7. ПРОСТРАНСТВО НА СЪСТОЯНИЯТА И ПРОЛОГ

Въведение. Редица задачи могат да се представят чрез множество състояния на средата, в която се развива задачата. Когато всяко състояние се изобрази чрез връх на граф, а преминаванията от състояние в състояние – чрез дъга, която свързва съответните на състоянията върхове от графа, задачата е представена в пространството на състоянията. Решението на задачата в пространството на състоянията представлява търсене на път в графа от начален връх към обявен за цел връх от графа на състоянията.

Разграничават се два етапа в решаването на такива задачи:

- формулиране на задачата в пространството на състоянията и
- избор на метод за решаване задачата.

Преди построяването на пространството на състоянията е необходимо да се изяснят следните въпроси:

-какви величини ще се използват, за да се опише състоянието на средата?

-какви са възможните действия за преминаване от едно състояние в друго?

-съществуват ли състояния, които са недопустими?

Самото построяване обикновено започва от началното състояние. Към него се прилагат всички възможни промени и се генерират състоянията, които могат да следват това състояние (наследници на началното състояние). От новогенерираните състояния се отстраняват недопустимите.

Същата процедура се прилага и към всяко новопоявило се състояние. Състояния, които вече са били получени и техните наследници са генерирани, не се обработват. С това се гарантира сходимостта на процеса.

Методът за решаване на задачата включва като основен компонент търсене на път в граф, важен момент от който е начинът за обработване на различните варианти. Най-често се използва търсенето в дълбочина или търсенето в ширина.

Ако на всеки преход може да се постави тегло, в пространството на състоянията могат да се дефинират и оптимизационни задачи, например да се намери път с минимално тегло. Оптимизационни задачи могат да се дефинират и без наличието на тегла, например да се намери най-късият път.

Описанието на състоянията често представлява сложна структура от данни и за целта е удобно да се използва функтор (**functor**). Функторът позволява сложната структура да се третира като единно цяло. Той има име и аргументи, които са с фиксирани места, както при аргументите на предиката. Разликата е, че с тези аргументи не могат да се извършват други преобразувания освен да се заместват с променливи и стойности. Клаузите третират функтора като единичен обект. При унифицирането на сложните обекти се унифицират една след друга всичките им съставни части.

Тъй като функторът се дефинира от програмиста, необходимо е функторът да се опише и това става на специално място в програмата.

Постановка. Нека разгледаме задачата за фермера, овцата, вълка и зелката, които са на източния бряг на реката и искат да преминат на нейния западен бряг. На разположение е лодка, но в нея могат да се поберат само двама. Тъй като само фермерът може да гребе, той трябва да реши задачата, като пренася със себе си по един от останалите. Задачата се усложнява, тъй като ако козата и зелката останат на едната страна на реката сами, козата ще изяде зелката. Ако вълкът и козата останат насаме, вълкът ще изяде козата. Налага се фермерът да пресича реката няколко пъти, за да оцелеят всички.

Нека означим състоянието на системата с функтора STATE (F, W, G, C), в която променливите се отнасят за местоположението съответно на фермера (F), вълка (W), козата (G), зелката (C). Всяка от тези променливи има по две стойности – east или west. Тогава началното състояние се изобразява като STATE (east, east, east, east), а целевото състояние – със STATE (west, west, west, west). В нашия функтор позициите на величините и техните стойности са фиксирани по следния начин: на първа позиция е местоположението на фермера, на втора позиция – местоположението на вълка, на трета позиция - местоположението на козата, и четвърта позиция - местоположението на зелката.

Упътване. Дефинира се оператор за противоположност на двата бряга на реката opposite(west, east) и opposite(east,west) с цел да се дефинират опасните разположения на обектите. Тогава функторът предоставя възможност опасните състояния да се изобразят по следния начин.

Едното опасно състояние state(F,_,X,X) :- opposite (F, X). Козата ще изяде зелката, защото двата X означават, че те са от една и съща страна на реката, а в същото време стойностите на F и X са различни, т.е. фермерът не е на същия бряг на реката.

Другото опасно състояние е state(F,X,X,_) :- opposite (F, X). Записът означава вълкът и козата са на един и същ бряг, а фермерът е на отсрещния бряг на реката.

Забраната за избиране на опасните състояния може да изглежда по следния начин:

```
unsafe(state(F,X,X,_)):- opposite(F,X),!.   %опасно състояние. Вълкът ще изяде козата.
unsafe(state(F,_,X,X)):- opposite(F,X),!.   %опасно състояние. Козата ще изяде зелката.
```

Допустимите ходове за преминаване в следващо безопасно състояние изглеждат така:

```
move(state(X,X,G,C), state(Y,Y,G,C)):-opposite(X,Y). %фермерът + вълкът
move(state(X,W,X,C), state(Y,W,Y,C)):-opposite(X,Y). %фермерът + козата
move(state(X,W,G,X),state(Y,W,G,Y)):-opposite(X,Y).  %фермерът + зелката
move(state(X,W,G,C),state(Y,W,G,C)):-opposite(X,Y).  %фермерът
```

Възможните безопасни състояния са следните:

STATE (east, east, east, east) STATE (east, east, east, west)
STATE (west, east, west, east) STATE (west, west, east, west)

STATE (east, east, west, east) STATE (east, west, east, west)
STATE **(**west, east, west, west**)** STATE **(**west, west, west, west**)**

Запомнянето в списък на отработените състояния с цел да не се повтори някое от тях се осъществава със следния запис.

path(NextState,GoalState,[NextState|VisitedPath], Path),!.

Проверката дали текущото състояние не е в списъка на отработените става с известния ни оператор member.

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

В началото на програмата се поставя предикатът go с два аргумента – началното състояние и целевото състояние.

Начинът за търсене на решение може да се опише със следните правила:

```
go(StartState, GoalState):-
   path(StartState, GoalState, [StartState], Path),
   write("Решението е: "),nl,
   write_path(Path).

path(StartState, GoalState, VisitedPath, Path):-
    move(StartState, NextState),              %търси следващо състояние
    not(unsafe(NextState)),                   %безопасно ли е състоянието
    not(member(NextState, VisitedPath)),      %повтаря ли се състоянието
    path(NextState,GoalState,[NextState|VisitedPath], Path),!.
path(GoalState, GoalState, Path, Path).  %достигнато е крайното състояние
```

Примерната програма изглежда по следния начин, като към нея са добавени и оператори за извеждане на решението.

Програма:

```
go(StartState, GoalState):-
   path(StartState, GoalState, [StartState], Path),
   write("Решението е: "),nl,
   write_path(Path).
go(X,Y):- !.

path(StartState, GoalState, VisitedPath, Path):-
   move(StartState, NextState),              %търси преместване
   not(unsafe(NextState)),                   %безопасно ли е състоянието
   not(member(NextState, VisitedPath)),      %повтаря ли се състоянието
   path(NextState,GoalState,[NextState|VisitedPath], Path),!.
path(GoalState, GoalState, Path, Path).  %достигнато е крайното състояние

move(state(X,X,G,C), state(Y,Y,G,C)):-opposite(X,Y).    %фермера + вълка
move(state(X,W,X,C), state(Y,W,Y,C)):-opposite(X,Y).    %фермера + козата
move(state(X,W,G,X),state(Y,W,G,Y)):-opposite(X,Y).     %фермера + зелката
move(state(X,W,G,C),state(Y,W,G,C)):-opposite(X,Y).     %фермера
```

```
opposite(east,west).
opposite(west,east).

unsafe(state(F,X,X,_)):- opposite(F,X),!.   %вълкът ще изяде козата
unsafe(state(F,_,X,X)):- opposite(F,X),!.   %козата ще изяде зелката

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

write_path([H1,H2|T]) :-
   write_move(H1,H2),
   write_path([H2|T]).

write_path([]).

write_move(state(X,W,G,C), state(Y,W,G,C)) :- !,
   write("Фермерът пресича реката от ",X," до ",Y),nl.
write_move(state(X,X,G,C), state(Y,Y,G,C)) :- !,
   write("Фермерът пренася вълка от ", X, " до ", Y),nl.
write_move(state(X,W,X,C), state(Y,W,Y,C)) :- !,
   write("Фермерът пренася козата от ", X, " до ", Y),nl.
write_move(state(X,W,G,X), state(Y,W,G,Y)) :- !,
   write("Фермерът пренася зелката от ", X, " до ", Y),nl.
```

Работа на Пролог. Извикването на примерната програма става с

Reconsulted from: C:\My Documents\Visual Prolog 6 Examples\pie\EXE\fermer_goat_w_c.pro

След поставянето на цел:

go(state(west,west,west,west), state(east,east,east,east))

Полученото решение е следното:

```
Фермерът пренася козата от east до west
Фермерът пресича реката от west до east
Фермерът пренася зелката от east до west
Фермерът пренася козата от west до east
Фермерът пренася вълка от east до west
Фермерът пресича реката от west до east
Фермерът пренася козата от east до west
1 solution
```

Задание

1. Разучете програмата и я трасирайте. Опитайте се да намерите друг вариант на програмата за решаване на същата задача.

2. Съставете програма за пренареждане на колонка от разноцветни кубчета с еднакви размери. На всяка стъпка може да се мести само едно кубче. Кубчето може да се вземе само ако над него няма друго кубче. Върху масата могат да бъдат едновременно не повече от три кубчета.

Забележка. Удобно е колонките от кубчета да се представят чрез списъци.

3. Съставете програма за решаване на следната задача. Да се подредят цифрите в следната таблица във възходящ ред. Допустими са само ходове, при които цифра се премества само в съседно квадратче, ако то е празно.

Например: Начално състояние: Целево състояние:

5	4	7
	6	1
3	2	8

1	2	3
4	5	6
7	8	

Забележка. За удобство може да се разглежа движението на празната клетка.

Внимание. Съществуват случаи, за които задачата няма решение. Нека наречем инверсия всеки случай, когато една цифра предхожда по-малка от нея цифра. За горния пример инверсиите за цифрата 5 са четири на брой. Задачата няма решение, ако броят на инверсиите в началното и крайното състояния са с различна четност.

4. Съставете програма за движение в лабиринт. Представете лабиринта като мрежа от квадратчета, възможностите за преминаване от едно квадратче в друго – като отвор в стената между двете квадратчета. Целта е да се намери изход от лабиринта.

5. Зададена е карта с *n* града. Да се намери път, който започва от начален град А, преминава през всички градове и завърша в град А. Нито един град, с изключение на началния, не бива да се посещава повече от един път.

Забележка. Пробвайте програмата с малък брой градове.

6. Нека в задачата от предната точка пътищата между всяка двойка градове са с отбелязана дължина в километри. Съставете програма за намиране на път с минимална дължина.