Государственный комитет Российской федерации по связи и информатизации
Сибирский государственный университет телекоммуникаций и информатики

М.С.Тарков

Программирование на Турбо-Прологе

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Новосибирск
1999

Тарков М.С., к.т.н.
Методические указания предназначены для студентов инженерно-технических факультетов, изучающих логическое программирование в 4-м семестре. Они содержат необходимые сведения по программированию на языке Турбо-Пролог, задачи для контрольных работ №1 - №6 и рекомендуемую литературу.

Кафедра прикладной математики и кибернетики.
Для специальности 2305, 2306, 2307.
Список литературы - 3 наименования.

Рецензент: Лебеденко Л.Ф.
Утверждено редакционно-издательским советом СибГУТИ в качестве методических указаний.

Содержание

1. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА ТУРБО-ПРОЛОГ

1.1. **Введение**

Турбо-Пролог - это осуществленная компанией Borland International реализация языка программирования высокого уровня Пролог компиляторного типа. Ее отличает большая скорость компиляции и счета. Турбо-Пролог предназначен для выдачи ответов, которые он логически выводит при посредстве своих мощных внутренних процедур.

Вызов самого Турбо-Пролога осуществляется приказом

PROLOG

Главное меню Турбо_Пролога высвечивает 7 доступных пользователю опций (команд) в верхней части экрана:

1.Запуск программы на счет (Run).
2.Трансляция программы (Compile).
3.Редактирование текста программы (Edit).
4.Задание опций компилятора (Options).
5.Работа с файлами (Files).
6.Настройка системы (Setup).
7.Выход из системы (Quit).

1.2.Запуск на счет программы, записанной на Турбо_Прологе

Набейте текст программы WELCOME.PRO:

```
predicates
   hello
goal
   hello.
clauses
   hello :- write("Welcome to Turbo Prolog!"),nl.
```

Данная программа имеет целью дать необходимые навыки в использовании меню системы и основных команд редактора. После ввода программы нажмите клавишу Esc. Задайте теперь команду Run и пронаблюдайте за строками в окне Message и за результатом работы программы в окне Dialog.

Первая строка в окне сообщений указывает на то, что началась трансляция программы WELCOME.PRO трансляция задается автоматически при задании команды Run. Вторая строка сигнализирует о трансляции предиката hello.

Чтобы записать программу на диск, нажмите Esc, затем выберите команду Files и подкоманду Save во вновь появившемся меню. В результате будет высвечено либо заданное по умолчанию имя файла (WORK.PRO), либо то имя, которое вы присвоили файлу (в данном случае следует ввести имя WELCOME.PRO).

Просмотр каталога осуществляется подкомандой Directory команды Files. Загрузка - подкомандой Load команды Files.

1.3.Основные понятия языка Турбо-Пролог

Турбо-Пролог - это декларативный язык, программы на котором содержат объявления логических взаимосвязей, необходимых для решения задачи. В Турбо_Прологе рассматриваются отношения между утверждениями и объектами, характерные для логики предикатов.

В программах на Прологе существует три типа предложений (clauses): факт, правило вывода, цель. Каждое предложение должно заканчиваться точкой. Факт - утверждение, истинность которого безусловна. Например,

```
likes(mary,apples).   /* Мэри любит яблоки */
```

или

```
male(bob)            /* Боб - мужчина */
parent(bob,ann).     /* Боб - родитель Энн */
```

Правило - утверждение, зависящее от условий.Например,

```
child(ann,bob) :- parent(bob,ann). /* Энн - дитя Боба,
                                      если Боб - родитель Энн */
```

или

```
father(X,Y) :-parent(X,Y),male(X  )./* Для всех X и Y
                                       X является отцом Y,если
                                       X является родителем Y и
                                       X - мужчина */
```

Цель - вопрос пользователя к системе о том, какие утверждения являются истинными.

Для указанных выше примеров на вопрос

```
child(ann,bob)  /* является ли Энн ребенком Боба ?*/
```

будет выдан ответ

```
true  /* истина */,
```

а на вопрос

```
father(X,ann)  /* кто является отцом Энн ? */
```

будет выдан ответ

```
X = Bob       /* отцом Энн является Боб */.
```

На все поставленные вопросы Пролог пытается ответить с помощью фактов и правил вывода. Он решает задачу, просматривая программу сверху вниз и слева направо. Сначала анализируется цель и ведется поиск такого

факта или правила вывода, с помощью которого она может быть достигнута. При нахождении такого факта после соответствующей подстановки переменных Пролог переходит к анализу следующей цели при условии, что предыдущая достигнута (доказана). Если некоторая цель последняя, доказательство заканчивается. При отсутствии нужных фактов, но наличии правила вывода, которое могло быть применено, цель заменяется условием этого правила с соответствующей подстановкой переменных. Теперь условием выполнения цели становится доказательство условия (правой части) правила вывода. Процесс нахождения соответствия между целью и фактом или правилом называется у н и ф и к а ц и е й. В ходе унификации Пролог ищет все альтернативные решения.

Программа Турбо-Пролога включает определенные разделы, не все из которых являются обязательными:

```
domains
   /*(домены) - раздел объявлений*/;
database
  /* описания предикатов динамической базы данных */
predicates
 /* описания предикатов */
goal
 /* целевое утверждение */
clauses
 /* утверждения - факты и правила */
```

В программе по крайней мере должны быть разделы predicates и clauses.

Раздел domains напоминает объявление данных в традиционных (императивных) языках, например таких, как Паскаль и Си. Существуют следующие типы доменов:

char (символьный),
integer (целый),
real (вещественный),
string (строковый),
symbol (для цепочки из букв, цифр и символов подчеркивания с первой строчной буквой либо цепочки знаков в двойных кавычках),
file (файловый).

По отношению к именам объектов (идентификаторам) в Прологе используются следующие правила :

1) имя может включать латинские буквы, цифры и символ подчеркивания, причем первым символом не должна быть цифра;
2) имена символических констант должны начинаться со строчной буквы;
3) в имени можно использовать одновременно и строчные и прописные буквы.

Рассмотрим примеры программ на языке Турбо-Пролог.

Пример 1.Родственные отношения

```
domains
  s=symbol   /* объект s имеет тип symbol */
predicates
  parent(s,s)
  female(s)
  male(s)
  mother(s,s)
  father(s,s)
  ancestor(s,s)
    child(s,s)
clauses
  parent(pam,bob).      /* Пам - родитель Боба */
  parent(tom,bob).
  parent(tom,liz).
  parent(bob,ann).
  parent(bob,pat).
  parent(pat,jim).

  female(pam).       /* Пам - женщина */
  female(liz). female(ann). female(pat).

  male(tom).       /* Том - мужчина */
  male(bob). male(jim).

  child(Y,X):-      /* Y - отпрыск X, если X - родитель Y */
    parent(X,Y).

  mother(X,Y):-             /* X - мать Y, если */
    parent(X,Y),female(X). /* X - родитель Y и X - женщина */

  father(X,Y):-           /* X - отец Y, если */
    parent(X,Y),male(X).  /* X - родитель Y и X - мужчина */

  ancestor(X,Z):-           /* X - предок Z, если */
       parent(X,Z).        /* X - родитель Z */
  ancestor(X,Z):-             /* X - предок Z, если */
     parent(X,Y),ancestor(Y,Z). /* X - родитель Y и
                                Y - предок Z */
```

Пример 2.Ориентированный граф

В программе, текст которой приведен ниже, устанавливается: есть ли дуга между вершинами; связаны ли две вершины противоположно направленными дугами; образуют ли три следующие друг за другом дуги треугольник:

```
domains
  point=symbol
predicates
  arc_(point,point)
  line_(point,point)
  tr(point,point,point)
clauses
/* описание существующих дуг в графе */
  arc_(a,c). arc_(b,c). arc_(c,d). arc_(d,a). arc_(d,e). arc_(c,b). arc_(a,f).
/* условие того, что вершины A и B связаны двумя противоположно
направленными дугами */
  line_(A,B):-arc_(A,B),arc(B,A).
/* условие того, что три следующих друг за другом дуги образуют
треугольник */
  tr(X,Y,Z):-arc_(X,Y),arc_(Y,Z),arc_(Z,X).
```

После запуска программы на выполнение в окне Dialog экрана появится строка Goal:. Введем, например line(X,Y). В результате получим: X=b,Y=c,X=c,Y=b 2 Solutions, т.е. найдено два решения. Точно так же можно ставить вопросы для отношений arc_ и tr. Например, если ввести: arc_(a,b), результатом будет false (ложь), а если ввести arc(a,f), -
true (истина). На вопрос arc_(F,c) Пролог выдаст два решения: F=a и F=b.

Пример 3.Отношение "нравится"

```
predicates
  likes(symbol,symbol)
clauses
  likes(ellen, tennis).   /* Эллен нравится теннис */
  likes(john, football).    likes(tom, baseball).    likes(eric, swimming).
  likes(mark, tennis).
   /* Биллу нравится то же, что и Тому */
  likes(bill, Activity) if likes(tom, Activity).
```

Пример 4. Отношение «может купить»

```
predicates
  can_buy(symbol, symbol)  /* отношение "может купить" */
  person(symbol)          /* отношение "субъект */
  car(symbol)            /* отношение "марка автомобиля" */
  likes(symbol, symbol)
  for_sale(symbol)       /* отношение "продается" */

clauses
  can_buy(X, Y) :- person(X), car(Y), likes(X, Y),  for_sale(Y).
  person(kelly). person(judy).
  car(lemon). car(hot_rod).

  likes(kelly, hot_rod). likes(judy, pizza).

  for_sale(pizza). for_sale(lemon). for_sale(hot_rod).
```

1.4.Варианты заданий

1.Родственные отношения.

Кроме родственных отношений parent (родитель) и ancestor (предок) программа должна содержать хотя бы одно из следующих отношений:

1.1. brother (брат);

1.2. sister (сестра);

1.3. grand-father (дедушка);

1.4. grand-mother (бабушка);

1.5. uncle (дядя);

2.Ориентированные графы.

Описать граф. Задать отношения, позволяющие определить наличие в графе:

2.1. путей между произвольной парой вершин;

2.2. многоугольников с заданным числом сторон (например, четырехугольников).

3.Отношения likes ("нравится") и can_buy ("может купить").

Описать указанные отношения для следующих комбинаций "субъекты - предметы":

3.1.субъекты - фрукты;

3.2.субъекты - марки автомобилей;

3.3.субъекты - фильмы;

3.4.субъекты - книги.

2. ОПЕРАЦИИ НАД СПИСКАМИ И АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

2.1.Представление списков

Список - последовательность из произвольного числа элементов. Список является основной структурой данных в Прологе. Элементы списка разделяются запятыми и заключаются в квадратные скобки. Любой список представляет собой:

- либо пустой список (атом []);
- либо непустой список - структуру, состоящую из двух частей:
  - первый элемент - голова (Head) списка;
  - второй элемент - хвост (Tail) списка.

В общем случае голова списка может быть любым объектом языка Пролог, а хвост - обязательно должен быть списком. Поскольку хвост - список, то он либо пуст, либо имеет свои собственные голову и хвост.

Для повышения наглядности программ в Прологе предусматриваются специальные средства для списковой нотации, позволяющие представлять списки в виде

[Элемент1, Элемент2,...]

или

[Голова | Хвост]

или

[Элемент1, Элемент2,... | Остальные].

Здесь знак | используется для отделения начала списка от конца.

2.2.Операции над списками

2.2.1.Принадлежность к списку (member)

Отношение принадлежности записывается в виде двух предложений:

```
member(X,[X|Tail]).
member(X,[_|Tail]) :- member(X,Tail).
```

Оно основано на следующих соображениях: либо X - голова списка, либо X принадлежит хвосту этого списка.

2.2.2.Сцепление (конкатенация) списков (conc)

Обозначается через conc(L1,L2,L3). Здесь L1 и L2 - два списка, L3 - список, получаемый при их сцеплении. Определение отношения conc содержит два случая:

- если первый аргумент - пуст, то второй и третий аргументы представляют собой один и тот же список:

conc([],L,L).

- если первый аргумент отношения conc не пуст, то он имеет голову и хвост - [X|L1]. Результат сцепления - список [X|L3], где L3 - получен после сцепления списков L1 и L2:

conc([X|L1],L2,[X|L3]) :- conc(L1,L2,L3).

Примеры сцепления заданных списков.

Goal: conc([a,b,c],[x,y,z],L).

L = [a,b,c,x,y,z]

Goal: conc([a,[b,c],d],[a,[],b],L).

L = [a,[b,c],d,a,[],b]

Программу для conc можно применить "в обратном направлении" - для разбиения списка на две части. Например:

Goal: conc(L1,L2,[a,b,c]).

L1=[]
L2=[a,b,c]

L1=[a]
L2=[b,c]

L1=[a,b]
L2=[c]

L1=[a,b,c]
L2=[]

Используя conc можно определить отношение принадлежности следующим эквивалентным способом:

member(X,L) :- conc(_,[X|_],L).

Здесь символом "_" обозначены анонимные переменные (переменные, встречающиеся в предложении только по одному разу).

2.2.3.Добавление элемента (append)

Наиболее простой способ добавления элемента в список - вставить его в начало так, чтобы он стал его новой головой. Процедура добавления определяется в форме факта

append(X,L,[X|L]).

2.2.4. Удаление элемента (remove)

Имеем два случая:

- если X - голова списка, то результат удаления - хвост списка;

- если X - в хвосте списка, то его нужно удалить оттуда.

В результате отношение remove определяется так:

```
remove(X,[X|Tail],Tail).
remove(X,[Y|Tail],[Y|Tail1]) :- remove(X,Tail,Tail1).
```

Если в списке встречается несколько вхождений элемента X, то remove сможет исключить их все при помощи возвратов.

2.3.Арифметические действия

Турбо-Пролог располагает двумя числовыми типами доменов: целыми и действительными числами. Четыре основные арифметические операции - это сложение, вычитание, умножение и деление. Для их реализации в Турбо-Прологе используются предикаты.

Пример 1. Поиск нужного элемента в списке

```
domains
    i=integer
    s=symbol
    i_list=i*  /* определение типа "список целых чисел" */
    s_list=s*  /* определение типа "список атомов" */
predicates
    /* предикат member определяется над списками двух типов */
    member(i,i_list)
    member(s,s_list)
clauses
    member(Head,[Head|_]).
    member(Head,[_|Tail]):-member(Head,Tail).
```

Пример 2. Определение суммы элементов списка

```
domains
    i=integer
    i_list=i*
predicates
    sum_list(i_list,i)
clauses
    /* Если список пуст, то сумма его элементов равна нулю */
    sum_list([],0).
    /* Иначе найти сумму элементов хвоста списка
            и прибавить к ним голову  */
    sum_list([H|T],Sum,Number):- sum_list(T,Sum1,Number1), Sum=H+Sum1.
```

Пример 3. Реализация арифметики

```
domains
    i=integer
    r=real
predicates
    add(i,i,i)
    sub(i,i,i)
    mul(i,i,i)
    div(i,i,i)
    fadd(r,r,r)
    fsub(r,r,r)
    fmul(r,r,r)
    fdiv(r,r,r)

clauses

    add(X,Y,Z) :- Z=X+Y.      sub(X,Y,Z):-  Z=X-Y.
    mul(X,Y,Z):- Z=X*Y.     div(X,Y,Z):-  Z=X/Y.
    fadd(X,Y,Z) :-Z=X+Y.    fsub(X,Y,Z):- Z=X-Y.
    fmul(X,Y,Z):-Z=X*Y.     fdiv(X,Y,Z):-Z=X/Y.
```

2.3. Варианты заданий

1. Определить максимальный элемент в списке.
2. Определить число элементов в списке.
3. Определить произведение элементов списка.
4. Исключить из списка отрицательные элементы.
5. Выполнить сортировку элементов списка по возрастанию.
6. Даны два списка, имеющие ненулевое пересечение. Построить список, включающий все элементы указанных двух списков без повторений.
7. Определить отношение

   Обращение(Список, Обращенный список),

которое располагает элементы списка в обратном порядке.
8. Определить отношение

   перевод(Список1, Список2)

для перевода списка чисел от 0 до 9 в список соответствующих слов.
9. Определить отношение

   разбиение_списка(Список, Список1, Список2)

так, чтобы оно распределяло элементы списка между двумя списками Список1 и Список2, длины которых отличаются друг от друга не более чем на единицу.

10. Определить отношение

пересечение(Список1, Список2, Список3),

где элементы списка Список3 являются общими для списков Список1 и Список2.

11. Определить отношение

разность(Список1, Список2, Список3),

где элементы списка Список3 принадлежат Списку1, но не принадлежат Списку2.

12. Определить отношение

element_mult(List1,List2,List3),

в котором элементы списка List3 равны произведениям соответствующих элементов списков List1 и List.

13. Определить отношение

shift(List1,List2)

таким образом, чтобы список List2 представлял собой список List1, "циклически сдвинутый" влево на один символ.

14. Треугольное число с индексом N - это сумма всех натуральных чисел до N включительно. Напишите программу, задающую отношение

triangle(N,T),

истинное, если T - треугольное число с индексом N.

15. Написать программу, задающую предикат

power(X,N,V),

истинный, если $V = X^N$.

16. Написать программу, задающую отношение

fib(N,F),

для определения N - го числа Фибоначчи F.

17. Написать программу вычисления скалярного произведения векторов inner_product(X,Y,V), где X и Y - списки целых чисел, V - скалярное произведение.

18. Написать программу

fact(N,F),

вычисляющую факториал F числа N.

3. ВВОД И ВЫВОД В ТУРБО-ПРОЛОГЕ

3.1.Взаимодействие с файлами

Ввод и вывод в Прологе организуется с помощью специальных предикатов чтения и записи, которые могут рассматриваться как аналоги соответствующих подпрограмм в языках Паскаль и Си.

В общем случае пролог- программа взаимодействует с несколькими файлами (в том числе с "псевдофайлом" keyboard (клавиатура) и

"псевдофайлом" screen (экран)). Она считывает данные из нескольких входных файлов, называемых входными потоками, и выводит данные в несколько выходных файлов, называемых выходными потоками.

В каждый момент выполнения программы лишь два файла являются "активными": один для ввода, другой для вывода. В начальный момент эти два потока соответствуют терминалу. Текущий входной поток может быть заменен на другой файл name_of_file посредством цели

readdevice(name_of_file).

Такая цель всегда успешна (если только с файлом name_of_file все в порядке), а в качестве побочного эффекта происходит переключение ввода с предыдущего входного потока на файл name_of_file.

Пример 1. Использование предиката readdevice.

Приведенная ниже последовательность целей считывает информацию из файла myfile, а затем переключает ввод обратно на терминал.

```
...
 openread(myfile,"myfile.txt"),
/* файл myfile открывается для чтения */
 readdevice(myfile),
/* стандартный входной поток связывается с файлом myfile */
 filepos(myfile,X,0),
/* текущий указатель устанавливается на позицию X, которая отсчитывается от начала файла myfile */
 readchar(Y),
/* из позиции X читается символ и назначается переменной Y */
 readdevice(keyboard),
/* стандартный входной поток связывается с клавиатурой */
...
```

Текущий выходной поток может быть изменен при помощи цели вида

writedevice(name_of_file).

Следующая последовательность целей выводит значение переменной X в файл outfile.txt и после этого перенаправляет последующий вывод обратно на терминал.

Пример 2.

```
...
openwrite(outfile,"outfile.txt"),
/* файл outfile открывается для записи */
writedevice(outfile),
```

/* стандартный выходной поток связывается с файлом outfile */
write(X),
closefile(outfile),
/* закрытие файла outfile */
writedevice(screen),
/* стандартный выходной поток связывается с экраном дисплея */
...

Кроме вышеупомянутых используются следующие стандартные предикаты работы с файлами:

1) openappend(логическое имя файла,
физическое имя файла)
- подобен openread, но файл открывается для дополнения;

2) openmodify(логическое имя файла,
физическое имя файла)
- подобен openread, но файл открывается для модификации (чтения и записи);

3) filepos(логическое имя файла,позиция,режим) - устанавливает текущий указатель на заданную позицию в указанном файле. Объект, указывающий на позицию, должен быть вещественного типа (real), а режим - целого типа (integer). Если режим равен 0, то позиция отсчитывается от начала файла; если 1, то - от текущего значения указателя; если 2, то - от конца файла. Этот предикат используется для прямого доступа к файлу.

4) readchar(Var) - считывает символ с входного потока (по умолчанию с клавиатуры) и присваивает его переменной Var;

5) readint(Var) - считывает целое число и присваивает его переменной Var;

6) readln(Var) - считывает символы с входного потока до нажатия клавиши Enter. Введенные символы присваиваются переменной Var, которая должна быть строковой (string) или символьной (symbol).

7) readreal(Var) - считывает вещественное число;

8) write(Arg1, Arg2, ...) - выводит значения аргументов на текущее устройство (по умолчанию, на экран дисплея). Аргументы Arg1, Arg2, ... могут быть константами или переменными, которым заранее присвоены требуемые значения.

9) nl - вызывает перевод каретки в начало следующей строки.

В предикате write можно использовать символы, начинающиеся со знака \. Они имеют специальные значения:

\k - символы, имеющие ASCII код числа k;

\n - возврат каретки и перевод строки;

\t - табуляция.

Файлы могут обрабатываться только последовательно.

Каждый запрос на чтение из входного файла приводит к чтению в текущей позиции текущего входного потока. После этого текущая позиция будет перемещена на следующий, еще не прочитанный элемент данных. Следующий запрос на чтение приведет к считыванию, начиная с этой новой текущей позиции.

Запись производится точно так же: каждый запрос на вывод информации приведет к тому, что она будет присоединена к концу текущего выходного потока.

Пример 3. Запись символов в файл myfile.f, который создается на текущем диске.

```
domains
  file=myfile
/* объявляется логическое имя файла myfile */
predicates
  read_in_loop
goal
  openwrite(myfile,"myfile.f"),  writedevice(myfile), not(read_in_loop),
  closefile(myfile), writedevice(screen),
  write("\n запись в файл myfile.f произведена \n ").
clauses
  read_in_loop:- readchar(X), X<>'#',!, write(X), read_in_loop.
```

Пример 4. Чтение символа из файла и вывод его на экран дисплея.

```
domains
  file=infile
predicates
  position
goal
  write(" С каким файлом Вы хотите работать ? \n "),
  readln(Fname), openread(infile,Fname), position.
clauses
  position:- readdevice(keyboard), nl,write("Введите номер позиции: "),
        readreal(X), readdevice(infile), filepos(infile,X,0),
        readchar(Y), write(" Здесь записан символ:",Y), position.
```

3.2.Форматный вывод writef.

writef(Format,Arg1,...,Argn) - подобен write, но осуществляет форматированный вывод в соответствии с параметром Format, который представляется в виде

%p

Возможные значения p :

d - нормальное десятичное число (символы и целые числа);
u - беззнаковое целое;
s - строка (атомы или строки);
c - символ (символы и целые);
g - вещественное в самом коротком формате;
e - вещественное в экспоненциальном представлении;
f - вещественное в десятичном представлении (задается по умолчанию);
x - шестнадцатиричное число (символы и целые числа).

3.3.Ввод и вывод чисел и символов

Пример 5.Вычисление куба числа, вводимого с терминала.

```
domains
       i=integer
predicates
         process(i)
         cube
clauses
         cube:-  write("Next number,please:"), readint(X), process(X).
          process(N):- C=N*N*N, write("Cube",N,"is equal",C,"\n"), cube.
goal
       cube.
```

Пример 6. Считывание целых чисел с терминала и занесение их в список

```
domains
         list=integer*
predicates
         readlist(list)
goal
         readlist(TheList),write("\n The list is: ", TheList).
clauses
         readlist([H|T]):-readint(H),!,readlist(T).     readlist([]).
```

После набора каждого целого числа нужно нажать ENTER. Завершение программы происходит по любому символу, который не является целым числом, плюс ENTER.

Пример 7. Вывод списков

```
domains
        i_list=integer*
        n_list=symbol*
predicates
        writelist(i_list)
        writelist(n_list)
clauses
        writelist([]).
        writelist([H|T]):-  write(H," "),writelist(T).
```

3.4.Обработка строк.

Стандартные предикаты обработки строк в Турбо-Прологе:

1.concat(Str1,Str2,Str1_2) - утверждает, что Str1_2 - это объединение строк Str1 и Str2; при этом по крайней мере два параметра должны быть определены заранее.

2.frontchar(Str1_2,Char,Str2) - работает в соответствии с уравнением:

Str1_2 = Char U Str2

3.frontstr(Length,InpString,StartString,RestString) - назначает первые Length символов строки InpString в строку StartString, остальные - в строку RestString.

4.fronttoken(String,Token,RestString)

String - входная строка;

Token - первый атом строки (последовательность символов до пробела);

RestString - остаток строки.

5.str_len(String,Length) - определяет длину строки.

Пример 8. Преобразование строки в список символов.

```
domains
        charlist=char*
predicates
        name(string,charlist)
clauses
        name(" ",[]).
        name(S,[H|T]):-frontchar(S,H,S1), name(S1,T).
```

Пример 9. Преобразование строки в список атомов.

```
domains
        namelist=name*
        name=symbol
predicates
        string_namelist(string,namelist)
clauses
        string_namelist(S,[H|T]):-fronttoken(S,H,S1),!,string_namelist(S1,T).
        string_namelist(_,[]).
```

3.5.Встроенный предикат findall

Встроенный предикат findall(X,P,L) порождает список L всех объектов X, удовлетворяющих цели P.

Пример 10. Вычисление среднего возраста

```
domains
  name, address = string
  age = integer
  list = age*
predicates
  person(name, address, age)
  sumlist(list, age, integer)
goal
  findall(Age, person(_, _, Age), L), sumlist(L, Sum, N),    Ave = Sum/N,
write("Average =", Ave), nl.

clauses
  sumlist([], 0, 0).
  sumlist([H|T], Sum, N) :- sumlist(T, S1, N1), Sum=H+S1, N=1+N1.
  person("Sherlock Holmes", "22B Baker Street", 42).
  person("Pete Spiers", "Apt. 22, 21st Street", 36).
  person("Mary Darrow", "Suite 2, Omega Home", 51).
```

3.6. Варианты заданий

1. Создать символьные файлы f и g. Записать в файл h сначала компоненты файла f, затем компоненты файла g с сохранением порядка.
2. Создать файл f, содержащий целые числа. Переписать в файл g отрицательные числа и определить их количество.

3. Создать символьный файл t. Удалить из текста файла t предпоследний элемент.

4. Создать файл f из натуральных чисел. Найти количество нечетных чисел.

5. Создать файл f, содержащий целые числа. Записать в файл g максимальное и минимальное числа из файла f.

6. Создать файл, состоящий из действительных чисел. Найти их сумму и произведение.

7. Ввести символьную строку с терминала. Определить в ней количество слов.

8. Написать простую программу-калькулятор, которая выполняет четыре арифметических действия над целыми числами, вводимыми с терминала.

9. Написать программу, считывающую с терминала произвольные предложения и выводящую их на терминал в форматированном виде,в котором все группы идущих подряд пробелов заменены на одиночные пробелы.

10. Ввести строку с терминала. Преобразовать введенную строку в список слов, упорядоченных лексикографически.

11. Ввести строку с терминала. Вычислить минимальную и максимальную длины слов строки.

12. Создать текстовый файл f. Преобразовать этот файл в список слов, упорядоченных по длине, и вывести этот список на терминал.

4. БАЗЫ ДАННЫХ В ТУРБО ПРОЛОГЕ

4.1.Встроенные предикаты для работы с базами данных

Пролог-программу можно рассматривать как реляционную базу данных, т.е. описание некоторого множества отношений. Описание отношений присутствует либо в явном виде (факты), либо в неявном виде (правила).

Встроенные предикаты дают возможность корректировать эту базу данных (БД) в процессе выполнения программы. Это делается:

1) добавлением к программе (в процессе вычислений) новых фактов;

2) вычеркиванием из нее уже существующих фактов.

Следующие цели (предикаты) выполняют операции над БД:

1.Цель assert(d) всегда успешна и добавляет факт d к базе данных;

2.Цель retract(d) удаляет факт, сопоставимый с d;

3.asserta(d) - обеспечивает запись в базу данных нового факта перед имеющимися фактами для заданного отношения;

4.assertz(d) - обеспечивает запись в базу данных нового факта после всех имеющихся фактов для заданного отношения.

Объявление динамической базы данных, в которую факты могут добавляться во время выполнения программы или выбираться из файла

посредством предиката consult, осуществляется посредством ключевого слова database.

Пример 1.

```
domains
  tip,fun=symbol
  x,kol=integer

database
  ms(tip,fun,x,kol)
/* tip - тип микросхемы; fun - реализуемая функция;
  x   - число входов;   kol - число элементов */

clauses
  ms(k155la3,i_ne,2,4). ms(k155la4,i_ne,3,3). ms(k155la1,i_ne,4,2).
  ms(k155ln1,ne,1,6). ms(k155le1,ili,2,4). ms(k155li3,i,3,3).
```

Ввод строки assertz(ms(k155ir1,rg,4,4)) приводит к добавлению факта s(k155ir1,rg,4,4) в базу данных.

Ввод строки retract(ms(Q,i_ne,W,E)) приведет к удалению из БД всех объектов для элементов типа i_ne.

Для сохранения БД на диске в заданном файле (например, mybase.dba) необходимо ввести строку

```
save("mybase.dba").
```

Затем в этой же или в другой программе на основе полученного файла можно создать новую БД. Для этого необходимо ввести строку

```
consult("mybase.dba").
```

В результате существующая БД дополняется объектами из файла mybase.dba.

Для сохранения поименованной БД используется предикат

```
save(DosFileName,DataBaseName).
```

Для создания поименованной БД из фактов, расположенных в файле, используем предикат

```
consult(DosFileName,DataBaseName).
```

4.2. Накопление в базе данных ответов на вопросы

Одним из полезных применений предиката asserta является накопление уже вычисленных ответов на вопросы. Пусть, например, в программе определен предикат

```
solve(Problem,Solution).
```

Мы можем теперь задать вопрос и потребовать, чтобы ответ на него был запомнен для того, чтобы облегчить получение ответов на будущие вопросы:

solve(Problem,Solution), asserta(solve(Problem,Solution)).

Если в первой из приведенных целей будет успех, ответ (Solution) будет сохранен, а затем использован так же, как и любое другое пр ложение, при ответе на дальнейшие вопросы. Преимущество такого "запоминания" состоит в том, что на дальнейшие вопросы, сопоставимые с добавленным фактом, ответ будет получен, как правило, значительно быстрее, чем в первый раз. Ответ теперь будет получен как факт, а не как результат вычислений, требующих, возможно, длительного времени.

4.3. Задание итерации

Цель repeat порождает новую ветвь вычислений. Для этого она должна быть определена в программе следующим образом

```
repeat.
repeat :- repeat.
```

Пример 2. Простейший итерационный процесс.

Данная программа изменяет значение счетчика counter(i) в базе данных от i=0 до i=100.

```
database
  counter(integer)
predicates
  repeat
  count
goal
  count.
clauses
  repeat. repeat :- repeat.
  count :- assert(counter(0)),fail.
  count :- repeat, counter(X), Y=X+1, retract(counter(X)), asserta(counter(Y)),
        write(Y,"\n"), Y=100.
```

Пример 3. Порождение таблицы умножения

В базе данных формируется таблица умножения в виде совокупности термов вида prod(X,Y,Z), где X и Y - сомножители, а Z - произведение. Эта таблица сохраняется в файле tabl.dba при помощи цели save("tabl.dba").

```
domains
  i=integer
```

```
    list=i*
  predicates
    repeat
    member(i,list)
    tabl
    run
  database
    prod(i,i,i)
    counter(i)

  clauses
    repeat.  repeat :- repeat.
    member(X,[X|_]). member(X,[_|L]) :- member(X,L).
    tabl :- repeat, L=[1,2,3,4,5,6,7,8,9],
          member(X,L), member(Y,L),Z=X*Y,
          assert(prod(X,Y,Z)), counter(V),W=V+1, retract(counter(V)),
          asserta(counter(W)),W=81.
    run :- assert(counter(0)), tabl, retract(counter(_)), save("tabl.dba").
  goal
    run.
```

Пример 4. Считывание БД из файла.

Таблица умножения, сформированная в примере 3, считывается целью consult("tabl.dba") из файла tabl.dba и выводится на терминал.

```
  domains
    i=integer
  database
    prod(i,i,i)
Goal: consult("tabl.dba"),prod(X,Y,Z). /* Цель задается с терминала*/
                                       /*   в окне диалога */
```

Пример 5. База данных о читателях.

Формируется БД о читателях в виде совокупности термов вида reader(Фамилия,Номер_читательского_билета, Дата_посещения_библиотеки). Далее определяется количество читателей, посетивших библиотеку 10- го "числа".

```
  domains
    i=integer
  database
    reader(symbol,i,i)
```

```
  counter(i)
predicates
  wr(i,i)
  repeat
  inbase
  count(i,i)
clauses
  repeat. repeat :- repeat.
  count(X,Y) :- counter(X),Y=X+1, retract(counter(X)), assert(counter(Y)).
  inbase :-                /* Занесение информации*/
        repeat,            /*   о читателях в БД */
        write('&'),
        readln(Name),   /* считывание фамилии читателя */
        readint(Ticket),  /* считывание номера чит. билета */
        readint(Date), /* считывание даты посещения */
        assert(reader(Name,Ticket,Date)),
        count(_,Y),Y=5.   /* количество читателей = 5 */
  wr(D,Y) :-                /* подсчет числа читателей*/
        retract(reader(_,_,D)), /*   с датой посещения D */
        count(Y,Y1), wr(D,Y1).
  wr(_,Y) :- counter(Y),        /* Выдача числа читателей*/
          write("Count=",Y),!.    /*    на терминал    */
goal
  assert(counter(0)), inbase,
  save("reader.dba"), /* БД сохраняется в файле reader.dba */
  asserta(counter(0)), wr(10,Y).
```

4.4. Варианты заданий

1.Создать БД, содержащую сведения о пассажирах:
Ф.И.О., количество мест, вес багажа.
Определить, есть ли пассажиры, багаж которых занимает 1 место и вес багажа больше 30 кг.

2.Создать БД о студентах вашей группы:
Фамилия, Имя, Год рождения.
Получить список студентов старше 20 лет.

3.Соэдать БД, содержащую сведения:
Ф.И.О., профессия, оклад.
Найти среднемесячную заработную плату для инженеров.

4.Создать БД о группе студентов:
Фамилия, Имя.
Выяснить, имеются ли в группе однофамильцы.

5.Создать БД со сведениями о файлах:
спецификация файла, дата создания, размер файла.
Получить сведения о файлах, имеющих размер более 5 блоков.

6.Создать БД о металлах:
Наименование, Удельная проводимость, Удельная стоимость.
Найти металлы с максимальной проводимостью и минимальной стоимостью.

7.Создать БД с расписанием движения поездов:
Номер поезда,
Пункт назначения,
Время отправления,
Время в пути,
Стоимость билета.
Найти номер и время отправления самого скорого поезда до Москвы.

8.Создать БД с расписанием движения самолетов:
Номер рейса,
Пункт отправления,
Пункт прибытия,
Время отправления,
Время в пути,
Стоимость билета.
Определить маршрут движения из Новосибирска в Нью-Йорк, время в пути и стоимость проезда.

9. Создать БД с таблицей игр чемпионата по футболу:
Первая команда, Вторая команда, Счет игры.
Определить чемпиона.

10. Создать БД с книжным каталогом:
Ф.И. автора, Название книги, Издательство, Год издания.
Найти все книги, изданные в издательстве "Наука" после 1990 года.

11. Создать БД со сведениями о стоимости товаров:
Наименование товара, Стоимость товара.
Определить суммарную стоимость указанных в БД товаров, найти товары с максимальной и минимальной стоимостями.

5. ОПЕРАЦИИ НА ГРАФАХ

5.1. Представление ориентированных графов в Прологе

Способ 1.

Каждая дуга графа записывается в виде отдельного предложения. Например,

arca(a,b). arca(b,c).

или (граф с взвешенными дугами)

arca(s,t,1). arca(t,v,3). arca(v,u,2).

Способ 2.

Граф представляется в виде списка дуг. Например,

G = [arca(a,b), arca(b,c), arca(b,d), arca(c,d)]

или

G = [arca(s,t,3), arca(t,v,1), arca(v,u,2), arca(u,t,5), arca(t,u,2)]

Способ 3.

Граф представляется как один объект. Графу соответствует пара множеств - множество вершин и множество дуг. Для объединения множеств в пару будем применять функтор graph, а для записи дуги - arca. Например,

G = graph([a,b,c,d], [arca(a,b), arca(b,d), arca(b,c), arca(c,d)])

Всюду, где это возможно, для простоты записи программы будем представлять графы способом 1 или способом 2.

5.2. Операции на графах

Типичными операциями на графе являются следующие:

- найти путь между двумя заданными вершинами графа;
- найти подграф, обладающий заданными свойствами (например, построить остовное дерево графа).

5.2.1. Поиск пути в графе

Определим отношение

path(A,Z,P),

где P - ациклический путь между вершинами A и Z в графе G, представленном следующими дугами:

arca(a,b). arca(b,c). arca(c,d). arca(b,d).

Один из методов поиска пути основан на следующих соображениях:

- Если A = Z, то положим P = [A];

- Иначе нужно найти ациклический путь P1 из произвольной вершины Y в Z, а затем найти путь из A в Y, не содержащий вершин из P1.

Введем отношение

path1(A,P1,P),

означающее, что P1 - завершающий участок пути P.

Между path и path1 имеет место соотношение:

path(A,Z,P) :- path1(A,[Z],P).

Рекурсивное определение отношения path1 вытекает из следующих посылок:

- "граничный случай": начальная вершина пути P1 совпадает с начальной вершиной A пути P;

-в противном случае должна существовать такая вершина X, что:) Y - вершина, смежная с X, 2) X - не содержится в P1, 3) для P выполняется отношение path(A,[Y|P1],P).

Пример 1. Программа нахождения пути.

```
domains
        s=symbol
        list=s*
predicates
        path(s,s,list)
        path1(s,list,list)
        member(s,list)
        arca(s,s)
clauses
        arca(a,b). arca(b,c). arca(c,d). arca(b,d).
        member(X,[X|_]).
        member(X,[_|Tail]):- member(X,Tail).
        path(A,Z,Path):- path1(A,[Z],Path).
        path1(A,[A|Path1],[A|Path1]).
        path1(A,[Y|Path1],Path):- arca(X,Y), not(member(X,Path1)),
                path1(A,[X,Y|Path1],Path).
```

Чтобы отношения path и path1 могли работать со стоимостями (весами), их нужно модифицировать введением дополнительного аргумента для каждого пути:

```
path(A,Z,P,C),
path1(A,P1,C1,P,C),
```

где C и C1 - стоимости путей P и P1 соответственно.

Отношение смежности arca включает дополнительный аргумент - стоимость дуги:

```
arca(X,Y,C).
```

Пример 2. Программа построения пути минимальной стоимости между заданными вершинами графа.

```
domains
        i=integer
        s=symbol
        list=s*
database
        db_path(s,s,list,i)
predicates
        path(s,s,list,i)
        path1(s,list,i,list,i)
        member(s,list)
        arca(s,s,i)
        db0(s,s)
        db(s,s)
clauses
        arca(a,b,1).  arca(b,c,3).  arca(c,d,1).  arca(b,d,7).  arca(a,d,1).
        member(X,[X|_]).
        member(X,[_|Tail]):- member(X,Tail).
        path(A,Z,Path,C):- path1(A,[Z],0,Path,C).
        path1(A,[A|Path1],C,[A|Path1],C).
        path1(A,[Y|Path1],C1,Path,C):- arca(X,Y,CXY), not(member(X,Path1)),
                C2=C1+CXY, path1(A,[X,Y|Path1],C2,Path,C).
   /* поиск пути минимальной стоимости между вершинами X и Y */
        db0(X,Y) :-path(X,Y,P,C), assert(db_path(X,Y,P,C)).
        db(X,Y):-db_path(X,Y,P,C), path(X,Y,MP,MC), MC<C,!,
                retract(db_path(X,Y,P,C)), assert(db_path(X,Y,MP,MC)), db(X,Y).
        db(_,_).
goal
 /* определяется путь MP минимальной стоимости MC между вершинами a  и
    b */
        db0(a,d), db(a,d), db_path(a,d,MP,MC), write("\nMP=",MP,"\nMC=",MC).
```

5.2.2. Построение остовного дерева

Граф называется связным, если между любыми двумя его вершинами существует путь. Остовное дерево графа G=(V,E) - это связный граф T=(V,E1) без циклов,в котором E1 - подмножество E.

Определим процедуру

osttree(e,T),

где T - остовное дерево графа G (G - связный граф),e - произвольно выбранное ребро графа G.

Остовное дерево можно строить так: 1) начать с произвольного ребра графа G; 2) добавлять новые ребра, постоянно следя за тем, чтобы не образовывались циклы; 3) продолжать этот процесс до тех пор, пока не обнаружится, что нельзя присоединить ни одного ребра, поскольку любое новое ребро порождает цикл. Отсутствие цикла обеспечивается так: ребро присоединяется к дереву лишь в том случае, когда одна из его вершин уже содержится в строящемся дереве, а другая пока еще не включена в него.

Основное отношение, используемое в программе,

expand(Tree1,Tree).

Здесь Tree - остовное дерево, полученное добавлением множества ребер из G к дереву Tree1.

Пример 3. Построение остовного дерева.

```
domains
        s=symbol
        list=s*
        a=arca(s,s)
        tree=a*
predicates
        member(a,tree)
        arca(s,s)
        osttree(a,tree)
        expand(tree,tree)
        ap_arca(tree,tree)
        node(s,tree)
clauses
        arca(a,b). arca(b,c). arca(c,d).    arca(b,d).
        member(X,[X|_]).
        member(X,[_|Tail]):-member(X,Tail).
        osttree(arca(A,B),Tree) :- expand([arca(A,B)],Tree).
        expand(Tree1,Tree) :- ap_arca(Tree1,Tree2), expand(Tree2,Tree).
        expand(Tree,Tree) :-not(ap_arca(Tree,_)).
        ap_arca(Tree,[arca(A,B)|Tree]) :- arca(A,B), node(A,Tree),
not(node(B,Tree));
                arca(A,B),node(B,Tree), not(node(A,Tree)).
        node(A,Tree) :- member(arca(A,_),Tree); member(arca(_,A),Tree).
```

5.3. Варианты заданий

1. Определить, является ли связным заданный граф.

2. Найти все вершины графа, к которым существует путь заданной длины от выделенной вершины графа.

3. Найти все вершины графа, достижимые из заданной.

4. Подсчитать количество компонент связности заданного графа.

5. Найти диаметр графа, т.е. максимум расстояний между всевозможными парами его вершин.

6. Найти такую нумерацию вершин орграфа, при которой всякая дуга ведет от вершины с меньшим номером к вершине с большим номером.

7. Дан взвешенный граф. Построить остовное дерево минимальной стоимости.

8. Определить является ли граф гамильтоновым. Найти гамильтонов цикл, т. е. цикл, проходящий через все вершины графа.

9. Задана система односторонних дорог. Найти путь, соединяющий города A и B и не проходящий через заданное множество городов.

10. В заданном графе указать все его четырехвершинные полные подграфы.

11. Известно, что заданный граф - не дерево. Проверить, можно ли удалить из него одну вершину (вместе с инцидентными ей ребрами), чтобы в результате получилось дерево.

6. ОСНОВНЫЕ СТРАТЕГИИ РЕШЕНИЯ ЗАДАЧ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

6.1.Пространство состояний задачи

Пространством состояний задачи является граф, вершины которого соответствуют ситуациям, встречающимся в задаче (" проблемные ситуации"), а решение задачи сводится к поиску пути в этом графе.

Пространство состояний задачи определяет "правила игры":

- вершины пространства состояний соответствуют ситуациям;

- дуги соответствуют разрешенным ходам или действиям, или шагам решения задачи.

Конкретная задача определяется: пространством состояний; стартовой вершиной; целевым условием (т. е. условием, к выполнению которого нужно стремиться).

Целевыми называются вершины, удовлетворяющие целевым условиям.

Каждому разрешенному ходу или действию можно приписать его стоимость. В тех случаях, когда каждый ход имеет стоимость, мы заинтересованы в отыскании решения минимальной стоимости.

Стоимость решения - это сумма стоимостей дуг, из которых состоит решающий путь - путь из стартовой вершины в целевую.

Будем представлять пространство состояний при помощи отношения

after(X,Y),

которое истинно тогда, когда в пространстве состояний существует разрешенный ход из вершины X в вершину Y. Будем говорить, что Y – это преемник вершины X. Если с ходами связаны их стоимости, мы добавим третий аргумент, стоимость хода C:

after(X,Y,C).

Эти отношения можно задавать в программе явным образом при помощи набора соответствующих фактов. Однако, такой принцип непрактичен, поэтому отношение следования after обычно определяется неявно, при помощи правил вычисления вершин преемников некоторой заданной вершины.

Пример 1. Задача манипулирования кубиками.

Проблемная ситуация - список столбиков. Каждый столбик - список кубиков, из которых он составлен. Кубики упорядочены в списке таким образом, что самый верхний кубик находится в голове списка. "Пустые" столбики изображаются как пустые списки.

Отношение следования вытекает из правила:

Ситуация Sit2 - преемник ситуации Sit1, если в Sit1 имеется два столбика Stolb1 и Stolb2 такие, что верхний кубик из Stolb1 можно поставить сверху на Stolb2 и получить тем самым Sit2.

Поскольку все ситуации - списки столбиков, правило транслируется на Пролог так:

```
after(Stolbs,[Stolb1, [Up1|Stolb2],Rest]) :-
        delete([Up1|Stolb1],Stolbs,Stolbs1),
        delete(Stolb2,Stolbs1,Rest).
delete(X,[X|L],L).
delete(X,[Y|L],[Y|L1]) :- delete(X,L,L1).
```

Здесь:

- Stolbs - множество столбиков в ситуации Sit1;
- Stolbs1 - множество столбиков без первого столбика;
- Rest - множество столбиков без первого и второго.

Целевое условие в данном примере имеет вид:

```
goal(Sit) :- member([a,b,c],Sit).
```

Алгоритм поиска программируется как отношение

```
solve(Start,Solution),
```

где

- Start - стартовая вершина пространства состояний,
- Solution - путь, ведущий из вершины Start в любую целевую вершину.

Для конкретного примера обращение к Пролог- системе имеет вид:

```
solve([[c,a,b],[],[]],Solution).
```

Исходная ситуация

[[c,a,b],[],[]].

Целевые ситуации:

[[a,b,c],[],[]]
[[],[a,b,c],[]]
[[],[],[a,b,c]]

Список Solution представляет собой план преобразования исходного состояния в состояние, в котором три кубика поставлены друг на друга в указанном порядке : [a,b,c].

6.2. Стратегия поиска в глубину

Основные стратегии поиска решающего пути - поиск в глубину и поиск в ширину. Идея алгоритма поиска в глубину заключается в следующем. Чтобы найти путь Solution из заданной вершины B в некоторую целевую вершину, необходимо:

- если B - целевая вершина, то положить Solution = [B], или
- если для исходной вершины B существует вершина - преемник B1, такая, что можно провести путь Solution1 из B1 в целевую вершину, то положить

Solution = [B|Solution1].

На Прологе это правило имеет вид:

```
solve(B,[B]) :- goal(B).
solve(B,[B|Solution1]) :- after(B,B1), solve(B1,Solution1).
```

Эта программа реализует поиск в глубину, т.е. когда алгоритму поиска в глубину надлежит выбрать из нескольких вершин ту, в которую следует перейти для продолжения поиска, он предпочитает самую глубокую из них. Самая глубокая вершина - та, которая расположена дальше других от стартовой вершины.

Поиск в глубину наиболее адекватен рекурсивному стилю программирования, принятому в Прологе. Обрабатывая цели Пролог - система сама просматривает альтернативы именно в глубину.

Описанная процедура поиска в глубину страдает одним серьезным недостатком - она не работает в пространстве состояний, имеющем циклы.

Добавим к нашей процедуре механизм обнаружения циклов. Ни одну из вершин, уже содержащихся в пути, построенном из стартовой вершины в текущую, не следует вторично рассматривать в качестве возможной альтернативы продолжения поиска. Это правило можно сформулировать в виде отношения

in_depth(Path,Node,Solution).

Здесь:

- Node - вершина (состояние), из которой необходимо найти путь до цели;
- Path - список вершин (путь) между стартовой вершиной и Node;

- Solution - путь, продолженный до целевой вершины.

Для облегчения программирования вершины в списках, представляющих пути, будут расставляться в обратном порядке. Аргумент Path нужен для того,

(1) чтобы не рассматривать тех преемников вершины Node, которые уже встречались (обнаружение циклов);

(2) чтобы облегчить построение решающего пути Solution.

Программа поиска в глубину без зацикливания имеет вид:

```
solve(Node,Solution) :-in_depth([],Node,Solution).
in_depth(Path,Node,[Node|Path]) :- goal(Node).
in_depth(Path,Node,Solution) :- after(Node,Node1), not(member(Node1,Path)),
                    in_depth([Node|Path],Node1,Solution).
```

Предложенная процедура, снабженная механизмом обнаружения циклов, будет успешно находить пути в конечных пространствах состояний, т. е. в пространствах с конечным числом вершин. Если же пространство состояний бесконечно, то алгоритм поиска в глубину может "потерять" цель, двигаясь вдоль бесконечной ветви графа. Чтобы предотвратить бесцельное блуждание, добавим в процедуру поиска в глубину ограничение на глубину поиска. Тогда эта процедура будет иметь следующие аргументы:

```
in_depth2(Node,Solution,MaxDepth).
```

Не разрешается вести поиск на глубине большей, чем MaxDepth. Программная реализация этого ограничения сводится к уменьшению на единицу величины предела глубины при каждом рекурсивном обращении к in_depth2 и к проверке, что этот предел не стал отрицательным. В результате получаем следующую программу:

```
in_depth2(Node,[Node],_) :- goal(Node).
in_depth2(Node,[Node|Solution],MaxDepth) :-
        MaxDepth > 0, after(Node,Node1),
        Max1=MaxDepth - 1, in_depth2(Node1,Solution,Max1).
```

В отличие от предыдущей процедуры, где порядок вершин обратный, здесь порядок вершин - прямой.

Пример 2. Решение задачи манипулирования кубиками методом поиска в глубину.

```
domains
    s=symbol
    list=s*
    l2=list*
    l3=l2*
predicates
    tgoal(l2)
    delete(list,l2,l2)
```

```
        member(l2,l3)
        after(l2,l2)
        solve(l2,l3)
        in_depth(l3,l2,l3)
        write_list(l3)
clauses
   tgoal([[a,b,c],[],[]]).   /* целевые ситуации */
   tgoal([[],[a,b,c],[]]). tgoal([[],[],[a,b,c]]).
   after(Stolbs,[Stolb1,[Up1|Stolb2]|Rest]):-
        delete([Up1|Stolb1],Stolbs,Stolbs1), delete(Stolb2,Stolbs1,Rest).
        delete(X,[X|L],L).
        delete(X,[Y|L],[Y|L1]) :- delete(X,L,L1).
        member(X,[X|_]).
        member(X,[_|Tail]):- member(X,Tail).
        solve(Node,Solution) :- in_depth([],Node,Solution).
        in_depth(Path,Node,[Node|Path]) :- tgoal(Node).
        in_depth(Path,Node,Solution) :-
                after(Node,Node1), not(member(Node1,Path)),
                in_depth([Node|Path],Node1,Solution).
      write_list([X|Rest]):-   /* вывод решения на терминал */
                write("\n",X),readchar(_), write_list(Rest).
        write_list([]).
goal
        solve([[c,a,b],[],[]],Solution), write_list(Solution).
```

На терминал выводится последовательность ситуаций, начиная с целевой и кончая стартовой (обратный порядок ситуаций).

6.3. Стратегия поиска в ширину

Стратегия поиска в ширину предусматривает переход в первую очередь к вершинам, ближайшим к стартовой. При поиске в ширину приходится сохранять все множество альтернативных путей-кандидатов. Таким образом, цель

```
in_width(Paths,Solution)
```

истинна только тогда, когда в множестве Paths существует такой путь,который может быть продолжен вплоть до целевой вершины.

Представим каждый путь списком вершин, перечисленных в обратном порядке, т.е. головой списка будет самая последняя из порожденных вершин, а последним элементом списка будет стартовая вершина. Поиск начинается с одноэлементного множества кандидатов

```
[[StartNode]].
```

Чтобы выполнить поиск в ширину при заданном множестве путей-кандидатов, нужно:

- если голова первого пути - это целевая вершина, то взять этот путь в качестве решения;

- иначе удалить первый путь из множества кандидатов и породить множество всех возможных продолжений этого пути на один шаг; множество продолжений добавить в конец множества кандидатов, а затем выполнить поиск в ширину с полученным новым множеством.

Пример 3. Решение задачи манипулирования кубиками методом поиска в ширину.

```
domains
        s=symbol
        list=s*
        l2=list*
        l3=l2*
        l4=l3*
predicates
        tgoal(l2)
        delete(list,l2,l2)
        member(l2,l3)
        conc(l4,l4,l4)
        after(l2,l2)
        solve(l2,l3)
        in_width(l4,l3)
        new_node(l2,l3,l3)
        write_list(l3)
clauses
   tgoal([[a,b,c],[],[]]). tgoal([[],[a,b,c],[]]). tgoal([[],[],[a,b,c]]).
   after(Stolbs,[Stolb1,[Up1|Stolb2]|Rest]):-
        delete([Up1|Stolb1],Stolbs,Stolbs1), delete(Stolb2,Stolbs1,Rest).
        delete(X,[X|L],L).
        delete(X,[Y|L],[Y|L1]) :- delete(X,L,L1).
        member(X,[X|_]).
        member(X,[_|Tail]):- member(X,Tail).
        conc([],L,L).
        conc([X|L1],L2,[X|L3]) :- conc(L1,L2,L3).
solve(Start,Solution) :- in_width([[Start]],Solution).
in_width([[Node|Path]|_],[Node|Path]) :- tgoal(Node).
in_width([[B|Path]|Paths],Solution) :- findall(S, new_node(B,S,Path),  NewPaths),
      /* NewPaths - ациклические продолжения пути [B|Path] */
        conc(Paths,NewPaths,Paths1),!, in_width(Paths1,Solution);
      in_width(Paths,Solution).  % случай, когда у B нет преемника
```

```
new_node(B,S,Path) :- after(B,B1), not(member(B1,[B|Path])), S=[B1,B|Path].
write_list([X|Rest]):-write("\n",X),readchar(_), write_list(Rest).
write_list([]).
goal
        solve([[c,a,b],[],[]],Solution), write_list(Solution).
```

6.4. Варианты заданий

1. Решить задачу о перевозке через реку волка, козы и капусты методом поиска в глубину. (Предполагается, что вместе с человеком в лодке помещается только один объект и что человеку приходится охранять козу от волка и капусту от козы.)
2. Решить предыдущую задачу методом поиска в ширину.
3. Решить задачу о коммивояжере методом поиска в глубину.
4. Решить головоломку "игра в восемь" методом поиска в ширину.
5. Решить задачу о восьми ферзях методом поиска в ширину. Расставить 8 ферзей на шахматной доске таким образом, что ни один из ферзей не находится под боем другого.)
6. Задача раскраски карты состоит в приписывании каждой стране на заданной карте одного из четырех заданных цветов с таким расчетом, чтобы ни одна пара соседних стран не была окрашена в одинаковый цвет. Решить задачу методом поиска в глубину.
7. Задача о ханойской башне (упрощенный вариант):

   Имеется три колышка 1, 2 и 3 и три диска a,b и c (a - наименьший из них, c - наибольший). Изначально все диски находятся на колышке 1. Задача состоит в том, чтобы переложить все диски на колышек 3. На каждом шагу можно перекладывать только один диск, причем никогда нельзя помещать больший диск на меньший. Решить эту задачу методом поиска в ширину.

Литература

1. Братко И. Программирование на языке Пролог для искусственного интеллекта. - М.: Мир, 1990.- 560 с.
2. Ин Ц., Соломон Д. Использование Турбо-Пролога. - М.: Мир, 1993. - 608 с.
3. Скляров В.А. Программное и лингвистическое обеспечение персональных ЭВМ. Системы общего назначения: Справ. пособие.- Минск: Выш. шк., 1992.- 462 с.