Вступ

Даний методичний посібник є продовженням серії двох попередніх посібників [1,2], в яких викладені методичні вказівки до розв'язання олімпіадних задач державних, обласних і міських олімпіад, що пропонувалися в 1986-1996 роках. Особливістю даного посібника є "швидка реакція" на розкриття особливостей розвязання олімпіадних задач, що пропонувалися на державній олімпіаді юних інформатиків в 1998 році і міській олімпіаді м.Ужгорода в листопаді місяці 1998 року. Програмні установки в посібнику носять як принциповий характер, так і містять розгорнутий методичний коментар. Також відбитий досвід пошукової роботи в виявленні нових оптимальних методів розв'язання задач підвищеної складності. Деякі з приведених в посібнику завдань вже пропрацьовані з учнями спецщколи при Ужгородському державному інституті інформатики, економіки і права, Ужгородській гімназії.

Автори вважають , що даний посібник буде корисним не тільки учням, а і студентам, що вивчають інформатику на молодших курсах.

Завдання 2-го етапу (міської) олімпіади юних інформатиків для 8-9 класів.

Завдання 1. Розмін монет.

Скласти програму, яка б визначала:

Скількома способами можна подати розмін монети номіналом N на K монет номіналами N_1, N_2, ... , N_k.

Розв'язання

Спочатку беремо скільки можна монет номіналом N_1, а потім на що залишилося набераємо монети номіналом N_2 і т.д. Нехай останні взяті монети будуть номіналом N_{z1}. Нашим слідуючим кроком буде: відкидаємо одну монету номіналом N_{z1} і на остачу набераємо монети починаючи з номіналу N_{z1+1}. Нехай останні взяті монети будуть номіналом N_{z2}. Тоді знову відкидаємо одну монету номіналом N_{z2} і на остачу набераємо монети починаючи з номіналу N_{z2+1}. Таким чином ми пройдемо всі можливі варіанти.

Позначення:
NN - монета яку розмінюють;
N[1..m] - номінали якими розмінюють;
Ost - поточна остача;
B[1..m],C[1..m] - набор і кількості цього набору відповідно;
p - кількість типів відрізків;
count - кількість варіантів;

Текст програми:

```
var                                          {описання змінних}
N : array[1..20] of real;
  b,c : array[1..20] of integer;
  NN,Ost : integer;
  count,z,i,p,m : integer;

procedure Vivod_result;
begin
    writeln('Всього варіантів=',count);
    halt;
end;

procedure Way;                          {Вивід знайденого шляху}
begin
    for i:=1 to p do
        writeln(b[i],' - ',c[i]);
    writeln;
    inc(count);                         {Збільшуємо кількість на 1}
end;

begin
```

Ввід даних;

```
Ost:=NN;
    p:=0;
    z:=1;
    while true do
    begin
        for i:=z to m do
        begin
          if N[i]<Ost then              {знаходимо інший варінт}
          begin
            inc(p);
            b[p]:=i;
            c[p]:=trunc(Ost/N[i]);
            Ost:=Ost-c[p]*N[i];
            if Ost=0 then Way;
          end;
        end;
            if p=0 then Vivod_result;
            dec(c[p]);                   {Крок назад}
            z:=b[p]+1;                   {Відкидаємо останню взяту монету}
            Ost:=Ost+N[b[p]];
            if c[p]=0 then dec(p);
    end;
end.
```

Завдання 2. Заміна.

В послідовності символів x[1], x[2], ... , x[n] визначити наявність в ній підряд розміщених один за одним символів "abcd" (вказати кількість вкладень k=0,1, ...), і якщо k>0, то виконати заміну символів "abcd" на "ijklmn".

Розв'язання

Позначення:
X - введена строка;
K - кількість вкладень.
P - позиція 'abcd' в X;

Текст програми:

```
var                                     {описання змінних}
x : string;
  k,p : integer;

begin
    readln(x);
    k:=0;
    while pos('abcd',x)>0 do
    begin
        p:=pos('abcd',x);
        delete(x,p,4);
```

```
        insert('ijklmn',x,p);
        inc(k);
    end;
    writeln('Кількість вкладень:', k);
    writeln(x);
end.
```

Завдання 3. Довга арифметика. (множення "вручну")

Скласти програму, яка б реалізовувала по-розрядний процес множення двох чисел AxB заданих послідовністю символів a[1], a[2], ... , a[n] і b[1], b[2], ... , b[m]. Результат записати як послідовність символів x[1], x[2], ... , x[n+m].

Розв'язання

Позначення:
A[1..n],B[1..m] - масиви, що перемножаються;
D[1..n,1..n+m] - таблиця, де в кожному рядку містяться результати множення числа з масиву A на числа з масиву B;
X[1..n] - масив, де міститься кінцевий результат множення;
Наприклад:

		9	**2**	**3**
			7	**3**
	2	**7**	**6**	**9**
6	**4**	**6**	**1**	
6	**7**	**3**	**7**	**9**

1-ий рядок - масив B;
2-ий рядок - масив A;
3-,4-ий рядки - матриця D;
5-ий рядок - масив X.

Текст програми:

```
var                                          {описання змінних}
  a,b,x : array[1..10] of byte;
  n,m,i,j,s,p,z : integer;
  d : array[1..10,1..20] of byte;

begin
    write('n=');                             {Ввід данних}
    readln(n);
    write('m=');
    readln(m);
    for i:=1 to n do read(a[i]);
    writeln;
    for i:=1 to m do read(b[i]);
    writeln;

    for i:=n downto 1 do                     {Множення в таблицю D}
    begin
        p:=0;
```

```
        for j:=m downto 1 do
        begin
            z:=a[i]*b[j]+p;
            d[i,i+j]:=z mod 10;
            p:=trunc(z/10);
        end;
    end;
    p:=0;
    for i:=n+m downto 1 do              {Сумування стовпців масиву D}
    begin
        s:=p;
        for j:=n downto 1 do
        begin
            s:=s+d[j,i];
        end;
        x[i]:=s mod 10;
        p:=trunc(s/10);
    end;

    j:=1;
    while x[j]=0 do inc(j);        {Знаходимо перший не нульовий елемент масиву
X}
    for i:=j to n+m do write(x[i]);  {вивід результату }
end.
```

Завдання 2-го етапу (міської) олімпіади юних інформатиків для 10-11 класів.

Завдання 1.Відрізки.

Вмістити відрізки довжинами L_1, L_2, ... , L_k в відрізок L з найменшим залишком.

Розв'язання

Аналогічна до 1-ої задачі за 8-9 клас.
Позначення:
LL - відрізок в який вкладають;
L[1..n] - відрізки, які вкладаються;
Min_Ost - мінімальна остача;
Ost - поточна остача;
B[1..n],C[1..n] - набор і кількості цього набору відповідно;
B1[1..n],C1[1..n] - найкращий набор і кількості цього набору відповідно;
p - кількість типів відрізків;
p1 - кількість типів відрізків при найкращому наборі;

Текст програми:

```
var                                          {описання змінних}
  L : array[1..20] of real;
  b,c,b1,c1 : array[1..20] of integer;
```

```
  LL,Ost,Min_Ost : real;
  z,i,p,p1,n : integer;

procedure Vivod_result;
begin
    writeln('Мінімальний залишок=',Min_Ost);
    writeln('Відрізки :');
    for i:=1 to p1 do
       writeln('Відрізок ',b1[i],' в кількості ',c1[i]);
    halt;
end;

procedure SAVE;
begin
    Min_Ost:=Ost;
    p1:=p;
    for i:=1 to p1 do
    begin
        b1[i]:=b[i];
        c1[i]:=c[i];
    end;
end;

begin
```
Ввід даних;
```
    Ost:=LL;
    p:=0;
    Min_ost:=LL;
    z:=1;
    while true do
    begin
        for i:=z to n do
        begin
          if L[i]<Ost then            {знаходимо слідующий варіант}
          begin
            inc(p);
            b[p]:=i;
            c[p]:=trunc(Ost/L[i]);
            Ost:=Ost-c[p]*L[i];
            if Min_Ost>Ost then SAVE;
          end;
        end;
            if p=0 then Vivod_result;
            dec(c[p]);                  {Крок назад}
            z:=b[p]+1;                  {Відкидаємо останній взятий відрізок}
            Ost:=Ost+L[b[p]];
            if c[p]=0 then dec(p);
    end;
```

end.

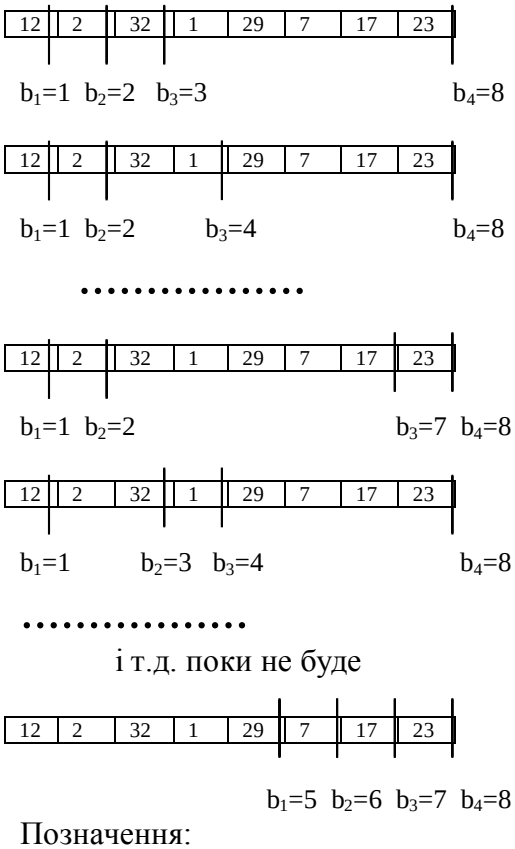
Завдання 2.Вантаж.

В черзі для відправки знаходяться n контейнерів вагою p_1, p_2, ... , p_n. Транспортер проводить погрузку вантажу на k автомашин вантажністю Q. Не порушуючи черговості провести погрузку вантажу так, щоб максимальне завантаження окремого транспорту було мінімальним.

Розв'язання

Спочатку присвоюємо B[i]=i (i=1, ... , k-1), тобто на 1-шу погружаємо 1-ий груз, на 2-гу - 2-ий, ... , на (k-1)-ий - (k-1)-ий, а на k-ий - всі інші. Потім на 1-шу погружаємо 1-ий груз, на 2-гу - 2-ий, ... , на (k-1)-ий - (k-1)-ий і k-ий (B[k-1]=k), а на k-ий транспорт - всі інші. І т.д. поки не буде B[k-1]=n-1. А далі збільшуємо на 1 значення B[k-2], а B[k-1] стане рівним B[k-2]+1. І т.д. дійдемо до випадку коли для i=k-1, ... , p буде виконуватися B[i]=n-k+i. Тоді B[p-1] збільшуємо на 1, а решту (i=p, ... ,k-1) ставимо підряд за ним. І т.д. поки всі B[i] (i=k-1, ... , 1) не стануть рівні n-k+i.

Наприклад:
n=8; k=4;



. . . . . .

. . . . . .

і т.д. поки не буде

Позначення:
k - кількість машин;
n - кількість вантажів;
L[1..n] - масив вантажів;
B[0..k] - масив поточного розподілу вантажів по машинам;
B1[0..k] - масив найкращого розподілу вантажів по машинам;

Min_m - значення найбільш завантаженої машини при найкращому розподілі;
Sp - значення найбільш завантаженої машини при поточному розподілі;

Текст програми:

```
var                                          {описання змінних}
L,b,b1 : array[0..20] of integer;
   k,i,j,p,n,Min_m,q : integer;

procedure Vivod_result;          {Вивід результату}
begin
  if Min_m<30000 then
  begin
     writeln('Мінімальне значення = ',Min_m);
     writeln('Машини :');
     for i:=1 to k do
        writeln('В ',i,'-ту машину завантажуємо вантажі з ',b1[i-1]+1,' до ',b1[i]);
  end
  else writeln('Погрузка неможлива');
     halt;
end;

procedure Obrob;
var
   Sm,Sp : integer;
begin
     Sp:=0;
     for i:=1 to k do          {знаходимо значення найбільш завантаженої машини }
     begin
          Sm:=0;
          for j:=b[i-1]+1 to b[i] do
             Sm:=Sm+L[j];
          if Sm>Sp then Sp:=Sm;
     end;
     if (Sp<Min_m)and(Sp<=Q) then {якщо воно менше за поточне оптимальне}
     begin
          Min_m:=Sp;                    {то запам'ятовуємо його}
            for i:=1 to k-1 do
               b1[i]:=b[i];
     end;
end;

begin
Ввід даних;
for i:=1 to k-1 do
       b[i]:=i;
     b[0]:=0; b[k]:=n;
     b1[0]:=0; b1[k]:=n;
     Min_m:=30000;         {присвоюємо йому якесь велике значення}
```

```
    while true do
    begin
          p:=k-1;
          while b[p]=n-k+p do {знаходимо перше p для якого b[p]<>n-k+p }
              dec(p);
          if p=0 then Vivod_result;
          inc(b[p]);                {збільшуємо його на одиницю}
          for i:=p+1 to k-1 do      {всі інші ставимо підряд за ним}
             b[i]:=b[p]+i-p;
          Obrob;
    end;
end.
```

Завдання 3.Розклад маршрутів.

Дано розклад всіх автобусних маршрутів міста. Їхати можна з пересадками. Скласти програму, що визначить:

а) необхідний найменший час подорожі з станції X до станції Y.

б) всі станції до яких можна добратися з станції X до k хвилин за маршрутом з найменшим часом подорожі.

Розв'язання

Метод рішення аналогічний методу описаному в задачі "Водопровід".

За) Позначення:

A[1..n,1..n] - таблиця відстаней;
B[1..p] - масив, який показує поточний шлях;
B1[1..p1] - масив, який показує найменший шлях;
C[1..n] - показує чи зайнята вершина;
s - поточна довжина;
s1 - найменша довжина;
p - поточна кількість вершин в масиві B;
p1 - показує кількість вершин в найменшому шляху;

Текст програми:

```
var                                          {описання змінних}
a : array[1..100,1..100] of integer;
  n,s,s1,x,y,p,i,j,z,p1 : integer;
  b,b1,c : array[1..100] of integer;

procedure Obrob;              {запоминаємо кращий шлях }
begin
       s1:=s;
       p1:=p;
       for i:=1 to p1 do b1[i]:=b[i];
end;

procedure Vivod_rez;          {Вивід результату}
begin
   writeln('Min=',s1);
```

```
    for i:=1 to p1 do
       writeln(b1[i],' ');
    halt;
end;

begin
```
Ввід даних;
```
s1:=10000; s:=0;
    for i:=1 to n do
     for j:=i+1 to n do
        a[j,i]:=a[i,j];
    for i:=1 to n do        {всі вільні}
          c[i]:=0;
   b[1]:=x; p:=1; c[x]:=1;          {x - початок , y - кінець}
    while true do
    begin
       if b[p]<>y              {знаходження слідуючої вершини починаємо з z}
         then z:=b[p+1]+1
         else z:=n+1;
       while ((a[b[p],z]=0)or(c[z]=1)or(s+a[b[p],z]>s1))and(z<=n) do
            inc(z);            {знаходження слідуючої вершини}
       if z>n then
       begin
           c[b[p]]:=0; b[p+1]:=0; dec(p);             {крок назад}
           if p>0 then s:=s-a[b[p],b[p+1]];  {відкидаємо останню вершину}
           if p=0 then Vivod_rez;
       end
       else begin
           inc(p); c[z]:=1; b[p]:=z;
           s:=s+a[b[p-1],z];                  {добавляємо z до поточної гілки}
           if z=y then Obrob;
       end;
    end;
end.
```

3б) Позначення:
A[1..n,1..n] - таблиця відстаней;
B[1..p] - поточний шлях;
B1[1..p1] - показує шлях-розвязок;
C[1..n] - показує чи зайнята вершина;
k - введена довжина;
s - поточна довжина;
p - поточна кількість вершин в масиві B;

Текст програми:

```
var                                          {описання змінних}
  a : array[1..100,1..100] of integer;
  n,k,s,s1,x,y,p,i,j,z,p1 : integer;
```

```
  b,b1,c : array[1..100] of integer;


procedure Vivod_rez;              {Вивід результату}
begin
    writeln('Вершина=',y);
    for i:=1 to p1 do
       write(b1[i],' ');
       writeln;
end;

procedure Zmina;                  {беремо за y слідуючу вершину}
begin
        inc(y);
        for i:=1 to n do
        begin
           c[i]:=0;
           b[i]:=0
        end;
        c[x]:=1; p:=1; b[1]:=x;
        s:=0;
        if y=x then inc(y);
        if y>n then halt;
end;

procedure Obrob;
begin
        p1:=p;
        for i:=1 to p1 do b1[i]:=b[i];
        Vivod_rez;
end;

begin
```
Ввід даних;
```
    if x=1
     then y:=2
     else y:=1;
for i:=1 to n do
     for j:=i+1 to n do
        a[j,i]:=a[i,j];
    for i:=1 to n do        {Всі вільні}
           c[i]:=0;
    s:=0;
    b[1]:=x; p:=1; c[x]:=1;          {x - початок, y - кінець}
    while true do
    begin
       if b[p]<>y              {знаходження слідуючої вершини починаємо з z}
         then z:=b[p+1]+1
```

```
        else z:=n+1;
      while ((a[b[p],z]=0)or(c[z]=1)or(s+a[b[p],z]>k))and(z<=n) do
          inc(z); {знаходження слідуючої вершини}
      if z>n then       {якщо не існує}
      begin
         c[b[p]]:=0; b[p+1]:=0; dec(p);          {крок назад}
         if p>0 then s:=s-a[b[p],b[p+1]];      {відкидуємо останню взяту вершину}
         if p=0 then Zmina;
      end
      else begin
         inc(p); c[z]:=1; b[p]:=z;
         s:=s+a[b[p-1],z];          {добавляємо z до поточної гілки}
         if z=y then Obrob;
      end;
   end;
end.
```

Нижче приведені задачі, які були висунуті на державній олімпіаді в 1998р.

Водопровід

Місто складається з N районів ($1 \le N \le 100$). Кожен район має свердловину для отримання води. Кожні дві свердловини з'єднані між собою трубою. По кожній трубі вода може текти тільки в одному напрямку. Внаслідок енергетичної кризи в кожен момент часу працює тільки одна свердловина. Оскільки система проектувалася без передбачення такого режиму роботи, деякі райони міста інколи залишаються без води.

Завдання: Напишіть програму, яка визначить чи можна, змінивши напрямок проходження води по всіх трубах, що приєднані до однієї із свердловин, добитись безперервного водопостачання в місті.

Розв'язання:

- Беремо 1-шу вершину і за допомогою *методу повного перебору* будуємо всі можливі шляхи руху води, враховуючи зміну напрямку в трубах що приєднані до неї. Якщо існує така гілка яка закінчується в 1-шій вершині, тоді це означає, що всі свердловини, які входять до даної гілки можуть постачатись водою із 1-шої вершини. Це ми фіксуємо в масиві M[1..N]. Якщо пробираючи всі гілки ми визначимо, що кількість вершин, які можуть забезпечуватися водою дорівнює N, то розв'язком буде 1-ша вершина. Якщо ні, то пробираємо всі свердловини по порядку і за вказаним вище правилом визначаємо чи може вона бути розв'язком, чи ні. Елементи поточної гілки записуються в масиві B[1..N+1].

Метод повного перебору організовуємо слідуючим чином :
вибираємо якусь свердловину i_1, яку хочемо перевірити чи буде вона розв'язком нашої задачі. Тому приймаємо $B[1]=i_1$. Шукаємо першу попорядку свердловину в яку може течи з i_1-ої вода, перед цим, звичайно, міняємо напрямки води по всім трубам (так як це сказано в умові) по яким зв'язана i_1-ша свердловина з іншими свердловинами. Нехай це буде i_2. Тоді присвоюємо $B[2]=i_2$. Потім переглядаємо всі попорядку свердловини (тобто 1, 2, 3, …,n, але вже не

розглядаючи i_1 та i_2) і знаходимо в яку з i_1-ої може течи вода. Нехай це буде i_3. Тоді присвоюємо B[3]=i_3. І так далі ми дійдемо до свердловини i_x (B[x]= i_x) з якої вода не може течи ні в яку іншу не використану свердловину. Тоді якщо із свердловини i_x вода може течи в свердловину i_1, то в масиві M[1..N] зазначимо, що свердловина i_1 може забезпечувати водою свердловини i_2,..,i_x. Далі повертаємось назад до свердловини i_{x-1} і знаходимо інше значення i_x. І так далі. Якщо ми визначимо, що всі елементи масиву M[1..N] дорівнюють false, то це означає, що свердловина i_1 буде розв'язком. Це все робить нижеприведена процедура STEP.

Рішення задачі на мові Паскаль:

```
Var                                                    {описання змінних}
 a : array[1..4,1..4] of integer;
 b : array[1..4] of integer;
 m,c : array[1..4] of boolean;
 s,r,n,p,l,i : integer;

procedure STEP;
var
  d : 0..1;
begin
  r:=b[l+1]+1;                                  {розглядаємо елементи від b[l+1]}
    if b[l]=p
      then d:=1
      else d:=0;
    if (a[b[l],p]=0) then
    begin
       for i:=1 to l do
       begin
          if m[b[i]] then
          begin
            s:=s+1;
            m[b[i]]:=false;
          end;
       end;
       if s=n then
       begin
         Вивід результату;                {результат знайдено}
         halt;
       end;
    end;
  while (a[b[l],r]=d) or (b[l]=r) or c[r] do {проглядаємо вершини поки не
виповниться одна з умов}
      begin
        r:=r+1;
        if r>n then break;
    end;
      if r<=n
```

```
    then                                    {дописуємо в масив B ще й
елемент з номером r}
      begin
        l:=l+1;
        b[l]:=r;
        c[r]:=true;
      end
    else                                    {повертаємось назад}
      begin
        c[b[l]]:=false;
        b[l+1]:=0;
        l:=l-1;
      end;
end;

begin
Ввід даних;
 p:=0;
 while true do
 begin
  p:=p+1;                                   {розглядаємо слідуючу вершину}
  s:=0;
  for i:=1 to n do
      m[i]:=true;
  l:=1;
  b[l]:=p;                  {за корінь ставимо вершину p}
  c[p]:=true;               {позначаємо, що p-та вершина уже
використовується}
  if p>n then               {перевіряємо чи розглянуто всі вершини}
  begin
    write(«Розв'язку немає»);
    halt;
  end;
  while l>0 do                                  {слідуючий крок}
   STEP;
 end;
end.
```

Годинник

Жителі планети Олімпія полюбляють літати в гості на інші планети. Вчені планети розробили годинник, що може налагоджуватися для відліку часу на будь-якій планеті. Цей годинник складається з кульок, лотка (черги) і трьох чаш: секундної, хвилинної і годинної. В кожен момент часу кількість кульок в чаші показує час (секунди, хвилини та години відповідно). Кожну секунду перша кулька з черги потрапляє в секундну чашу. Якщо секундна чаша наповнилась (кількість кульок дорівнює кількості секунд в хвилині на цій планеті), то ця кулька переходить до хвилинної чаші, а решта кульок переходять з секундної чаші в кінець черги в порядку, зворотному до їх надходження до секундної чаші. Аналогічно, при наповненні хвилинної чаші остання кулька

переходить до годинної чаші, а решта кульок з хвилинної чаші переходить в кінець черги в порядку, зворотному до їх надходження до хвилинної чаші. Якщо заповнюється годинна чаша, то всі кульки з неї переходять в кінець черги в порядку, зворотному до їх надходження в годинну чашу. Всі кульки пронумеровані і в початковий момент часу містяться в черзі.

Завдання: Написати програму, яка буде обчислювати мінімальну кількість діб, необхідних для того, щоб початкове положення кульок в черзі повторилося.

Розв'язання:

Змінні:

S – кількість секунд в хвилині;

M – кількість хвилин в годині;

H – кількість годин в добі;

K – загальна кількість кульок.

$S,M,H \le 80$, $S+M+H-2 \le K \le 1000$.

Сам етап розв'язання ділимо на такі частини:

- Знаходимо розміщення кульок після першої доби.

Тобто, після першої доби, коли в чашах, які показують скільки годин, хвилин, секунд, не буде нічого міститися і всі кульки будуть в черзі, то ми можемо визначити нові позиції кульок. Таблиця C(1..K) буде показувати, що i-тий елемент після першої доби буде знаходитися на C(i)-тому місці в черзі.

- Знаходимо період через який буде i-та кулька на своєму i-тому місці в черзі.

Ми уже знаємо, що i-та кулька після першої доби буде міститися на C(i)-тій позиції, після другої - на C(C(i))-тій позиції, після третьої – на C(C(C(i)))-тій і т.д. Таким чином ми можемо визначити через який період вона буде знову на i-тому місці в черзі. Зразу ж ми можемо зменшити кількість операцій. Щоб не визначати період для кожної кульки ми можемо визначаючи період для кульки i_0 враховувати властивість, що всі кульки на позиції яких i_0-та кулька побуває будуть мати той же період, що і вона. Тобто якщо позиція кульки i_0 змінюється таким чином :

$$i_0, i_1, i_2, \dots, i_d$$

то періоди для кульок $i_0, i_1, i_2, \dots, i_d$ будуть одинакові і будуть рівними періоду i_0-ої кульки (тобто будуть рівними d).

Наприклад позиція 1-ої кульки буде мінятись так:

$$1, 5, 2, 9, 1$$

це означає, що період 1-ої кульки рівний 4 і періоди 5-ої, 2-ої та 9-ої кульок будуть рівні 4.

Тому нам досить фіксувати період для i_0-ої кульки, а періоди для кульок $i_1, i_2, \dots, i_d$ ми можемо не розглядати.

- Знаходимо найменше спільне кратне для всіх періодів.

Найменше спільне кратне для всіх періодів будемо знаходити за такою схемою. Нехай A_1, A_2, A_3 і т. д. – знайдені періоди. Для A_i та A_j ми можемо знайти їх найбільший спільний дільник. Ділимо A_j на знайдене значення. І так пророблюємо для всіх $i=1,\dots,p-1$ та $j=i+1,\dots,p$, де p – кількість значень

періодів. Проробивши ці дії ми отримаємо, що найменше спільне кратне буде дорівнювати $\prod_{i=1}^{n} A_i$.

Легко догадатися, що знайдене нами найменше спільне кратне буде показувати мінімальну кількість діб, необхідних для того, щоб початкове положення кульок в черзі повторилося.

Рішення задачі на мові Паскаль:

```
Var                                                    {описання змінних}
  s,m,h,k,m1,h1,i,j,p,p1,n : integer;    {l1[1..h], l2[1..m] – масиви з номерами кульок,
які знаходяться}
  l1,l2,c,c1 : array[1..81] of integer;          {в годинній і хвилинній чаші
відповідно}
  a : array[1..80] of real;
  t : array[1..80] of boolean;
  ss : extended;

function NSD(x,y : real): real;                  {функція знаходження найбільшого}
begin                                                 {  спільного дільника }
 while x<>y do
 begin
   if x>y
    then x:=x-y
    else y:=y-x;
 end;
 NSD:=x;
end;                                                  {кінець функції знаходження НСД}

begin
Ввід данних;
  for i:=1 to k do
    c[i]:=i;                                          {c[i] – номер кульки на і-тій позиції}
  h1:=0;                                        {початок блоку знаходження позицій кульок
за одну добу}
  while h1<h do                                    {поки годинна чаша не заповниться}
  begin
    m1:=0;                                        {m1,h1 – поточні години й хвилини
відповідно}
    while m1<=m do                                 {поки хвилинна чаша не заповниться}
    begin
      m1:=m1+1;
      l2[m1]:=c[s+1];                  {першою в хвилинну чашу ставимо кульку з
номером s+1}
      for i:=1 to k-h1-m1 do           {початок блоку перерозподілу кульок після
кожної хвилини}
      begin                             {тобто ми зменшуємо кількість операцій не
використовуючи}
```

```
      if i+s+1>k-h1-m1+1            {секундну чашу, а зразу переносячи перші s
кульок в кінець}
        then c1[i]:=c[i+s-k+h1+m1]          {черги}
        else c1[i]:=c[i+s+1];                {c1[1..k] – допоміжний масив}
    end;                                {кінець блоку перерозподілу}
    for i:=1 to k-h1-m1 do
    begin
      c[i]:=c1[i];
      c1[i]:=0;
    end;
  end;

  h1:=h1+1;
  l1[h1]:=l2[m+1];{при наповненні хвилинної чаші остання кулька переходить
до годинної чаші,}
  for i:=1 to m do          { а решта кульок з хвилинної чаші переходить в кінець
черги в порядку,}
    c[k-h1-m+i]:=l2[m+1-i];          {зворотному до їх надходження до хвилинної
чаші }

 end;                                {кінець блоку знаходження позицій кульок за
одну добу}

 for i:=1 to h1 do          {при наповненні годинної чаші, кульки які знаходяться
в ній ставимо}
  c[k-h1+i]:=l1[h1+1-i];            {в кінець черги}

 for i:=1 to k do
  t[i]:=true;

 n:=0;                                          {знаходження періодів}
 for i:=1 to k do                               {n – кількість різних періодів}
 begin
  if t[i] then                                  {t[i] означає чи не знаємо ми період
i-тої кульки}
  begin                                         {тому будемо його визначати}
    p1:=1;                                      {p1 – лічильник періода}
    p:=c[i];
    t[p]:=false;
    while i<>p do                               {рахуємо період для i-тої кульки}
    begin
      p1:=p1+1;
      p:=c[p];
      t[p]:=false;
    end;
    n:=n+1;                                     {кількість періодів збільшуємо на
одиницю}
    a[n]:=p1;                                   {отримуємо новий період значення
якого p1}
```

```
   end;
  end;                                  {кінець визначенню періода}

  for i:=1 to n-1 do                    {знаходження найбільшого
спільного}
   for j:=i+1 to n do                   {кратного для значень масиву A[1..n]}
   begin                                {сам цей процес детально описаний
вище}
    a[j]:=a[j]/NSD(a[i],a[j]);
   end;
  ss:=1;
  for i:=1 to n do      {значення ss і буде показувати мінімальну кількість діб,
необхідних для}
     ss:=ss*a[i];     {того, щоб початкове положення кульок в черзі повторилося}
     Вивід результату;
end.
```

Охорона

Дано N (1≤N≤100) пронумерованих об’єктів, що охороняються і з’єднані K шляхами (1≤K≤4950). З будь-якого об’єкта на будь-який інший можна проїхати шляхами. Рух кожним шляхом можливий в обидва боки. Необхідно розташувати підрозділ так, щоб найвіддаленіший об’єкт досягався якомога швидше.

Завдання: Напишіть програму, що знаходить найкраще місце для підрозділу охорони. Підрозділ може знаходитися як на одному з об’єктів так і на шляху між об’єктами.

Розв’язання:

- Шукаємо найменші відстані між об’єктами і записуємо їх в масив VD[1..N,1..N].
- Знаходимо для кожного об’єкту номер найвіддаленішого об’єкту і записуємо це в масив NVD[1..N] (для і-того об’єкту найвіддаленішим буде вершина з номером NVD[i]).
- І нарешті знаходження потрібного місця для підрозділу охорони.

Це робиться слідуючим чином. Розглядаємо пари об’єктів (i,j), які з’єднані між собою дорогою. Можливі чотири випадки:

1а) Відстань до найдальшого об’єкту для і-того об’єкту проходить через j-тий об’єкт.

1б) Відстань до найдальшого об’єкту для і-того об’єкту не проходить через j-тий об’єкт.

2а) Відстань до найдальшого об’єкту для j-того об’єкту проходить через і-тий об’єкт.

2б) Відстань до найдальшого об’єкту для j-того об’єкту не проходить через і-тий об’єкт.

Тоді можливі варіанти:

1. Коли виконується 1а) і 2б), то найкраще місце для підрозділу охорони серед цих двох об’єктів буде на j-тому об’єкті.
2. Коли виконується 1б) і 2а) – то на і-тому об’єкті.

3. Коли виконується 1б) і 2б) – то там (на і-тому чи j-тому), де найменша відстань до найвіддаленішого об’єкту.
4. Коли виконується 1а) і 2а), то найкраще місце для підрозділу охорони серед цих двох об’єктів буде міститися на дорозі між і-тим та j-тим об’єктом. Знайдемо спочатку d1=(vd[i,nvd[i]]+vd[j,nvd[j]]-vd[i,j])/2. Тоді початкове місце для підрозділу буде на d1-nvd[i] кілометрі від об’єкту і до об’єкту j. Потім, пробираючи відстані від цього положення до всіх об’єктів, ми коректуємо дане положення і таким чином знаходимо найкраще місце для охорони.

Пробираючи всі можливі пари об’єктів (i,j), які з’єднані шляхом ми і знайдемо місце від якого до найвіддаленішого об’єкту відстань буде найменша.

Рішення задачі на мові Паскаль:

```
Var                                                    {описання змінних}
 a,vd : array[1..4,1..4] of integer;
 b,nvd : array[1..4] of integer;
 c : array[1..4] of boolean;
 l,m,x,z,zn,n,p,i,i1,j,k,h : integer;
 r,r1,d,d1 : real;
 t : boolean;

function MIN(g1,g2 : integer) : integer;          {функція, яка знаходить номер
одного}
begin                                             {з двух об’єктів для якого відстань}
  if vd[g1,nvd[g1]]>vd[g2,nvd[g2]]                 {до найдальшого буде найменшою}
   then MIN:=g2
   else MIN:=g1;
end;

procedure STEP;                          {функція за допомогою якої робиться
повний перебор}
begin                                {сам метод перебору описаний в попередній задачі}
  x:=b[z+1]+1;
   while (a[b[z],x]=0) or (b[z]=x) or c[x] do
    begin
       x:=x+1;
       if x>n then break;
    end;
      if x<=n
      then
        begin
          z:=z+1;
          b[z]:=x;
          c[x]:=true;
          if (vd[p,x]=0) or (vd[p,x]>vd[p,b[z-1]]+a[b[z-1],x])
            then vd[p,x]:=vd[p,b[z-1]]+a[b[z-1],x];
        end
      else
```

```
        begin
          c[b[z]]:=false;
          b[z+1]:=0;
          z:=z-1;
        end;
end;

begin
Ввід даних;
  p:=0;     {початок блоку знаходження найменшої відстані між об'єктами}
  while true do
  begin
    p:=p+1;                    {знаходження здійснюється за допомогою методу}
    z:=1;                      {повного перебора описаного вище}
    b[z]:=p;
    c[p]:=true;
    if p>n then
      break;
    while z>0 do
     STEP;
  end;                  {кінець блоку знаходження найменшої відстані між
об'єктами}

  for i:=1 to n do              {початок блоку знаходження номеру
найвіддаленішого об'єкту}
    nvd[i]:=1;                        {для кожного об'єкту}

  for i:=1 to n do
  begin
    for j:=1 to n do
    begin
      if vd[i,nvd[i]]<vd[i,j]
       then nvd[i]:=j;
    end;
  end;                          {кінець блоку знаходження найвіддаленішого
об'єкту}

d:=vd[1,nvd[1]]+1;     {Далі йде знаходить найкращого місця для підрозділу
охорони,}
  for i:=1 to n-1 do              {маючи найменші відстані між об'єктами та}
  begin                                   {номери найвіддаленіших об'єктів}
    for j:=i+1 to n do
    begin
      if (a[i,j]>0) and (vd[i,j]=a[i,j]) then
      begin
        if (vd[i,nvd[i]]=vd[j,nvd[i]]+vd[i,j]) and (vd[j,nvd[j]]<>vd[i,nvd[j]]+vd[j,i]) and
(vd[j,nvd[j]]<=d) then
        begin                                                    {виконується 1а)
та 2б)}
```

```
        l:=j;
        m:=j;              {найкраще місце для підрозділу охорони буде на j-тому
об'єкті}
        d:=vd[j,nvd[j]];
        r:=0;
      end;
      if (vd[i,nvd[i]]<>vd[j,nvd[i]]+vd[i,j]) and (vd[j,nvd[j]]=vd[i,nvd[j]]+vd[j,i]) and
(vd[i,nvd[i]]<=d) then
      begin                                                {виконується 1б) та 2а)}
        l:=i;
        m:=i;              {найкраще місце для підрозділу охорони буде в і-тому
об'єкті}
        d:=vd[i,nvd[i]];
        r:=0;
      end;
      t:=(vd[j,nvd[j]]<>vd[i,nvd[j]]+vd[j,i]) and (vd[MIN(i,j),nvd[MIN(i,j)]]<=d);
      if (vd[i,nvd[i]]<>vd[j,nvd[i]]+vd[i,j]) and (t) then
      begin                                                {виконується 1б) та 2б)}
       h:=MIN(i,j);
       l:=h;
       m:=h;     {найкраще місце для підрозділу охорони буде або на і-тому або
на j-тому об'єкті,}
       d:=vd[h,nvd[h]]; {тобто там від якої найменша відстань до
найвіддаленішої}
       r:=0;
      end;
      if (vd[i,nvd[i]]=vd[j,nvd[i]]+vd[i,j]) and (vd[j,nvd[j]]=vd[i,nvd[j]]+vd[j,i]) then
      begin                                                {виконується 1а) та
2а)}
       d1:=(vd[i,nvd[i]]+vd[j,nvd[j]]-vd[i,j])/2;
       if d1<d then
       begin
       r1:=vd[i,nvd[i]];
{найкраще місце для підрозділу охорони буде міститися на дорозі між і-тим та j-
тим об'єктами. Знайдемо спочатку d1=(vd[i,nvd[i]]+vd[j,nvd[j]]-vd[i,j])/2. Тоді
початкове місце дляпідрозділу буде на d1-nvd[i] кілометрі від об'єкту з номером
і до об'єкту з номером j}
       for  i1:=1 to n do                                  {коректування знайденого
положення}
       begin
          if (i1<>i)and(i1<>j) then
          begin
             if vd[i1,i]+r1<vd[i1,j]+vd[i,j]-r1
              then zn:=-1
              else zn:=1;
             if (zn=-1)and(vd[i1,i]+r1>d1) then
             begin
                d1:=(vd[i1,i]+r1+d1)/2;
                r1:=d1-vd[i1,i];
```

```
                end;
                if (zn=1)and(vd[i1,j]+vd[i,j]-r1>d1) then
                begin
                   d1:=(vd[i1,j]+vd[i,j]-r1+d1)/2;
                   r1:=vd[i,j]-(d1-vd[i1,j]);
                end;
             end;
          end;
          if d1<d then
          begin
            l:=i;
            m:=j;
            d:=d1;
            r:=r1;
          end;
        end;
       end;
     end;
   end;
  end;
  Вивід результату;
end.
```

Список використаної літератури:

1. Міца О.В. «Методичні вказівки до рішення задач підвищеної складності». Перша частина. Ужгород 1996.
2. Міца О.В. «Методичні вказівки до задач підвищеної складності на мові Паскаль». Друга частина. Ужгород 1997.

Міца О.В. Методичні вказівки до рішення задач підвищенної складності (на прикладі міської та державної олімпіад 1998 р.). Третя частина. Видрукувано на видавничому комплексі Ужгородського державного інституту інформатики, економіки і права. Відповідальний за випуск Василенко Е.Ю. Тираж - 50 екз.