } else {
15              return sum(dcur, l, m, x, m) + sum(dcur + 1, m + 1, r, m + 1, y);
16          }
17      }
18  }
19
```

Формат вхідних даних

Єдине ціле число $1 \leq N \leq 10^6$ – розмір масиву.

Формат вихідних даних

Єдине ціле число – відповідь на задачу.

Приклади

stdin	stdout
4	13
10	115

Розв'язання задачі «J. Скільки разів?»

Із принципу роботи запитів дерева відрізків випливає те, що весь інтервал вихідного запиту розбивається на менші інтервали, кожен із яких відповідає якійсь вершині у дереві відрізків. Для певної вершини і певного інтервалу запиту ми можемо сказати, що просумуємо значення з цієї вершині в тому випадку, коли інтервал, за який відповідає вершина, повністю належить інтервалу-запиту, а інтервал її батьківської вершини - ні. Для вершини, що відповідає за інтервал $[L; R]$ масиву на інтервалі $[0;\ N-1]$, існує $L \cdot (N-1-R)$ запитів, що повністю містять у собі інтервал $[L; R]$.

Отже, для знаходження відповіді на задачу, необхідно для кожної вершини дерева відрізків додати кількість запитів, що містять інтервал цієї вершини, та відняти кількість запитів, що містять інтервал батьківської вершини.

Асимптотика розв'язку $O(N)$.

K. Множина, підмножина, XOR та якесь число

Професор Кілобайтко має множину цілих чисел. Інколи ця його множина збільшується, а інколи він проводить над нею певні досліди, як, наприклад, узяти якусь підмножину цих чисел та вирахувати побітову суму цих чисел за модулем 2, або, простіше кажучи, знайти XOR усіх чисел із цієї підмножини. Назвемо результат цієї операції числом X. Також кожен дослід супроводжується певним числом K. Щоб отримати результати досліду, професор Кілобайтко вираховує значення $X \operatorname{XOR} K$. Звичайно, результати досліду тим кращі, чим більше буде отримане число.

Допоможіть професорові наперед визначити, який найкращий та найбільший результат досліду він може отримати.

Формат вхідних даних

У першому рядку задано натуральне число $1 \leq N \leq 2 \cdot 10^5$ – кількість запитів.

Кожен із наступних N рядків містить опис запиту одного з двох типів:

A t, де $0 \leq t \leq 10^{18}$ – число, що має бути додане до множини чисел професора.

G k, де $0 \leq k \leq 10^{18}$ – число, з яким професор збирається провести дослід.

Формат вихідних даних

Для кожно запиту другого типу вам необхідно вивести одне ціле число в окремому рядку - найбільший результат досліду, який професор може отримати.

Приклади

stdin	stdout
7 A 5 A 3 G 0 G 4 A 7 G 15 G 6	6 7 15 7

Примітка

XOR від порожньої множини вважати рівним нулю.

Розв'язання задачі «K. Множина, підмножина, XOR та якесь число»

Спочатку необхідно навчитися розв'язувати цю задачу для запитів G 0. Тобто - знайти підмножину, що дає найбільший XOR сама собою. Щоб розбити це, використаємо такий жадібний алгоритм: переберемо біти від старших до молодших, для поточного біта оберемо будь-яке ще не додане до відповіді число, що містить цей біт, додамо його до відповіді, а всі інші такі ж числа проксоримо із цим числом. Щоб показати, що це дійсно працює, по-перше, треба показати, що заміна числа на його XOR із якимось іншим числом із множини не псує відповідь (справді, якщо ми беремо числа A та A^B, то можемо отримати будь який результат із вихідних, тобто 0, A, B, A^B). Також очевидним є те, що якщо ми маємо єдине число із встановленим старшим бітом, то його обов'язково треба брати у відповідь, бо сума за всіма іншими бітами заздалегідь менша. З такого жадібного алгоритму випливає те, що у відповідь ми візьмемо не більше ніж логарифм чисел. Назвемо їх базисом. Очевидно, що після виконання такого алгоритму всі інші числа стануть нулями, а отже, їх можна відкинути. Таким чином, ми можемо додавати по одному числу в множину та підтримувати відповідь і базис. При додаванні числа, спочатку необхідно видалити з нього ті біти, що є значущими в базисі, а потім, якщо число не стало нулем, додати його до базису та проксорити ті числа із базису, які мають встановленим його старший біт. Варто зазначити, що

наведений алгоритм є не що інше, як алгоритм Гауса для рівнянь за модулем 2. Та враховуючи те, що ми працюємо з числами, усе це можна виконувати на бітових операціях.

Повертаючись до вихідної задачі, для відповіді на запит $\mathrm{G}\,K$ для довільного числа K, можна зімітувати додавання числа K до базису, для цього треба встановити числу K найбільший біт, якого немає в жодного іншого числа, виконати операцію додавання цього числа до множини, отримати відповідь та відкотити зміни до попереднього стану.

Інший спосіб відповіді на запит можна показати таким кодом:

```
ans = k
for (x in basis)
  if (ans ^ x) > ans:
    ans ^= x
return ans
```

Ідея цього коду полягає в тому, що ми обираємо лише ті числа у відповідь, що збільшують її.

Асимптотика розв’язку $O(N \log X)$.

Літня школа з програмування (Ужгород, 31 липня – 7 серпня 2016 року) : Матеріали лекцій, умови та розбір задач / За ред. Олександра Міци, Сергія Оришича. – Ужгород: Видавництво «ФОП Сабов А.М.», 2017. – 164 с.

ISBN 978-617-7344-46-8

Навчальне видання

Літня школа з програмування

(Ужгород, 31 липня – 7 серпня 2016 року)

Матеріали лекцій, умови та розбір задач

За редакцією Олександра Міци, Сергія Оришича

Підписано до друку 09.08.2017 р.
Формат 60x84/16. Папір офс. Гарнітура: Times New Roman.
Друк офс. Ум.друк.арк 10,23. Обл.-вид. арк. 17,09
Тираж 200 шт. Замовлення №38.

Видавництво «ФОП Сабов А.М.».
м. Ужгород, вул. Університетська, 21/220.
Тел./факс: (0312) 64-37-22.
Свідоцтво суб'єкта видавничої справи
ДК № 4815 від 25.02.2015 р.
Друк: ФОП Сабов А.М., тел. 050-43-22-437

УДК 004.42(042.3)
М-70