Московский авиационный институт
(государственный технический университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа №3

по курсу
«Логическое программирование»

Выполнил: Баскаков О.А.
Группа: 08-306
№ по списку: 2
Руководитель: Левинская М.А.
Оценка:
Дата:

Москва
2011 г.

Задание

Написать и отладить Пролог-программу решения задачи искусственного интеллекта, используя технологию поиска в пространстве состояний.
Вариант №3:
Железнодорожный сортировочный узел устроен как стек. На левой стороне собрано некоторое число вагонов двух типов (черные и белые), в произвольном порядке, по n штук каждого. Тупик может вместить все вагоны. Пользуясь тремя сортировочными операциями: слева в тупик, из тупика направо, слева направо минуя тупик), собрать все вагоны на левой стороне, так, чтобы типы чередовались.

Реализация

Состояние задается тремя списками – вагонами слева, справа и в тупике. Например, стартовая позиция:

```
start([L,S,R]) :- L = [b,w,w,b,b,w], S = [], R = [].
```

Финальная – когда все вагоны справа и чередуются:

```
final([Left, Stack, Right]) :- empty(Stack), empty(Left), test(Right).
```

Возможные операции сортировки определяет предикат move:

```
move([[H|Left], Stack, Right], [Left, [H|Stack], Right]).
move([[H|Left], Stack, Right], [Left, Stack, [H|Right]]).
move([Left, [H|Stack], Right], [Left, Stack, [H|Right]]).
```

Необходимо реализовать 3 поиска:
% 1 предикат поиска в глубину
% 2 предикат поиска в ширину
% 3 с итерационным заглублением
Например, так выглядит простейший вариант DFS:

```
dfs(S,S, [S]).
dfs(S,F, [S|Tail]) :- move(S, S2), dfs(S2, F, Tail).
```

Если задать лабиринт и функцию перехода:

```
% лабиринт задается дверьми как граф
door(b, c).
door(a, b).
door(a, e).
door(c, d).
door(c, f).
door(f, e).
door(e, r).
door(r, z).
door(f, z).
move(X, Y):- door(X,Y).
```

Найдем все пути из a в z:

```
?- dfs(a,z,Path).
Path = [a, b, c, f, e, r, z] ;
Path = [a, b, c, f, z] ;
Path = [a, e, r, t, z] ;
false.
```

Вывод списка решения в виде таблицы:

```
show_answer([_]):-!.
show_answer([A,B|Tail]):-
        show_answer([B|Tail]),nl,write(B),write(' -> '),write(A).
```

Для решения поставленной задачи добавляют предикаты продолжения пути prolong(), счетчика итераций int() и встроенный findall().

Реализация поиска на swi-prolog:

```
#! /usr/bin/prolog -s !#

% DFS, Depth-first search
prolong([Temp|Tail],[New,Temp|Tail]):-
        move(Temp,New),not(member(New,[Temp|Tail])).

dpth([Finish|Tail],[Finish|Tail]) :- final(Finish).
dpth(TempWay,Way):-
        prolong(TempWay,NewWay),
        dpth(NewWay,Way).

% BFS, Breadth-first search
bdth([[Finish|Tail]|_],[Finish|Tail]) :- final(Finish).
bdth([TempWay|OtherWays],Way):-
        findall(W,prolong(TempWay,W),Ways),
        append(OtherWays,Ways,NewWays),
        bdth(NewWays,Way).

% Iter
int(1).
int(N):-int(M),N is M+1.

search_iter(Start,                                                  Way):-
int(Level),(Level>100,!;id([Start],Way,Level)).

id([Finish|Tail],[Finish|Tail],0) :- final(Finish).
id(TempWay,Way,N):-N>0,
        prolong(TempWay,NewWay),N1 is N-1,
        id(NewWay,Way,N1).
search_iter(Start, Way):- int(Lev),(Level>100,!;id([Start],Way,Lev)).

d(X) :- start2(S),dpth([S], Path),  Path = [X|_].
b(X) :- start2(S),bdth([[S]], Path),  Path = [X|_].
i(X) :- start2(S),search_iter(S, Path),  Path = [X|_].
```

Тестирование:

```
oleg@debian:~/LP$ ./3.pro
% /home/oleg/LP/3.pro compiled 0.00 sec, 11,412 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.10.1)
Copyright (c) 1990-2010 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.

?- d(DFS).
DFS = [[], [], [w, b, w]] ;
DFS = [[], [], [w, b, w]] ;
DFS = [[], [], [w, b, w]] ;
DFS = [[], [], [w, b, w]] ;
false.

?- b(BFS).
BFS = [[], [], [w, b, w]] ;
BFS = [[], [], [w, b, w]] ;
BFS = [[], [], [w, b, w]] ;
BFS = [[], [], [w, b, w]] ;
false.
```

```
?- i(ITER).
ITER = [[], [], [w, b, w]] ;
ITER = [[], [], [w, b, w]] ;
ITER = [[], [], [w, b, w]] ;
ITER = [[], [], [w, b, w]] ;
true.

?-
% halt
```

Заметим, что все поиски дают одинаковые решения.

Сравним время выполнения поисков:

Поставим отсечение на первом решении.

```
taste_dfs(X) :- d2(X),!, show_answer(X).
taste_bfs(X) :- b2(X),!, show_answer(X).
taste_iter(X) :- i2(X),!, show_answer(X).
```

Результаты

```
oleg@debian:~/LP$ ./3.pro

?- time(taste_dfs(_)).

[[w,w,b,b,b,w,w],[],[]] -> [[w,b,b,b,w,w],[w],[]]
[[w,b,b,b,w,w],[w],[]] -> [[b,b,b,w,w],[w,w],[]]
[[b,b,b,w,w],[w,w],[]] -> [[b,b,b,w,w],[w],[w]]
[[b,b,b,w,w],[w],[w]] -> [[b,b,w,w],[b,w],[w]]
[[b,b,w,w],[b,w],[w]] -> [[b,w,w],[b,b,w],[w]]
[[b,w,w],[b,b,w],[w]] -> [[w,w],[b,b,b,w],[w]]
[[w,w],[b,b,b,w],[w]] -> [[w,w],[b,b,w],[b,w]]
[[w,w],[b,b,w],[b,w]] -> [[w],[w,b,b,w],[b,w]]
[[w],[w,b,b,w],[b,w]] -> [[w],[b,b,w],[w,b,w]]
[[w],[b,b,w],[w,b,w]] -> [[w],[b,w],[b,w,b,w]]
[[w],[b,w],[b,w,b,w]] -> [[],[w,b,w],[b,w,b,w]]
[[],[w,b,w],[b,w,b,w]] -> [[],[b,w],[w,b,w,b,w]]
[[],[b,w],[w,b,w,b,w]] -> [[],[w],[b,w,b,w,b,w]]
[[],[w],[b,w,b,w,b,w]] -> [[],[],[w,b,w,b,w,b,w]]
% 133,784 inferences, 0.078 CPU in 0.091 seconds (85% CPU, 1719887
Lips)
true.

?- time(taste_bfs(_)).

[[w,w,b,b,b,w,w],[],[]] -> [[w,b,b,b,w,w],[w],[]]
[[w,b,b,b,w,w],[w],[]] -> [[b,b,b,w,w],[w],[w]]
[[b,b,b,w,w],[w],[w]] -> [[b,b,w,w],[w],[b,w]]
[[b,b,w,w],[w],[b,w]] -> [[b,b,w,w],[],[w,b,w]]
[[b,b,w,w],[],[w,b,w]] -> [[b,w,w],[b],[w,b,w]]
[[b,w,w],[b],[w,b,w]] -> [[w,w],[b],[b,w,b,w]]
[[w,w],[b],[b,w,b,w]] -> [[w],[b],[w,b,w,b,w]]
[[w],[b],[w,b,w,b,w]] -> [[w],[],[b,w,b,w,b,w]]
[[w],[],[b,w,b,w,b,w]] -> [[],[],[w,b,w,b,w,b,w]]
% 14,475,154 inferences, 8.545 CPU in 8.592 seconds (99% CPU, 1693931
Lips)
true.
```

```
?- time(taste_iter(_)).

[[w,w,b,b,b,w,w],[],[]] -> [[w,b,b,b,w,w],[w],[]]
[[w,b,b,b,w,w],[w],[]] -> [[b,b,b,w,w],[w],[w]]
[[b,b,b,w,w],[w],[w]] -> [[b,b,w,w],[w],[b,w]]
[[b,b,w,w],[w],[b,w]] -> [[b,b,w,w],[],[w,b,w]]
[[b,b,w,w],[],[w,b,w]] -> [[b,w,w],[b],[w,b,w]]
[[b,w,w],[b],[w,b,w]] -> [[w,w],[b],[b,w,b,w]]
[[w,w],[b],[b,w,b,w]] -> [[w],[b],[w,b,w,b,w]]
[[w],[b],[w,b,w,b,w]] -> [[w],[],[b,w,b,w,b,w]]
[[w],[],[b,w,b,w,b,w]] -> [[],[],[w,b,w,b,w,b,w]]
% 109,057 inferences, 0.070 CPU in 0.081 seconds (87% CPU, 1550117
Lips)
true.

?-
% halt
```

Замечания

Можно обратить внимание на тот факт, что поиск в ширину и с итер. заглублением обычно выдают одинаковые оптимальные ответы, но BFS работает медленно и имеет высокие требования к памяти. Как следствие, на больших тестах он не работает.
Поиск в глубину дает неоптимальный ответ, т.к. зависит от порядка move.
Перестановкой можно добиться такого результата:

```
?- time(taste_dfs(_)).
[[w,w,b,b,b,w,w],[],[]] -> [[w,b,b,b,w,w],[],[w]]
[[w,b,b,b,w,w],[],[w]] -> [[b,b,b,w,w],[w],[w]]
[[b,b,b,w,w],[w],[w]] -> [[b,b,w,w],[w],[b,w]]
[[b,b,w,w],[w],[b,w]] -> [[b,w,w],[b,w],[b,w]]
[[b,w,w],[b,w],[b,w]] -> [[w,w],[b,b,w],[b,w]]
[[w,w],[b,b,w],[b,w]] -> [[w],[b,b,w],[w,b,w]]
[[w],[b,b,w],[w,b,w]] -> [[w],[b,w],[b,w,b,w]]
[[w],[b,w],[b,w,b,w]] -> [[],[b,w],[w,b,w,b,w]]
[[],[b,w],[w,b,w,b,w]] -> [[],[w],[b,w,b,w,b,w]]
[[],[w],[b,w,b,w,b,w]] -> [[],[],[w,b,w,b,w,b,w]]
% 27,206 inferences, 0.015 CPU in 0.018 seconds (84% CPU, 1796781
Lips)
true.
```

Выводы

В работе реализованы классические алгоритмы обхода вершин графа. Такое решение задачи далеко до настоящего искусственного интеллекта, но в большинстве случаев позволяет найти приемлемый ответ. Человек мог бы решать задачу более творчески, заранее оценив конфигурации, не приводящие в допустимое состояние.
Показательно, что выполнение декларативной программы при наличии поиска с возвратом(Backtracking) и отсечений(Cuting) сильно зависит от порядка следования определений функции(Clause), в некоторых случаях приводя к зацикливанию.
Наиболее эффективным решением данной задачи оказывается DFS, т.к. можно заранее расположить предикаты перехода move оптимальным образом. Он является «родным» алгоритмом пролога, т.к. используется им при обходе дерева решений.