

# Prohledávání stavového prostoru

Aleš Horák

E-mail: [hales@fi.muni.cz](mailto:hales@fi.muni.cz)  
<http://nlp.fi.muni.cz/uui/>

Obsah:

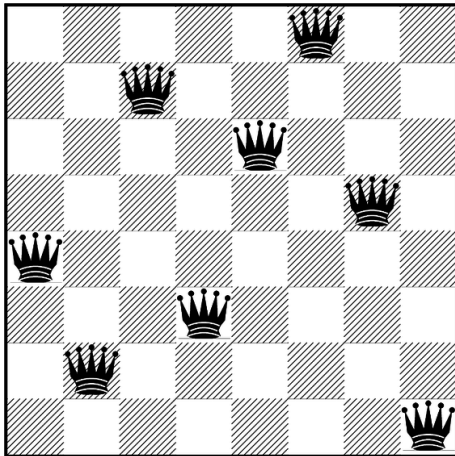
- Problém osmi dam
- Prohledávání stavového prostoru
- Neinformované prohledávání

# Problém osmi dam

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

# Problém osmi dam

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



celkem pro 8 dam existuje 92 různých řešení

# 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]

# 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]).`

# 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 =  $64 \cdot 63 \cdot 62 \dots \cdot 57 \approx 1.8 \times 10^{14}$

# Problém osmi dam II

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

omezení stavového prostoru – každá dáma má svůj sloupec

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



# Problém osmi dam II

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

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

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

```
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

k souřadnicím  $x$  a  $y$   $\longrightarrow$  přidáme i souřadnice diagonály  $u$  a  $v$

$$u = x - y$$

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

$$v = x + y$$

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

po každém umístění dámy aktualizujeme [seznamy volných pozic](#)

# Problém osmi dam III

k souřadnicím  $x$  a  $y$   $\longrightarrow$  přidáme i souřadnice diagonály  $u$  a  $v$

$$u = x - y$$

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

$$v = x + y$$

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

po každém umístění dámy aktualizujeme [seznamy volných pozic](#)

počet možností u řešení III = 2057

# Problém osmi dam III

k souřadnicím  $x$  a  $y$   $\longrightarrow$  přidáme i souřadnice diagonály  $u$  a  $v$

$$u = x - y \qquad D_x = [1..8] \qquad \longrightarrow \qquad D_u = [-7..7]$$

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

po každém umístění dámy aktualizujeme **seznamy volných pozic**

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).
```

```
% když del nenajde Item, končí neúspěchem
```

```
del(Item,[Item|List],List).
```

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

# Problém osmi dam III

k souřadnicím  $x$  a  $y$   $\longrightarrow$  přidáme i souřadnice diagonály  $u$  a  $v$

$$u = x - y \qquad D_x = [1..8] \qquad \longrightarrow \qquad D_u = [-7..7]$$

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

po každém umístění dámy aktualizujeme seznamy volných pozic

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).
```

```
% když del nenajde Item, končí neúspěchem
```

```
del(Item,[Item|List],List).
```

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

Problém  $n$  dam pro  $n = 100$ :

# Problém osmi dam III

k souřadnicím  $x$  a  $y$   $\longrightarrow$  přidáme i souřadnice diagonály  $u$  a  $v$

$$u = x - y \qquad D_x = [1..8] \qquad \longrightarrow \qquad D_u = [-7..7]$$

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

po každém umístění dámy aktualizujeme seznamy volných pozic

počet možností u řešení III = 2057

```
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).
```

```
% když del nenajde Item, končí neúspěchem
```

```
del(Item,[Item|List],List).
```

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

Problém  $n$  dam pro  $n = 100$ :

řešení I ...  $10^{400}$

# Problém osmi dam III

k souřadnicím  $x$  a  $y$   $\longrightarrow$  přidáme i souřadnice diagonály  $u$  a  $v$

$$u = x - y \qquad D_x = [1..8] \qquad \longrightarrow \qquad D_u = [-7..7]$$

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

po každém umístění dámy aktualizujeme seznamy volných pozic

počet možností u řešení III = 2057

```
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).
```

```
% když del nenajde Item, končí neúspěchem
```

```
del(Item,[Item|List],List).
```

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

Problém  $n$  dam pro  $n = 100$ :

řešení I ...  $10^{400}$       řešení II ...  $10^{158}$

# Problém osmi dam III

k souřadnicím  $x$  a  $y$   $\longrightarrow$  přidáme i souřadnice diagonály  $u$  a  $v$

$$u = x - y \qquad D_x = [1..8] \qquad \longrightarrow \qquad D_u = [-7..7]$$

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

po každém umístění dámy aktualizujeme seznamy volných pozic

počet možností u řešení III = 2057

```
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).
```

```
% když del nenajde Item, končí neúspěchem
```

```
del(Item,[Item|List],List).
```

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

Problém  $n$  dam pro  $n = 100$ :

řešení I ...  $10^{400}$

řešení II ...  $10^{158}$

řešení III ...  $10^{52}$

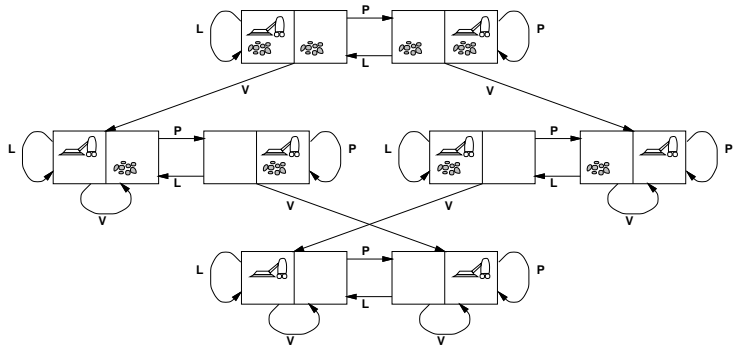


# Prohledávání stavového prostoru

## Řešení problému prohledáváním stavového prostoru:

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

# Problém agenta Vysavače

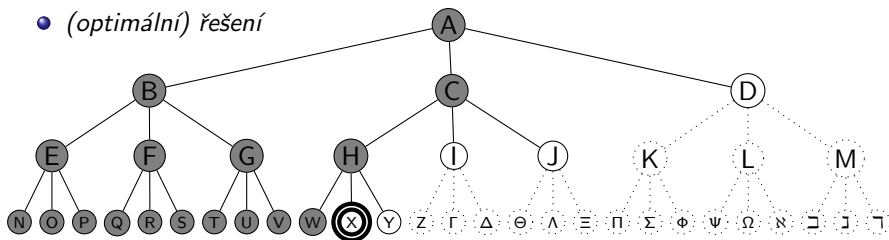


- máme dvě **místnosti** (L, P)
- jeden **vysavač** (v L nebo P)
- v každé místnosti je/není špína
- počet **stavů** je  $2 \times 2^2 = 8$
- **akce** = {doLeva, doPrava, Vysávej}

# Problém agenta Vysavače

## Prohledávací strategie – 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

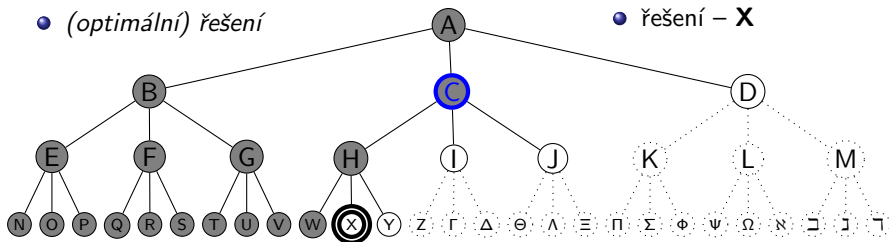


# Problém agenta Vysavače

## Prohledávací strategie – 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

- **A**
- např. uzel **C**:
  - stav – 
  - rodič – **A**
  - akce – **doPrava**
  - hloubka – **1**
  - cena – **1**
- řešení – **X**



# Další příklad – posunovačka

počáteční stav (např.)

|   |   |   |
|---|---|---|
| 7 | 2 | 4 |
| 5 |   | 6 |
| 8 | 3 | 1 |

→ ... →

cílový stav

|   |   |   |
|---|---|---|
|   | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

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

## Další příklad – posunovačka

počáteční stav (např.)

|   |   |   |
|---|---|---|
| 7 | 2 | 4 |
| 5 |   | 6 |
| 8 | 3 | 1 |

→ ... →

cílový stav

|   |   |   |
|---|---|---|
|   | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

- hra na čtvercové šachovnici  $m \times m$  s  $n = m^2 - 1$  očíslovanými kameny
- příklad pro šachovnici  $3 \times 3$ , posunování osmi kamenů (8-posunovačka)
- **stavy** – pozice všech 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**



# 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

# Řešení problému prohledáváním

Kostrá 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:

složitost závisí na:

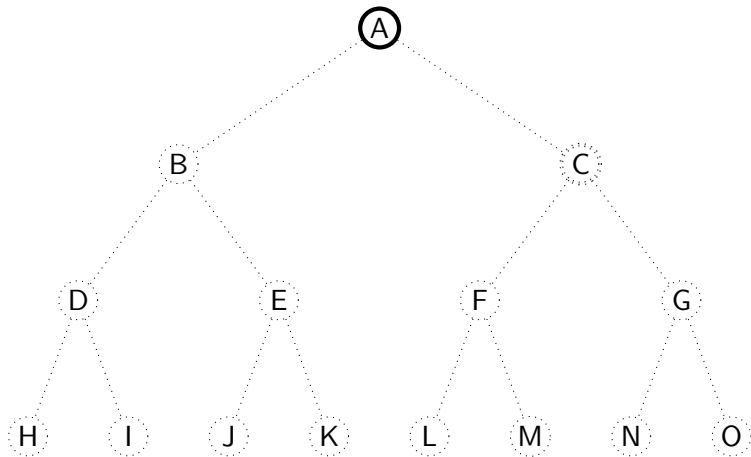
- úplnost
  - optimálnost
  - časová složitost
  - prostorová složitost
- $b$  – faktor **větvení** (branching factor)
  - $d$  – hloubka cíle (goal depth)
  - $m$  – maximální hloubka větve/délka cesty (maximum depth/path, může být  $\infty$ ?)

# Neinformované prohledávání

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

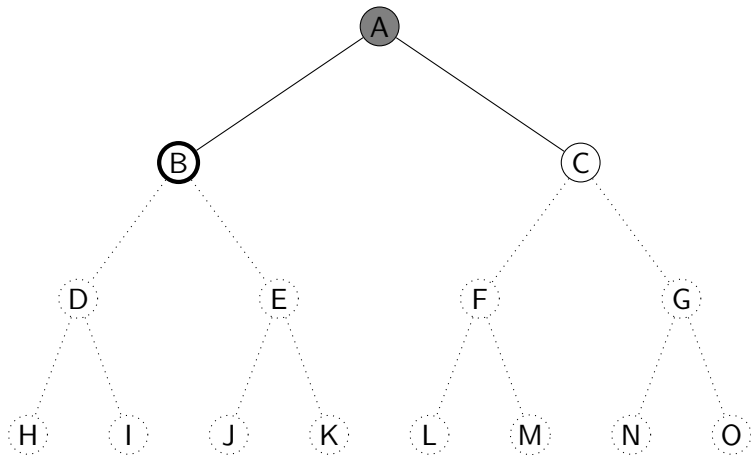
# Prohledávání do hloubky

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



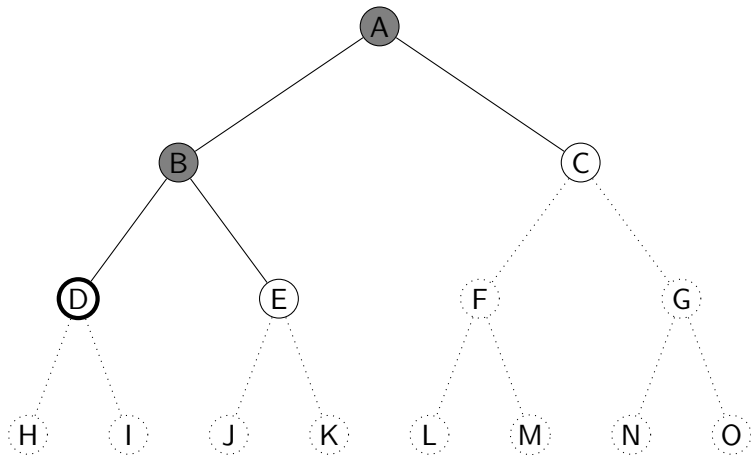
# Prohledávání do hloubky

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



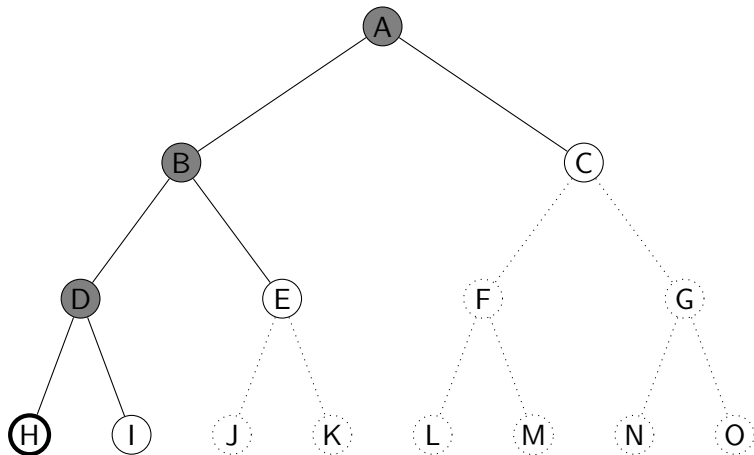
# Prohledávání do hloubky

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



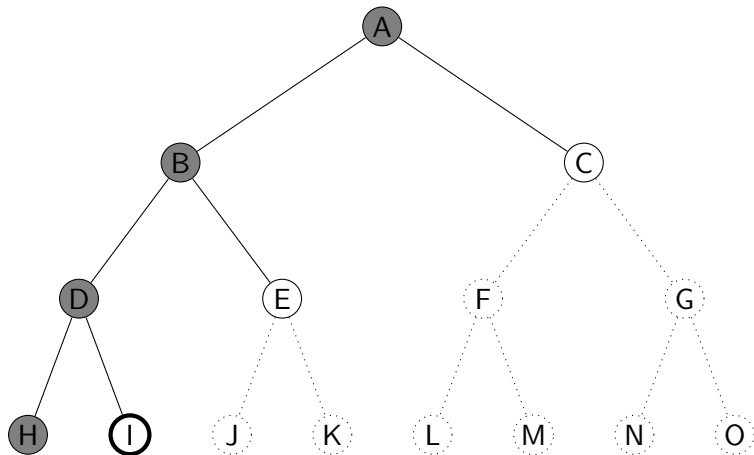
# Prohledávání do hloubky

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



# Prohledávání do hloubky

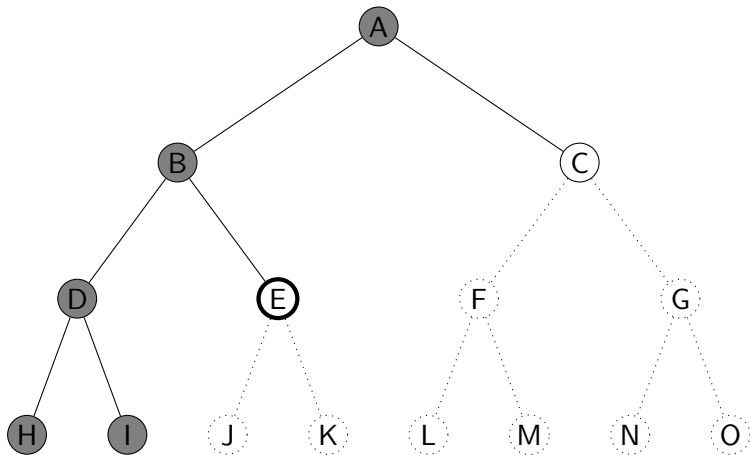
Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)





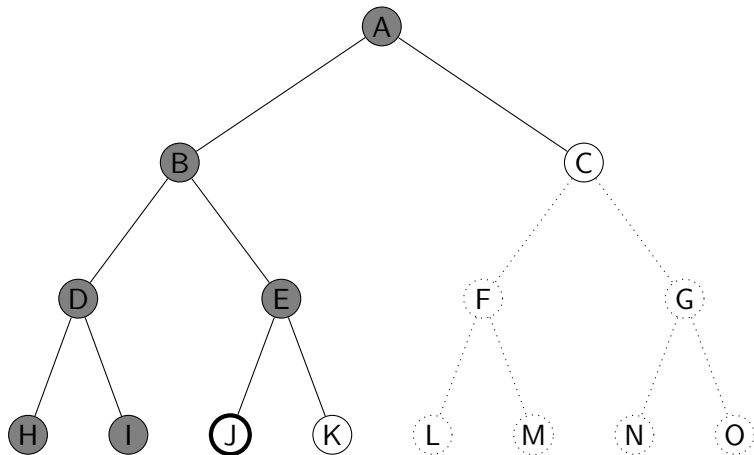
# Prohledávání do hloubky

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



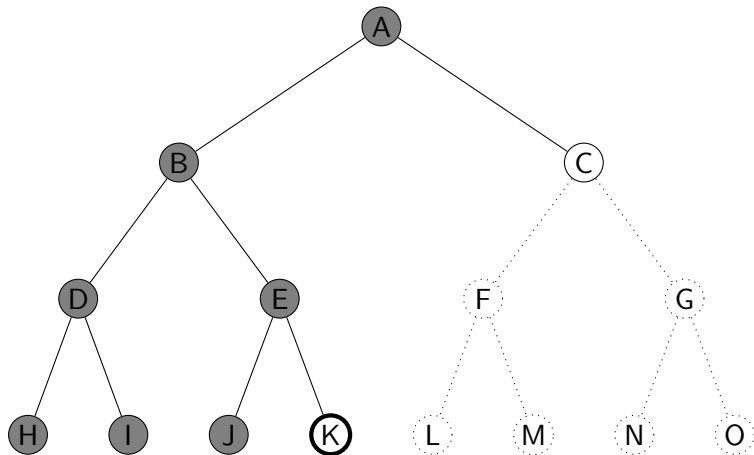
# Prohledávání do hloubky

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



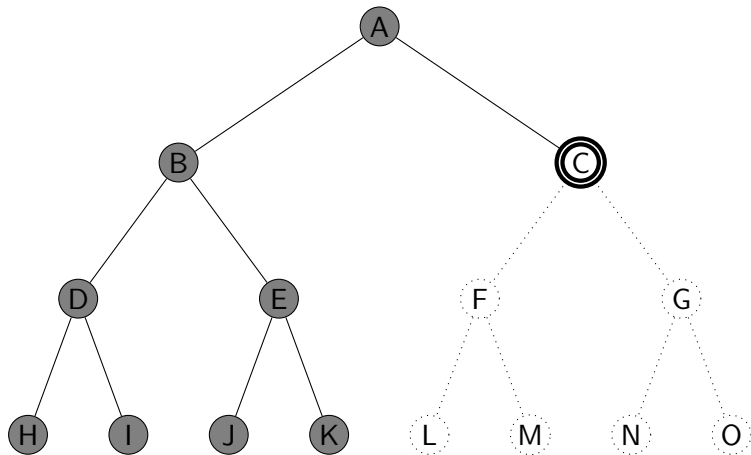
# Prohledávání do hloubky

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



# 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) × Prolog – využití rekurze

# Prohledávání do hloubky

procedurální programovací jazyk – uzly se uloží do **zásobníku** (fronty LIFO) × 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),  
  \+ member(Node1,Path),depth_first_search([Node|Path],Node1,Sol).
```

# Prohledávání do hloubky – vlastnosti

*úplnost*

*optimálnost*

*časová složitost*

*prostorová složitost*

# Prohledávání do hloubky – vlastnosti

*úplnost*

**není** úplný (nekonečná větev, cykly)

*optimálnost*

*časová složitost*

*prostorová složitost*



# Prohledávání do hloubky – vlastnosti

|                             |                                     |
|-----------------------------|-------------------------------------|
| <i>úplnost</i>              | není úplný (nekonečná větev, cykly) |
| <i>optimálnost</i>          | není optimální                      |
| <i>časová složitost</i>     |                                     |
| <i>prostorová složitost</i> |                                     |

# Prohledávání do hloubky – vlastnosti

|                             |                                     |
|-----------------------------|-------------------------------------|
| <i>úplnost</i>              | není úplný (nekonečná větev, cykly) |
| <i>optimálnost</i>          | není optimální                      |
| <i>časová složitost</i>     | $O(b^m)$                            |
| <i>prostorová složitost</i> |                                     |

# Prohledávání do hloubky – vlastnosti

|                             |                                     |
|-----------------------------|-------------------------------------|
| <i>úplnost</i>              | není úplný (nekonečná větev, cykly) |
| <i>optimálnost</i>          | není optimální                      |
| <i>časová složitost</i>     | $O(b^m)$                            |
| <i>prostorová složitost</i> | $O(bm)$ , lineární                  |

# Prohledávání do hloubky – vlastnosti

|                             |                                     |
|-----------------------------|-------------------------------------|
| <i>úplnost</i>              | není úplný (nekonečná větev, cykly) |
| <i>optimálnost</i>          | není optimální                      |
| <i>časová složitost</i>     | $O(b^m)$                            |
| <i>prostorová složitost</i> | $O(bm)$ , lineární                  |

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

# Prohledávání do hloubky s limitem

Řešení nekonečné větve – použití “zarážky” = limit hloubky  $\ell$

```
solution(Node,Solution) :- depth_first_search_limit(Node,Solution, $\ell$ ).
```

```
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).
```

# Prohledávání do hloubky s limitem

Řešení nekonečné větve – použití “zarážky” = limit hloubky  $\ell$

```
solution(Node,Solution) :- depth_first_search_limit(Node,Solution, $\ell$ ).
```

```
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

# Prohledávání do hloubky s limitem

Řešení nekonečné větve – použití “zarážky” = limit hloubky  $\ell$

```
solution(Node,Solution) :- depth_first_search_limit(Node,Solution, $\ell$ ).
```

```
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

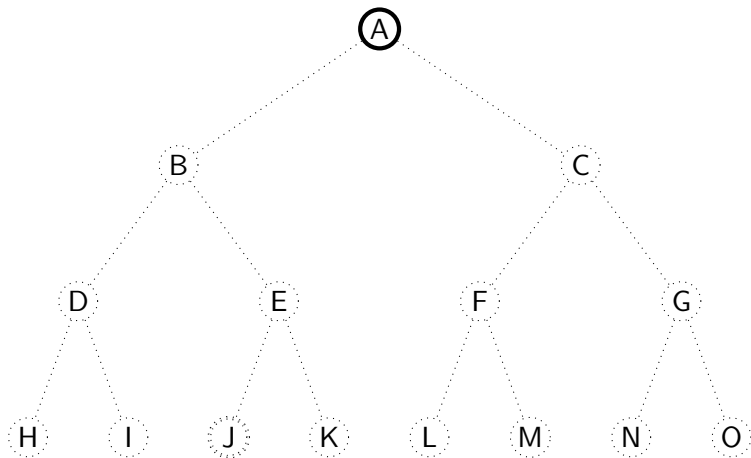
## Vlastnosti:

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

dobrá volba limitu  $\ell$  – podle znalosti problému

# Prohledávání do šířky

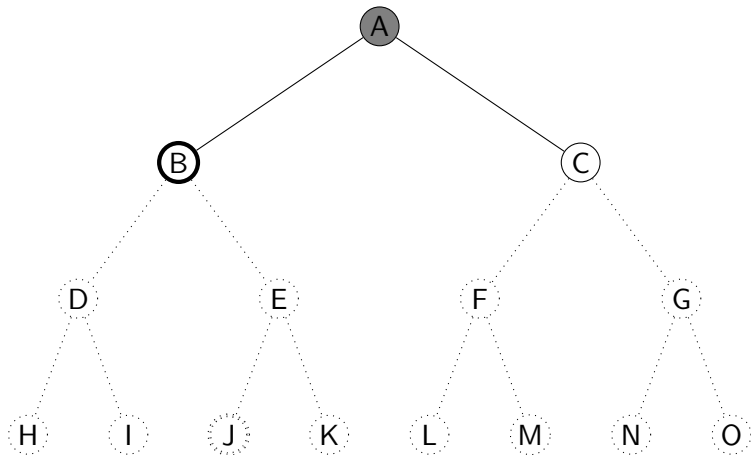
Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.  
(*Breadth-first Search, BFS*)





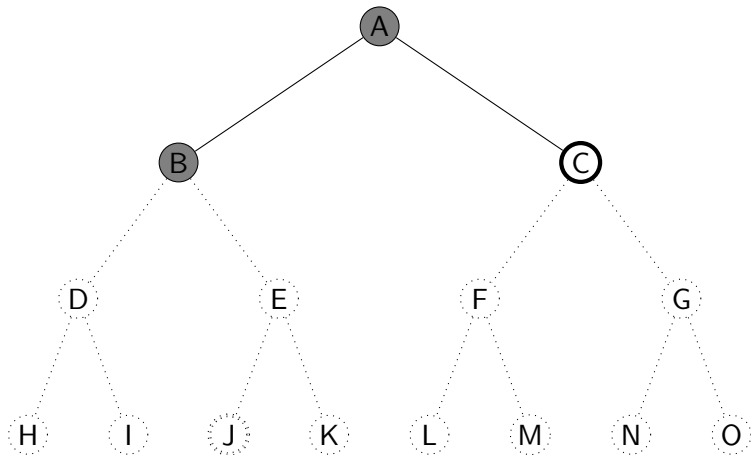
# Prohledávání do šířky

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



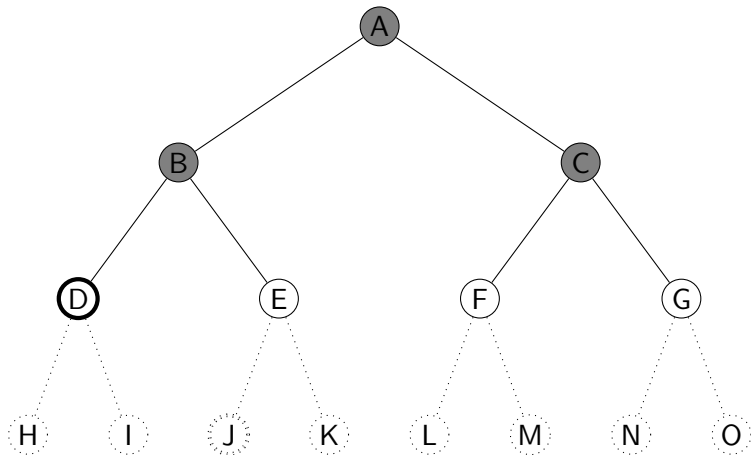
# Prohledávání do šířky

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



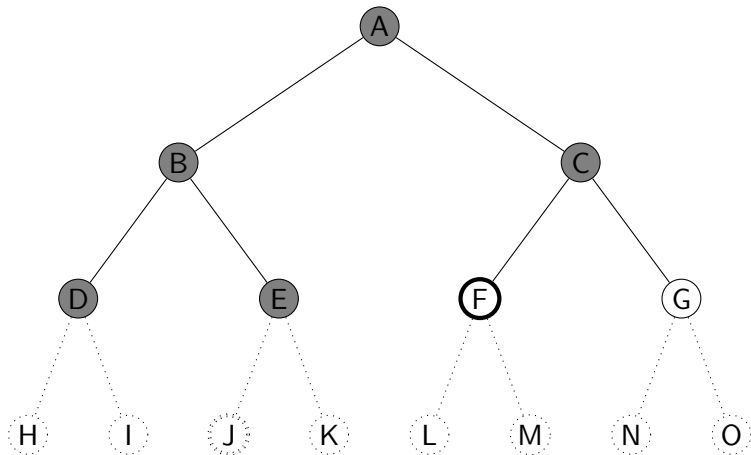
# Prohledávání do šířky

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



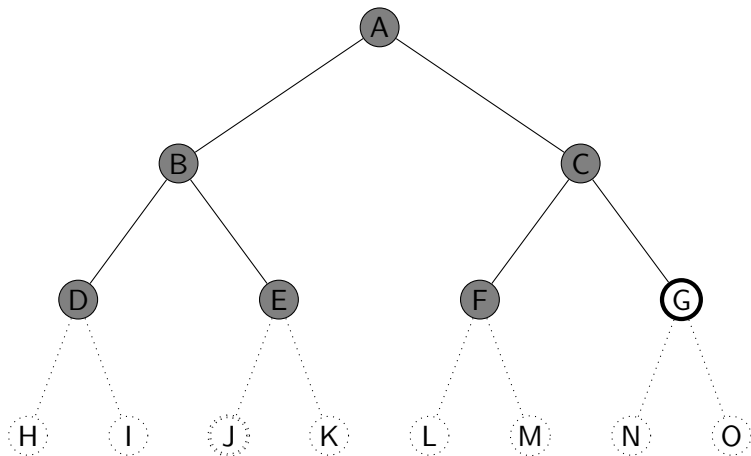
# Prohledávání do šířky

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



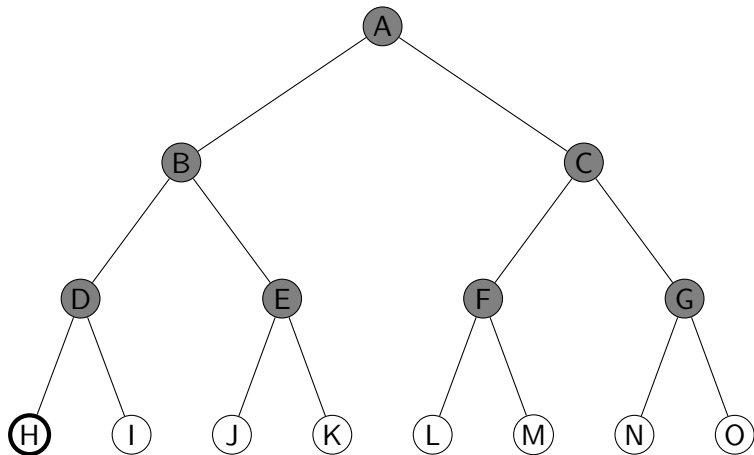
# Prohledávání do šířky

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



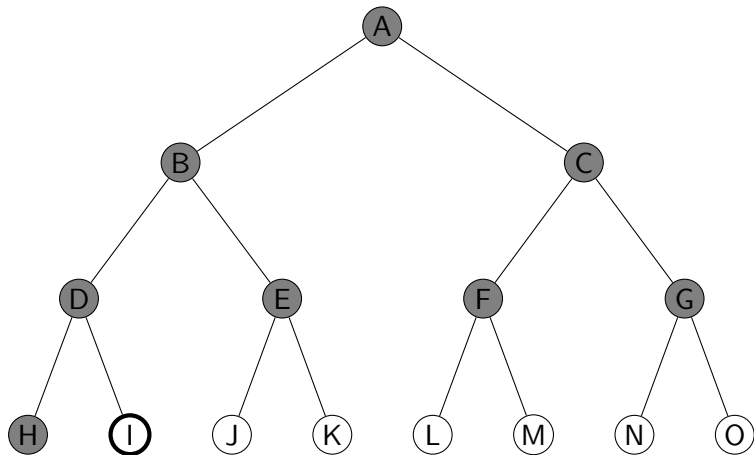
# Prohledávání do šířky

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



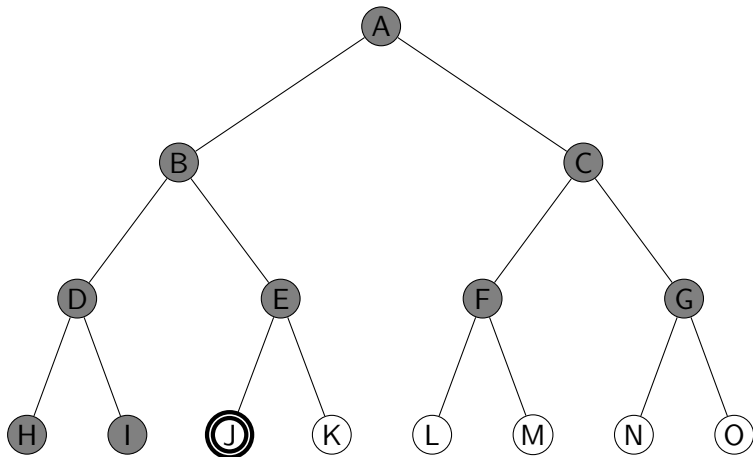
# Prohledávání do šířky

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



# 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) × 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),\+ member(M,[N|Path])), NewPaths),  
    NewPaths\=[], append(Paths,NewPaths,Path1), !,  
    breadth_first_search(Path1,Solution); breadth_first_search(Paths,Solution).
```

# Prohledávání do šířky

procedurální programovací jazyk – uzly se uloží do **fronty** (FIFO) × 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), \+ 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**

# Prohledávání do šířky

procedurální programovací jazyk – uzly se uloží do **fronty** (FIFO) × 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), \+ member(M,[N|Path])), NewPaths),
```

```
  NewPaths\=[], append(Paths,NewPaths,Path1), !, p :- a,b;c. ⇔ p :- (a,b);c.
```

```
  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**

# Prohledávání do šířky

procedurální programovací jazyk – uzly se uloží do **fronty** (FIFO) × 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), \+ member(M,[N|Path])), NewPaths),
```

```
  NewPaths\=[], append(Paths,NewPaths,Path1), !, p :- a,b;c. ⇔ p :- (a,b);c.
```

```
  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**

## Vylepšení:

- **append** → **append\_dl**
- seznam cest:

|  |   |  |
|--|---|--|
| <pre>[[a]] [[b,a],[c,a]] [[c,a],[d,b,a],[e,b,a]] [[d,b,a],[e,b,a],[f,c,a],[g,c,a]]</pre> | → | <pre>l(a) t(a,[l(b),l(c)]) t(a,[t(b,[l(d),l(e)]),l(c)]) t(a,[t(b,[l(d),l(e)]),t(c,[l(f),l(g)])])</pre> |
|--|---|--|

# Prohledávání do šířky – vlastnosti

*úplnost*

*optimálnost*

*časová složitost*

*prostorová složitost*

# Prohledávání do šířky – vlastnosti

*úplnost*

je úplný (pro konečné  $b$ )

*optimálnost*

*časová složitost*

*prostorová složitost*

# Prohledávání do šířky – vlastnosti

*úplnost*

je úplný (pro konečné  $b$ )

*optimálnost*

je optimální podle délky cesty/**není** optimální podle obecné ceny

*časová složitost*

*prostorová složitost*

# Prohledávání do šířky – vlastnosti

*úplnost*

je úplný (pro konečné  $b$ )

*optimálnost*

je optimální podle délky cesty/**není** optimální podle obecné ceny

*časová složitost*

$1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$ ,  
exponenciální v  $d$

*prostorová složitost*



# Prohledávání do šířky – vlastnosti

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

# Prohledávání do šířky – vlastnosti

|                             |  |
|-----------------------------|--|
| <i>úplnost</i>              | je úplný (pro konečné $b$ )  |
| <i>optimálnost</i>          | je optimální podle délky cesty/ <b>není</b> optimální podle obecné ceny              |
| <i>časová složitost</i>     | $1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$ ,<br>exponenciální v $d$ |
| <i>prostorová složitost</i> | $O(b^{d+1})$ (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 → potřebujeme **informované** strategie prohledávání.

# Prohledávání podle ceny

- BFS je optimální pro rovnoměrně ohodnocené stromy × **prohledávání podle ceny** (**Uniform-cost Search**) je optimální pro **obecné ohodnocení**
- fronta uzlů se udržuje **uspořádaná** podle ceny cesty

# Prohledávání podle ceny

- BFS je optimální pro rovnoměrně ohodnocené stromy × **prohledávání podle ceny** (**Uniform-cost Search**) je optimální pro **obecné ohodnocení**
- fronta uzlů se udržuje **uspořádaná** podle ceny cesty

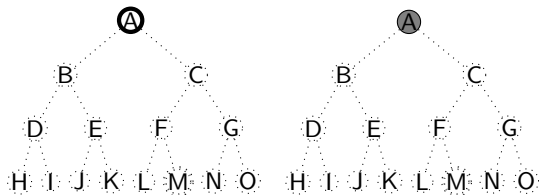
## Vlastnosti:

|                             |  |
|-----------------------------|--|
| <i>úplnost</i>              | je úplný (pro $\text{cena} \geq \epsilon$ )  |
| <i>optimálnost</i>          | je optimální (pro $\text{cena} \geq \epsilon$ , $g(n)$ roste)  |
| <i>časová složitost</i>     | počet uzlů s $g \leq C^*$ , $O(b^{1+\lceil C^*/\epsilon \rceil})$ , kde<br>$C^*$ ... cena optimálního řešení |
| <i>prostorová složitost</i> | počet uzlů s $g \leq C^*$ , $O(b^{1+\lceil C^*/\epsilon \rceil})$  |

# Prohledávání s postupným prohlubováním

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

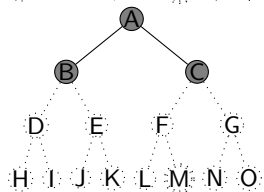
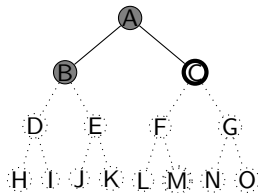
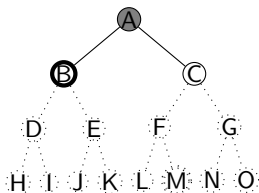
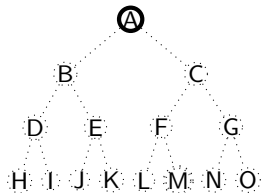
limit=0



# Prohledávání s postupným prohlubováním

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

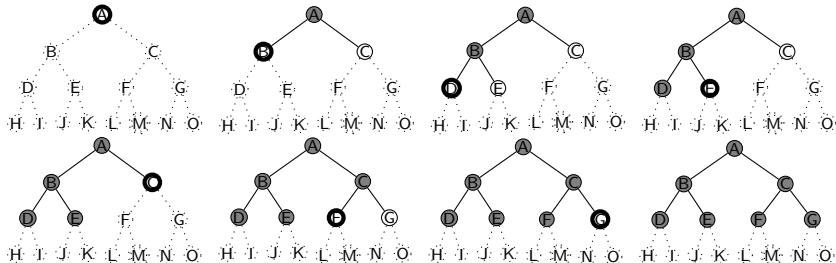
limit=1



# Prohledávání s postupným prohlubováním

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

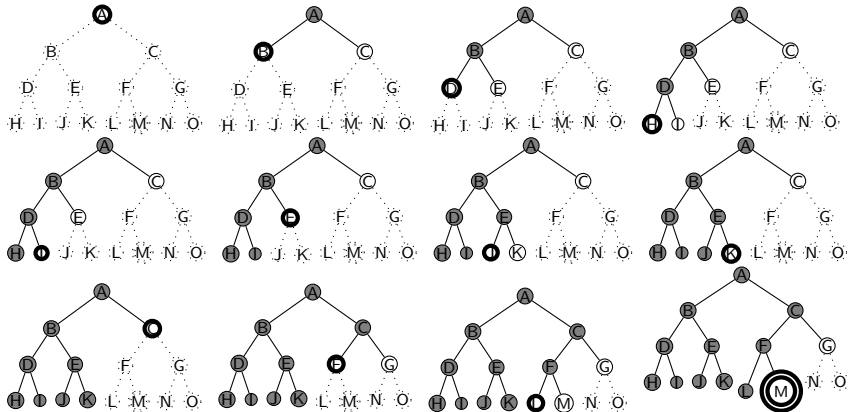
limit=2



# Prohledávání s postupným prohlubováním

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

limit=3





# Prohledávání s postupným prohlubováním – vlastnosti

|                             |   |
|-----------------------------|---|
| <i>úplnost</i>              | je úplný (pro konečné $b$ )                                     |
| <i>optimálnost</i>          | je optimální (pro $g(n)$ rovnoměrně neklesající funkce hloubky) |
| <i>časová složitost</i>     | $d(b) + (d - 1)b^2 + \dots + 1(b^d) = O(b^d)$                   |
| <i>prostorová složitost</i> | $O(bd)$   |

# Prohledávání s postupným prohlubováním – vlastnosti

*úplnost* je úplný (pro konečné  $b$ )

*optimálnost* je optimální (pro  $g(n)$  rovnoměrně neklesající funkce hloubky)

*časová složitost*  $d(b) + (d - 1)b^2 + \dots + 1(b^d) = O(b^d)$

*prostorová složitost*  $O(bd)$

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

# Prohledávání s postupným prohlubováním – vlastnosti

|                             |   |
|-----------------------------|---|
| <i>úplnost</i>              | je úplný (pro konečné $b$ )                                     |
| <i>optimálnost</i>          | je optimální (pro $g(n)$ rovnoměrně neklesající funkce hloubky) |
| <i>časová složitost</i>     | $d(b) + (d - 1)b^2 + \dots + 1(b^d) = O(b^d)$                   |
| <i>prostorová složitost</i> | $O(bd)$   |

- kombinuje výhody BFS a DFS:
  - nízké paměťové nároky – lineární
  - optimálnost, úplnost
- zdánlivé plýtvání opakovaným generováním

ALE generuje o jednu úroveň míň, např. pro  $b = 10, d = 5$ :

$$N(\text{IDS}) = 50 + 400 + 3\,000 + 20\,000 + 100\,000 = 123\,450$$

$$N(\text{BFS}) = 10 + 100 + 1\,000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

# Prohledávání s postupným prohlubováním – vlastnosti

|                             |   |
|-----------------------------|---|
| <i>úplnost</i>              | je úplný (pro konečné $b$ )                                     |
| <i>optimálnost</i>          | je optimální (pro $g(n)$ rovnoměrně neklesající funkce hloubky) |
| <i>časová složitost</i>     | $d(b) + (d - 1)b^2 + \dots + 1(b^d) = O(b^d)$                   |
| <i>prostorová složitost</i> | $O(bd)$   |

- kombinuje výhody BFS a DFS:
  - nízké paměťové nároky – lineární
  - optimálnost, úplnost
- zdánlivé plýtvání opakovaným generováním

ALE generuje o jednu úroveň míň, např. pro  $b = 10, d = 5$ :

$$N(\text{IDS}) = 50 + 400 + 3\,000 + 20\,000 + 100\,000 = 123\,450$$

$$N(\text{BFS}) = 10 + 100 + 1\,000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

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

## Shrnutí vlastností algoritmů neinformovaného prohledávání

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