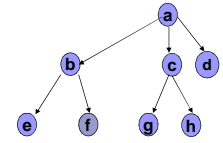


Prohledávání grafů

1

Prohledávání stromu:g1

```
hr(a,b). hr(a,c). hr(a,d).
hr(b,e). hr(b,f).
hr(c,g). hr(c,h).
cesta(X,X).
cesta(Z,D):-hr(Z,N),cesta(N,D).
```

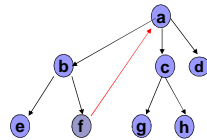


```
% ?- cesta(a,X).
% X = a ; X = b ;
% X = e ; X = f ; X = c ; X = g ; X = h ;
% X = d ; no
% Pořadí určuje SLD strategie, tj. „prohledávání do hloubky“
```

Umělá inteligence I.

Prohledávání grafu:g1a

```
hr(a,b). hr(a,c). hr(a,d).
hr(b,e). hr(b,f). hr(f,a)
hr(c,g). hr(c,h).
cesta(X,X).
cesta(Z,D):-hr(Z,N),cesta(N,D).
?- cesta(a,X).
```

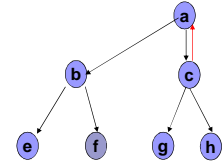


```
X = a ; X = b ; X = e ; X = f ;
X = a ; X = b ; X = e ; X = f ;
X = a ; X = b ; X = e ; X = f ; ...
% k uzlům c, d, g, h se SLD strategie „prohledávání do hloubky“
nikdy nedostane
?- cesta(f,c).
```

Umělá inteligence I.

Prohledávání obecného grafu:g1b

```
hr(a,b). hr(a,c).
hr(b,e). hr(b,f).
hr(c,a). hr(c,g). hr(c,h).
cesta(X,X).
cesta(Z,D):-hr(Z,N),cesta(N,D).
% ?- cesta(a,X).
% X = a ; X = b ; X = e ; X = f ; X = a ; X = b ; X = e ; X = f ; X = a ; ...
% ?- cesta(a,g). Abort
```

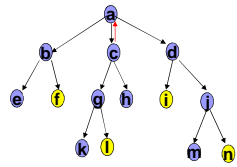


Jak ovlivní výpočet změna pořadí : hr(c,g), hr(c,h), hr(c,a). ? viz **g1c**

Jak řešit tento problém? Je třeba
 * buď omezit hloubku
 * zkontrolovat, že se nevracíme do bodu, kudy jsme už prošli.

Prohledávání do omezené hloubky:g2

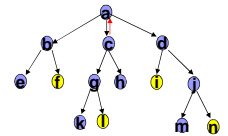
```
/* Forma popisu orient.grafu:
seznam hran hr(Poc_uzel,Konc_uzel)
a seznam cilovych uzlu cil(Uzel) */
hr(a,b). hr(a,c). hr(a,d).
hr(b,e). hr(b,f). hr(c,g).
hr(c,h). hr(c,a).
hr(g,k). hr(g,l). hr(d,i).
hr(d,j). hr(j,m). hr(j,n).
/* Výčet cílových uzlů */
cil(f). cil(i). cil(l). cil(n).
% omezena_hl(+VychoziciUzel, -Reseni, +Max_hloubka)
omezena_hl(Uzel,[Uzel], _):- cil(Uzel).
omezena_hl(Uzel,[Uzel|CastReseni], Max_Hl):-
Max_Hl > 0,hr(Uzel,Naslednik),Max1 is Max_Hl -1,
omezena_hl(Naslednik, CastReseni, Max1 ).
```



Umělá inteligence I.

Prohledávání do omezené hloubky:g2

```
hr(a,b). hr(a,c). hr(a,d).
hr(b,e). hr(b,f). hr(c,g).
hr(c,h). hr(c,a).
hr(g,k). hr(g,l). hr(d,i).
hr(d,j).
hr(j,m). hr(j,n).
cil(d).cil(f).cil(i).cil(l). cil(n).
omezena_hl(Uzel,[Uzel], _):- cil(Uzel).
omezena_hl(Uzel,[Uzel|CastReseni], Max_Hl):-
Max_Hl > 0,h(Uzel,Naslednik),Max1 is Max_Hl -1,
omezena_hl(Naslednik, CastReseni, Max1 ).
?- omezena_hl(a,Kudy,2). Kudy= [a,b,f]; [a,b,i]; No
?- omezena_hl(a,Kudy,5). Kudy= [a,b,f]; [a,c,g,l];
[a,c,a,b,f]; [a,c,a,c,g,l]; [a,c,a,d,i];[a,c,a,d,j,n];
[a,d,i]; [a,d,j,n]; No
```



Umělá inteligence I.

Prohledávání grafu se záznamem cesty 1: g2

```

hr(a,b).    hr(a,c).    hr(a,d).
hr(b,e).    hr(b,f).    hr(c,g).
hr(c,h).    hr(c,a).
hr(g,k).    hr(g,l).    hr(d,i).
hr(d,j).    hr(j,m).    hr(j,n).
cil(d).     cil(f).     cil(i).     cil(l).     cil(n).
% c_hl(+Vychozi_uzel,-Cilovy_uzel,-Cesta)
c_hl(X,X,[]) :- cil(X).
c_hl(X,Y,[M|L]) :- hr(X,M),c_hl(M,Y,L).
?- c_hl(a,X,S).    X=f, S=[b,f]; X=l, S=[c,g,l];
                  X=f,S=[c,a,b,f]; X=l,S=[c,a,c,g,l]; X=f, S=[c,a,c,a,b,f];
                  X=l,S=[c,a,c,a,c,g,l]; X=f, S=[c,a,c,a,c,a,b,f]; ...

```

■ Můžeme zkontrolovat, že M není v L?

Prohledávání grafu se záznamem cesty 2

```

c_hl_1(X,X,[]) :- cil(X).
c_hl_1(X,Y,[M|L]) :-
hr(X,M),not member(M,L),c_hl_1(M,Y,L).

```

■ ?- c_hl_1(a,e,S).

?- c_hl_1(a,Kam,Jak). No

Důvod? **not** je použito na seznam, jehož hodnotou je proměnná. V takovém případě odpovídající dotaz **vždy uspěje!!!** Např.

?-member(a,S). S = [a|L]

Tedy naopak **not member(X,L)** selže.

Prohledávání grafu se záznamem cesty 2: g2

```

hr(a,b).    hr(a,c).    hr(a,d).
hr(b,e).    hr(b,f).    hr(c,g).
hr(c,h).    hr(c,a).
hr(g,k).    hr(g,l).    hr(d,i).
hr(d,j).    hr(j,m).    hr(j,n).
cil(d).     cil(f).     cil(i).     cil(l).     cil(n).
% c_hl_n(+Vychozi_uzel,-Cilovy_uzel,-Cesta)
c_hl_n(X,X,[]) :- cil(X).
c_hl_n(X,Y,[M|L]) :- hr(X,M),c_hl_n(M,Y,L),not member(M,L).

```

■ ?- c_hl_n(a,X,S). X=f, S=[b,f]; X=l, S=[c,g,l]; X=f,S=[c,a,b,f]; **abort**

Kontrola zacyklení: g3

```

/* efektivnější řešení:kontrola výskytu
s pomocným seznamem UvodReseni */
res(VychoziU,Reseni) :-
prohlizeni_hl([VychoziU],VychoziU,Reseni).
prohlizeni_hl(UvodReseni,Uzel,UvodReseni):- cil(Uzel).
prohlizeni_hl(UvodReseni,Uzel,Reseni):-
hr(Uzel,NaslU),
not member(NaslU, UvodReseni),
prohlizeni_hl([NaslU|UvodReseni],NaslU,Reseni).

```

S = [f, b, a] Yes (0.00s cpu, solution 1, maybe more)
S = [l, g, c, a] Yes (0.00s cpu, solution 2, maybe more)
S = [i, d, a] Yes (0.00s cpu, solution 3, maybe more)
S = [n, j, d, a] Yes (0.00s cpu, solution 4, maybe more)
No (0.00s cpu)

Predikáty 2. Řádu setof (+Z,+Cil(Z,X),-S)

Fixuje X a nalézá neprázdný seznam S všech instancí za Z, pro které platí Cil(Z, X):

- seznam S chápe jako uspořádanou množinu, tj. bez opakování
- pokud řešení pro Cil(Z,X) neexistuje, pak výsledkem volání predikátu setof byl seznam prázdný, a proto program hlásí neúspěch

Příklad (g4):

```

hr(a,b).    hr(a,c).    hr(a,d).
hr(b,e).    hr(b,f).
hr(c,g).    hr(c,h).    hr(c,a).

```

```

?- setof(Z,hr(a,Z),S).    S = [b,c,d]
?- setof(Z,hr(X,Z),S).
Z = Z    X = a    S = [b, c, d]    Yes (0.00s cpu, solution 1, maybe more)
Z = Z    X = b    S = [e, f]      Yes (0.03s cpu, solution 2, maybe more)
Z = Z    X = c    S = [a, g, h]    Yes (0.03s cpu, solution 3)
?- setof(X, hr(d, X), S).    No (0.00s cpu)

```

Predikáty 2. Řádu bagof (+Z,+Cil(Z,X),-S)

Proměnně lze ve výrazu +Cil(Z,X) **existenčně kvantifikovat** pomocí značky ^

```

hr(a,b).    hr(a,c).    hr(a,d).
hr(b,e).    hr(b,f).
hr(c,g).    hr(c,h).    hr(c,a).

```

```

?- setof(Z,X^hr(X,Z),S).    S = [a, b, c, d, e, f, g, h]

```

bagof(+Z,+Cil(Z,_),-S) nalézá seznam S všech instancí za Z, které jsou řešením pro cíl Cil(Z,_), prvky se v S mohou opakovat --> rychlejší než setof

```

?- bagof(Z, h(a,Z), S).    S = [ b, c, d ]
% Pozor ! Z uzlu d nevede žádná hrana, proto
?- bagof(Z, h(d,Z), S).    No.

```

Predikáty 2. řádu

findall (+Term(Z),+Cil(Z),-S)

podobný jako **bagof**, ovšem **vždy končí úspěchem** :

nalézá seznam S (s opakováním) všech takových **Z** instancí výrazu **Term**, pro které platí **Cil(Z)**

v případě, že **neexistuje** objekt splňující **Cil(Z)**, je **S = []**

Příklad:

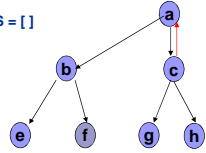
```
h(a,b).      h(a,c).
h(b,e).      h(b,f).
h(c,g).      h(c,h).      h(c,a).
```

```
?- findall(symetricky(X,Y), (h(X,Y),h(Y,X)), S).
```

```
S = [symetricky(a,c)]
```

```
?- findall(symetricky(b,Y), (h(b,Y),h(Y,b)), S). S = []
```

```
?- findall(Z, h(f,Z), S). S = []
```



15 / 23

Umělá inteligence I.



Prohledávání stavového prostoru do šířky

pridej_za(+Open,+Uzel,-Vysledek) nalezne všechny přímé násl.

uzlu **Uzel**, tento seznam přidá za **Open** a tak vznikne **Vysledek**.

vlna(+Open,Z) hledá cestu, t.ž. začíná v některém prvku seznamu **Open** a končí v cílovém uzlu **Z** (**Open** odpovídá seznamu **Open nodes** používaném v klas. implementaci **prohl.stav.prost.**)

dotaz **?-c_sir(a,X)**. Najde "nejbližší" cílový uzel dostupný z **a**

```
c_sir(X,X) :- cil(X).
```

```
c_sir(X,Y) :- vlna([X],Y).
```

```
vlna([],_) :- write('Vic cil.uzlu neni.').nl.
```

```
vlna([H|_],H) :- cil(H).
```

```
vlna([Y|L],Z) :- pridej_za(L,Y,N), vlna(N,Z).
```

```
pridej_za(L,Y,N) :- primi_naslednici(Y,S), append(L,S,N).
```

```
primi_naslednici(Y,S) :- bagof(Z,hr(Y,Z),S),!.
```

```
primi_naslednici(Y,[]).
```

16 / 23

Umělá inteligence I.



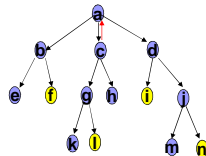
Prohledávání grafu do šířky: g2

```
?-c_sir(a,X).
```

```
X = f; X = i; X = l; X = n;
```

```
X = f; X = i; X = l; X = n;
```

```
X = f; X = i; X = l; X = n;
```



Cvičení:

Upravte program tak, aby podával i informace o cestě použité k dosažení cíle!

15 / 23

Umělá inteligence I.



Prohledávání grafu - potřebné prostředky Prologu

- **stromy a acyklické grafy** - stačí čistý Prolog
- **konečné grafy s případným cyklem** - Prolog + **negace ***
- **nekonečné grafy** (lokálně konečné) -
Prolog + **negace*** + **predikáty 2. řádu ****

* **nutná pro kontrolu cyklu**

** **prohledávání do šířky**

16 / 23

Umělá inteligence I.

