МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

ДВНЗ «УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

Математичний факультет

Кафедра кібернетики і прикладної математики

ЛОГІЧНЕ ПРОГРАМУВАННЯ

Методичні рекомендації до вивчення курсу

УЖГОРОД – 2013

Логічне програмування: методичні рекомендації до вивчення курсу для студентів спеціальності «Прикладна математика» математичного факультету УжНУ / Розробник: М.М. Повідайчик. – Ужгород: Видавництво УжНУ «Говерла», 2013. – 63 с.

У методичних рекомендаціях до курсу «Логічне програмування» розглянуто основні підходи до написання програм на мові Пролог, приведено приклади розв'язування задач як базового, так і підвищеного рівня складності. Розглянуті приклади можуть бути основою для розробки студентами власних програмних проектів.

Розробник: Повідайчик М.М., к.е.н., доцент кафедри кібернетики і прикладної математики математичного факультету УжНУ

Рецензенти:

- Головач Й.Г., д.т.н., професор кафедри кібернетики і прикладної математики математичного факультету УжНУ;
- Брила А.Ю., к.ф.-м.н., доцент кафедри системного аналізу і теорії оптимізації математичного факультету УжНУ

Рекомендовано кафедрою кібернетики і прикладної математики.
Протокол № 3 від 30 листопада 2012 року.

Рекомендовано Вченою радою математичного факультету.
Протокол №4 від 24 грудня 2012 року.

ЗМІСТ

ВСТУП

«Логічне програмування» є вибірковою дисципліною для студентів спеціальності «Прикладна математика», що читається у 5 семестрі у обсязі 3 кредитів, в тому числі 54 аудиторних годин (36 лекційних та 18 практичних год.) та 36 годин самостійної роботи.

Мета вивчення дисципліни «Логічне програмування» – ознайомлення студентів з основами логічного програмування, опанування мови програмування Пролог, моделювання деяких задач штучного інтелекту за допомогою мови Пролог.

Завдання дисципліни «Логічне програмування» полягають у формуванні у студентів знань, умінь та навичок розробки алгоритмів евристичного пошуку, експертних систем, моделювання деяких задач штучного інтелекту та ін.

В результаті вивчення даного курсу студент повинен

- знати: синтаксис і семантику пролог-програм, операції над структурами даних у пролог-програмах, теоретичні основи моделювання задач штучного інтелекту;
- вміти: програмувати та розробляти бази знань на мові Пролог.

ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Змістовий модуль 1. Мова програмування Пролог.

Тема 1. Вступ до мови логічного програмування Пролог. (2 год.)

Історія виникнення і розвитку Прологу. Японський проект ЕОМ п’ятого покоління. Імперативні та декларативні мови програмування. Галузі використання Прологу. Переваги та недоліки мови Пролог.

Тема 2. Логічні основи Прологу. (2 год.)

Хорновські диз’юнкти. Принцип резолюцій. Алгоритм уніфікації. Процедура доведення теорем методом резолюцій для Хорновських диз’юнктів. Особливості роботи з негативними даними в Пролозі.

Тема 3. Основні поняття Прологу. (2 год.)

Факти і правила. Внутрішні і зовнішні цілі. Предикати. Вільні і пов’язані змінні. Анонімна змінна. Відсікання. «Зелені» і «червоні» відсікання. Семантичні моделі Прологу: декларативна і процедурна.

Тема 4. Рекурсія. (2 год.)

Рекурсія. Переваги і недоліки рекурсії. Хвостова рекурсія. Організація циклів на основі рекурсії. Обчислення факторіала. Числа Фібоначчі.

Тема 5. Основи Visual Prolog. (2 год.)

Структура програми на Visual Prolog. Директиви компілятора. Домени: стандартні, спискові, складні. Альтернативні домени.

Тема 6. Вбудовані предикати. (2 год.)

Предикати вводу/виводу. Предикати перетворення типів даних.

Тема 7. Керування виконанням програми на Пролозі. (2 год.)

Метод пошуку в глибину. Відкат після невдачі. Відсікання і відкат. Метод пошуку, визначений користувачем.

Тема 8. Списки. Операції над списками. (2 год.)

Списки. Рекурсивне визначення списку. Операції над списками. Предикат `findall`. Списки списків.

Тема 9. Сортування списків. (2 год.)

Алгоритми сортування списків: бульбашковий, вибором, вставкою, злиттям, швидке сортування.

Тема 10. Множини. (2 год.)

Реалізація множин в Пролозі. Операції над множинами: представлення множини списком, приналежність елемента множині, об'єднання, перетин, різниця, доповнення множин.

Тема 11. Символьні дані. (2 год.)

Вбудовані предикати опрацювання символьних даних. Перетворення рядка символів у список.

Тема 12. Дерева. (2 год.)

Бінарні дерева, двійкові довідники та операції над ними.

Тема 13. Файли. (2 год.)

Опис файлового домену. Вбудовані предикати для роботи з файлами. Запис інформації у файл. Читання інформації з файлу. Переписування даних з файлу у файл.

Тема 14. Динамічні бази даних. (2 год.)

Робота з динамічними базами даних: додавання фактів у базу, видалення фактів з бази. Проекти «Телефонний довідник», «Словник».

Тема 15. Евристичний пошук. (2 год.)

Пошук у «глибину». Пошук у «ширину». Евристичний пошук.

Тема 16. Програмування логічних задач. (2 год.)

Проекти «Вовк, коза, капуста», «8 королев», «Задача Ейнштейна».

Тема 17. Пролог і штучний інтелект. (2 год.)

Застосування Прологу в галузі штучного інтелекту. Тест Тюрінга. Проект «Електронний психотерапевт».

Тема 18. Експертні системи. (2 год.)

Загальна схема експертної системи. Оболонка експертної системи. Класифікація експертних систем. Проект «Визначення тварини».

І. ЗАВДАННЯ БАЗОВОГО РІВНЯ СКЛАДНОСТІ

1.1. Факти та правила.

Завдання 1.1. На рис. 1 зображено зв'язки у деякій родині. Описати зв'язки між членами родини фактами `батько` та `одружений`. Написати правила для відношень `жіноча стать`, `чоловіча стать`, `мати`, `брат`, `дід`.

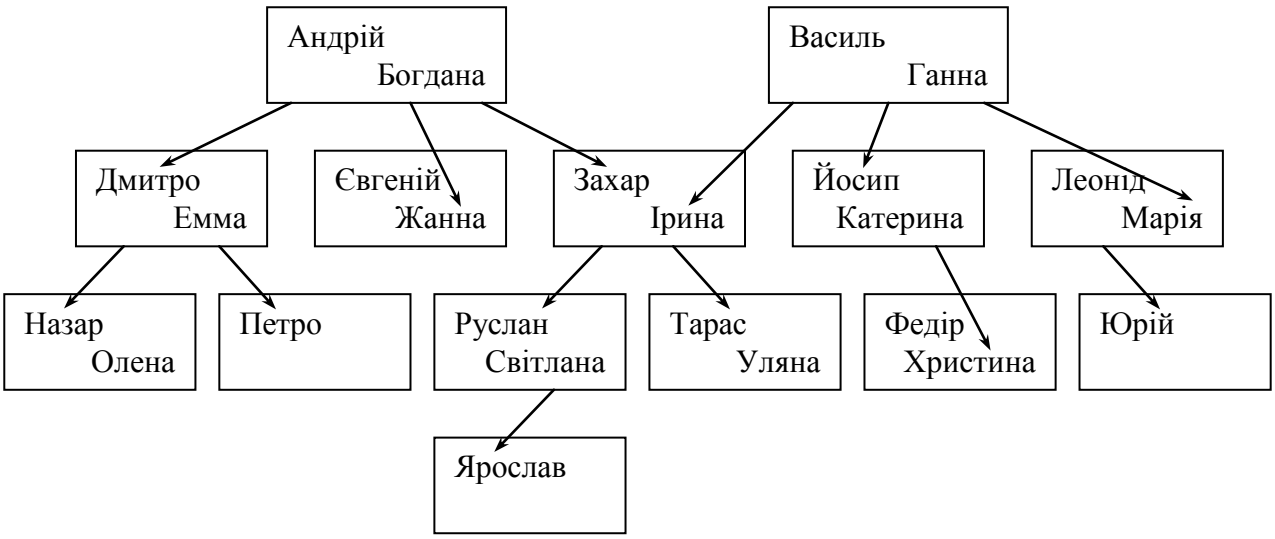

Рис 1. Задача «Родина»

Розв'язання. Для описання зв'язків між членами родини використовуються факти `батько(Батько,Дитина)` та `одружений(Чоловік,Дружина)`.

Відношення `жіноча стать` реалізується предикатом `жін_стать(Жінка)`.

```
жін_стать(Жінка):-
   str_len(Жінка,N),        % Кількість букв у імені Жінки
   subchar(Жінка,N,Буква), % Остання буква імені Жінки
   жін_буква(Буква).

жін_буква('а').
жін_буква('я').
```

Вбудовані предикати `str_len(Жінка,N)` та `subchar(Жінка,N,Буква)` знаходять кількість букв та останню букву параметра `Жінка`. Якщо це буква «а» або «я» (відношення `жін_буква`), то предикат виконується успішно.

Приклад.

```
goal жін_стать(клеопатра).
```

Результат: `yes`

Відношення `чоловіча стать` реалізується предикатом `чол_стать(Чоловік)` (`not` – логічне заперечення).

```
чол_стать(Чоловік):-
   not(жін_стать(Чоловік)).
```

Приклад.

```
goal чол_стать(федір).
```

Результат: `yes`

Відношення мати задається таким правилом: *A* буде матір'ю для *B*, якщо *A* має чоловіка *C* і *C* – батько для *B*.

```
мати(Мати,Дитина):-
   одружений(Батько,Мати),
   батько(Батько,Дитина).
```

Приклад.

```
goal мати(ірина,Дитина).
```
Результат:
```
Дитина=руслан
Дитина=тарас
```

Приклад.

```
goal мати(Мати,ірина).
```
Результат: `Мати=ганна`

Відношення брат задається таким правилом: *A* буде братом для *B*, якщо *A* і *B* мають батька *C*, *A* і *B* різні, *A* – чоловічої статі.

```
брат(Чоловік,Дехто):-
   батько(Батько,Чоловік),
   батько(Батько,Дехто),
   not(Чоловік=Дехто),
   чол_стать(Чоловік).
```

Приклад.

```
goal брат(юрій,Брат_або_сестра).
```
Результат: `No Solution`

Приклад.

```
goal брат(дмитро,Брат_або_сестра).
```
Результат:
```
Брат_або_сестра=жанна
Брат_або_сестра=захар
```

Приклад.

```
goal брат(Брат,дмитро).
```
Результат: `Брат=захар`

Відношення дід задається такими правилами (процедурою):

Правило 1. *A* буде дідом для *B*, якщо *B* має батька *C*, а батько для *C* – *A*.

Правило 2. Або *A* буде дідом для *B*, якщо *B* має матір *C*, а батько для *C* – *A*.

```
дід(Дід,Онук):-
   батько(Батько,Онук),
   батько(Дід,Батько).
дід(Дід,Онук):-
   батько(Батько,Онук),
   одружений(Батько,Мати),
   батько(Дід,Мати).
```

Приклад.

```
goal дід(андрій,Онук).
```

Результат: Онук=назар
Онук=петро
Онук=руслан
Онук=тарас

Приклад.

```
goal дід(Дід,тарас).
```
Результат: Дід=андрій
Дід=василь

Лістинг 1.1. Факти та правила.

```
DATABASE
батько(symbol батько,symbol дитина)
одружений(symbol чоловік,symbol дружина)
CLAUSES
батько(андрій,дмитро).      батько(андрій,жанна).
батько(андрій,захар).       батько(василь,ірина).
батько(василь,йосип).       батько(василь,марія).
батько(дмитро,назар).       батько(дмитро,петро).
батько(захар,руслан).       батько(захар,тарас).
батько(йосип,христина).     батько(леонід,юрій).
батько(руслан,ярослав).
одружений(андрій,богдана).  одружений(василь,ганна).
одружений(дмитро,емма).     одружений(євгеній,жанна).
одружений(захар,ірина).     одружений(йосип,катерина).
одружений(леонід,марія).    одружений(назар,олена).
одружений(руслан,світлана). одружений(тарас,уляна).
одружений(федір,христина).

PREDICATES
жін_стать(symbol жінка)
жін_буква(char буква_а_або_я)
чол_стать(symbol чоловік)
CLAUSES
жін_буква('а').
жін_буква('я').

жін_стать(Жінка):-
   str_len(Жінка,N),        % Кількість букв у імені Жінки
   subchar(Жінка,N,Буква), % Остання буква імені Жінки
   жін_буква(Буква).

чол_стать(Чоловік):-
   not(жін_стать(Чоловік)).

PREDICATES
nondeterm мати(symbol мати,symbol дитина)
nondeterm брат(symbol брат,symbol брат_або_сестра)
nondeterm дід(symbol дід,symbol онук_або_онука)
```

```
CLAUSES
мати(Мати,Дитина):-
    одружений(Батько,Мати),
    батько(Батько,Дитина).

брат(Чоловік,Дехто):-
    батько(Батько,Чоловік),
    батько(Батько,Дехто),
    not(Чоловік=Дехто),
    чол_стать(Чоловік).

дід(Дід,Онук):-
    батько(Батько,Онук),
    батько(Дід,Батько).
дід(Дід,Онук):-
    батько(Батько,Онук),
    одружений(Батько,Мати),
    батько(Дід,Мати).

GOAL
    дід(Дід,тарас).
```

Завдання для самостійної роботи.

1. Описати відношення сестра, двоюрідний брат, племінник.

1.2. Рекурсивне визначення предикатів.

Завдання 1.2. Дано натуральне `Число`. Визначити такі відношення: `просте(Число)` (має рівно 2 дільники), `складене(Число)` (має більше 2-х дільників), `досконале(Число)` (сума всіх дільників, що менші за число, дорівнює числу), `надлишкове(Число)` (сума всіх дільників, що менші за число, більша за число), `недостатнє(Число)` (сума всіх дільників, що менші за число, менша за число).

Розв'язання. Для визначення предиката `просте(Число)` використовується відношення `має_дільник(Число,Дільник)`, яке перевіряє, чи має `Число` дільники у діапазоні від 2 до `Дільник`.

```
має_дільник(Число,Дільник):-
   Дільник>1,
   Число mod Дільник = 0,!.
має_дільник(Число,Дільник):-
   Дільник>2,
   Дільник1=Дільник-1,
   має_дільник(Число,Дільник1).
```

Правило 1. `Число` має `Дільник`, якщо `Дільник` більший за 1 і `Число` ділиться на `Дільник` без остачі.

Правило 2. Інакше – рекурсивно перевірити для дільника (`Дільник` – 1).

Приклад. Перевірити, чи має число 23 дільники у діапазоні від 2 до 11.

```
goal має_дільник(23,11).
```
Результат: `no`

Предикат `просте(Число)` (`Число` > 1) виконується успішно, якщо `Число` не має дільників у діапазоні від 2 до [`Число/2`] (тут [x] – ціла частина від x).

```
просте(Число):-
   Число>1,
   Дільник=Число div 2,
   not(має_дільник(Число,Дільник)).
```

Приклад. Перевірити, чи просте число 23.

```
goal просте(23).
```
Результат: `yes`

Предикат `складене(Число)` (`Число` > 1) виконується успішно, якщо `Число` не просте.

```
складене(Число):-
   Число>1,
   not(просте(Число)).
```

Приклад. Перевірити, чи складене число 24.

```
goal складене(24).
```
Результат: `yes`

Для визначення предиката `досконале(Число)` використовується відношення `сума_дільників(Число,Дільник,Сума)`, яке шукає суму дільників у діапазоні від 1 до `Дільник`.

```
сума_дільників(_,1,1):-!.
сума_дільників(Число,Дільник,Сума):-
   Дільник>1,
   Число mod Дільник = 0,
   Дільник1=Дільник-1,
   сума_дільників(Число,Дільник1,Сума1),
   Сума=Сума1+Дільник,!.
сума_дільників(Число,Дільник,Сума):-
   Дільник>1,
   Число mod Дільник <> 0,
   Дільник1=Дільник-1,
   сума_дільників(Число,Дільник1,Сума),!.
```

Правило 1. Якщо `Дільник` = 1, то `Сума` = 1.

Правило 2. Якщо `Дільник` > 1 і `Число` ділиться на `Дільник` без остачі, то рекурсивно знайти суму дільників у діапазоні від 1 до `Дільник – 1` і до результату додати `Дільник`.

Правило 3. Інакше – до результату не додавати `Дільник`.

Приклад. Знайти суму дільників числа 24 у діапазоні від 1 до 12.

```
goal сума_дільників(24,12,Сума).
```

Результат: `Сума=36`

Предикат `досконале(Число)` (`Число` > 1) виконується успішно, якщо сума дільників у діапазоні від 1 до [`Число/2`] рівна `Числу`.

```
досконале(Число):-Число>1,
   Дільник=Число div 2,
   сума_дільників(Число,Дільник,Сума),
   Число=Сума.
```

Приклад. Перевірити, чи досконале число 28.

```
goal досконале(28).
```

Результат: `yes`

Аналогічно визначається предикат `надлишкове(Число)`.

```
надлишкове(Число):-
   Число>1,
   Дільник=Число div 2,
   сума_дільників(Число,Дільник,Сума),
   Число<Сума.
```

Приклад. Перевірити, чи надлишкове число 12.

```
goal надлишкове(12).
```

Результат: `yes`

Предикат `недостатнє(Число)` визначає додатне число як не досконале і не надлишкове.

```
недостатнє(Число):-
   Число>0,
   not(досконале(Число)),
   not(надлишкове(Число)).
```

Приклад. Перевірити, чи недостатнє число 10.

```
goal недостатнє(10).
```

Результат: yes

Лістинг 1.2. Рекурсивне визначення предикатів.

```
PREDICATES
просте(integer число)
складене(integer число)
має_дільник(integer число,integer дільник_більший_1)
досконале(integer число)
сума_дільників(integer число,integer дільник,integer сума)
надлишкове(integer число)
недостатнє(integer число)

CLAUSES

просте(Число):-
   Число>1,
   Дільник=Число div 2,
   not(має_дільник(Число,Дільник)).

складене(Число):-
   Число>1,
   not(просте(Число)).

має_дільник(Число,Дільник):-
   Дільник>1,
   Число mod Дільник = 0,!.
має_дільник(Число,Дільник):-
   Дільник>2,
   Дільник1=Дільник-1,
   має_дільник(Число,Дільник1).

досконале(Число):-
   Число>1,
   Дільник=Число div 2,
   сума_дільників(Число,Дільник,Сума),
   Число=Сума.

сума_дільників(_,1,1):-!.
сума_дільників(Число,Дільник,Сума):-
   Дільник>1,
   Число mod Дільник = 0,
   Дільник1=Дільник-1,
   сума_дільників(Число,Дільник1,Сума1),
   Сума=Сума1+Дільник,!.
сума_дільників(Число,Дільник,Сума):-
   Дільник>1,
   Число mod Дільник <> 0,
```

```
    Дільник1=Дільник-1,
    сума_дільників(Число,Дільник1,Сума),!.

надлишкове(Число):-
    Число>1,
    Дільник=Число div 2,
    сума_дільників(Число,Дільник,Сума),
    Число<Сума.

недостатнє(Число):-
    Число>0,
    not(досконале(Число)),
    not(надлишкове(Число)).

GOAL
    просте(31).
```

Завдання для самостійної роботи.

1. Описати відношення дружні числа (наприклад, 220 і 284) – числа, у яких сума дільників одного рівна іншому числу (враховуються тільки дільники, які менші за число).
2. Рекурсивно описати знаходження *n*!.
3. У завданні 1.1 описати відношення пращур та нащадок.

1.3. Деякі задачі зі списками.

Завдання 1.3. Дано список цілих чисел. Описати відношення, що друкують елементи списку на екрані, знаходять суму, кількість та мінімальний елемент списку, видаляють елемент списку та виводять всі перестановки елементів.

Розв'язання. Для описання списку цілих чисел використовується домен `сп=integer*`.

1) Надрукувати елементи списку:

Правило 1. Якщо список порожній, то завершити (вбудований предикат `nl` – перехід на новий рядок).

Правило 2. Якщо список розділяється на голову і хвіст, то надрукувати голову (вбудований предикат `write`), а далі надрукувати рекурсивно хвіст.

```
надрукувати([]):-nl,!.
надрукувати([Гол|Хв]):-
   write(Гол),надрукувати(Хв),!.
```

Приклад. Надрукувати список [1,2,3,4,5].

```
goal надрукувати([1,2,3,4,5]).
```
Результат: `12345`

2) Знайти суму елементів списку:

Правило 1. Якщо список порожній, то сума рівна 0.

Правило 2. Якщо список розділяється на голову і хвіст, то рекурсивно знаходиться сума елементів хвоста і додається до результату голова.

```
сума([],0):-!.
сума([Гол|Хв],Сума):-
   сума(Хв,Сума1),
   Сума=Гол+Сума1,!.
```

Приклад. Знайти суму елементів списку [1,2,3,4,5].

```
goal сума([1,2,3,4,5],Сума).
```
Результат: `Сума=15`

3) Знайти кількість елементів списку:

Правило 1. Якщо список порожній, то кількість рівна 0.

Правило 2. Якщо список розділяється на голову і хвіст, то рекурсивно знаходиться кількість елементів хвоста і додається до результату 1.

```
кількість([],0):-!.
кількість([_|Хв],К):-
   кількість(Хв,К1),
   К=К1+1,!.
```

Приклад. Знайти кількість елементів списку [3, –1,5, –3,7].

```
goal кількість([3,-1,5,-3,7],Кільк).
```
Результат: `Кільк=5`

4) Знайти мінімальний елемент списку:

Правило 1. Якщо список містить один елемент `Ел`, то `Ел` – мінімальний елемент.

Правило 2. Якщо список розділяється на голову і хвіст, а мінімальний елемент хвоста більший за голову, то голова – мінімальний елемент.

Правило 3. Інакше – результатом є мінімальний елемент хвоста.

```
мінімум([Ел],Ел):-!.
мінімум([Гол|Хв],Гол):-
   мінімум(Хв,Мін),
   Гол<Мін,!.
мінімум([_|Хв],Мін):-
   мінімум(Хв,Мін),!.
```

Приклад. Знайти мінімальний елемент списку [3, –1,5, –3,7].

```
goal мінімум([3,-1,5,-3,7],Мін).
```
Результат: `Мін=-3`

5) Видалити елемент списку:

Правило 1. Якщо список розділяється на голову та хвіст і елемент, що видаляється, співпадає з головою, то хвіст – результат.

Правило 2. Видалити елемент з хвоста та до результату додати голову.

```
видалити(Гол,[Гол|Хв],Хв).
видалити(Ел,[Гол|Хв],[Гол|Хв1]):-
   видалити(Ел,Хв,Хв1).
```

Приклад. Видалити 2 зі списку [1,2,3].

```
goal видалити(2,[1,2,3],Сп).
```
Результат: `Сп=[1,3]`

Оскільки предикат `Видалити` недетермінований, то його можна використовувати для виведення всіх елементів списку.

Приклад. Вивести елементи списку [1,2,3].

```
goal видалити(X,[1,2,3],Сп).
```
Результат: `X=1, Сп=[2,3]`
`X=2, Сп=[1,3]`
`X=3, Сп=[1,2]`

6) Вивести всі перестановки елементів списку:

Правило 1. Якщо список порожній, то – завершити.

Правило 2. Видалити елемент зі списку та зробити перестановку елементів, що залишилися, до результату добавити елемент, що був видалений.

```
перестановка([],[]).
перестановка(Сп,[Гол|Хв]) :-
   видалити(Гол,Сп,Сп1),
   перестановка(Сп1,Хв).
```

Приклад. Вивести перестановку елементів списку [1,2,3].

```
goal перестановка([1,2,3],Сп).
```
Результат: `Сп=[1,2,3]`
`Сп=[1,3,2]`
`Сп=[2,1,3]`
`Сп=[2,3,1]`
`Сп=[3,1,2]`
`Сп=[3,2,1]`

Лістинг 1.3. Деякі задачі зі списками

```
DOMAINS
сп=integer*
PREDICATES
надрукувати(сп)
сума(сп,integer)
кількість(сп,integer)
мінімум(сп,integer)
nondeterm видалити(integer,сп,сп)
nondeterm перестановка(сп,сп)
CLAUSES
надрукувати([]):-nl,!.
надрукувати([Гол|Хв]):-
   write(Гол),надрукувати(Хв),!.

сума([],0):-!.
сума([Гол|Хв],Сума):-
   сума(Хв,Сума1),
   Сума=Гол+Сума1,!.

кількість([],0):-!.
кількість([_|Хв],К):-
   кількість(Хв,К1),
   К=К1+1,!.

мінімум([Ел],Ел):-!.
мінімум([Гол|Хв],Гол):-
   мінімум(Хв,Мін),
   Гол<Мін,!.
мінімум([_|Хв],Мін):-
   мінімум(Хв,Мін),!.

видалити(Гол,[Гол|Хв],Хв).
видалити(Ел,[Гол|Хв],[Гол|Хв1]):-
   видалити(Ел,Хв,Хв1).

перестановка([],[]).
перестановка(Сп,[Гол|Хв]) :-
   видалити(Гол,Сп,Сп1),
   перестановка(Сп1,Хв).
GOAL
   надрукувати([1,2,3,4,5]).
```

Завдання для самостійної роботи.

1. Знайти суму додатних елементів списку.
2. Знайти кількість елементів списку з інтервалу (*a*, *b*).
3. Знайти найбільший від'ємний елемент списку.

1.4. Робота із символьними даними.

Завдання 1.4. Для заданого речення (рядка символів):

1) сформувати список слів;
2) знайти кількість голосних літер;
3) знайти найдовше слово;
4) зробити найпростіший (дослівний) переклад (з російської мови на українську).

Розв'язання.

1) Побудову списку слів реалізовано відношенням `список_слів(Рядок,Список)`, де `Рядок` – вхідний параметр, `Список` – вихідний:

Правило 1. Якщо `Рядок` порожній, то `Список` також буде порожній.

Правило 2. Якщо з `Рядка` можна виділити перше `Слово` (використовується вбудований предикат `fronttoken(Рядок,Слово,Остаток)`, який `Рядок` розділяє на перше `Слово` і `Остаток`), то будується `Список` слів з `Остатку`, а потім до результату додається перше `Слово`.

```
список_слів("",[]):-!.
список_слів(Рядок,[Слово|Список]):-
   fronttoken(Рядок,Слово,Остаток),
   список_слів(Остаток,Список),!.
```

Приклад. Задано `Речення` = "це речення перетворимо у список". Сформувати з нього `Список` слів.

```
goal Речення="це речення перетворимо у список",
     список_слів(Речення,Список).
```

Результат: `Список=["це","речення","перетворимо","у","список"]`

2) Знаходження кількості голосних літер реалізовано відношенням `кількість_голосних(Рядок,Кількість)`, де `Рядок` – вхідний параметр, `Кількість` – вихідний:

Правило 1. Якщо `Рядок` порожній, то `Кількість` рівна 0.

Правило 2. Якщо перший символ `Рядка` голосна літера (використовується вбудований предикат `frontchar(Рядок,Символ,Остаток)`, який `Рядок` розділяє на перший `Символ` і `Остаток`, а також предикат `належить(Символ,['а','о','у','і','и','е'])`, який перевіряє належність `Символа` списку голосних літер (див. 1.6)), то рекурсивно знаходиться кількість голосних літер у `Остатку`, а до результату додається 1.

Правило 3. Інакше – знаходиться кількість голосних літер у `Остатку`.

```
кількість_голосних("",0):-!.
кількість_голосних(Рядок,Кількість):-
   frontchar(Рядок,Символ,Остаток),
   належить(Символ,['а','о','у','і','и','е']),
   кількість_голосних(Остаток,Кількість1),
   Кількість=Кількість1+1,!.
кількість_голосних(Рядок,Кількість):-
   frontchar(Рядок,_,Остаток),
   кількість_голосних(Остаток,Кількість),!.

   належить(Ел,[Ел|_]):-!.
   належить(Ел,[_|Хв]):-
      належить(Ел,Хв),!.
```

Приклад. У Реченні з попереднього прикладу знайти кількість голосних літер.

```
goal Речення="це речення перетворимо у список",
     кількість_голосних(Речення,Кількість).
```

Результат: Кількість=11

3) Знаходження найдовшого слова речення реалізовано відношенням найдовше_слово(Рядок,Слово), яке для вхідного параметра Рядок будує Список слів, а потім предикатом найдовше_слово_списку(Список,Слово) знаходиться відповідне Слово:

Правило 1. Якщо Список, складається з одного Слова, то це Слово і є результатом.

Правило 2. Якщо список розділяється на Голову і Хвіст, то у Хвості рекурсивно знаходиться найдовше Слово. Якщо довжина цього Слова менша за довжину Голови (використовується вбудований предикат str_len), то результатом буде Голова.

Правило 3. Інакше – результатом буде найдовше Слово у Хвості.

```
найдовше_слово(Рядок,Слово):-
   список_слів(Рядок,Список),
   найдовше_слово_списку(Список,Слово),!.

   найдовше_слово_списку([Слово],Слово):-!.
   найдовше_слово_списку([Голова|Хвіст],Голова):-
      найдовше_слово_списку(Хвіст,Слово),
      str_len(Голова,Д1),
      str_len(Слово,Д2),
      Д1>Д2,!.
   найдовше_слово_списку([_|Хвіст],Слово):-
      найдовше_слово_списку(Хвіст,Слово),!.
```

Приклад. У Реченні з попереднього прикладу знайти найдовше слово.

```
goal Речення="це речення перетворимо у список",
     найдовше_слово(Рядок,Слово).
```

Результат: Слово=перетворимо

4) Дослівний переклад реалізовано відношенням перекласти_речення, який використовує факти динамічної бази даних переклад:

Правило 1. Якщо вхідний параметр – порожній рядок, то вихідний параметр – також порожній рядок.

Правило 2. Якщо з Речення, що перекладається, можна виділити Слово, яке наявне у базі даних, то рекурсивно перекладається Остаток речення, а потім до результату добавляється переклад Слова.

Правило 3. Інакше (якщо Слово не задано у базі даних) – Слово добавляється до результату без перекладу.

```
перекласти_речення("","",):-!.
перекласти_речення(Речення,Речення1):-
   fronttoken(Речення,Слово,Остаток),
   переклад(Слово,Слово1),
   перекласти_речення(Остаток,Остаток1),
   frontchar(Остаток2,' ',Остаток1),
   fronttoken(Речення1,Слово1,Остаток2),!.
```

```
перекласти_речення(Речення,Речення1):-
   fronttoken(Речення,Слово,Остаток),
   перекласти_речення(Остаток,Остаток1),
   frontchar(Остаток2,' ',Остаток1),
   fronttoken(Речення1,Слово,Остаток2),!.
```

Приклад. Перекласти з російської мови на українську речення: "Если дискриминант равен 0, то квадратное уравнение имеет 1 решение".

```
goal Речення="если дискриминант равен 0, то квадратное уравнение имеет 1 решение",
     перекласти_речення(Речення,Речення1).
```

Результат: `Речення1="якщо дискримінант рівний 0 , то квадратне рівняння має 1 розв'язок"`

Лістинг. 1.4. Робота із символьними даними.

```
DOMAINS
сп=string*
символи=char*

PREDICATES
список_слів(string,сп)
кількість_голосних(string,integer)
належить(char,символи)
найдовше_слово(string,string)
найдовше_слово_списку(сп,string)

CLAUSES
список_слів("",[]):-!.
список_слів(Рядок,[Слово|Список]):-
   fronttoken(Рядок,Слово,Остаток),
   список_слів(Остаток,Список),!.

кількість_голосних("",0):-!.
кількість_голосних(Рядок,Кількість):-
   frontchar(Рядок,Символ,Остаток),
   належить(Символ,['а','о','у','і','и','е']),
   кількість_голосних(Остаток,Кількість1),
   Кількість=Кількість1+1,!.
кількість_голосних(Рядок,Кількість):-
   frontchar(Рядок,_,Остаток),
   кількість_голосних(Остаток,Кількість),!.

   належить(Ел,[Ел|_]):-!.
   належить(Ел,[_|Хв]):-
      належить(Ел,Хв),!.

найдовше_слово(Рядок,Слово):-
   список_слів(Рядок,Список),
   найдовше_слово_списку(Список,Слово),!.
```

```
   найдовше_слово_списку([Слово],Слово):-!.
   найдовше_слово_списку([Голова|Хвіст],Голова):-
      найдовше_слово_списку(Хвіст,Слово),
      str_len(Голова,Д1),
      str_len(Слово,Д2),
      Д1>Д2,!.
   найдовше_слово_списку([_|Хвіст],Слово):-
      найдовше_слово_списку(Хвіст,Слово),!.

DATABASE
переклад(string,string)
predicates
перекласти_речення(string,string)

CLAUSES
переклад("если","якщо").
переклад("дискриминант","дискримінант").
переклад("равен","рівний").
переклад("квадратное","квадратне").
переклад("уравнение","рівняння").
переклад("имеет","має").
переклад("решение","розвязок").

перекласти_речення("","").-!.
перекласти_речення(Речення,Речення1):-
   fronttoken(Речення,Слово,Остаток),
   переклад(Слово,Слово1),
   перекласти_речення(Остаток,Остаток1),
   frontchar(Остаток2,' ',Остаток1),
   fronttoken(Речення1,Слово1,Остаток2),!.
перекласти_речення(Речення,Речення1):-
   fronttoken(Речення,Слово,Остаток),
   перекласти_речення(Остаток,Остаток1),
   frontchar(Остаток2,' ',Остаток1),
   fronttoken(Речення1,Слово,Остаток2),!.

GOAL
   Речення="це речення перетворимо у список",
   список_слів(Речення,Список).
```

Завдання для самостійної роботи.

1. Із даного речення (рядка символів) сформувати список слів, які мають подвоєння літер.
2. У реченні (рядку символів) зустрічаються цілі числа. Знайти найменше з них.
3. У завданні перекладу передбачити таке використання прийменників «у» або «в», яке б забезпечило чергування голосних і приголосних літер.

1.5. Структурні домени.

Завдання 1.5. Представити структурними доменами такі об'єкти як точка, відрізок, трикутник. Розв'язати такі задачі:

1) знайти середину відрізка;
2) знайти довжину відрізка;
3) знайти периметр трикутника;
4) знайти площу трикутника;
5) перевірити, чи трикутник прямокутний.

Розв'язання. Будемо використовувати такі структурні домени:

`тч=точ(real,real)` – точка двовимірного простору;

`вд=відр(тч,тч)` – відрізок, заданий двома точками;

`тр=трик(тч,тч,тч)` – трикутник, заданий трьома точками.

1) Точку $K(x_0, y_0)$, середину відрізка AB, де $A(x_1, y_1)$, $B(x_2, y_2)$, будемо знаходити за формулами: $x_0 = (x_1 + x_2)/2$, $y_0 = (y_1 + y_2)/2$.

Приклад. Знайти середину відрізка AB, де $A(-2, 3)$, $B(6, 5)$.

```
goal A=точ(-2,3),B=точ(6,5),
   V=відр(A,B),
   середина(V,K).
```

Результат: `K=точ(2,4)`

2) Довжину відрізка AB, де $A(x_1, y_1)$, $B(x_2, y_2)$, будемо знаходити за формулою: $|AB| = ((x_1 - x_2)^2 + (y_1 - y_2)^2)^{½}$.

Приклад. Знайти довжину відрізка AB, де $A(0, 0)$, $B(3, 4)$.

```
goal A=точ(0,0),B=точ(3,4),
   V=відр(A,B),
   довжина(V,L).
```

Результат: `L=5`

3) Периметр трикутника ABC, будемо знаходити за формулою: $P = |AB| + |AC| + |BC|$.

4) Площу трикутника ABC, будемо знаходити за формулою Герона: $S = (p(p - |AB|)(p - |AC|)(p - |BC|))^{½}$, де $p = P/2$.

Приклад. Знайти периметр та площу трикутника ABC, де $A(0, 0)$, $B(0, 4)$, $C(3, 0)$.

```
goal A=точ(0,0),B=точ(0,4),C=точ(3,0),
   T=трик(A,B,C), периметр(T,P), площа(T,S).
```

Результат: `P=12, S=6`

5) Для перевірки, чи трикутник ABC є прямокутним, будемо використовувати теорему, обернену до теореми Піфагора: якщо $|AB|^2 = |AC|^2 + |BC|^2$, то трикутник – прямокутний.

Приклад. Перевірити, чи трикутник ABC прямокутний, де $A(0, 0)$, $B(0, 4)$, $C(3, 0)$.

```
goal A=точ(0,0),B=точ(0,4),C=точ(3,0),
   T=трик(A,B,C), прямокутний(T).
```

Результат: `yes`

Лістинг 1.5. Структурні домени.

```
DOMAINS
   тч=точ(real,real)
   вд=відр(тч,тч)
   тр=трик(тч,тч,тч)
```

```
PREDICATES
   середина(вд,тч)
   довжина(вд,real)
   периметр(тр,real)
   площа(тр,real)
   прямокутний(тр)
   теорема_Піфагора(real,real,real)
CLAUSES
середина(відр(точ(X1,Y1),точ(X2,Y2)), точ(X0,Y0)):-
   X0=(X1+X2)/2,
   Y0=(Y1+Y2)/2,!.

довжина(відр(точ(X1,Y1),точ(X2,Y2)),L):-
   L=sqrt((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)),!.

периметр(трик(A,B,C),P):-
   довжина(відр(A,B),L1),
   довжина(відр(B,C),L2),
   довжина(відр(A,C),L3),
   P=L1+L2+L3,!.

площа(трик(A,B,C),S):-
   периметр(трик(A,B,C),P),
   довжина(відр(A,B),L1),
   довжина(відр(B,C),L2),
   довжина(відр(A,C),L3),
   Pp=P/2,
   S=sqrt(Pp*(Pp-L1)*(Pp-L2)*(Pp-L3)),!.

прямокутний(трик(A,B,C)):-
   довжина(відр(A,B),L1),
   довжина(відр(B,C),L2),
   довжина(відр(A,C),L3),
   теорема_Піфагора(L1,L2,L3),!.

теорема_Піфагора(L1,L2,L3):-
   L1*L1+L2*L2=L3*L3,!.
теорема_Піфагора(L1,L2,L3):-
   L1*L1+L3*L3=L2*L2,!.
теорема_Піфагора(L1,L2,L3):-
   L2*L2+L3*L3=L1*L1,!.
GOAL A=точ(0,0),B=точ(0,4),C=точ(3,0),
   T=трик(A,B,C), прямокутний(T).
```

Завдання для самостійної роботи.

1. Описати структурний домен `вектор`.
2. Знайти довжину вектора, суму, різницю, скалярний добуток, векторний добуток двох векторів.
3. Реалізувати для двох векторів відношення `рівні`, `колінеарні` та `перпендикулярні`.

1.6. Деякі операції над множинами.

Завдання 1.6. Реалізувати такі операції над множинами:

1) перевірити належність елемента множині;
2) перетворити список у множину;
3) перевірити, чи *A* підмножина *B*;
4) перевірити, чи множини *A* і *B* рівні;
5) знайти об'єднання 2-х множин;
6) знайти перетин 2-х множин;
7) знайти різницю 2-х множин.

Розв'язання. Множину будемо представляти списком, елементи якої не повторюються.

1) Перевірку належності елемента множині реалізує предикат `належить(Ел,Сп)`:

Правило 1. Елемент `Ел` буде належати множині, яка представлена списком `Сп`, якщо він співпадає з головою списку

Правило 2. … або належить хвосту списку.

Приклад.

```
goal належить(3,[1,2,3,4,5]).
```

Результат: `yes`

2) Перетворює список у множину предикат `список_множина(Сп,Мн)`:

Правило 1. Якщо список порожній, то і множина буде порожня.

Правило 2. Якщо список розділяється на голову та хвіст і голова належить хвосту, то перетворити хвіст у множину.

Правило 3. Інакше – перетворити хвіст у множину та добавити до неї голову списку.

Приклад.

```
goal список_множина([1,3,5,3],A).
```

Результат: `A=[1,5,3]`

3) Перевірку підмножини реалізує предикат `підмножина(A,B)`:

Правило 1. Порожня множина буде підмножиною довільної множини.

Правило 2. Якщо множина `A`, яка представлена списком, розділяється на голову та хвіст і голова належить множині `B`, то рекурсивно перевірити, чи хвіст буде підмножиною `B`.

Приклад.

```
goal підмножина( [1,3,5],[2,1,5,3,4]).
```

Результат: `yes`

4) Перевірку рівності множин реалізує предикат `рівні(A,B)`: множини `A` і `B` рівні, якщо `A` підмножина `B`, а `B` підмножина `A`.

Приклад.

```
goal рівні([1,2,3,4,5],[2,1,5,3,4]).
```

Результат: `yes`

5) Об'єднання двох множин реалізує предикат `обєднання(A,B,C)`:

Правило 1. Об'єднанням порожньої множини з будь-якою множиною і буде ця множина.

Правило 2. Якщо множина `A`, яка представлена списком, розділяється на голову та хвіст і голова належить множині `B`, то рекурсивно об'єднати хвіст з множиною `B`.

Правило 3. Інакше – об'єднати хвіст з множиною `B` та добавити до нової множини голову списку.

Приклад.

```
goal обєднання([1,3,5],[2,3,4,5],C).
```
Результат: `C=[1,2,3,4,5]`

6) Перетин двох множин реалізує предикат `перетин(A,B,C)`:

Правило 1. Перетином порожньої множини з будь-якою множиною буде порожня множина.

Правило 2. Якщо множина `A`, яка представлена списком, розділяється на голову та хвіст і голова належить множині `B`, то рекурсивно перетнути хвіст з множиною `B` та добавити до нової множини голову списку.

Правило 3. Інакше – перетнути хвіст з множиною `B`.

Приклад.

```
goal перетин([1,3,5],[2,3,4,5],C).
```
Результат: `C=[3,5]`

7) Знаходження різниці двох множин реалізує предикат `різниця(A,B,C)`:

Правило 1. Різницею порожньої множини з будь-якою множиною буде порожня множина.

Правило 2. Якщо множина `A`, яка представлена списком, розділяється на голову та хвіст і голова належить множині `B`, то рекурсивно знайти різницю хвоста з множиною `B`.

Правило 3. Інакше – знайти різницю хвоста з множиною `B` та добавити до нової множини голову списку.

Приклад.

```
goal різниця([1,3,5],[2,3,4,5],C).
```
Результат: `C=[1]`

Лістинг 1.6. Деякі операції над множинами.

```
DOMAINS
сп=integer*
PREDICATES
належить(integer,сп)
список_множина(сп,сп)
підмножина(сп,сп)
рівні(сп,сп)
обєднання(сп,сп,сп)
перетин(сп,сп,сп)
різниця(сп,сп,сп)
CLAUSES
належить(Ел,[Ел|_]):-!.
належить(Ел,[_|Хв]):-
   належить(Ел,Хв),!.

список_множина([],[]):-!.
список_множина([Гол|Хв],Множ):-
   належить(Гол,Хв),
   список_множина(Хв,Множ),!.
список_множина([Гол|Хв],[Гол|Хв1]):-
   список_множина(Хв,Хв1),!.
```

```
підмножина([],_):-!.
підмножина([Гол|Хв],Множ):-
   належить(Гол,Множ),
   підмножина(Хв,Множ),!.

рівні(A,B):-
   підмножина(A,B),
   підмножина(B,A),!.

обєднання([],Множ,Множ):-!.
обєднання([Гол|Хв],Множ,Множ1):-
   належить(Гол,Множ),
   обєднання(Хв,Множ,Множ1),!.
обєднання([Гол|Хв],Множ,[Гол|Хв1]):-
   обєднання(Хв,Множ,Хв1),!.

перетин([],_,[]):-!.
перетин([Гол|Хв],Множ,[Гол|Хв1]):-
   належить(Гол,Множ),
   перетин(Хв,Множ,Хв1),!.
перетин([_|Хв],Множ,Множ1):-
   перетин(Хв,Множ,Множ1),!.

різниця([],_,[]):-!.
різниця([Гол|Хв],Множ,Множ1):-
   належить(Гол,Множ),
   різниця(Хв,Множ,Множ1),!.
різниця([Гол|Хв],Множ,[Гол|Хв1]):-
   різниця(Хв,Множ,Хв1),!.

GOAL
   належить(3,[1,2,3,4,5]).
```

Завдання для самостійної роботи.

1. Реалізувати операцію доповнення множини.
2. Знайти декартовий добуток двох множин.

1.7. Методи сортування списків.

Завдання 1.7. Реалізувати такі методи сортування:
1) бульбашкове сортування;
2) сортування вибором;
3) сортування вставкою;
4) сортування злиттям;
5) швидке сортування.

Розв'язання. У всіх предикатах, що реалізують методи сортування перший параметр – це вхідний список, а другий – вихідний (упорядкований по неспаданню).

1) Метод бульбашкового сортування реалізує предикат `бульбашкове_сортування(Сп1,Сп2)`, який використовує відношення `перестановка`. Предикат `перестановка(Сп1,Сп3)` виконується успішно, якщо у вхідному списку `Сп1` знайдуться два послідовні елементи, що не упорядковані по неспаданню. Тоді вихідний список `Сп3` отримується зі списку `Сп1` перестановкою цих елементів.

Приклад.

```
goal перестановка([1,2,-3,4,5],Сп).
```
Результат: `Сп=[1,-3,2,4,5]`

Алгоритм бульбашкового сортування:

Правило 1. Якщо `перестановка(Сп1,Сп3)` виконується успішно, то рекурсивно сортувати `Сп3`.

Правило 2. Інакше – вхідний список упорядкований.

2) Метод сортування вибором реалізує предикат `сортування_вибором`, який використовує відношення `мінімальний(Сп,Мін)`, яке шукає у вхідному списку мінімальний елемент, та `видалити(Мін,Сп,Сп1)`, яке із списку `Сп` видаляє цей елемент і повертає список `Сп1`.

Приклад.

```
goal мінімальний([1,2,-3,4,5],Мін).
```
Результат: `Мін=-3`

Приклад.

```
goal видалити(-3,[1,2,-3,4,5],Сп).
```
Результат: `Сп=[1,2,4,5]`

Алгоритм сортування вибором:

Правило 1. Якщо вхідний список порожній, то вихідний також буде порожній.

Правило 2. Інакше – видалити мінімальний елемент `Мін`, рекурсивно упорядкувати отриманий `Сп1`. У вихідному списку елемент `Мін` розмістити на початку.

3) Метод сортування вставкою реалізує предикат `сортування_вставкою`, який використовує відношення `вставка`. Предикат `вставка(Ел,Сп,Сп1)` вставляє елемент `Ел` у відсортований список `Сп`, не змінюючи впорядкування.

Приклад.

```
goal вставка(4,[1,2,3,5],Сп).
```
Результат: `Сп=[1,2,3,4,5]`

Алгоритм сортування вставкою:

Правило 1. Якщо вхідний список порожній, то вихідний також буде порожній.

Правило 2. Інакше – розділити список на голову і хвіст, рекурсивно упорядкувати хвіст та вставити голову у впорядкований список.

4) Метод сортування злиттям реалізує предикат `сортування_злиттям(Сп1,Сп2)`, який використовує відношення `виділити_неспадний` та `злити_списки`. Предикат `виділити_неспадний(Сп1,Сп3,Сп4)` розділяє список `Сп1` на неспадний список `Сп3` та остаток `Сп4`. Предикат `злити_списки(Сп3,Сп5,Сп2)` із відсортованих списків `Сп3` і `Сп5` формує упорядкований список `Сп2`.

Приклад.

```
goal виділити_неспадний([1,2,-3,4,5],Сп1,Сп2).
```
Результат: `Сп1=[1,2], Сп2=[-3,4,5]`

Приклад.

```
goal злити_списки([1,2],[-3,4,5],Сп).
```
Результат: `Сп=[-3,1,2,4,5]`

Алгоритм сортування злиттям:

Правило 1. Якщо вхідний список порожній, то вихідний також буде порожній.

Правило 2. Інакше – розділити список на неспадний і остаток, рекурсивно упорядкувати остаток та злити його з неспадним.

5) Метод швидкого сортування реалізує предикат `швидке_сортування`, який використовує відношення `розділити_менші_неменші` та `обєднати_списки` (див. 1.6). Предикат `розділити_менші_неменші(Ел,Сп,Сп1,Сп2)` розділяє список `Сп` на список `Сп1`, елементи якого менші за `Ел`, та список `Сп2`, елементи якого не менші за `Ел`.

Приклад.

```
goal розділити_менші_неменші(1,[2,3,-3,4,5,1,-1],Сп1,Сп2).
```
Результат: `Сп1=[-3,-1], Сп2=[2,3,4,5,1]`

Алгоритм сортування злиттям:

Правило 1. Якщо вхідний список порожній, то вихідний також буде порожній.

Правило 2. Інакше – розділити список на голову і хвіст, розділити хвіст на список, елементи якого менші за голову, та список, елементи якого не менші за голову, рекурсивно упорядкувати ці списки та об'єднати.

Лістинг 1.7. Методи сортування списків.

```
DOMAINS
сп=integer*
PREDICATES
бульбашкове_сортування(сп,сп)
перестановка(сп,сп)
CLAUSES
бульбашкове_сортування(Сп1,Сп2):-
   перестановка(Сп1,Сп3),
   бульбашкове_сортування(Сп3,Сп2),!.
бульбашкове_сортування(Сп,Сп):-!.

перестановка([Ел1,Ел2|Хв],[Ел2,Ел1|Хв]):-
   Ел1>Ел2,!.
перестановка([Гол|Хв1],[Гол|Хв2]):-
   перестановка(Хв1,Хв2),!.
```

```
PREDICATES
сортування_вибором(сп,сп)
мінімальний(сп,integer)
видалити(integer,сп,сп)

CLAUSES
сортування_вибором([],[]):-!.
сортування_вибором(Сп,[Мін|Хв]):-
   мінімальний(Сп,Мін),
   видалити(Мін,Сп,Сп1),
   сортування_вибором(Сп1,Хв),!.

мінімальний([Мін],Мін):-!.
мінімальний([Гол|Хв],Мін):-
   мінімальний(Хв,Мін),
   Мін<=Гол,!.
мінімальний([Гол|_],Гол):-!.

видалити(Гол,[Гол|Хв],Хв):-!.
видалити(Ел,[Гол|Хв1],[Гол|Хв2]):-
   видалити(Ел,Хв1,Хв2),!.

PREDICATES
сортування_вставкою(сп,сп)
вставка(integer,сп,сп)
CLAUSES
сортування_вставкою([],[]):-!.
сортування_вставкою([Гол|Хв],Сп):-
   сортування_вставкою(Хв,Сп1),
   вставка(Гол,Сп1,Сп),!.

%Параметр2 - список, упорядкований по неспаданню
вставка(Ел,[],[Ел]):-!.
вставка(Ел,[Гол|Хв],[Ел,Гол|Хв]):-
   Ел<=Гол,!.
вставка(Ел,[Гол|Хв1],[Гол|Хв2]):-
   вставка(Ел,Хв1,Хв2),!.

PREDICATES
сортування_злиттям(сп,сп)
виділити_неспадний(сп,сп,сп)
злити_списки(сп,сп,сп)
CLAUSES
сортування_злиттям([],[]):-!.
сортування_злиттям(Сп1,Сп2):-
   виділити_неспадний(Сп1,Сп3,Сп4),
   сортування_злиттям(Сп4,Сп5),
   злити_списки(Сп3,Сп5,Сп2),!.
```

```
виділити_неспадний([],[],[]):-!.
виділити_неспадний([Ел],[Ел],[]):-!.
виділити_неспадний([Ел1,Ел2|Хв],[Ел1],[Ел2|Хв]):-
   Ел1>Ел2,!.
виділити_неспадний([Гол|Хв],[Гол|Хв1],Сп):-
   виділити_неспадний(Хв,Хв1,Сп),!.

%Параметр1, Параметр2 – списки, упорядковані по неспаданню
злити_списки([],Сп,Сп):-!.
злити_списки(Сп,[],Сп):-!.
злити_списки([Гол1|Хв1],[Гол2|Хв2],[Гол1|Хв3]):-
   Гол1<=Гол2,!,
   злити_списки(Хв1,[Гол2|Хв2],Хв3),!.
злити_списки(Сп,[Гол|Хв1],[Гол|Хв2]):-
   злити_списки(Сп,Хв1,Хв2),!.

PREDICATES
швидке_сортування(сп,сп)
розділити_менші_неменші(integer,сп,сп,сп)
обєднати_списки(сп,сп,сп)
CLAUSES
швидке_сортування([],[]):-!.
швидке_сортування([Гол|Хв],Сп):-
   розділити_менші_неменші(Гол,Хв,Сп1,Сп2),
   швидке_сортування(Сп1,Сп3),
   швидке_сортування(Сп2,Сп4),
   Сп5=[Гол|Сп4],
   обєднати_списки(Сп3,Сп5,Сп),!.

розділити_менші_неменші(_,[],[],[]):-!.
розділити_менші_неменші(Ел,[Гол|Хв],[Гол|Хв1],Сп):-
   Ел>Гол,!,
   розділити_менші_неменші(Ел,Хв,Хв1,Сп),!.
розділити_менші_неменші(Ел,[Гол|Хв],Сп,[Гол|Хв1]):-
   розділити_менші_неменші(Ел,Хв,Сп,Хв1),!.

обєднати_списки([],Сп,Сп):-!.
обєднати_списки([Гол|Хв],Сп,[Гол|Хв1]):-
   обєднати_списки(Хв,Сп,Хв1),!.

GOAL
   швидке_сортування([1,2,3,-3,4,5,1,-1],Сп).
```

Результат: Сп=[-3,-1,1,1,2,3,4,5]

Завдання для самостійної роботи.

1. Реалізувати метод сортування включенням:

Перший елемент утворює відсортований список, а кожний наступний «включається» у нього таким чином, щоб не порушити впорядкованість.

ІІ. ЗАВДАННЯ ПІДВИЩЕНОГО РІВНЯ СКЛАДНОСТІ

2.1. Робота з динамічною базою даних.

Завдання 2.1. Інформацію про студентів задано предикатами динамічної бази даних `студент(Номер залікової, Прізвище, Курс, Стипендія)` та `сесія(Номер залікової, Предмет, Оцінка)`. Описати предикати, що:

1) виводять на екран прізвище та стипендію студентів вказаного курсу;
2) знаходять сумарну стипендію студентів вказаного курсу;
3) знаходять найбільшу стипендію серед студентів вказаного курсу та відповідне прізвище студента;
4) знаходять кількість відмінників;
5) збільшують стипендію відмінникам на вказану суму.

Розв'язання. Нехай у файлі `"Студенти.txt"` міститься інформація про студентів:

```
студент(11110,"Дмитрук",1,1000)
студент(11111,"Романюк",1,1100)
студент(11112,"Сидорук",1,1000)
студент(11113,"Григорюк",2,1200)
студент(11114,"Богданюк",2,1300)
студент(11115,"Степанюк",2,0)
студент(11116,"Михайлюк",2,1000)
студент(11117,"Назарюк",3,0)
студент(11118,"Борисюк",3,1200)
студент(11119,"Василюк",3,1100)
сесія(11110,"Історія",4)
сесія(11110,"Політологія",5)
сесія(11110,"Філософія",4)
сесія(11111,"Історія",5)
сесія(11111,"Політологія",5)
сесія(11111,"Філософія",5)
сесія(11112,"Історія",4)
сесія(11112,"Політологія",4)
сесія(11112,"Філософія",5)
сесія(11113,"Іноземна мова",4)
сесія(11113,"Інформатика",5)
сесія(11113,"Соціологія",5)
сесія(11114,"Іноземна мова",5)
сесія(11114,"Інформатика",5)
сесія(11114,"Соціологія",5)
сесія(11115,"Іноземна мова",3)
сесія(11115,"Інформатика",3)
сесія(11115,"Соціологія",5)
сесія(11116,"Іноземна мова",3)
сесія(11116,"Інформатика",5)
сесія(11116,"Соціологія",4)
сесія(11117,"Вища математика",3)
сесія(11117,"Правознавство",4)
сесія(11117,"Політекономія",3)
сесія(11118,"Вища математика",5)
сесія(11118,"Правознавство",4)
сесія(11118,"Політекономія",4)
сесія(11119,"Вища математика",5)
сесія(11119,"Правознавство",5)
сесія(11119,"Політекономія",5)
```

1) Для виводу на екран прізвища та стипендії студентів вказаного курсу достатньо використати предикат динамічної бази даних `студент`.

```
надрукувати(Курс):-
   write("Прізвище  ","Стипендія"),nl,
   студент(_,Прізв,Курс,Стип),
   writef("%-10%-5\n",Прізв,Стип),fail.
надрукувати(_):-!.
```

Предикат `надрукувати` із вхідним параметром `Курс` звертається до динамічної бази даних та серед всіх студентів знаходить першого, який навчається на заданому курсі. Далі за допомогою вбудованого предиката `writef` виводиться прізвище та стипендія на екран (формат `"%-10%-5\n"` задає відступи для змінних `Прізв, Стип` та переводить курсор на наступний рядок). Вбудований предикат `fail` ініціює відкат, таким чином Пролог перегляне всіх

студентів. Для успішного завершення предиката `надрукувати` використовується друге правило процедури.

Приклад. Зчитаємо файл `"Студенти.txt"` та виведемо інформацію про другокурсників.

```
goal consult("Студенти.txt"),
     надрукувати(2).
```

Результат:
```
Прізвище  Стипендія
Григорюк  1200
Богданюк  1300
Степанюк  0
Михайлюк  1000
```

2) Для знаходження сумарної стипендії студентів вказаного курсу у відношенні `сумарна_стипендія` із вхідним параметром `Курс` та вихідним параметром `Всього` використаємо вбудований предикат `findall`, який формує `Список` потрібних стипендій. Далі за допомогою предиката `сума` (див. 1.3) знаходимо суму елементів списку.

```
сумарна_стипендія(Курс,Всього):-
   findall(Стип,студент(_,_,Курс,Стип),Список),
   сума(Список,Всього),!.

   сума([],0):-!.
   сума([Гол|Хв],Сума):-
      сума(Хв,Сума1),
      Сума=Гол+Сума1,!.
```

Приклад. Знайдемо сумарну стипендію другокурсників.

```
goal consult("Студенти.txt"),
     сумарна_стипендія(2,Всього).
```

Результат: `Всього=3500`

3) На відміну від попередньої задачі, для знаходження найбільшої стипендії серед студентів вказаного курсу не будемо використовувати списки.

```
максимальна_стипендія(Курс,_,_):-
   початкова_макс_стип,
   студент(_,Прізв,Курс,Стип),
   змінити_макс_стип(Прізв,Стип),
   fail.
максимальна_стипендія(_,Прізв,Стип):-
   retract(макс_стип(Прізв,Стип)),!.

   початкова_макс_стип:-
      assert(макс_стип("",0)),!.

   змінити_макс_стип(Прізв,Стип):-
      макс_стип(_,Стип1),
      Стип>Стип1,
      retractall(макс_стип(_,_)),
      assert(макс_стип(Прізв,Стип)),!.
```

Відношення максимальна_стипендія із вхідним параметром Курс та вихідними параметрами Прізв, Стип використовує предикат динамічної бази даних макс_стип, у якому зберігається поточна найбільша стипендія та прізвище відповідного студента. Для цього за допомогою предиката початкова_макс_стип задається початкове значення 0. Далі проглядаються всі записи студент(_,Прізв,Курс,Стип) і за допомогою предиката змінити_макс_стип, якщо значення змінної Стип більше за поточну стипендію, то старе значення макс_стип видаляється і запам'ятовується нове прізвище і стипендія.

Приклад. Знайдемо максимальну стипендію серед другокурсників.

```
goal consult("Студенти.txt"),
     максимальна_стипендія(2,Прізв,Стип).
```

Результат: Прізв=Богданюк, Стип=1300

4) Для знаходження кількості відмінників необхідно використати обидва предикати динамічної бази даних: студент та сесія. Спочатку створимо недетермінований предикат відмінник, який повертає Н_зал (номер залікової) та Прізв (прізвище) студента, що має всі оцінки 5.

```
відмінник(Н_зал,Прізв):-
   студент(Н_зал,Прізв,_,_),
   findall(Оц,сесія(Н_зал,_,Оц),Список),
   всі_5(Список).

   всі_5([]):-!.
   всі_5([5|Хв]):-всі_5(Хв),!.
```

За допомогою предиката findall будується Список оцінок (Оц). Предикат всі_5 виконується успішно, якщо всі елементи списку рівні 5.

Приклад. Знайдемо всіх відмінників.

```
goal consult("Студенти.txt"),
     відмінник(Н_зал,Прізв).
```

Результат: Н_зал=11111, Прізв=Романюк
Н_зал=11114, Прізв=Богданюк
Н_зал=11119, Прізв=Василюк

Далі за допомогою предиката findall будується Список прізвищ відмінників та знаходиться кількість елементів списку (див. 1.3).

```
кількість_відмінників(Кільк):-
   findall(Прізв,відмінник(_,Прізв),Список),
   кількість(Список,Кільк),!.

   кількість([],0):-!.
   кількість([_|Хв],Кільк):-
      кількість(Хв,Кільк1),
      Кільк=1+Кільк1,!.
```

Приклад. Знайдемо кількість відмінників.

```
goal consult("Студенти.txt"),
     кількість_відмінників(Кільк).
```

Результат: Кільк=3

5) Для збільшення стипендії відмінникам використовується описане раніше відношення відмінник та предикат збільшити_стипендію, який за вказаним номером залікової (Н_зал) знищує попередню інформацію про студента-відмінника та добавляє на початок бази даних (для уникнення зациклення) поновлену інформацію.

```
збільшити_стипендію_відмінникам(Сума):-
   відмінник(Н_зал,_),
   збільшити_стипендію(Н_зал,Сума),
   fail.
збільшити_стипендію_відмінникам(_):-!.
   збільшити_стипендію(Н_зал,Сума):-
      retract(студент(Н_зал,Прізв,Курс,Стип)),
      Стип1=Стип+Сума,
      asserta(студент(Н_зал,Прізв,Курс,Стип1)),!.
```

Приклад. Збільшимо стипендію відмінникам на 50 од.

```
goal consult("Студенти.txt"),
     збільшити_стипендію_відмінникам(50),
     save("Студенти.txt"),
     надрукувати(_).
```

Результат:

```
Прізвище  Стипендія
Василюк   1150
Богданюк  1350
Романюк   1150
Дмитрук   1000
Сидорук   1000
Григорюк  1200
Степанюк  0
Михайлюк  1000
Назарюк   0
Борисюк   1200
```

Лістинг 2.1. Робота з динамічною базою даних.

```
DOMAINS
ціле = integer
цілі = ціле*
рядок = string
рядки = рядок*
DATABASE
студент(ціле н_зал,рядок прізв,ціле курс,ціле стип)
сесія(ціле н_зал,рядок предмет,ціле оцінка)
макс_стип(рядок прізв,ціле стип)

PREDICATES
надрукувати(ціле курс)
сумарна_стипендія(ціле курс,ціле стип)
   сума(цілі,ціле)
максимальна_стипендія(ціле курс,рядок прізв,ціле стип)
   початкова_макс_стип
   змінити_макс_стип(рядок прізв,ціле стип)
```

```
кількість_відмінників(ціле кількість)
   nondeterm відмінник(ціле н_зал,рядок прізв)
      всі_5(цілі)
   кількість(рядки,ціле)
збільшити_стипендію_відмінникам(ціле сума)
   збільшити_стипендію(ціле н_зал,ціле сума)

CLAUSES
надрукувати(Курс):-
   write("Прізвище  ","Стипендія"),nl,
   студент(_,Прізв,Курс,Стип),
   writef("%-10%-5\n",Прізв,Стип),
   fail.
надрукувати(_):-!.

сумарна_стипендія(Курс,Всього):-
   findall(Стип,студент(_,_,Курс,Стип),Список),
   сума(Список,Всього),!.

   сума([],0):-!.
   сума([Гол|Хв],Сума):-
      сума(Хв,Сума1),
      Сума=Гол+Сума1,!.

максимальна_стипендія(Курс,_,_):-
   початкова_макс_стип,
   студент(_,Прізв,Курс,Стип),
   змінити_макс_стип(Прізв,Стип),
   fail.
максимальна_стипендія(_,Прізв,Стип):-
   retract(макс_стип(Прізв,Стип)),!.

   початкова_макс_стип:-
      assert(макс_стип("",0)),!.

   змінити_макс_стип(Прізв,Стип):-
      макс_стип(_,Стип1),
      Стип>Стип1,
      retractall(макс_стип(_,_)),
      assert(макс_стип(Прізв,Стип)),!.

кількість_відмінників(Кільк):-
   findall(Прізв,відмінник(_,Прізв),Список),
   кількість(Список,Кільк),!.

   відмінник(Н_зал,Прізв):-
      студент(Н_зал,Прізв,_,_),
      findall(Оц,сесія(Н_зал,_,Оц),Список),
      всі_5(Список).
```

```
      всі_5([]):-!.
      всі_5([5|Хв]):-
         всі_5(Хв),!.

   кількість([],0):-!.
   кількість([_|Хв],Кільк):-
      кількість(Хв,Кільк1),
      Кільк=1+Кільк1,!.

збільшити_стипендію_відмінникам(Сума):-
   відмінник(Н_зал,_),
   збільшити_стипендію(Н_зал,Сума),
   fail.
збільшити_стипендію_відмінникам(_):-!.

   збільшити_стипендію(Н_зал,Сума):-
      retract(студент(Н_зал,Прізв,Курс,Стип)),
      Стип1=Стип+Сума,
      asserta(студент(Н_зал,Прізв,Курс,Стип1)),!.

GOAL
   %1
   consult("Студенти.txt"),
   надрукувати(2).
```

Завдання для самостійної роботи.

1. Знайти середній бал вказаного студента.
2. Знайти найбільшу стипендію серед студентів-трійочників.
3. Збільшити на *a*% стипендію студентам вказаного курсу, які не мають трійок.

2.2. Обчислення числово-літерних виразів.

Завдання 2.2. Задано числово-літерний вираз у інфіксній формі. Перетворити вираз у префіксну форму та обчислити його значення при заданих змінних.

Розв'язання. Для представлення виразу у префіксній формі, використовується структурний домен `пр_вираз`:

```
пр_вираз = число(дійсне); змінна(рядок); плюс(пр_вираз,пр_вираз);
   мінус(пр_вираз,пр_вираз); добуток(пр_вираз,пр_вираз);
   частка(пр_вираз,пр_вираз)
```

Приклади.

Число 5 – `число(5)`; змінна a – `змінна("a")`;

вираз "a * (b + c)" – `добуток(змінна("a"),плюс(змінна("b"),змінна("c")))`.

Значення змінних зберігається у відношенні `значення` динамічної бази даних.

Перетворення виразу із інфіксної форми (`Вираз`) у префіксну (`Пр_Вираз`) реалізує процедура `префіксна_форма(Вираз,Пр_Вираз)`, в якій враховано такі випадки:

1) вираз це число;
2) вираз це змінна;
3) вираз взятий у дужки;
4) вираз розкладається на суму;
5) вираз розкладається на різницю;
6) вираз розкладається на добуток;
7) вираз розкладається на частку.

Предикат `префіксна_форма` використовує відношення `розбити(Вираз, Початок,Знак,Остача,Дужки)`, яке числово-літерний `Вираз` розділяє на `Початок` та `Остачу` відносно символу `Знак` (+, –, *, /), при цьому враховується кількість відкритих і закритих дужок:

1) якщо `Вираз` починається із символу `Знак` і кількість незакритих дужок рівна 0, то розбиття успішно завершено;
2) якщо `Вираз` починається із відкриваючої дужки, то рекурсивно його розділити, збільшивши кількість незакритих дужок на 1;
3) якщо `Вираз` починається із закриваючої дужки, то рекурсивно його розділити, зменшивши кількість незакритих дужок на 1;
4) інакше – якщо `Вираз` починається із `Символу`, відмінного від дужок, то рекурсивно його розділити і добавити `Символ` до результату.

Приклад. Розділити вираз `"(2*b+a)/b"` на дільники.

```
goal Вираз="(2*b+a)/b",
     розбити(Вираз,Початок,"/",Остаток,0).
```

Результат: `Початок=(2*b+a), Остаток=b`

Приклад. Перевірити, чи можна розділити вираз `"(2*b+a)/b"` на доданки.

```
goal Вираз="(2*b+a)/b",
     розбити(Вираз,Початок,"+",Остаток,0).
```

Результат: `no`

Обчислення префіксного виразу реалізовано процедурою `обчислити(Пр_Вираз, Число)`, у якій враховано такі випадки:

1) якщо префіксний вираз містить тільки `Число`, то це `Число` і буде результатом;

2) якщо префіксний вираз містить тільки змінну, то результатом буде значення цієї змінної;
3) якщо префіксний вираз має вигляд плюс(Пр_Вираз1,Пр_Вираз2), то рекурсивно обчислюються значення для префіксних виразів Пр_Вираз1 та Пр_Вираз2, а результатом буде сума цих значень;
4) аналогічно розглядаються випадки для структур мінус, добуток, частка (в останньому випадку також враховується заборона ділення на 0).

Приклад. Обчислити значення виразу (2*b+a)/b при a = 3, b = -2.5.

```
goal assert(значення("a",3)),
   assert(значення("b",-2.5)),
   Вираз="(2*b+a)/b",
   префіксна_форма(Вираз,Пр_Вираз),
   обчислити(Пр_Вираз,Число),
   write(Число),nl.
```

Результат: 0.8

Даний вираз буде мати таке відображення:

```
частка(плюс(добуток(число(2),змінна("b")),змінна("a")),змінна("b"))
```

Лістинг 2.2. Обчислення числово-літерних виразів.

```
DOMAINS
   ціле = integer
   дійсне = real
   рядок = string
   пр_вираз = число(дійсне);змінна(рядок);
      плюс(пр_вираз,пр_вираз);мінус(пр_вираз,пр_вираз);
      добуток(пр_вираз,пр_вираз);частка(пр_вираз,пр_вираз)
DATABASE
значення(рядок,дійсне)
PREDICATES
префіксна_форма(рядок,пр_вираз)
розбити(рядок,рядок,рядок,рядок,ціле)
обчислити(пр_вираз,дійсне)
CLAUSES
префіксна_форма(Вираз,Пр_Вираз):-
   str_int(Вираз,Число),
   Пр_Вираз=число(Число),!.
префіксна_форма(Вираз,Пр_Вираз):-
   значення(Вираз,_),
   Пр_Вираз=змінна(Вираз),!.
префіксна_форма(Вираз,Пр_Вираз):-
   concat("(",Вираз1,Вираз),
   concat(Вираз2,")",Вираз1),
   префіксна_форма(Вираз2,Пр_Вираз),!.
префіксна_форма(Вираз,Пр_Вираз):-
   розбити(Вираз,Доданок,"+",Остача,0),
   префіксна_форма(Доданок,Пр_Вираз1),
   префіксна_форма(Остача,Пр_Вираз2),
   Пр_Вираз=плюс(Пр_Вираз1,Пр_Вираз2),!.
```

```
префіксна_форма(Вираз,Пр_Вираз):-
   розбити(Вираз,Доданок,"-",Остача,0),
   префіксна_форма(Доданок,Пр_Вираз1),
   префіксна_форма(Остача,Пр_Вираз2),
   Пр_Вираз=мінус(Пр_Вираз1,Пр_Вираз2),!.
префіксна_форма(Вираз,Пр_Вираз):-
   розбити(Вираз,Множник,"*",Остача,0),
   префіксна_форма(Множник,Пр_Вираз1),
   префіксна_форма(Остача,Пр_Вираз2),
   Пр_Вираз=добуток(Пр_Вираз1,Пр_Вираз2),!.
префіксна_форма(Вираз,Пр_Вираз):-
   розбити(Вираз,Множник,"/",Остача,0),
   префіксна_форма(Множник,Пр_Вираз1),
   префіксна_форма(Остача,Пр_Вираз2),
   Пр_Вираз=частка(Пр_Вираз1,Пр_Вираз2),!.

розбити(Вираз,"",Знак,Остача,0):-
   concat(Знак,Остача,Вираз),!.
розбити(Вираз,Початок,Знак,Остача,Дужки):-
   frontchar(Вираз,'(',Вираз1),
   Дужки1=Дужки+1,
   розбити(Вираз1,Початок1,Знак,Остача,Дужки1),
   frontchar(Початок,'(',Початок1),!.
розбити(Вираз,Початок,Знак,Остача,Дужки):-
   frontchar(Вираз,')',Вираз1),
   Дужки1=Дужки-1,
   розбити(Вираз1,Початок1,Знак,Остача,Дужки1),
   frontchar(Початок,')',Початок1),!.
розбити(Вираз,Початок,Знак,Остача,Дужки):-
   frontchar(Вираз,Символ,Вираз1),
   Символ<>'(',Символ<>')',
   розбити(Вираз1,Початок1,Знак,Остача,Дужки),
   frontchar(Початок,Символ,Початок1),!.

обчислити(число(Число),Число):-!.
обчислити(змінна(Вираз),Число):-
   значення(Вираз,Число),!.
обчислити(плюс(Пр_Вираз1,Пр_Вираз2),Число):-
   обчислити(Пр_Вираз1,Число1),
   обчислити(Пр_Вираз2,Число2),
   Число=Число1+Число2,!.
обчислити(мінус(Пр_Вираз1,Пр_Вираз2),Число):-
   обчислити(Пр_Вираз1,Число1),
   обчислити(Пр_Вираз2,Число2),
   Число=Число1-Число2,!.
обчислити(добуток(Пр_Вираз1,Пр_Вираз2),Число):-
   обчислити(Пр_Вираз1,Число1),
   обчислити(Пр_Вираз2,Число2),
   Число=Число1*Число2,!.
```

```
обчислити(частка(Пр_Вираз1,Пр_Вираз2),Число):-
   обчислити(Пр_Вираз1,Число1),
   обчислити(Пр_Вираз2,Число2),Число2<>0,
   Число=Число1/Число2,!.
обчислити(частка(Пр_Вираз1,Пр_Вираз2),0):-
   обчислити(Пр_Вираз1,_),
   обчислити(Пр_Вираз2,Число2),Число2=0,
   write("Ділення на нуль!"),nl,!,fail.
GOAL assert(значення("a",3)),
   assert(значення("b",-2.5)),
   Вираз="(2*b+a)/b",
   префіксна_форма(Вираз,Пр_Вираз),
   обчислити(Пр_Вираз,Число),
   write(Число),nl.
```

Завдання для самостійної роботи.

1. Модифікувати домен пр_вираз для представлення степенів з натуральними показниками.
2. Розробити програму спрощення виразів.

2.3. Алгоритм Крускала побудови мінімального кістякового дерева.

Завдання 2.3. Задано список точок. Побудувати мінімальне кістякове дерево для повного графа, що задається цими точками.

Розв'язання. Для розв'язання задачі використаємо алгоритм Крускала:

Спочатку поточна множина ребер встановлюється порожньою. Потім, поки це можливо, проводиться така операція: з усіх ребер, додавання яких до вже наявної множини не викличе появи в ній циклу, вибирається ребро мінімальної ваги і додається до цієї множини. Коли таких ребер більше немає, алгоритм завершено. Підграф даного графа, що має всі його вершини і знайдену множину ребер, утворює кістякове дерево мінімальної ваги.

У програмі використовуються структурні домени:

`тч=тч(integer,integer)` – точка двовимірного простору;

`сп_тч=тч*` – список точок;

`сп_сп_тч=сп_тч*` – список множин (списків) точок

`рб=рб(тч,тч,real)` – ребро, що задається двома точками, та його довжина;

`сп_рб=рб*` – список ребер.

Також у програмі використовуються деякі предикати над списками: `обєднати_списки`, `швидке_сортування`, `менше_більшерівно`, `належить_списку` (див. 1.6, 1.7).

Цільовий предикат `старт`, що має вхідний параметр `Сп_тч` – список точок та вихідний параметр `Сп_рб` – мінімальне кістякове дерево, задане списком ребер, виконує такі дії:

1) зі списку точок `Сп_тч` будується повний граф, що задається списком ребер `Сп_рб1`;
2) за допомогою алгоритму швидкого сортування зі `Сп_рб1` формується упорядкований список ребер `Сп_рб2`;
3) із кожної точки із списку `Сп_тч` утворюється множина. Ці множини по одній точці записуються у список `Сп_сп_тч`;
4) на основі списку множин `Сп_сп_тч` та упорядкованого списку ребер `Сп_рб2` будується мінімальне кістякове дерево `Сп_рб`.

Приклад. Дано точки *A*(2,0), *B*(2,5), *C*(0,4), *D*(2,4), *E*(5,5). Побудувати повний граф із цих точок.

```
goal Сп_тч=[тч(2,0),тч(2,5),тч(0,4),тч(2,4),тч(5,5)],
     повний_граф(Сп_тч,Сп_рб1).
```

Результат:
```
Сп_рб1=[рб(тч(2,0),тч(2,5),5), рб(тч(2,0),тч(0,4),4.47),
        рб(тч(2,0),тч(2,4),4), рб(тч(2,0),тч(5,5),5.83),
        рб(тч(2,5),тч(0,4),2.23), рб(тч(2,5),тч(2,4),1),
        рб(тч(2,5),тч(5,5),3), рб(тч(0,4),тч(2,4),2),
        рб(тч(0,4),тч(5,5),5.09), рб(тч(2,4),тч(5,5),3.16)]
```

Приклад. Упорядкувати ребра із попереднього прикладу за неспаданням довжини.

```
goal Сп_тч=[тч(2,0),тч(2,5),тч(0,4),тч(2,4),тч(5,5)],
     повний_граф(Сп_тч,Сп_рб1),
     швидке_сортування(Сп_рб1,Сп_рб2).
```

Результат:
```
Сп_рб2=[рб(тч(2,5),тч(2,4),1), рб(тч(0,4),тч(2,4),2),
        рб(тч(2,5),тч(0,4),2.23), рб(тч(2,5),тч(5,5),3),
        рб(тч(2,4),тч(5,5),3.16), рб(тч(2,0),тч(2,4),4),
        рб(тч(2,0),тч(0,4),4.47), рб(тч(2,0),тч(2,5),5),
        рб(тч(0,4),тч(5,5),5.09), рб(тч(2,0),тч(5,5),5.83)]
```

Приклад. Дано точки *A*(2,0), *B*(2,5), *C*(0,4), *D*(2,4), *E*(5,5). Сформувати множини з кожної точки.

```
goal Сп_тч=[тч(2,0),тч(2,5),тч(0,4),тч(2,4),тч(5,5)],
     точки_множини(Сп_тч,Сп_сп_тч).
```

Результат: `Сп_сп_тч=[[тч(2,0)],[тч(2,5)],[тч(0,4)],[тч(2,4)],[тч(5,5)]]`

Розглянемо детальніше п. 4 предикату `старт`. Його реалізує відношення `мін_кістякове_дерево(Сп_сп_тч,Сп_рб2,Сп_рб)`, яке складається з трьох правил.

Правило 1. Якщо упорядкований список ребер `Сп_рб2` порожній, то список `Сп_рб` також буде порожнім.

Правило 2 опрацьовує випадок, коли голову списку `Сп_рб2` – ребро `рб(Тч1,Тч2,Дов_рб)` від точки `Тч1` до точки `Тч2` потрібно включити у мінімальне кістякове дерево. Для цього список множин `Сп_сп_тч` розбивається на множину (список точок) `Сп_тч1`, якому належить точка `Тч1`, та на список інших множин `Сп_сп_тч1` (це реалізовано предикатом `належить_множині_тч`). Якщо, аналогічно, цю операцію можна повторити для списку множин `Сп_сп_тч1` і точки `Тч2`, то розглядуване ребро належить різним множинам вершин, отже, не утворює циклу.

Правило 3. Інакше – голова списку ребер `Сп_рб2` не включається у мінімальне кістякове дерево.

Приклад. Побудуємо мінімальне кістякове дерево для множини точок *A*(2,0), *B*(2,5), *C*(0,4), *D*(2,4), *E*(5,5).

```
goal Сп_тч=[тч(2,0),тч(2,5),тч(0,4),тч(2,4),тч(5,5)],
     старт(Сп_тч,Мін_к_д).
```

Результат: `Мін_к_д=[рб(тч(2,5),тч(2,4),1), рб(тч(0,4),тч(2,4),2),`
`рб(тч(2,5),тч(5,5),3), рб(тч(2,0),тч(2,4),4)]`

Лістинг 2.3. Алгоритм Крускала побудови мінімального кістякового дерева.

```
DOMAINS
тч=тч(integer,integer)
сп_тч=тч*
сп_сп_тч=сп_тч*
рб=рб(тч,тч,real)
сп_рб=рб*

PREDICATES
старт(сп_тч,сп_рб)
повний_граф(сп_тч,сп_рб)
сп_рб_точки(тч,сп_тч,сп_рб)
обєднати_списки(сп_рб,сп_рб,сп_рб)
обєднати_списки(сп_тч,сп_тч,сп_тч)
швидке_сортування(сп_рб,сп_рб)
менше_більшерівно(рб,сп_рб,сп_рб,сп_рб)
точки_множини(сп_тч,сп_сп_тч)
мін_кістякове_дерево(сп_сп_тч,сп_рб,сп_рб)
належить_множині_тч(тч,сп_сп_тч,сп_тч,сп_сп_тч)
належить_списку(тч,сп_тч)
```

```
CLAUSES
старт(Сп_тч,Сп_рб):-
   повний_граф(Сп_тч,Сп_рб1),
   швидке_сортування(Сп_рб1,Сп_рб2),
   точки_множини(Сп_тч,Сп_сп_тч),
   мін_кістякове_дерево(Сп_сп_тч,Сп_рб2,Сп_рб),!.

повний_граф([_],[]):-!.
повний_граф([Тч|Сп_тч],Сп_рб):-
   повний_граф(Сп_тч,Сп_рб1),
   сп_рб_точки(Тч,Сп_тч,Сп_рб2),
   обєднати_списки(Сп_рб2,Сп_рб1,Сп_рб),!.

сп_рб_точки(_,[],[]):-!.
сп_рб_точки(Тч1,[Тч2|Сп_тч],[рб(Тч1,Тч2,Дов_рб)|Сп_рб]):-
   сп_рб_точки(Тч1,Сп_тч,Сп_рб),
   Тч1=тч(Аб1,Ор1),
   Тч2=тч(Аб2,Ор2),
   Дов_рб=sqrt((Аб1-Аб2)*(Аб1-Аб2)+(Ор1-Ор2)*(Ор1-Ор2)),!.

обєднати_списки([],Сп,Сп):-!.
обєднати_списки([Гл|Хв],Сп1,[Гл|Сп2]):-
   обєднати_списки(Хв,Сп1,Сп2),!.

швидке_сортування([],[]):-!.
швидке_сортування([Гл|Хв],Сп):-
   менше_більшерівно(Гл,Хв,Сп_м,Сп_бр),
   швидке_сортування(Сп_м,Сп_м1),
   швидке_сортування(Сп_бр,Сп_бр1),
   Сп1=[Гл|Сп_бр1],
   обєднати_списки(Сп_м1,Сп1,Сп),!.

менше_більшерівно(_,[],[],[]):-!.
менше_більшерівно(Рб,[Гл|Хв],[Гл|Сп_м],Сп_бр):-
   Рб=рб(_,_,Дов1),
   Гл=рб(_,_,Дов2),
   Дов1 > Дов2,
   менше_більшерівно(Рб,Хв,Сп_м,Сп_бр),!.
менше_більшерівно(Рб,[Гл|Хв],Сп_м,[Гл|Сп_бр]):-
   менше_більшерівно(Рб,Хв,Сп_м,Сп_бр),!.

точки_множини([],[]):-!.
точки_множини([Гл|Хв],[[Гл]|Сп_сп]):-
   точки_множини(Хв,Сп_сп),!.

мін_кістякове_дерево(_,[],[]):-!.
мін_кістякове_дерево(Сп_сп_тч,[рб(Тч1,Тч2,Дов_рб)|Сп_рб1],
[рб(Тч1,Тч2,Дов_рб)|Сп_рб2]):-
   належить_множині_тч(Тч1,Сп_сп_тч,Сп_тч1,Сп_сп_тч1),
```

```
   належить_множині_тч(Тч2,Сп_сп_тч1,Сп_тч2,Сп_сп_тч2),
   обєднати_списки(Сп_тч1,Сп_тч2,Сп_тч3),
   Сп_сп_тч3=[Сп_тч3|Сп_сп_тч2],
   мін_кістякове_дерево(Сп_сп_тч3,Сп_рб1,Сп_рб2),!.
мін_кістякове_дерево(Сп_сп_тч,[_|Сп_рб1],Сп_рб2):-
   мін_кістякове_дерево(Сп_сп_тч,Сп_рб1,Сп_рб2),!.

належить_множині_тч(Тч,[Гл|Хв],Гл,Хв):-
   належить_списку(Тч,Гл),!.
належить_множині_тч(Тч,[Гл|Хв],Сп_тч,[Гл|Сп_сп_тч]):-
   належить_множині_тч(Тч,Хв,Сп_тч,Сп_сп_тч),!.

належить_списку(Тч,[Тч|_]):-!.
належить_списку(Тч,[_|Хв]):-
   належить_списку(Тч,Хв),!.

GOAL
   старт([тч(2,0),тч(2,5),тч(0,4),тч(2,4),тч(5,5)],Мін_к_д).
```

Завдання для самостійної роботи.

1. Реалізувати алгоритм Прима побудови мінімального кістякового дерева:

Побудова починається з дерева, що включає в себе одну (довільну) вершину. Протягом роботи алгоритму дерево розростається, поки не охопить всі вершини початкового графа. На кожному кроці алгоритму до поточного дерева приєднується найлегше з ребер, що з’єднує вершину з побудованого дерева і вершину не з дерева.

2.4. Логічна задача «Місіонери та людоїди».

Завдання 2.4. На лівому березі річки знаходяться 3 місіонери та 3 людоїди. У їхньому розпорядженні 2-місний човен. Розробити план переправи їх на правий берег, при якому місіонерів завжди буде не менше за людоїдів.

Розв'язання. Для розв'язання задачі використовується структурний домен `стан(місіонери_зліва, людожери_зліва, місіонери_справа, людожери_справа, берег)` (параметр `берег` має значення –1 – зліва, 1 – справа). Результатом програми буде список як ланцюг станів від початкового до кінцевого, заданих умовою задачі.

Для уникнення зациклення використовується предикат `належить_списку(Стан, Стани)` (див. 1.6), де `Стан` – поточний стан, `Стани` – список пройдених станів. Для виведення плану переправи використовуються предикати `вивести` та `переправа`.

Для генерації можливих станів використовується недетермінований предикат `переплисти(Стан1,Стан2)`, у якому враховано місткість човна (може розміститися 1 або 2 людини). Даний предикат використовує відношення `перевірити`, яке перевіряє виконання заданих в умові обмежень.

Приклад. Вивести можливі переправи з початкового стану.

```
goal Стан1=стан(3,3,0,0,-1),
     переплисти(Стан1,Стан2).
```

Результат:
```
Стан2=стан(3,2,0,1,1)
Стан2=стан(3,1,0,2,1)
Стан2=стан(2,2,1,1,1)
```

Тобто, на правий берег може переплисти 1 людоїд, 2 людоїди або 1 людоїд і 1 місіонер.

Алгоритм головного предикату `розвязання`.

Правило 1. Якщо із початкового стану можна потрапити у кінцевий, то задача розв'язана.

Правило 2. Інакше: із початкового стану `Стан1` переплисти у `Стан3` такий, що не належить списку пройдених станів, та рекурсивно побудувати план переправи із стану `Стан3` до кінцевого стану `Стан2`.

У відношенні третій параметр містить список пройдених станів, а четвертий – вихідний, що повертає план переправи.

```
розвязання(Стан1,Стан2,Стани,[Стан2|Стани]):-
   переплисти(Стан1,Стан2),!.
розвязання(Стан1,Стан2,Стани,Стани1):-
   переплисти(Стан1,Стан3),
   not(належить_списку(Стан3,Стани)),
   розвязання(Стан3,Стан2,[Стан3|Стани],Стани1),!.
```

Приклад.

```
goal Стан1=стан(3,3,0,0,-1),Стан2=стан(0,0,3,3,1),
     розвязання(Стан1,Стан2,[Стан1],Стани),
     вивести(Стани).
```

Результат:
```
Перепливає 0 місіонер(и) та 2 людоїд(и) з лівого берега на правий
Перепливає 0 місіонер(и) та 1 людоїд(и) з правого берега на лівий
Перепливає 0 місіонер(и) та 2 людоїд(и) з лівого берега на правий
Перепливає 0 місіонер(и) та 1 людоїд(и) з правого берега на лівий
Перепливає 2 місіонер(и) та 0 людоїд(и) з лівого берега на правий
Перепливає 1 місіонер(и) та 1 людоїд(и) з правого берега на лівий
```

```
Перепливає 2 місіонер(и) та 0 людоїд(и) з лівого берега на правий
Перепливає 0 місіонер(и) та 1 людоїд(и) з правого берега на лівий
Перепливає 0 місіонер(и) та 2 людоїд(и) з лівого берега на правий
Перепливає 1 місіонер(и) та 0 людоїд(и) з правого берега на лівий
Перепливає 1 місіонер(и) та 1 людоїд(и) з лівого берега на правий
```

Лістинг 2.4. Логічна задача «Місіонери та людоїди».

```
DOMAINS
стан = стан(integer місіонери_зліва,integer людожери_зліва,integer
місіонери_справа,integer людожери_справа,integer берег)
стани=стан*

PREDICATES %списки та інтерфейс
належить_списку(стан,стани)
вивести(стани)
переправа(стан,стан)

CLAUSES
належить_списку(Ел,[Ел|_]):-!.
належить_списку(Ел,[_|Хв]):-
   належить_списку(Ел,Хв).

вивести([_]):-!.
вивести([Стан1,Стан2|Стани]):-
   вивести([Стан2|Стани]),
   переправа(Стан2,Стан1),!.

переправа(стан(М1,Л1,_,_,-1),стан(М3,Л3,_,_,1)):-
   М=М1-М3,Л=Л1-Л3,
   write("Перепливає ",М," місіонер(и) та ",Л," людоїд(и) з лівого
берега на правий" ),nl,!.
переправа(стан(_,_,М2,Л2,1),стан(_,_,М4,Л4,-1)):-
   М=М2-М4,Л=Л2-Л4,
   write("Перепливає ",М," місіонер(и) та ",Л," людоїд(и) з правого
берега на лівий" ),nl,!.

PREDICATES
nondeterm переплисти(стан,стан)
перевірити(integer,integer,integer,integer)
розвязання(стан,стан,стани,стани)


CLAUSES
переплисти(стан(М1,Л1,М2,Л2,Б1),стан(М3,Л1,М4,Л2,Б2)):-
   Б2=-Б1,М3=М1-1*Б2,М4=М2+1*Б2,
   перевірити(М3,Л1,М4,Л2).
переплисти(стан(М1,Л1,М2,Л2,Б1),стан(М1,Л3,М2,Л4,Б2)):-
   Б2=-Б1,Л3=Л1-1*Б2,Л4=Л2+1*Б2,
   перевірити(М1,Л3,М2,Л4).
```

```
переплисти(стан(М1,Л1,М2,Л2,Б1),стан(М3,Л1,М4,Л2,Б2)):-
   Б2=-Б1,М3=М1-2*Б2,М4=М2+2*Б2,
   перевірити(М3,Л1,М4,Л2).
переплисти(стан(М1,Л1,М2,Л2,Б1),стан(М1,Л3,М2,Л4,Б2)):-
   Б2=-Б1,Л3=Л1-2*Б2,Л4=Л2+2*Б2,
   перевірити(М1,Л3,М2,Л4).
переплисти(стан(М1,Л1,М2,Л2,Б1),стан(М3,Л3,М4,Л4,Б2)):-
   Б2=-Б1,М3=М1-1*Б2,М4=М2+1*Б2,Л3=Л1-1*Б2,Л4=Л2+1*Б2,
   перевірити(М3,Л3,М4,Л4).

перевірити(М1,Л1,М2,Л2):-
   М1>=0,Л1>=0,М2>=0,Л2>=0,
   М1>=Л1,М2>=Л2,!.
перевірити(0,Л1,М2,Л2):-
   Л1>=0,М2>=0,Л2>=0,
   М2>=Л2,!.
перевірити(М1,Л1,0,Л2):-
   М1>=0,Л1>=0,Л2>=0,
   М1>=Л1,!.

розвязання(Стан1,Стан2,Стани,[Стан2|Стани]):-
   переплисти(Стан1,Стан2),!.
розвязання(Стан1,Стан2,Стани,Стани1):-
   переплисти(Стан1,Стан3),
   not(належить_списку(Стан3,Стани)),
   розвязання(Стан3,Стан2,[Стан3|Стани],Стани1),!.

GOAL
   Стан1=стан(3,3,0,0,-1),Стан2=стан(0,0,3,3,1),
   розвязання(Стан1,Стан2,[Стан1],Стани),
   вивести(Стани).
```

Завдання для самостійної роботи.

1. Модифікувати задачу для використання *n*-місного човна (*n* = 2, 3).
2. Дослідити, яку найбільшу кількість місіонерів та людоїдів можна переправити з використанням *n*-місного човна.

2.5. Логічна гра «Бики-корови».

Завдання 2.5. Зробити програмну реалізацію гри «Бики-корови», яка полягає у наступному:

1) користувач задумує 4-значне число з різними цифрами;
2) програма пропонує деяке 4-значне число;
3) користувач вказує кількість «биків» – цифр, що входять у задумане число та співпадають позиціями, та «корів» – цифр, що входять у задумане число але не співпадають позиціями.

Кроки 2-3 повторюються, доки програма не відгадає задумане число.

Розв'язання. Для формування числа використовується недетермінований предикат `цифра`, який буде повертати цифру 1, 2, …, 9, 0. У результаті буде формуватися хід програми у вигляді структури `хід([А,Б,В,Г])`, де `[А,Б,В,Г]` – список, що відображає число. Для збереження інформації про зроблені ходи використовується предикат динамічної бази даних `пройдений_хід(результат)`, де структура `результат=рез(хід([А,Б,В,Г]), Бики, Корови)` також містить інформацію про кількість «биків» та «корів», отриманих від користувача.

При розв'язанні задачі використовуються декілька предикатів опрацювання списків: `належить_списку`, `надрукувати_список` (див. 1.3), `кіл_точних_співпадінь` (кількість «биків»), `кіл_неточних_співпадінь` (кількість «корів»). Розглянемо два останні детальніше.

```
кіл_точних_співпадінь([],[],0):-!.
кіл_точних_співпадінь([Гол|Хв1],[Гол|Хв2],Кіл):-!,
   кіл_точних_співпадінь(Хв1,Хв2,Кіл1),Кіл=Кіл1+1,!.
кіл_точних_співпадінь([_|Хв1],[_|Хв2],Кіл):-
   кіл_точних_співпадінь(Хв1,Хв2,Кіл),!.
```

При виклику предиката `кіл_точних_співпадінь(Сп1,Сп2,Кільк)` вхідними параметрами є списки `Сп1` та `Сп2`, а вихідним `Кільк` – кількість «биків».

Приклад.

```
goal кіл_точних_співпадінь([1,2,3,4],[1,4,3,5],Кільк).
```

Результат: `Кільк=2` (елементи 1 і 3).

На відміну від попереднього відношення `кіл_неточних_співпадінь(Сп1, Сп2, Сп2, Кільк)` має три вхідні параметри-списки. При рекурсивному виклику даного предиката, у перших двох списках відбувається розбиття на голову і хвіст, тоді як третій параметр залишається без зміни.

```
кіл_неточних_співпадінь([],[],_,0):-!.
кіл_неточних_співпадінь([Гол|Хв1],[Гол|Хв2],Сп,Кіл):-!,
   кіл_неточних_співпадінь(Хв1,Хв2,Сп,Кіл),!.
кіл_неточних_співпадінь([Гол|Хв1],[_|Хв2],Сп,Кіл):-
   належить_списку(Гол,Сп),!,
   кіл_неточних_співпадінь(Хв1,Хв2,Сп,Кіл1),Кіл=Кіл1+1,!.
кіл_неточних_співпадінь([_|Хв1],[_|Хв2],Сп,Кіл):-
   кіл_неточних_співпадінь(Хв1,Хв2,Сп,Кіл),!.
```

Приклад.

```
goal
   кіл_неточних_співпадінь([1,2,3,4],[1,4,3,5],Кільк).
```

Результат: `Кільк=1` (елемент 4).

Розглянемо алгоритм розв'язання задачі, який представлений предикатом `старт`.

```
старт:-
   сформувати(Хід),
   перевірити(Хід),
   отримати_результат(Хід,Бики,Корови),
   добавити_у_бд(Хід,Бики,Корови),
   пройдений_хід(рез(_,4,0)).
```

1) викликається недетермінований предикат `сформувати(Хід)`, який послідовно перебирає можливі комбінації цифр:

```
Хід=хід([1,2,3,4])
Хід=хід([1,2,3,5])
...
Хід=хід([0,9,8,7])
```

2) предикат `перевірити(Хід)` виконується успішно, якщо сформований `Хід` задовольняє всі збережені у базі даних попередні результати. Тобто формується такий `Хід`, у якого кількість «биків» та «корів» відповідає попереднім `Ходам`;
3) предикат `отримати_результат(Хід,Бики,Корови)` організовує інтерфейс з користувачем: виводить число та зчитує кількість «биків» та «корів»;
4) далі у базі даних зберігається інформація про зроблений `Хід`;
5) гра закінчується, коли відгадано всі цифри.

Приклад. Нехай задумане число 4739.

```
Число: 1234
   Биків: 1
   Корів: 1
Число: 1356
   Биків: 0
   Корів: 1
Число: 2574
   Биків: 0
   Корів: 2
Число: 3287
   Биків: 0
   Корів: 2
Число: 4739
   Биків: 4
   Корів: 0
```

Лістинг 2.5. Логічна гра «Бики-корови».

```
DOMAINS
сп=integer*
хід=хід(сп)
результат=рез(хід,integer,integer)
результати=результат*

DATABASE
пройдений_хід(результат)
```

```
PREDICATES
належить_списку(integer,сп)
надрукувати_список(сп)
кіл_точних_співпадінь(сп,сп,integer)
кіл_неточних_співпадінь(сп,сп,сп,integer)
CLAUSES
належить_списку(Ел,[Ел|_]):-!.
належить_списку(Ел,[_|Хв]):-
   належить_списку(Ел,Хв).

надрукувати_список([]):-nl,!.
надрукувати_список([Гол|Хв]):-
   write(Гол),надрукувати_список(Хв),!.

%(Список1,Список2,Кількість)
%Визначає скільки елементів зі Список1 належать Список2
%та розміщені на однаковій позиції.
кіл_точних_співпадінь([],[],0):-!.
кіл_точних_співпадінь([Гол|Хв1],[Гол|Хв2],Кіл):-!,
   кіл_точних_співпадінь(Хв1,Хв2,Кіл1),
   Кіл=Кіл1+1,!.
кіл_точних_співпадінь([_|Хв1],[_|Хв2],Кіл):-
   кіл_точних_співпадінь(Хв1,Хв2,Кіл),!.

%(Список1,Список2,Список2,Кількість)
%Визначає скільки елементів зі Список1 належать Список2,
%але розміщені не на однаковій позиції.
кіл_неточних_співпадінь([],[],_,0):-!.
кіл_неточних_співпадінь([Гол|Хв1],[Гол|Хв2],Сп,Кіл):-!,
   кіл_неточних_співпадінь(Хв1,Хв2,Сп,Кіл),!.
кіл_неточних_співпадінь([Гол|Хв1],[_|Хв2],Сп,Кіл):-
   належить_списку(Гол,Сп),!,
   кіл_неточних_співпадінь(Хв1,Хв2,Сп,Кіл1),Кіл=Кіл1+1,!.
кіл_неточних_співпадінь([_|Хв1],[_|Хв2],Сп,Кіл):-
   кіл_неточних_співпадінь(Хв1,Хв2,Сп,Кіл),!.
PREDICATES
nondeterm цифра(integer)
nondeterm старт
nondeterm сформувати(хід)
перевірити(хід)
перевірити_всі(сп,результати)
отримати_результат(хід,integer,integer)
добавити_у_бд(хід,integer,integer)

CLAUSES
цифра(1). цифра(2). цифра(3).
цифра(4). цифра(5). цифра(6).
цифра(7). цифра(8). цифра(9). цифра(0).
```

```
старт:-
   сформувати(Хід),
   перевірити(Хід),
   отримати_результат(Хід,Бики,Корови),
   добавити_у_бд(Хід,Бики,Корови),
   пройдений_хід(рез(_,4,0)).

сформувати(хід([А,Б,В,Г])):-
   цифра(А),цифра(Б),А<>Б,
   цифра(В),А<>В,Б<>В,
   цифра(Г),А<>Г,Б<>Г,В<>Г.

перевірити(хід(Сп)):-
   findall(Рез,пройдений_хід(Рез),СпРез),
   перевірити_всі(Сп,СпРез),!.

перевірити_всі(_,[]):-!.
перевірити_всі(Сп,[рез(хід(Сп1),Бики,Корови)|Хв]):-
   кіл_точних_співпадінь(Сп,Сп1,Бики),
   кіл_неточних_співпадінь(Сп,Сп1,Сп1,Корови),
   перевірити_всі(Сп,Хв),!.

отримати_результат(хід(Список),Бики,Корови):-
   write("Число: "),надрукувати_список(Список),
   write("Биків: "),readint(Бики),
   write("Корів: "),readint(Корови),!.

добавити_у_бд(Хід,Бики,Корови):-
   assert(пройдений_хід(рез(Хід,Бики,Корови))),!.

GOAL
   старт.
```

Завдання для самостійної роботи.

1. Модифікувати предикат цифра, з метою формування числа випадковим чином.
2. Розробити програму для *n*-значного числа (*n* = 3, 4, 5).

2.6. Логічна головоломка Судоку.

Завдання 2.6. Написати програму розв'язання головоломки Судоку, що має такі правила. Ігрове поле складається з квадрата, розміром $n^2 \times n^2$, розділеного на менші квадрати $n \times n$ клітинок. У деяких з них вже на початку гри розташовані числа від 1 до n^2. Мета головоломки – необхідно заповнити вільні клітинки цифрами від 1 до n^2 так, щоб в кожному рядку, в кожному стовпці і в кожному малому квадраті кожна цифра зустрічалася лише один раз.

Розв'язання.

І спосіб. Для Судоку розмірності $n = 2$ використаємо повний перебір. Знайдені значення будемо зберігати у змінних `Xij`, ($i, j = 1, ..., n^2$). Деякі з них, що задані в умові завдання, будуть визначені до перебору. Інші генеруються предикатом `перестановка` (див. 1.3). Предикат `надрукувати` виводить списки зі змінними на екран.

Зазначимо, що всього перестановок з 4-х елементів є 4! = 24. Оскільки перестановки формуються для кожного рядка, стовпця і малого квадрата, то програма перебирає 24^{12} варіантів.

Програма, приведена нижче, демонструє розв'язання такого Судоку (0 – невідоме число):

```
1030
0400
0003
4000
```

Лістинг 2.6.1. Програма Судоку ($n = 2$).

```
DOMAINS
сп = reference integer*

PREDICATES
nondeterm видалити(integer,сп,сп)
nondeterm перестановка(сп,сп)
надрукувати(сп)

CLAUSES
видалити(Гол,[Гол|Хв],Хв).
видалити(Ел,[Гол|Хв],[Гол|Хв1]):-
   видалити(Ел,Хв,Хв1).

перестановка([],[]).
перестановка(Сп,[Гол|Хв]) :-
   видалити(Гол,Сп,Сп1),
   перестановка(Сп1,Хв).

надрукувати([]):-nl,!.
надрукувати([Гол|Хв]):-
   write(Гол),надрукувати(Хв),!.

PREDICATES
nondeterm судоку4
```

```
CLAUSES
судоку4:-
   X11=1,      X13=3,
         X22=4,
                     X34=3,
   X41=4,
   перестановка([1,2,3,4],[X11,X12,X13,X14]),
   перестановка([1,2,3,4],[X21,X22,X23,X24]),
   перестановка([1,2,3,4],[X31,X32,X33,X34]),
   перестановка([1,2,3,4],[X41,X42,X43,X44]),
   перестановка([1,2,3,4],[X11,X21,X31,X41]),
   перестановка([1,2,3,4],[X12,X22,X32,X42]),
   перестановка([1,2,3,4],[X13,X23,X33,X43]),
   перестановка([1,2,3,4],[X14,X24,X34,X44]),
   перестановка([1,2,3,4],[X11,X12,X21,X22]),
   перестановка([1,2,3,4],[X13,X14,X23,X24]),
   перестановка([1,2,3,4],[X31,X32,X41,X42]),
   перестановка([1,2,3,4],[X33,X34,X43,X44]),
   надрукувати([X11,X12,X13,X14]),
   надрукувати([X21,X22,X23,X24]),
   надрукувати([X31,X32,X33,X34]),
   надрукувати([X41,X42,X43,X44]),nl,fail.
судоку4.

GOAL
   судоку4.
```

В результаті отримаємо два розв'язки:

```
1234
3412
2143
4321


1234
3421
2143
4312
```

II спосіб. Для Судоку розмірності $n = 3$ повний перебір не дає результату через велику кількість варіантів ($9!^{27}$). Тому будемо використовувати керований перебір варіантів.

Відмітимо, що для зменшення лістингу програми використана директива `include` для підключення предикатів роботи зі списками (`кіл_елементів, видалити`) та множинами (`обєднання, різниця`).

Схема розв'язання задачі. Як задані числа, так і обчислені будемо зберігати у вигляді фактів бази даних `ч(Рядок,Стовпець,Число)`. Для вільних позицій будуть формуватися кандидати – числа, які не співпадають із заданими у відповідних рядках, стовпцях і малих квадратах. Вони будуть зберігатися у вигляді фактів бази даних `к(Рядок,Стовпець, СписокКандидатів)`.

Предикати `читати(9)` та `надрукувати_числа(9)` відповідно зчитують та друкують 9 рядків по 9 чисел у кожному. При зчитуванні невідомі числа позначаються нулями, відомі добавляються у базу даних.

Головний предикат – `судоку` – ітераційно шукає розв'язок задачі.

Правило 1:

1) видаляються всі кандидати попередньої ітерації;
2) формується новий набір кандидатів;
3) серед всіх наборів п. 2 вибирається набір (список) мінімальної розмірності (тобто список з одного, двох, трьох … елементів);
4) використовуючи недетермінований предикат `видалити` (див. 1.3) вибирається елемент списку з п. 3;
5) елемент з п. 4 добавляється у базу даних;
6) рекурсивно запускається наступна ітерація.

До п. 2. Формуванням набору кандидатів займаються предикати:

`кандидати` – проглядає всі рядки;

`кандидати1` – для заданого рядка переглядає всі елементи. Якщо для елемента з координатами `(Ряд,Стовпець)` у базі даних відсутній факт `ч(Ряд,Стовпець,_)`, то для цього елемента за допомогою вбудованого предиката `findall` формуються списки з чисел, що знаходяться з ним на одному рядку, стовпці та малому квадраті (використовується недетермінований предикат `число_квадрата`). За допомогою предикатів `обєднання` та `різниця` (див. 1.6) будується список кандидатів.

`кандидати2` – добавляє у базу даних непорожній список кандидатів.

До п. 3. Предикат `мін_набір_канд` повертає список спочатку з одного, потім з двох, а потім з трьох елементів (що достатньо для практичних задач).

До п. 5. Недетермінований предикат `добавити_число` добавляє факт у базу даних, використовуючи її як стек. Справа у тому, що кандидат з п. 4, може бути вибраний помилково. Це призведе до невдачі на наступних ітераціях і, відповідно, до відкату. Отже, факт, добавлений у п. 5 у базу даних, буде видалений.

Правило 2. Якщо знайдені всі 81 числа, то задача розв'язана.

Лістинг 2.6.2. Програма Судоку (*n* = 3).

```
include "d:\\Prolog\\Множини.PRO"
include "d:\\Prolog\\Список.PRO"
%domains
%сп = integer*

DATABASE
ч(integer,integer,integer)
к(integer,integer,сп)

PREDICATES
nondeterm судоку
кандидати(integer)
кандидати1(integer,integer)
кандидати2(integer,integer,сп)
nondeterm мін_набір_канд(integer,integer,сп)
nondeterm добавити_число(integer,integer,integer)
nondeterm число_квадрата(integer,integer,integer)
```

```
CLAUSES
судоку:-
   retractall(к(_,_,_)),
   кандидати(9),
   мін_набір_канд(Ряд,Стовпець,Числа),
   видалити(Число,Числа,_),% Елемент списку
   добавити_число(Ряд,Стовпець,Число),
   судоку.
судоку:-
   findall(Р,ч(Р,_,_),Всі),
   кіл_елементів(Всі,81),!.

кандидати(0):-!.
кандидати(Ряд):-
   Ряд1=Ряд-1,
   кандидати(Ряд1),
   кандидати1(Ряд,9),!.

кандидати1(_,0):-!.
кандидати1(Ряд,Стовпець):-
   ч(Ряд,Стовпець,_),
   Стовпець1=Стовпець-1,
   кандидати1(Ряд,Стовпець1),!.
кандидати1(Ряд,Стовпець):-
   Стовпець1=Стовпець-1,
   кандидати1(Ряд,Стовпець1),
   findall(Число,ч(Ряд,_,Число),ЧР),
   findall(Число,ч(_,Стовпець,Число),ЧС),
   findall(Число,число_квадрата(Ряд,Стовпець,Число),ЧК),
   обєднання(ЧР,ЧС,Множина1),
   обєднання(Множина1,ЧК,Множина2),
   різниця([1,2,3,4,5,6,7,8,9],Множина2,Множина3),
   кандидати2(Ряд,Стовпець,Множина3),!.

кандидати2(_,_,[]):-!. % Нема кандидатів
кандидати2(Ряд,Стовпець,Числа):-
   assertz(к(Ряд,Стовпець,Числа)),!. % 1,2 і більше кандидатів

мін_набір_канд(Ряд,Стовпець,Числа):-
   к(Ряд,Стовпець,Числа),
   Числа=[_].
мін_набір_канд(Ряд,Стовпець,Числа):-
   к(Ряд,Стовпець,Числа),
   Числа=[_,_].
мін_набір_канд(Ряд,Стовпець,Числа):-
   к(Ряд,Стовпець,Числа),
   Числа=[_,_,_].
```

```
добавити_число(Ряд,Стовпець,Число):-
    assertz(ч(Ряд,Стовпець,Число)).
добавити_число(Ряд,Стовпець,Число):-
    retract(ч(Ряд,Стовпець,Число)),fail.

число_квадрата(Р,С,Число):-
   ч(Ряд,Стовпець,Число),
   Ряд>=((Р-1)div 3)*3+1,
   Ряд<=((Р-1)div 3)*3+3,
   Стовпець>=((С-1)div 3)*3+1,
   Стовпець<=((С-1)div 3)*3+3.

PREDICATES %Інтерфейс
читати(integer)
задані_числа(integer,integer,string)
надрукувати_числа(integer)
надрукувати1(integer,integer)

CLAUSES
читати(0):-!.
читати(N):-
   readln(S),
   N1=10-N,
   задані_числа(N1,1,S),
   N2=N-1,
   читати(N2),!.

задані_числа(_,_,""):-!.
задані_числа(N,P,S):-
   frontstr(1,S,C,S1),
   str_int(C,Z),
   Z<>0,
   assert(ч(N,P,Z)),
   P1=P+1,
   задані_числа(N,P1,S1),!.
задані_числа(N,P,S):-
   frontstr(1,S,_,S1),
   P1=P+1,
   задані_числа(N,P1,S1),!.

надрукувати_числа(0):-!.
надрукувати_числа(Ряд):-
   Ряд1=Ряд-1,
   надрукувати_числа(Ряд1),
   надрукувати1(Ряд,9),!.
```

```
надрукувати1(_,0):-nl,!.
надрукувати1(Ряд,Стовпець):-
   ч(Ряд,Стовпець,Число),
   Стовпець1=Стовпець-1,
   надрукувати1(Ряд,Стовпець1),
   write(Число),!.
надрукувати1(Ряд,Стовпець):-
   Стовпець1=Стовпець-1,
   надрукувати1(Ряд,Стовпець1),
   write(0),!.

GOAL читати(9),
   судоку,
   надрукувати_числа(9),nl.
```

Приклад. Розв'язати «найскладніший судоку» (див. http://crosswords-world.net/mini-blog/sudoku/1833/)

```
005300000
800000020
070010500
400005300
010070006
003200080
060500009
004000030
000009700
```

Результат:

```
145327698
839654127
672918543
496185372
218473956
753296481
367542819
984761235
521839764
```

Завдання для самостійної роботи.

1. Розробити правила, що зменшують перебір (наприклад, якщо у рядку серед всіх кандидатів певне число зустрічається тільки один раз, то його слід добавити у базу даних).

2.7. Гра «Словесний ланцюг».

Завдання 2.7. Написати програму побудови мінімального ланцюга між двома заданими словами однакової довжини, проміжними ланками якого є слова, що відрізняються одне від одного однією літерою.

Розв'язання.

І спосіб. Реалізуємо алгоритм пошуку у глибину. У програмі використовуються факти динамічної бази даних `слово` (зчитується із зовнішнього файлу) та `мінімальний_ланцюг` для запам'ятовування поточного найкоротшого ланцюга.

Схема розв'язання. Розглянемо цільовий предикат `старт`.

Правило 1.

1) задається початкове значення факту `мінімальний_ланцюг` та зчитуються із файлу `слова`;
2) задаються початок ланцюга (`Слово1`) і його кінець (`Слово2`);
3) будується ланцюг від `Слово2` до `Слово1` (для зручності перегляду – адже ланцюг (список) будується з «права» на «ліво»);
4) обчислюється кількість слів ланцюга – довжина списку (див. 1.3);
5) перевіряється знайдене значення на мінімум (див. 2.1);
6) ініціюється відкат для перевірки інших можливих ланцюгів.

Правило 2 використовується для успішного завершення предикату `старт` та виведення результатів.

До п. 3. У предикаті `побудувати_ланцюг` використовується третій параметр – список пройдених слів – для уникнення зациклення. Предикат також використовує відношення `відстань`, яке знаходить «відстань» між двома словами – кількість неспівпадаючих літер.

Лістинг 2.7.1. Гра «Словесний ланцюг» (алгоритм пошуку у глибину).

```
DOMAINS
слова=symbol*

DATABASE
мінімальний_ланцюг(слова,integer)
слово(symbol)

PREDICATES
старт
задати_початкове_значення
відстань(symbol,symbol,integer)
nondeterm побудувати_ланцюг(symbol,symbol,слова,слова)
належить_списку(symbol,слова)
кількість_слів(слова,integer)
перевірка_на_мінімум(слова,integer)

CLAUSES
старт:-
   задати_початкове_значення,
   Слово1="шах",Слово2="мат",
   побудувати_ланцюг(Слово2,Слово1,[Слово2],Ланцюг),
   % Будуємо ланцюг між словами
   write(Ланцюг),nl,
   кількість_слів(Ланцюг,КількістьСлів),
   % Знаходимо кількість слів у маршруті
```

```
   перевірка_на_мінімум(Ланцюг,КількістьСлів),
   % Зберігаємо у базі даних, якщо ланцюг мінімальний
   fail.
старт:-
   retract(мінімальний_ланцюг(Ланцюг,_)),
   write("Мінімальний ланцюг:"),nl,write(Ланцюг),nl,!.

задати_початкое_значення:-
   retractall(мінімальний_ланцюг(_,_)),
   assert(мінімальний_ланцюг([],1000)),
   consult("d:\\slova.txt"),!.

% Предикат побудови ланцюга між словами (i,i,i,o).
% ПобудованийЛанцюг використовується для уникнення циклів
побудувати_ланцюг(Сл1,Сл2,Ланцюг,[Сл2|Ланцюг]):-
   відстань(Сл1,Сл2,1).
   % Якщо відстань між словами = 1, то ланцюг побудовано
побудувати_ланцюг(Сл1,Сл2,Ланцюг1,Ланцюг2):-
   відстань(Сл1,Сл2,Відстань),Відстань<>1,
   % Якщо відстань між словами <> 1
   слово(Сл3),відстань(Сл1,Сл3,1),
   % Шукаємо нове Слово3, віддалене від Слова1 на 1
   not(належить_списку(Сл3,Ланцюг1)),
   % Слово3 не повинно належати побудованому ланцюгу
   побудувати_ланцюг(Сл3,Сл2,[Сл3|Ланцюг1],Ланцюг2).
   % Будуємо ланцюг між Словом3 і Словом2

% Відстань між словами однакової довжини (кільк. неспівпадаючих букв)
відстань("","",0):-!.
відстань(Слово1,Слово2,Відстань):-
   frontchar(Слово1,Буква,Остаток1),
   frontchar(Слово2,Буква,Остаток2),!,
   % Слова починаються з однакової букви
   відстань(Остаток1,Остаток2,Відстань),!.
відстань(Слово1,Слово2,Відстань):-
   frontchar(Слово1,_,Остаток1),
   frontchar(Слово2,_,Остаток2),!,
   % Слова починаються з різних букв
   відстань(Остаток1,Остаток2,Відстань1),
   Відстань=Відстань1+1,!.

% Елемент належить списку, якщо належить голові або хвосту списку
належить_списку(Голова,[Голова|_]):-!.
належить_списку(Голова,[_|Хвіст]):-
   належить_списку(Голова,Хвіст),!.

% Кількість слів у списку рівна кількості слів хвоста списку + 1
кількість_слів([],0):-!.
```

```
кількість_слів([_|Хвіст],КількістьСлів):-
   кількість_слів(Хвіст,КількістьСлів1),
   КількістьСлів=КількістьСлів1+1,!.

перевірка_на_мінімум(_,КількістьСлів):-
   мінімальний_ланцюг(_,КількістьСлів1),
   КількістьСлів>=КількістьСлів1,!.
   % Якщо у БД збережений мінімальний ланцюг
перевірка_на_мінімум(Ланцюг,КількістьСлів):-
   % Інакше - замінити
   retractall(мінімальний_ланцюг(_,_)),
   assert(мінімальний_ланцюг(Ланцюг,КількістьСлів)),!.

GOAL
   старт.
```

Результат:

```
["шах","шар","цар","жар","вар","вир","сир","син","чин","чан","хан",
"лан","пан","пат","мат"]
```

… і ще 99 «ланцюгів».

```
Мінімальний ланцюг:
["шах","шар","вар","вир","сир","син","чин","чан","пан","пат","мат"]
```

II спосіб. Реалізуємо алгоритм пошуку у ширину.

Схема розв'язання. Розглянемо цільовий предикат `старт`.

1) зчитуються із файлу `слова`;
2) задаються початок ланцюга (`Слово1`) і його кінець (`Слово2`);
3) будується ланцюг;
4) виводяться результати.

До п. 3. У предикаті `пошук_у_ширину` третій параметр `[[Слово2]]` – це список ланцюгів (списків), які будуються від `Слово2`. Для цього використовується недетермінований предикат `слово_сусід`, який для заданого слова шукає його сусіда (що відрізняється однією літерою), а всіх сусідів у список записує вбудований предикат `findall`. Для побудови списку ланцюгів використовується відношення `сформувати`. Предикат `обєднати` об'єднує списки ланцюгів (див. 1.6). Цей процес рекурсивно продовжується доки не буде знайдено перший ланцюг між `Слово1` та `Слово2`. Із алгоритму пошуку у ширину випливає, що цей ланцюг буде найкоротшим.

Приклад. Знайти слова-сусіди до слова «сир».

```
goal consult("d:\\slova.txt"),
     Слово1="сир",
     слово_сусід(Слово1,Слово2).
```

Результат:

```
Слово2=вир
Слово2=мир
Слово2=син
```

Приклад. Продовжити список ланцюгів для ланцюга ["сир","син","чин"].

```
goal consult("d:\\slova.txt"),
     Ланц=["сир","син","чин"],
     Ланц=[Сл2|_],
     findall(Сл3,слово_сусід(Сл2,Сл3),СпСусідів),
     сформувати(СпСусідів,Ланц,СпЛанц).
```

Результат: СпСусідів=["вир","мир","син"]
СпЛанц=[["вир","сир","син","чин"],["мир","сир","син","чин"]]

Лістинг 2.7.2. Гра «Словесний ланцюг» (алгоритм пошуку у ширину).

```
DOMAINS
слова=symbol*
сп_слів=слова*
DATABASE
слово(symbol)
PREDICATES
старт
пошук_у_ширину(symbol,сп_слів,слова)
сформувати(слова,слова,сп_слів)
обєднати(сп_слів,сп_слів,сп_слів)
nondeterm слово_сусід(symbol,symbol)
відстань(symbol,symbol,integer)
належить_списку(symbol,слова)

CLAUSES
старт:-
   consult("d:\\slova.txt"),
   Слово1="шах",Слово2="мат",
   пошук_у_ширину(Слово1,[[Слово2]],Ланцюг),
   write(Ланцюг),nl.

% (i,i,o) Сл1 - Слово, до якого будуємо ланцюг,
% [[Сл2|Хв]|_] - Список ланцюгів, [Сл1,Сл2|Хв] - Ланцюг-результат
пошук_у_ширину(Сл1,[[Сл2|Хв]|_],[Сл1,Сл2|Хв]):-
   відстань(Сл1,Сл2,1),!.
   % Якщо відстань = 1 - ланцюг побудовано
пошук_у_ширину(Сл1,[[Сл2|Хв]|СпЛанц],Рез):-
   findall(Сл3,слово_сусід(Сл2,Сл3),СпСусідів),
   % Формуємо список сусідів
   сформувати(СпСусідів,[Сл2|Хв],СпЛанц1),
   % Формуємо список ланцюгів - включаємо всіх нових сусідів
   % (без пройдених)
   обєднати(СпЛанц,СпЛанц1,СпЛанц2),
   пошук_у_ширину(Сл1,СпЛанц2,Рез),!.
   % Продовжити пошук

сформувати([],_,[]):-!.
сформувати([Гол|Хв],Сп,[[Гол|Сп]|Рез]):-
   not(належить_списку(Гол,Сп)),
```

```
   сформувати(Хв,Сп,Рез),!.
сформувати([_|Хв],Сп,Рез):-
   сформувати(Хв,Сп,Рез),!.

обєднати([],Сп,Сп):-!.
обєднати([Гол|Хв],Сп,[Гол|Сп1]):-
   обєднати(Хв,Сп,Сп1),!.

% Слово1 сусід до Слово2, якщо відстань = 1 (і,о).
слово_сусід(Слово1,Слово2):-
   слово(Слово2),
   відстань(Слово1,Слово2,1).

% Відстань між словами однакової довжини (кільк. неспівпадаючих букв)
відстань("","",0):-!.
відстань(Слово1,Слово2,Відстань):-
   frontchar(Слово1,Буква,Остаток1),
   frontchar(Слово2,Буква,Остаток2),!,
   % Слова починаються з однакової букви
   відстань(Остаток1,Остаток2,Відстань),!.
відстань(Слово1,Слово2,Відстань):-
   frontchar(Слово1,_,Остаток1),
   frontchar(Слово2,_,Остаток2),!,
   % Слова починаються з різних букв
   відстань(Остаток1,Остаток2,Відстань1),
   Відстань=Відстань1+1,!.

% Елемент належить списку, якщо належить голові або хвосту списку
належить_списку(Голова,[Голова|_]):-!.
належить_списку(Голова,[_|Хвіст]):-
   належить_списку(Голова,Хвіст),!.

GOAL
   старт.
```

Результат:

```
["шах","шар","вар","вир","сир","син","чин","чан","пан","пат","мат"]
```

Завдання для самостійної роботи.

1. Написати програму побудови мінімального ланцюга між двома заданими словами різної довжини. При цьому словами-сусідами будуть як слова, що відрізняються одне від одного однією літерою, так і слова, у одного з яких є додаткова літера (гра-гора).

ПРОЕКТИ ДЛЯ ІНДИВІДУАЛЬНОЇ РОБОТИ

1. Розробити програму пошуку, добавлення та знищення елементів у AVL-дереві.
2. Розробити програму знаходження похідної функції, заданої у символьному вигляді.
3. Розробити програму знаходження розв’язку задачі Ейнштейна.
4. Розробити програму автоматичного доведення теорем алгебри логіки.
5. Розробити програму для гри у шаховому ендшпілі (король і тура проти короля).
6. Розробити програму для гри у шаховому ендшпілі (король і пішак проти короля).
7. Розробити програму імітації спілкування з комп’ютером на природній мові.
8. Розробити програму, що дозволяє робити запити до бази даних на природній мові.
9. Розробити програму, що робить семантичний аналіз речень української мови.
10. Розробити програму, що моделює роботу експерта у певній галузі.

ЛІТЕРАТУРА

1. Адаменко А., Кучуков А. Логическое программирование и Visual Prolog (с CD). – СПб.:«БХВ-Петербург», 2003. – С. 990.
2. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG = Prolog Programming For Artificial Intelligence. – М.: Вильямс, 2004. – 640 с.
3. Братко И. Программирование на языке Пролог для искусственного интеллекта. Пер. с англ. – М.: Мир, 1990. – 560 с.
4. Братко И. Алгоритмы искусственного интеллекта на языке Prolog, 3-е издание. : Пер. с англ. — М. : Издательский дом "Вильямс", 2004. — 640 с.
5. Малпас Дж. Реляционный язык Пролог и его применение. – М.: Наука, 1990. – 465 с.
6. Трушевський В. М. Технології та мови програмування для штучного інтелекту. Частина 1: Основи програмування мовою Prolog. – Львів, 2006.
7. Экспертные системы. Под. ред. Р. Форсайта. – М.: Радио и связь, 1987.
8. Янсон А. Турбо-Пролог в сжатом изложении. – М.: Мир, 1991.
9. www.visual-prolog.com

Підписано до друку 15.01.2013. Формат 60x84/16.
Гарнітура TimesNewRoman. Ум. друк. арк. 2,5.
Наклад 100 прим. Віддруковано на різографі.

Видавництво УжНУ «Говерла»
88000, м. Ужгород, вул. Капітульна, 18.
Свідоцтво про внесення до державного реєстру видавців виготівників, і розповсюджувачів видавничої продукції
Серія Зт №32 від 31 травня 2006 року