

Изкуствен интелект / интелигентни системи

*гл. асистент д-р Венета Табакова-Комсалова
ФМИ, ПУ „П. Хилендарски“, Пловдив*

УПРАЖНЕНИЕ 6

Стандартни предикати

Ø Съществуват три “стандартни” предиката за генериране на всички решения

- `findall/3` събира всички екземпляри в реда, в който те са намерени, но не обработва свободни променливи вдясно
- `setof/3` връща множество от екземпляри (списък в стандартния ред без повторения) и обработва свободни променливи вдясно
- `bagof/3` хибрид между `findall/3` и `setof/3`

Пример

```
likes(иван, вино).  
likes(стоян, бира).  
likes(гого, бира).  
likes(део, вино).  
likes(тони, бира).  
likes(тони, вино).
```



```
setof(X, likes(X,Y), S).
```

```
S = [гого, стоян, тони],
```

```
Y = бира
```

```
S = [део, иван, тони],
```

```
Y = вино
```

```
?-
```

```
setof(X, likes(X,Y), S).
```

```
% ?- setof(X, likes(X, Y), S).
```

```
% ?- setof((Y,S), setof(X,likes(X, Y), S), SS).
```

```
% ?- findall(X, likes(X, Y), S).
```

```
% ?- bagof(X, likes(X, Y), S).
```



```
setof((Y,S), setof(X, likes(X,Y), S), SS).
```

```
SS = [(бира,[гого, стоян, тони]), (вино,[део, иван, тони])]
```

```
?-
```

```
setof((Y,S), setof(X, likes(X,Y), S), SS).
```

Пример



findall(X,likes(X,Y),S).

S = [иван, стоян, гого, део, тони, тони]

?-

findall(X,likes(X,Y),**S**).



bagof(X,likes(X,Y),S).

S = [стоян, гого, тони],

Y = бира

S = [иван, део, тони],

Y = вино

?-


bagof(X,likes(X,Y),**S**).

Пример

- ∅ Променливите в Goal могат да се считат за свободни, освен ако те изрично не са свързани с Goal посредством квантора за съществуване:

$$Y^{\exists} Q$$

означава че “съществува Y такава че Q е true”,
където Y е някаква променлива в Пролог.

 `setof(X, Y^(likes(X,Y)), S).`

S = [гого, део, иван, стоян, тони]

?- `setof(X, Y^(likes(X,Y)), S).`

 `bagof(X, Y^likes(X,Y), S).`

S = [иван, стоян, гого, део, тони, тони]

?- `bagof(X, Y^likes(X,Y), S).`

 `findall(X, likes(X,Y), S).`

S = [иван, стоян, гого, део, тони, тони]

?- `findall(X, likes(X,Y), S).`

Стандартни предикати

`findall (Template, Enumerator, List)`

Намира всички екземпляри на `Template`, за които `Enumerator` е удовлетворим, и ги натрупва в списък по реда, в който те са намерени (т.е. първият намерен екземпляр се поставя като първи елемент на списъка, а последният - като последен елемент на списъка) и свързва `List` с този списък. Обикновено `List` е променлива, а `Template` е променлива или терм, чийто аргументи са променливи, но могат да бъдат и произволни

Стандартни предикати

- Ø `bagof/3` и `setof/3` имат същия формат като `findall/3`:

`bagof(Template, Enumerator, InstanceList)`

`setof(Template, Enumerator, InstanceSet)`

- Ø `bagof/3` прави разлика между променливи в `Enumerator`, които са несвързани и не се съдържат в `Template`. Това означава, че той може да изброява/пресмята свързванията за тези променливи

Пример 2

```
calendar(tom, algebra, monday) .  
calendar(tom, cooking, tuesday) .  
calendar(tom, english, wednesday) .  
calendar(sue, algebra, tuesday) .  
calendar(sue, history, wednesday) .  
calendar(sue, biology, thursday) .
```

```
bagof(Subject, calendar(Person, Subject, _),  
      Subjects)
```

***Subject** - споменава се в Template*

***Person** - искаме да свържем*

***'_'** - искаме да игнорираме*

Пример 2а

?- bagof(Subject, Day^calendar(Person, Subject, Day), Subjects).



bagof(Subject, calendar(Person, Subject, _), Subjects)

Person = sue,
Subjects = [biology]

Person = sue,
Subjects = [algebra]

Person = sue,
Subjects = [history]

Person = tom,
Subjects = [algebra]

Person = tom,
Subjects = [cooking]

Person = tom,
Subjects = [english]



bagof(Subject, Day^calendar(Person, Subject, Day), Subjects).

Person = sue,
Subjects = [algebra, history, biology]
Person = tom,
Subjects = [algebra, cooking, english]

?- bagof(Subject, Day^calendar(Person, Subject, Day), Subjects).

?- bagof(Subject, calendar(Person, Subject, _), Subjects)

Задача 1. Дефинирайте предикат, определящ N-я елемент в даден списък при известно N.

```
nmem(1,[X|_], X):-!.
```

```
nmem(N, [_|T], X):- N>1, N1 is N-1, nmem(N1,T,X).
```

Задача 2. Дефинирайте предикат, чрез който може да се вмъкне даден терм на N-тото място в даден списък.

`ins(1, X, T, [X|T]).`

`ins(N, X, [H|T], [H|L]) :- N > 1, N1 is N-1, ins(N1, X, T, L).`

`?- ins(3, g, [1,2,4,3,5,6,3], Y).`

Задача 3. Дефинирайте предикат, който проверява дали един списък се получава от друг чрез замяна на всеки елемент на първия списък, равен на терм A, с терма B

`subst(_,_,[], []).`

`subst(A,B, [A|T], [B|L]) :- !, subst(A,B,T,L).`

`subst(A,B, [X|T], [X|L]) :- subst(A,B,T,L).`

`?-subst(3, g, [1,2,4,3,5,6,3], Y)`

Задача 4. Дефинирайте предикат, който от даден числов списък генерира два списъка: единият, съдържащ елементите на даден списък, които са по-малки от дадено число X , а другият, съдържащ всички останали елементи на първия списък.

`split(_, [], [], []).`

`split(X, [H|T], [H|T1], T2):- H<X, !, split(X,T,T1,T2).`

`split(X, [H|T], T1, [H|T2]) :- split(X,T,T1,T2).`

`?- split(3,[1,2,4,3,5,6,3],B, Y)`

ЗАДАЧА 5: Осемте царици

Да се разположат N царици върху шахматна дъска с размер $N \times N$, така че нито една царица да не бие друга царица. Цариците могат да се местят хоризонтално, вертикално или по диагонал. Следващата диаграма показва решение за $N=4$ царици. Решението на тази задача може да се представи като специална пермутация на списъка $[1, 2, 3, 4]$. Например, решението показано по-горе може да се представи като $[3, 1, 4, 2]$, което означава, че в първия ред царицата се намира в третата колона, във втория ред царицата се намира в първа колона и т.н. Да се провери дали дадена пермутация е решение трябва да се изчисли дали пермутацията има две или повече царици на един и същ диагонал. Представянето само по себе си предпазва две или повече царици да са в един и същи ред или колона. Две царици са на един и същ диагонал тогава и само тогава, когато сумата от ред и колона е една и съща за двете царици. Те са на един и същи диагонал и тогава и само тогава, когато разликата на техните редове и колони е едно и също число.

	X	0	
0			X
X			0
	0	X	

```

1 solution([]).
2 solution([X/Y|Others]):-
3   solution(Others),
4   member(Y,[1,2,3,4,5,6,7,8]),
5   noattack(X/Y,Others).
6 noattack(_,[]).
7 noattack(X/Y,[X1/Y1|Others]):-
8   Y=\=Y1, %различни Y-координати
9   Y1-Y=\=X1-X, %различни диагонали
10  Y1-Y=\=X-X1,
11  noattack(X/Y,Others).
12 % инициализация на списъка от позициите на осемте царици
13 template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
14 /*Примерна цел е:
15 ?-template(S), solution(S).
16 */

```

S = [1/5, 2/2, 3/4, 4/7, 5/3, 6/8, 7/6, 8/1]

S = [1/3, 2/5, 3/2, 4/8, 5/6, 6/4, 7/7, 8/1]

S = [1/3, 2/6, 3/4, 4/2, 5/8, 6/5, 7/7, 8/1]

S = [1/5, 2/7, 3/1, 4/3, 5/8, 6/6, 7/4, 8/2]

S = [1/4, 2/6, 3/8, 4/3, 5/1, 6/7, 7/5, 8/2]

S = [1/3, 2/6, 3/8, 4/1, 5/4, 6/7, 7/5, 8/2]

S = [1/5, 2/3, 3/8, 4/4, 5/7, 6/1, 7/6, 8/2]

S = [1/5, 2/7, 3/4, 4/1, 5/3, 6/8, 7/6, 8/2]

S = [1/4, 2/1, 3/5, 4/8, 5/6, 6/3, 7/7, 8/2]

S = [1/3, 2/6, 3/4, 4/1, 5/8, 6/5, 7/7, 8/2]

S = [1/4, 2/7, 3/5, 4/3, 5/1, 6/6, 7/8, 8/2]

S = [1/6, 2/4, 3/2, 4/8, 5/5, 6/7, 7/1, 8/3]

Next 10 100 1,000 Stop

?- template(S), solution(S).

Задача 6. Разбиване на един долар на монети
(половин долар, четвърт долар, десетаче, петаче и
един цент)

<https://swish.swi-prolog.org/p/ChangeFor1Euro.pl>


```
solution([]).
solution([X/Y|Others]):-
solution(Others),
member(Y,[1,2,3,4,5,6,7,8]),
noattack(X/Y,Others).
noattack(_,[]).
noattack(X/Y,[X1/Y1|Others]):-
Y=\=Y1, %различни Y-координати
Y1-Y=\=X1-X, %различни диагонали
Y1-Y=\=X-X1,
noattack(X/Y,Others).
% инициализация на списъка от позициите на
осемте царици
template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
/*Примерна цел е: ?-template(S), solution(S). */
```

Задача 7. Четири къщи

/*Има 4 къщи на една и съща улица. Всеки от тях е дом на един от 4-ма души: Симо, Николай, Ангел и Радо.

Всеки от тях има професия: лекар, художник, ловец, треньор. Определете кой в коя къща живее и кой каква професия притежава.

Известно е, че:

Художникът живее до треньора.

Лекарят живее до Художника.

Ловецът е вляво от лекаря.

Треньорът не е до ловеца.

Художникът е отдясно на Симо.

Радо не е треньор.

Симо е до Николай.

Ангел не е до Радо. */

```

:- use_rendering(table,[header(house('№ на
къща', 'Име', 'Професия'))]).
houses([ house(first, _, _), house(second, _, _),
          house(third, _, _), house(forth, _, _ ) ]).
left_of(A, B, [A, B | _]).
left_of(A, B, [_ | Y]) :- left_of(A, B, Y).
right_of(A, B, [B, A | _]).
right_of(A, B, [_ | Y]) :- right_of(A, B, Y).
next_to(A, B, [A, B | _]).
next_to(A, B, [B, A | _]).
next_to(A, B, [_ | Y]) :- next_to(A, B, Y).

mymember(X, [X|_]).
mymember(X, [_|Y]) :- mymember(X, Y).
not_in(_, []).
not_in(X, [X|_]) :- !, fail.
not_in(X, [_|A]) :- not_in(X, A).
not_trainer(H) :- mymember(house(_, rado, hunter),
H).
not_trainer(H) :- mymember(house(_, rado, doctor),
H).
not_trainer(H) :- mymember(house(_, rado, painter)
,H).
not_trainer(H) :- mymember(house(_, rado, trainer),
H), !, fail.

```

```

print_houses([]).
print_houses([A|B]) :- write(A), nl, print_houses(B).

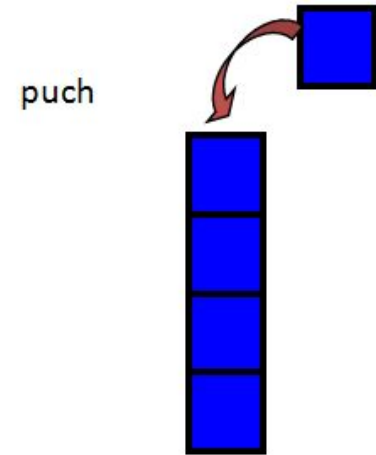
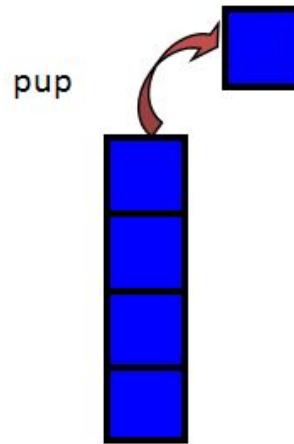
false(X) :- X, !, fail. false(_).

solve(H) :- houses(H), next_to(house(_, _, painter),
house(_, _, trainer), H),
next_to(house(_, _, doctor), house(_, _, painter), H),
left_of(house(_, _, hunter), house(_, _, doctor), H),
false(next_to(house(_, _, trainer),
house(_, _, hunter), H)),
right_of(house(_, _, painter), house(_, simo, _), H),
next_to(house(_, simo, _), house(_, nikolay, _), H),
false(next_to(house(_, angel, _), house(_, rado, _), H
))),
not_trainer(H), mymember(house(_, rado, _), H),
mymember(house(_, angel, _), H),
mymember(house(_, simo, _), H),
mymember(house(_, nikolay, _), H),
print_houses(H).

```

% ?- solve(H).

СТЕК



Основни операции

push - поставяне на елемент
push(X, Stack, [X|Stack]).

pop - вземане на елемент
*pop(Stack, _, _) :- empty_stack(Stack), write('Empty stack'), nl.
pop([X|Stack], X, Stack).*

empty_stack - проверка дали даден стек е празен
empty_stack([]).

ОПАШКА

Основни операции

push - поставяне на елемент

push(X, Queue, NewQueue) :- append(Queue, [X], NewQueue).

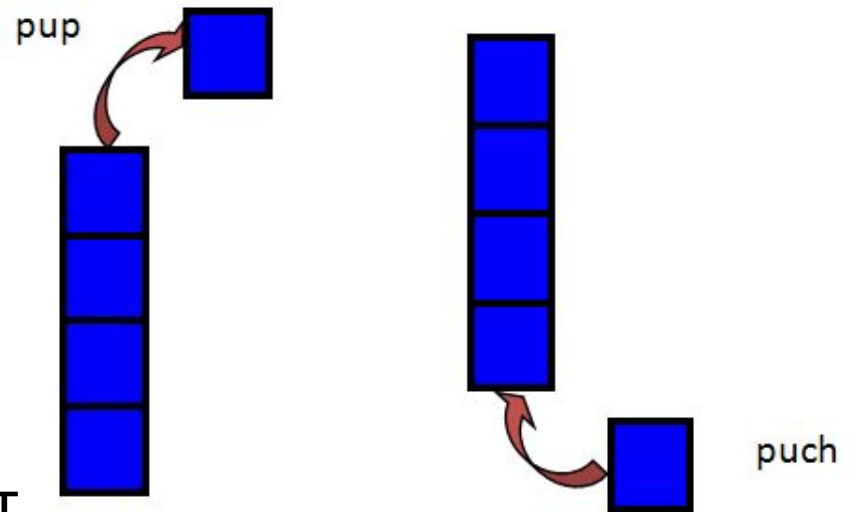
pop - вземане на елемент

pop(Queue, _, _) :- empty_queue (Queue), write('Empty queue'), nl.

pop([X|Queue], X, Queue).

empty_queue - проверка дали дадена опашка е празна

empty_queue([]).



Задача: Проверка дали даден елемент
принадлежи на даден стек/опашка

in_stack(X, Stack)

in_queue(X, Queue)

in_stack(X, Stack):-

\+ empty_stack(Stack),!,

pup(Stack, Y, RestStack),

(X=Y, ! ; in_stack(X, RestStack)).

ГРАФ

Нека разгледаме следния свързан граф:

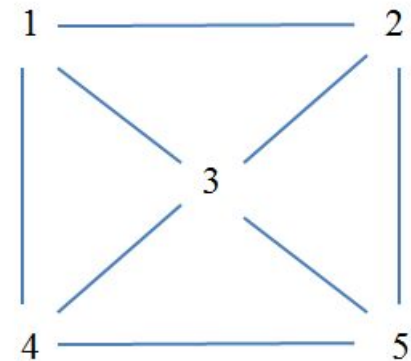
Дъгите граф могат да бъдат представени в Пролог като факти:

edge(1,2). edge(1,3). edge(1,4).

edge(2,3).

edge(2,5). edge(3,4). edge(3,5).

edge(4,5).



За да представим факта, че дъгите са двустранно свързани можем да добавим осем допълнителни „edge” факта (*edge(2,1); edge(3,1) и т.н.*) или можем да използваме правило като:

(*) *edge(X,Y):-edge(Y,X).*

!Това не е добра идея.

За да се види защо това не е добра идея, може да се опита следната цел:

?-edge(5,1).

Ще отбележим, че правилото (*) ще бъде извикано отново и отново в **безкраен цикъл**, така че целта няма да свърши. По-добър начин за справяне с това е да използваме следното правило:
`connected(X,Y):-edge(X,Y);edge(Y,X).`

Забелязва се използването на дезюнкцията „ ; “ в правилото. Това правило може да се представи със следните две правила:

connected(X,Y):-edge(X,Y).

connected(X,Y):-edge(Y,X).

Сега ще напишем дефиниция на предикат, който генерира път между два върха в един граф.

Пътят е представен чрез списък от върхове, през които трябва да се мине, за да се стигне от връх 1 до връх 5. Ето я дефиницията на предиката:

```
path(A,B,Path):- travel(A,B,[A],Q), reverse(Q,Path).
```

```
travel(A,B,P,[B|P]):- connected(A,B).
```

```
travel(A,B,Visited,Path):- connected(A,C), C\== B, \+member(C,Visited),  
    travel(C,B,[C|Visited],Path).
```

Декларативното прочитане на второто правило означава "Пътят от А до В е получен, ако А и В са свързани".

Декларативното прочитане на третото правило означава "Пътят от А до В е получен, ако е осигурено, че А е свързан с С, С е различен от В и В все още не е посещаван и се продължава да се търси път от С до В".

Поведението на този предикат е следното:

```
?-path(1,5,P).
```

```
P=[1,2,5];
```

```
P=[1,2,3,5];
```

```
P=[1,2,3,4,5];
```

```
P=[1,4,5];
```

```
P=[1,4,3,5];
```

```
P=[1,4,3,2,5];
```

```
P=[1,3,5];
```

```
P=[1,3,4,5];
```

```
P=[1,3,2,5];
```

```
no
```

Избягването на повтарящите се възли осигурява програмата да не зацikli безкрайно.

```
shortest(A,B,Path,Length):-  
  setof([P,L],path(A,B,P,L),Set),  
  Set=[_],  
  minimal(Set,[Path,Length]).
```

```
minimal([F|R],M):- min(R,F,M).
```

МИНИМАЛЕН ПЪТ

```
min([],M,M).
```

```
min([[P,L]|R],[_ ,M],Min):- L<M, !, min(R,[P,L],Min).
```

```
min([_|R],M,Min):- min(R,M,Min).
```

структурата граф и пътища в графи

```
edge(1,2). edge(1,4).  
edge(1,3). edge(2,3).  
edge(2,5). edge(3,4).  
edge(3,5). edge(4,5).  
connected(X,Y):- edge(X,Y); edge(Y,X).  
path(A,B,Path):- travel(A,B,[A],Q),  
reverse(Q,Path).  
    travel(A,B,P,[B | P]):- connected(A,B).  
    travel(A,B,Visited,Path):- connected(A,C),  
        C\==B, \+member(C,Visited),  
        travel(C,B,[C | Visited],Path).
```

Хамилтонов път в граф се нарича ацикличен път, съдържащ всички върхове на графа.

В графа от фигурата хамилтоновите пътища са:

$[a,b,c,d]$, $[a,b,d,c]$, $[d,c,b,a]$, $[c,d,b,a]$

Ще дефинираме отношение $\text{hamilton}(G,W)$, което намира хамилтонов път W в графа G , ако такъв съществува.

$\text{hamilton}(G,W):- \text{way}(A,B,G,W), \text{not}(\text{top}(C,G), \text{not member}(C,W)).$

$\text{top}(X,\text{graph}(V,R)):- \text{member}(X,V).$

$\text{way}(A,Z,G,P):- \text{way1}(A,[Z],G,P).$

$\text{way1}(A,[A|W],G,[A|W]).$

$\text{way1}(A,[Y|W],G,P):- \text{rib}(X,Y,G), \text{not member}(X,W),$

$\text{way1}(A,[X,Y|W],G,P).$

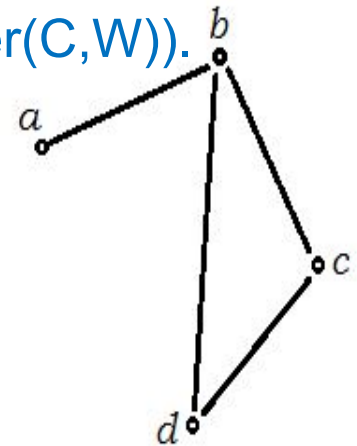
$\text{rib}(X,Y,\text{graph}(V,R)):- \text{member}(p(X,Y),R);$

$\text{member}(p(Y, X), R).$

Отношението $\text{way}(A,B,G,W)$ намира произволен път W от A до B в G , а целта $\text{not}(\text{top}(C, G), \text{not member}(C, W))$ проверява дали всеки връх C на графа G се съдържа в пътя W .

Отношението $\text{top}(X, G)$, определящо дали X е връх на графа G , може да се дефинира по следния начин: $\text{top}(X,\text{graph}(V,R)):- \text{member}(X, V).$

Ако всяко ребро на графа е свързано със стойност, възможно е да се търсят пътища с определена стойност.



ДЪРВЕТА

Покриващо дърво за графа $G=(V, R)$ е свързан граф $T=(V, R')$, където R' е такова подмножество на R , че в T няма цикли.

Даден е свързан неориентиран граф G , представен чрез списък от съставлящите го ребра. Ще напишем програма, която намира покриващо дърво T на G .

Ще дефинираме отношението `cov_tree(G, T)`, определящо покриващо дърво T на графа G .

T е покриващо дърво на графа G , ако:

- T е подмножество на G и
- T е дърво и
- T "покрива" G , т.е. всеки връх на G се съдържа в T .

Множеството от ребра T е дърво, ако:

- T е свързан граф и
- T не съдържа цикли.

cov_tree(G,T):-subset(G,T), tree(T), cover(T,G).

subset([],[]).

subset([X|L],S):- subset(L, L1), (S=L1; S=[X|L1]).

% T е подмножество на G

tree(T):- connected(T), not have_a_cycle(T).

% T е дърво

connected(T):- not (top(A,T), top(B,T), not way(A,B,T,_)).

% T е свързан граф

have_a_cycle(T):- rib(A, B,T), way(A,B,T,[A,X,Y|_]).

%T не съдържа цикли.

дължината на пътя е по-голяма от 1

cover(T,G):- not (top(A,G), not top(A, T)).

ТЪРСЕНЕ

Търсене в графи

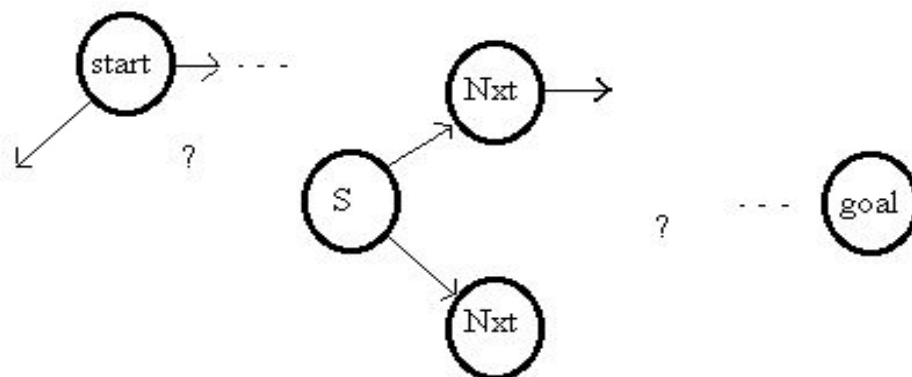
Нека да разгледаме ситуации, където търсенето може да бъде специфицирано със започване от начално състояние, генериране на ходове до следващи възможни състояния, проверка дали ходът е сигурен или позволен, проверка дали следващото състояние не е вече посещавано и след това търсенето продължава от това следващо състояние. На Пролог тази спецификация изглежда по следния начин:

```
solve(P):- start(Start), search(Start, [Start],Q),
reverse(Q,P).
search(S,P,P):- goal(S), !.
search(S,Visited,P):- next_state(S,Nxt), safe_state(Nxt),
no_loop(Nxt,Visited), search(Nxt,[Nxt|Visited],P).
no_loop(Nxt,Visited):- \+member(Nxt,Visited).
```

Това не е цялата програма. Този програмен фрагмент представлява един вид общо търсене. Нуждаем се от малко спецификация.

```
next_state(S,Nxt):- <fill in here>.  
safe_state(Nxt):- <fill in here>.  
no_loop(Nxt,Visited):- <fill in here>.  
start(...).  
goal(...).
```

Диаграмата разкрива търсенето по следния начин:



Като пример ще разгледаме задачата Осемте царици

			Q			
	Q					
					Q	
			Q			

```
solve(P):-
perm([1,2,3,4,5,6,7,8],P),
combine([1,2,3,4,5,6,7,8],P,S,D),
all_diff(S), all_diff(D).
```

```
combine([X1|X],[Y1|Y],[S1|S],[D1|D]):-
S1 is X1+Y1, D1 is X1-Y1,
combine(X,Y,S,D).
combine([],[],[],[]).
```

```
all_diff([X|Y]):-
\+member(X,Y), all_diff(Y).
all_diff([X]).
```

```
?solve(P).
P=[5,2,6,1,7,4,8,3]
```

```
?- setof(P,solve(P),Set), length(Set,L)
L=92
```

Задача 8. Екскурзия до Пловдив

/* Студенти от Историческия факултет на Великотърновския университет Асен, Васил, Мария и Елена решили да посетят и разгледат историческите забележителности на Пловдив като Европейска столица на културата. По пътя обаче се оказало, че четиримата трябва да пишат реферати за различни исторически периоди – тракийски, римски, български и възрожденски.

Пловдив, като град с богата хилядолетна история давал възможност на всеки от тях да задоволи интереса си и да събере необходимата информация. Затова те решили да се разделят и да посетят различни музеи.

Известни са следните факти:

1. Васил решил да посети природонаучния музей в града, тъй като наскоро беше посетил града и вече беше събрал нужната му информация.
 2. В историческия музей беше открита изложба от периода на Възраждането.
 3. Мария се интересува от периода на Първата и Втора българска държава, а Елена ще пише реферат за тракийския период
 4. В етнографския музей в момента няма експонати от римския период
 5. В археологическия музей няма експонати от Първата и Втора българска държава и Възраждането.
- Открийте кой студент кой музей е посетил и за кой исторически период ще пише реферат. */

Известни са следните факти:

1. Васил решил да посети природонаучния музей в града, тъй като наскоро беше посетил града и вече беше събрал нужната му информация.
 2. В историческия музей беше открита изложба от периода на Възраждането.
 3. Мария се интересува от периода на Първата и Втора българска държава, а Елена ще пише реферат за тракийския период
 4. В етнографския музей в момента няма експонати от римския период
 5. В археологическия музей няма експонати от Първата и Втора българска държава и Възраждането.
- Открийте кой студент кой музей е посетил и за кой исторически период ще пише реферат. */

st(ime, muzey, period).

S=[st(asen, muzey, period), st(vasil, muzey, period), st(maria, muzey, period), st(elena, muzey, period)]

member(st(vasil,priroda, _), S). member(st(_,istoria, vazrajthane), S).
not(member(st(_,etno, romski), S)).

Задача 9

За Коледа родителите на Петя, Лили, Иво, Камен и Вики решили да им направят за подарък музикален инструмент на който те могат да свирят (китара, синтезатор, барабани, цигулка, флейта), защото се обучават в Национално училище за музикално и танцово изкуство „Добрин Петков“, град Пловдив. В началото на месец декември в най-големия магазин за музикални инструменти в града “Black Friday”. Родителите побързали и били първите пет клиенти на магазина в този ден.

Родителят на детето, което свири на цигулка влезнал първи в магазина, веднага открил търсения инструмент и бил първи при отварянето на касата. Родителят на Лили влезнал втори в магазина, но трябвало да се консултира с продавач-консултант и когато отишъл на касата бил четвърти. Родителят на Иво се забавил в магазина, защото му трябвала количка, предвид големината на синтезатора, който купил и бил последен на опашката от тези родители. Родителят на едно от децата влезнал последен, но бързо открил флейта и затова бил на касата пред родителя на Лили. Вики получил барабани на Коледа, а баща му бил след родителя с цигулката на касата в магазина.

Кое дете на какъв инструмент свири? Кой родител кой по ред в бил на касата в магазина?

Дете	Инструмент на който свири	Родител влезнал в магазина	Родител излезнал от магазина
Петя	цигулка	1	1
Лили	китара	2	4
Иво	синтезатор		5
Камен	флейта		3
Вики	барабани		Не е 1 ->2