Cvičení 3

Programování
s omezujícími podmínkami

Roman Barták
Katedra teoretické informatiky a matematické logiky

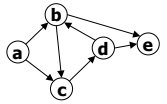
roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Domácí úkol

Vytvořte program pro nalezení jedné z nejkratších cest mezi zadanou dvojicí uzlů.

- **Databáze (graf):**

```
arc(a,b).
arc(a,c).
arc(b,c).
arc(b,e).
arc(c,d).
arc(d,b).
arc(d,e).
```



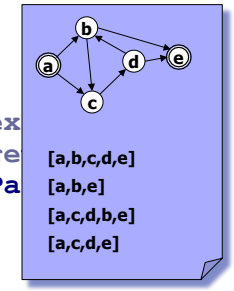
- **Očekávané odpovědi:**

```
?-shortest_path(a,a,P).
  P = [a]

?- shortest_path(a,e,P).
  P = [a,b,e]

?- shortest_path(e,b,P).
  no
```


Nejkratší cesta (naïvně)

- Najdeme **všechny cesty prohledáváním do hloubky** a potom vybereme nejkratší.

```
shortest_path(From,To, ShortestPath):-
  findall(Path,path(From,To,[],Path),AllPaths),
  shortest_list(AllPaths,ShortestPath).

path(From,From,Visited,Path):-!,
  revert([From|Visited],Path).
path(From,To,Visited,Path):-
  arc(From,Through),              % nex
  \+ member(Through,Visited),     % pre
  path(Through,To,[From|Visited],Pa
```



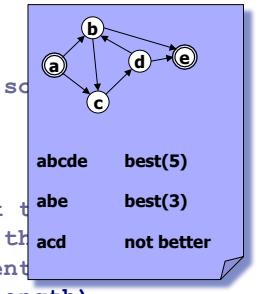
Nejkratší cesta (B&B)

- **Branch&Bound** prozkoumá všechny cesty.

```
shortest_pathBB(From,To,_Path):-
  bb_put(best,no_path),
  spathBB(From,To,[],0).
shortest_pathBB(_From,_To,Path):-
  bb_get(best,path(_,Path)).
```

```
can_be_shorter(_):-
    bb_get(best,no_path).
can_be_shorter(Length):-
    bb_get(best,path(BestLength,_)),
    Length<BestLength.
```

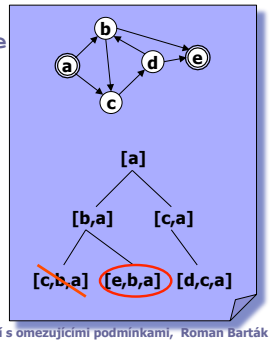
```
spathBB(From,From,Visited,Length):-!,
  revert([From|Visited],Path),
  bb_put(best,path(Length,Path)),% save so
  fail.
spathBB(From,To,Visited,OldLength):-
  NewLength is OldLength+1,
  can_be_shorter(NewLength),      % check t
  arc(From,Through),              % find th
  \+ member(Through,Visited),     % prevent
  spathBB(Through,To,[From|Visited],NewLength).
```



Nejkratší cesta (BFS)

- **Prohledávání do šířky** se spojováním

```
shortest_pathBFS(From,To,Path):-
  spathBFS([[From]],To,Path).

spathBFS([Visited|Rest],To,Path):-
  Visited = [N|_],
  (N=To ->                    % we found the path
      revert(Visited,Path)
  ;                           % expand the
      findall([N1|Visited],
          (arc(N,N1),
            \+ member(N1,Visited),
            \+ member([N1|_],Rest)),
          NewNodes),
      concat(Rest,NewNodes, Nodes),
      spathBFS(Nodes,To,Path)
  ).
```



Unifikace?

Přípomínka z Prologu:

```
?-3=1+2.
no
?-X=1+2
X=1+2;
no
?-3=X+1
no
```

Kde je problém?

- Term v Prologu nemá „význam", je to pouze syntaktická struktura!

Rádi bychom měli následující:

```
?-X=1+2.
X=3

?-3=X+1.
X=2

?-3=X+Y,Y=2.
X=1

?-3=X+Y,Y>=2,X>=1.
X=1
Y=2
```


Logické programy s podmínkami (CLP)

Pro každou proměnnou můžeme definovat její doménu.

- používají se pouze diskrétní domény
- které jsou mapovány na celá čísla

Potom definujeme mezi proměnnými podmínky.

- podmínky jsou relace mezi proměnnými
- například aritmetické výrazy

```
?-domain([X,Y],0,100),3#=X+Y,Y#>=2,X#>=1.
```

- Jinými slovy, popíšeme problém splňování podmínek.
- Chceme, aby systém sám nalezl hodnoty proměnných tak, že všechny podmínky jsou splněny.

```
X=1, Y=2
```


Příklad (naivně)

SEND+MORE=MONEY

Přiřaďte písmenům různé cifry tak, že platí SEND+MORE=MONEY a S≠0 a M≠0.

Idea:

generuj přiřazení s různými ciframi a kontroluj podmínku

```
solve_naive([S,E,N,D,M,O,R,Y]):-
   Digits1_9 = [1,2,3,4,5,6,7,8,9],
   Digits0_9 = [0|Digits1_9],
   member(S, Digits1_9),
   member(E, Digits0_9), E\=S,
   member(N, Digits0_9), N\=S, N\=E,
   member(D, Digits0_9), D\=S, D\=E, D\=N,
   member(M, Digits1_9), M\=S, M\=E, M\=N, M\=D,
   member(O, Digits0_9), O\=S, O\=E, O\=N, O\=D, O\=M,
   member(R, Digits0_9), R\=S, R\=E, R\=N, R\=D, R\=M, R\=O,
   member(Y, Digits0_9), Y\=S, Y\=E, Y\=N, Y\=D, Y\=M, Y\=O, Y\=R,
             1000*S + 100*E + 10*N + D +
             1000*M + 100*O + 10*R + E =:=
   10000*M + 1000*O + 100*N + 10*E + Y.
```

rovnost aritmetických výrazů

Příklad (lépe)
SEND+MORE=MONEY

```
solve_better([S,E,N,D,M,O,R,Y]):-
   Digits1_9 = [1,2,3,4,5,6,7,8,9],
   Digits0_9 = [0|Digits1_9],
   % D+E = 10*P1+Y
   member(D, Digits0_9),
   member(E, Digits0_9), E\=D,
   Y is (D+E) mod 10, Y\=D, Y\=E,
   P1 is (D+E) // 10, % carry bit

   % N+R+P1 = 10*P2+E
   member(N, Digits0_9), N\=D, N\=E, N\=Y,
   R is (10+E-N-P1) mod 10, R\=D, R\=E, R\=Y, R\=N,
   P2 is (N+R+P1) // 10,

   % E+O+P2 = 10*P3+N
   O is (10+N-E-P2) mod 10, O\=D, O\=E, O\=Y, O\=N, O\=R,
   P3 is (E+O+P2) // 10,

   % S+M+P3 = 10*M+O
   member(M, Digits1_9), M\=D, M\=E, M\=Y, M\=N, M\=R, M\=O,
   S is 9*M+O-P3,
   S>0,S<10, S\=D, S\=E, S\=Y, S\=N, S\=R, S\=O, S\=M.
```

Hodnoty některých písmen mohou být vypočteny z jiných písmen a platnost podmínky může být testována i když neznáme všechny písmena

0 s

Příklad (CLP)
SEND+MORE=MONEY

Filtrace nevhodných cifer může být dělána automaticky systémem řešení podmínek.

```
:-use_module(library(clpfd)).
solve(Sol):-
  Sol=[S,E,N,D,M,O,R,Y],
  domain([E,N,D,O,R,Y],0,9),
  domain([S,M],1,9),
            1000*S + 100*E + 10*N + D +
            1000*M + 100*O + 10*R + E #=
  10000*M + 1000*O + 100*N + 10*E + Y,
  all_different([S,E,N,D,M,O,R,Y]),
  labeling([],Sol).
```

přiřaď hodnoty (z domén) do proměnných použitím backtrackingu

0 s

- Poznámka: Je možné použít také model s přenosovými bity.

CLP(FD)

- Typická struktura programu CLP:

```
:-use_module(library(clpfd)).

solve(Sol):-
  declare_variables( Variables),
  post_constraints( Variables ),
  labeling( Variables ).
```

definice operátorů CLP, podmínek a řešících strategií

definice proměnných a jejich domén

definice omezení

deklarativní model problému

Řídící část
- prohledávání prostoru možných řešení
- výběr hodnot proměnných
- najde jedno, všechna nebo optimální řešení

Definice domén

- doména v SICStus Prologu je množina celých čísel
  - jiné hodnoty je potřeba mapovat na čísla
  - čísla mají přirozené uspořádání
- doména je často interval
  - `domain(SeznamProměnných,MinVal,MaxVal)`
  - definuje proměnné s počáteční doménou {MinVal, ...,MaxVal}
- doménu můžeme definovat pro proměnné samostatně (možno i sjednocení, průnik, doplněk intervalů)
  - `X in MinVal..MaxVal`
  - `X in (1..3) \/ (5..8) \/ {10}`

Reprezentace domén

- domény jsou reprezentovány jako seznam disjunktních intervalů
  - $[[Min_1|Max_1],[Min_2|Max_2],\ldots,[Min_n|Max_n]]$
  - $Min_i \leq Max_i < Min_{i+1} - 1$
- definice domény odpovídá unární podmínce
  - tj. je-li pro stejnou proměnnou definováno více domén, bere se jejich průnik (jako konjunkce unárních podmínek)

```
?-domain([X],1,20), X in 15..30.
X in 15..20
```


Specifikace podmínek
aritmetické

- klasické aritmetické podmínky s operacemi +,-, *, /, abs, min, max,… operace jsou vestavěné
- samozřejmě je potřeba použít nějaký porovnávací operátor #=, #<, #>, #=<, #>=, #\=

```
?-A+B #=< C-2.
```

- Co když definuji podmínku dříve než doménu?
  - pro nové proměnné z takové podmínky se bere nekonečná doména inf..sup