Prohledávání stavového prostoru

Aleš Horák

E-mail: hales@fi.muni.cz

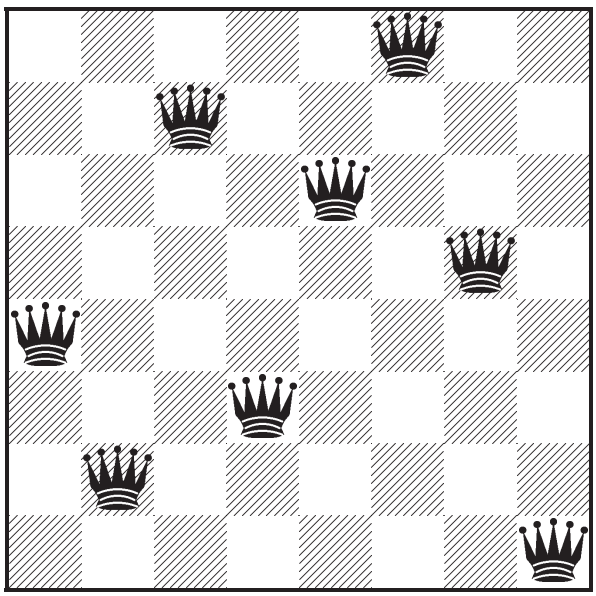
http://nlp.fi.muni.cz/uui/

Obsah:

➜ Problém osmi dam
➜ Prohledávání stavového prostoru
➜ Prohledávání do hloubky
➜ Prohledávání do šířky
➜ Prohledávání s postupným prohlubováním
➜ Shrnutí

Problém osmi dam

úkol: *Rozestavte po šachovnici 8 dam tak, aby se žádné dvě vzájemně neohrožovaly.*



Problém osmi dam I

datová struktura – osmiprvkový seznam **[X1/Y1, X2/Y2, X3/Y3, X4/Y4, X5/Y5, X6/Y6, X7/Y7, X8/Y8]**

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

```
solution(S) :− template(S), sol(S).

sol ([]).
sol([X/Y|Others]) :− sol(Others),
                     member(X,[1,2,3,4,5,6,7,8]),
                     member(Y,[1,2,3,4,5,6,7,8]),
                     noattack(X/Y,Others).

noattack(_,[]).
noattack(X/Y,[X1/Y1|Others]) :− X=\=X1, Y=\=Y1, Y1−Y=\=X1−X, Y1−Y=\=X−X1,
                                noattack(X/Y,Others).

template([X1/Y1, X2/Y2, X3/Y3, X4/Y4, X5/Y5, X6/Y6, X7/Y7, X8/Y8]).

?− solution(Solution).
    Solution     = [8/4, 7/2, 6/7, 5/3, 4/6, 3/8, 2/5, 1/1]    ;
    Solution     = [7/2, 8/4, 6/7, 5/3, 4/6, 3/8, 2/5, 1/1]    ;
    Yes
```


Problém osmi dam II

počet možností u řešení I $= 64 \cdot 63 \cdot 62 \ldots \cdot 57 \approx 1.8 \times 10^{14}$

omezení **stavového prostoru** – každá dáma má svůj sloupec

počet možností u řešení II $= 8 \cdot 7 \cdot 6 \ldots \cdot 1 = 40\,320$

```
solution(S) :− template(S), sol(S).

sol ([]).
sol([X/Y|Others]) :− sol(Others), member(Y,[1,2,3,4,5,6,7,8]),
                     noattack(X/Y,Others).
noattack(_,[]).
noattack(X/Y,[X1/Y1|Others]) :− Y=\=Y1, Y1−Y=\=X1−X, Y1−Y=\=X−X1,
                                noattack(X/Y,Others).

template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
```

PROBLÉM OSMI DAM III

souřadnice x a y $\longrightarrow$ souřadnice diagonály u a v

$u = x - v$ $\quad D_x = [1..8] \quad \longrightarrow \quad D_u = [-7..7]$

$v = x + v$ $\quad D_y = [1..8] \quad\quad D_v = [2..16]$

počet možností u řešení III $= 2\,057$

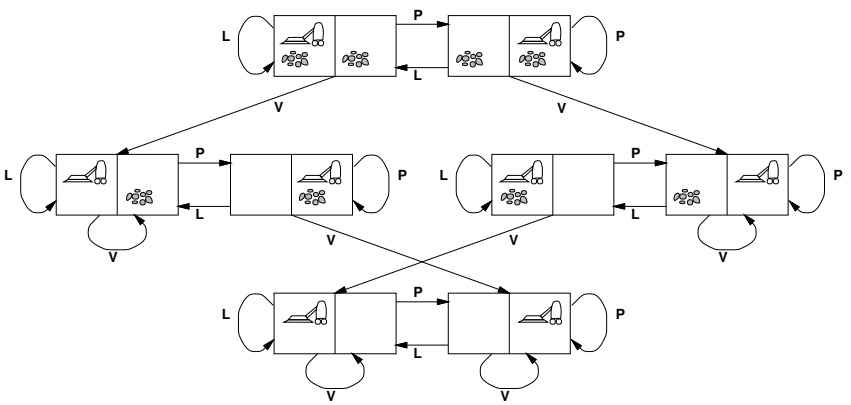
```
solution(YList) :- sol(YList   ,[1,2,3,4,5,6,7,8],[1,2,3,4,5,6,7,8],
                       [−7,−6,−5,−4,−3,−2,−1,0,1,2,3,4,5,6,7],
                       [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]).
sol  ([],[],  Dy,Du,Dv).
sol([Y|YList ],[ X|Dx1],Dy,Du,Dv) :- del(Y,Dy,Dy1), U is X−Y, del(U,Du,Du1), V is X+Y,
                                     del(V,Dv,Dv1), sol(YList,Dx1,Dy1,Du1,Dv1).

del(Item,[Item| List ], List ).
del(Item,[ First | List ],[ First | List1 ])  :- del(Item, List , List1 ).
```

Problém n dam pro $n = 100$: řešení I $\ldots 10^{400}$ řešení II $\ldots 10^{158}$ řešení III $\ldots 10^{52}$

PROHLEDÁVÁNÍ STAVOVÉHO PROSTORU

Řešení problému prohledáváním stavového prostoru:

- ➜ předpoklady – statické a deterministické prostředí, diskrétní stavy
- ➜ *stavový prostor*
- ➜ *počáteční stav* **init(State)**
- ➜ *cílová podmínka* **goal(State)**
- ➜ *přechodové akce* **move(State,NewState)**
- ➜ *prohledávací strategie*



Problém agenta Vysavače:

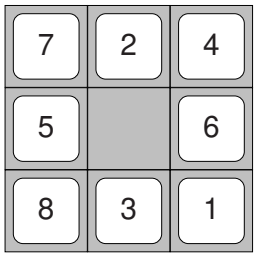
- ➜ máme dvě místnosti (L, P)
- ➜ jeden vysavač (v L nebo P)
- ➜ v každé místnosti je/není špína
- ➜ počet stavů je $2 \times 2^2 = 8$
- ➜ akce $=\{doLeva, doPrava, Vysávej\}$

Abstrakce prohledávání stavového prostoru

- ➜ *prohledávací strom*
- ➜ *kořenový uzel*
- ➜ *uzel* prohledávacího stromu:
  - *stav*
  - *rodičovský uzel*
  - *přechodová akce*
  - *hloubka uzlu*
  - *cena* – $g(n)$ cesty, $c(x, a, y)$ přechodu
- ➜ *(optimální) řešení*

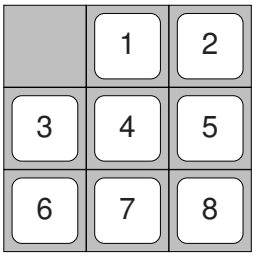
Další příklad – posunovačka

počáteční stav (např.)



$\longrightarrow \quad \ldots \quad \longrightarrow$

cílový stav



- ➜ hra na čtvercové šachovnici $m \times m$ s $n = m^2 - 1$ očíslovanými kameny
- ➜ příklad pro šachovnici 3×3, posunování osmi kamenů (8-posunovačka)
- ➜ stavy – pozice včech kamenů
- ➜ akce – “pohyb” prázdného místa

☞ Optimální řešení obecné n-posunovačky je NP-úplné

Počet stavů	u 8-posunovačky	…	$9!/2 = 181\,440$
	u 15-posunovačky	…	10^{13}
	u 24-posunovačky	…	10^{25}

Reálné problémy řešitelné prohledáváním

- ➜ hledání cesty z města A do města B
- ➜ hledání itineráře
- ➜ problém obchodního cestujícího
- ➜ návrh VLSI čipu
- ➜ navigace auta, robota, ...
- ➜ postup práce automatické výrobní linky
- ➜ návrh proteinů – 3D-sekvence aminokyselin
- ➜ Internetové vyhledávání informací

Řešení problému prohledáváním

Kostra algoritmu:

```
solution(Solution) :- init(State),solve(State,Solution).

solve(State,[State]) :- goal(State).
solve(State,[State|Sol]) :- move(State,NewState),solve(NewState,Sol).
```

move(State,NewState) – definuje prohledávací **strategii**

Porovnání strategií:

- ➜ úplnost
- ➜ optimálnost
- ➜ časová složitost
- ➜ prostorová složitost

složitost závisí na:

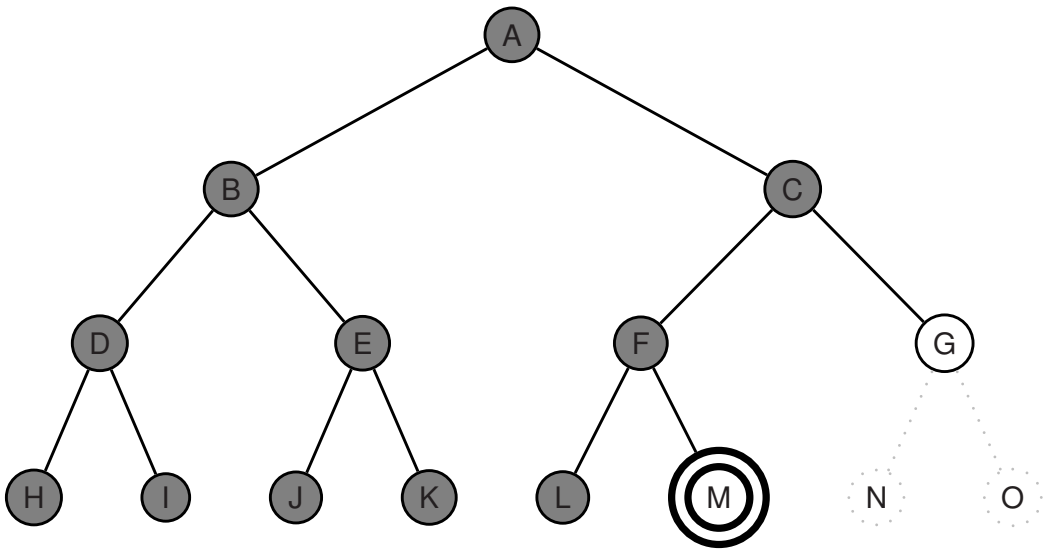
- ➜ b – faktor **větvení** (branching factor)
- ➜ d – hloubka cíle (goal depth)
- ➜ m – maximální hloubka větve/délka cesty (maximum depth/path)

Neinformované prohledávání

- prohledávání do hloubky
- prohledávání do hloubky s limitem
- prohledávání do šířky
- prohledávání podle ceny
- prohledávání s postupným prohlubováním

Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



PROHLEDÁVÁNÍ DO HLOUBKY

procedurální programovací jazyk – uzly se uloží do **zásobníku** (fronty LIFO) $\times$ Prolog – využití **rekurze**

```
solution(Node,Solution) :- depth_first_search([],Node,Solution).

depth_first_search(Path,Node,[Node|Path]) :- goal(Node).
depth_first_search(Path,Node,Sol) :- move(Node,Node1),
    not(member(Node1,Path)),depth_first_search([Node|Path],Node1,Sol).
```


PROHLEDÁVÁNÍ DO HLOUBKY – VLASTNOSTI

úplnost	**není** úplný (nekonečná větev, cykly)
optimálnost	**není** optimální
časová složitost	$O(b^m)$
prostorová složitost	$O(bm)$, lineární

Největší problém – nekonečná větev = nenajde se cíl, program neskončí!

PROHLEDÁVÁNÍ DO HLOUBKY S LIMITEM

Řešení nekonečné větve – použití "zarážky" = limit hloubky ℓ

```
solution(Node,Solution) :- depth_first_search_limit(Node,Solution,ℓ).

depth_first_search_limit(Node,[Node],_) :- goal(Node).
depth_first_search_limit(Node,[Node|Sol],MaxDepth) :- MaxDepth>0, move(Node,Node1),
    Max1 is MaxDepth−1,depth_first_search_limit(Node1,Sol,Max1).
```

neúspěch (**fail**) má dvě možné interpretace – **vyčerpání limitu** nebo **neexistenci řešení**

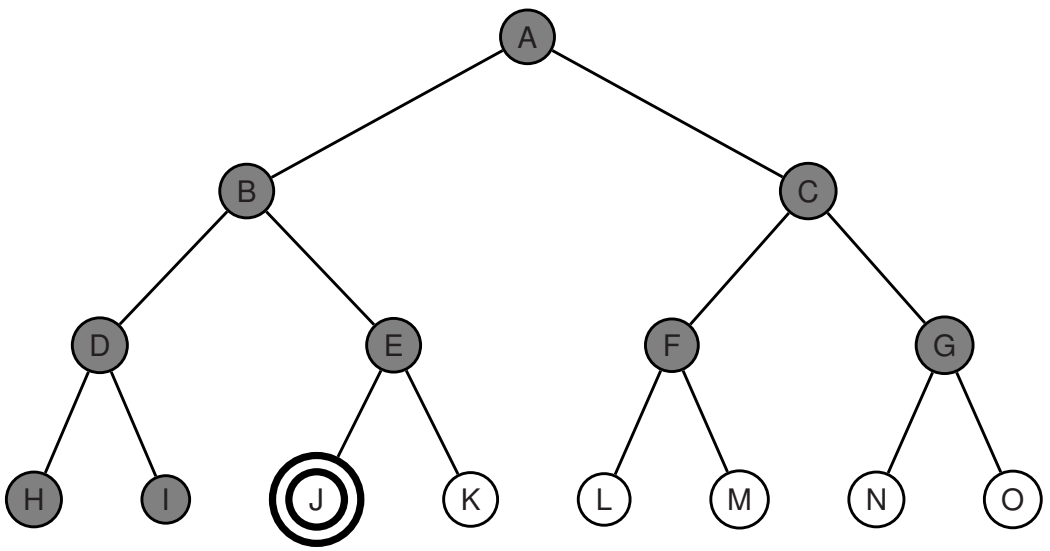
Vlastnosti:

úplnost	**není** úplný (pro $\ell < d$)
optimálnost	**není** optimální (pro $\ell > d$)
časová složitost	$O(b^\ell)$
prostorová složitost	$O(b\ell)$

dobrá volba limitu ℓ – podle znalosti problému

PROHLEDÁVÁNÍ DO ŠÍŘKY

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou. (*Breadth-first Search, BFS*)



PROHLEDÁVÁNÍ DO ŠÍŘKY

procedurální programovací jazyk – uzly se uloží do **fronty** (FIFO) $\times$ Prolog – udržuje **seznam cest**

```
solution(Start,Solution) :− breadth_first_search([[Start ]], Solution).

breadth_first_search([[Node|Path]|_],[ Node|Path]) :− goal(Node).
breadth_first_search([[N|Path]|Paths],Solution) :−
    bagof([M,N|Path],  (move(N,M),not(member(M,[N|Path]))), NewPaths),
    NewPaths\=[], append(Paths,NewPaths,Path1), !,
    breadth_first_search(Path1,Solution); breadth_first_search(Paths,Solution).
```

bagof(+Prom,+Cíl,-Sezn) *postupně vyhodnocuje* **Cíl** *a všechny vyhovující instance* **Prom** *řadí do seznamu* **Sezn**

p :- a,b;c. $\Leftrightarrow$ **p :- (a,b);c.**

Vylepšení:

➜ **append** $\longrightarrow$ **append_dl**

➜ seznam cest:

[[a]]	$\longrightarrow$	**l(a)**
[[b,a],[c,a]]		**t(a,[l(b),l(c)])**
[[c,a],[d,b,a],[e,b,a]]		**t(a,[t(b,[l(d),l(e)]),l(c)])**
[[d,b,a],[e,b,a],[f,c,a],[g,c,a]]		**t(a,[t(b,[l(d),l(e)]),t(c,[l(f),l(g)])])**

PROHLEDÁVÁNÍ DO ŠÍŘKY – VLASTNOSTI

úplnost	**je** úplný (pro konečné b)
optimálnost	**je** optimální podle délky cesty/**není** optimální podle obecné ceny
časová složitost	$1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, exponenciální v d
prostorová složitost	$O(b^{d+1})$ (každý uzel v paměti)

Největší problém – paměť:

Hloubka	Uzlů	Čas	Paměť
2	1100	0.11 sek	1 MB
4	111 100	11 sek	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hod	1 TB
10	10^{11}	129 dnů	101 TB
12	10^{13}	35 let	10 PB
14	10^{15}	3 523 let	1 EB

Ani čas není dobrý $\longrightarrow$ potřebujeme **informované** strategie prohledávání.

PROHLEDÁVÁNÍ PODLE CENY

➜ BFS je optimální pro rovnoměrně ohodnocené stromy $\times$ prohledávání podle ceny (Uniform-cost Search) je optimální pro **obecné ohodnocení**

➜ fronta uzlů se udržuje **uspořádaná** podle ceny cesty

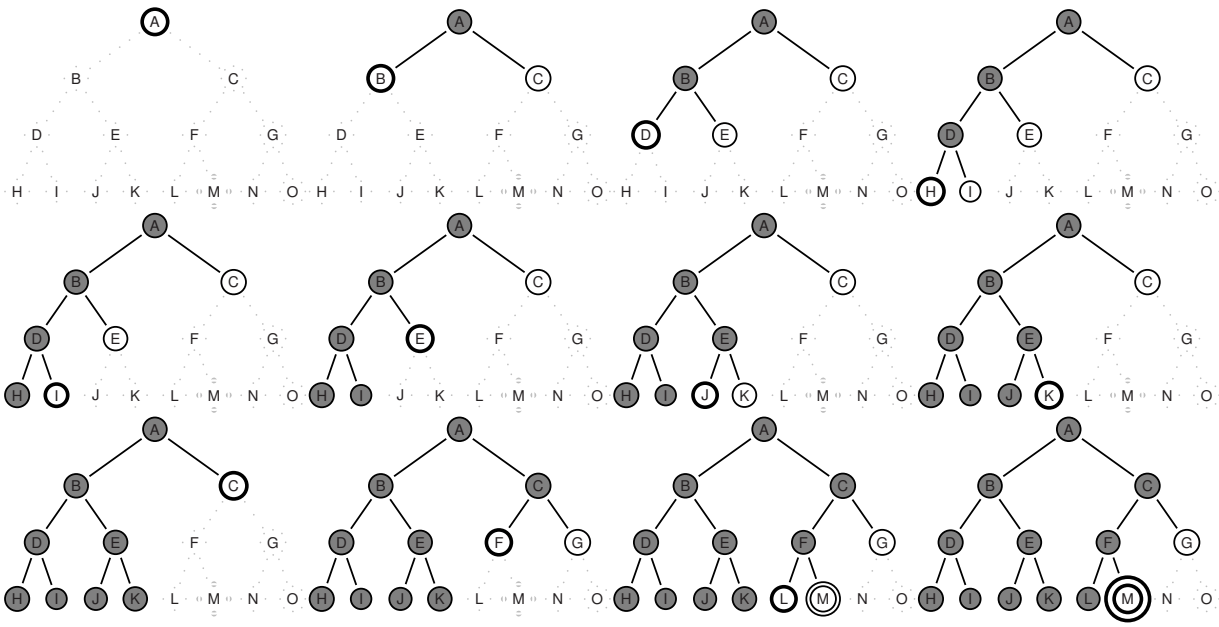
Vlastnosti:

úplnost	**je** úplný (pro cena $\geq \epsilon$)
optimálnost	**je** optimální (pro cena $\geq \epsilon$, $g(n)$ roste)
časová složitost	počet uzlů s $g \leq C^*$, $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, kde $C^*\ldots$ cena optimálního řešení
prostorová složitost	počet uzlů s $g \leq C^*$, $O(b^{1+\lfloor C^*/\epsilon \rfloor})$

PROHLEDÁVÁNÍ S POSTUPNÝM PROHLUBOVÁNÍM

prohledávání do hloubky s postupně se zvyšujícím limitem (Iterative deepening DFS, IDS)

limit=3



PROHLEDÁVÁNÍ S POSTUPNÝM PROHLUBOVÁNÍM – VLASTNOSTI

úplnost	**je** úplný (pro konečné b)
optimálnost	**je** optimální (pro $g(n)$ rovnoměrně neklesající funkce hloubky)
časová složitost	$d(b) + (d-1)b^2 + \ldots + 1(b^d) = O(b^d)$
prostorová složitost	$O(bd)$

➜ kombinuje výhody BFS a DFS:
- nízké paměťové nároky – lineární
- optimálnost, úplnost

➜ zdánlivé plýtvání opakovaným generováním
ALE generuje o jednu úroveň míň, např. pro $b = 10, d = 5$:

$$N(\mathrm{IDS}) = 50 + 400 + 3\,000 + 20\,000 + 100\,000 = 123\,450$$
$$N(\mathrm{BFS}) = 10 + 100 + 1\,000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

IDS je **nejvhodnější** neinformovaná strategie pro **velké prostory** a **neznámou hloubku** řešení.

SHRNUTÍ

Vlastnost	*do hloubky*	*do hloubky s limitem*	*do šířky*	*podle ceny*	*s postupným prohlubováním*
úplnost	ne	ano, pro $l \geq d$	ano*	ano*	ano*
optimálnost	ne	ne	ano*	ano*	ano*
časová složitost	$O(b^m)$	$O(b^\ell)$	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^d)$
prostorová složitost	$O(bm)$	$O(b\ell)$	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bd)$