Neinformované metody prohledávání stavového prostoru

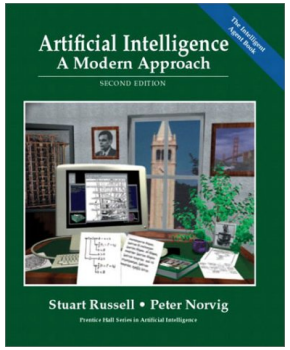
Michal Pěchouček

Department of Cybernetics
Czech Technical University in Prague

http://labe.felk.cvut.cz/~pechouc/kui/noinfosearch.pdf

Použitá literatura pro umělou inteligenci

:: Artificial Intelligence: A Modern Approach (Second Edition) by Stuart Russell and Peter Norvig, 2002 Prentice Hall.



http://aima.cs.berkeley.edu/

Úvodní poznámka o umělé inteligenci

Umělá inteligence jako:

- Přístup ke zkoumání a chápání inteligence
- Přístup ke psaní algoritmů s prvky umělé inteligence (která nemusí být nezbytně lidská):
  - Pro podporu rozhodování, optimalizaci rozhodovacích procesů, autonomii v rozhodování

Úvodní poznámka o umělé inteligenci

Umělá inteligence, jako věda o poznání lidského uvažování a povahy lidských znalostí pomocí modelování těchto na počítači, se dělí do základních větví:

- symbolický funkcionalismus – škola umělé inteligence, která je založeno na modelování inteligence pomocí modelování manipulace se symboly příklad: znalostní systémy, automatické dokazování, plánování
- konekcionismus – škola umělé inteligence, která je založeno na modelování inteligence pomocí velkého počtu stejných, fixně svázaných a interagujících výpočetních jednotek příklad: neuronové sítě
- robotický funkcionalismus (behavioralismus)– který je založen na předpokladu že kombinací velkého počtu specializovaných, ale neinteligentních procesů (černých krabiček) lze dosáhnout inteligentního chování příklad: inteligentní robotika
- hybridní a další přístupy: multi-agentní přístupy, genetické algoritmy, artificial life, ...

Úvodní poznámka o umělé inteligenci

- **Silná umělá inteligence** (podle Bretana): inteligence, která navozuje mentální stavy identické s mentálními stavu provázející lidské porozumění (extrémní definice)

Úvodní poznámka o umělé inteligenci

- **Silná umělá inteligence** (podle Bretana): inteligence, která navozuje mentální stavy identické s mentálními stavu provázející lidské porozumění (extrémní definice)
- **Slabá umělá inteligence** (podle Turinga): inteligence, která svými projevy není rozeznatelná od inteligence lidské

Úvodní poznámka o umělé inteligenci

- **Silná umělá inteligence** (podle Bretana): inteligence, která navozuje mentální stavy identické s mentálními stavu provázející lidské porozumění (extrémní definice)
- **Slabá umělá inteligence** (podle Turinga): inteligence, která svými projevy není rozeznatelná od inteligence lidské
- **Střední (middling) umělá inteligence** (podle Smithe): inteligence, která je založena na modelech znalostí a mechanismech uvažování identickými s modely a mechanismy používaných při projevech lidské inteligenci

Úvodní poznámka o umělé inteligenci

- **Silná umělá inteligence** (podle Bretana): inteligence, která navozuje mentální stavy identické s mentálními stavu provázející lidské porozumění (extrémní definice)
- **Slabá umělá inteligence** (podle Turinga): inteligence, která svými projevy není rozeznatelná od inteligence lidské
- **Střední (middling) umělá inteligence** (podle Smithe): inteligence, která je založena na modelech znalostí a mechanismech uvažování identickými s modely a mechanismy používaných při projevech lidské inteligenci

- **Turingův test** - test na přítomnost slabé umělé inteligence.

- **Turingův stroj** - příklad abstraktního stroje, pomocí něhož lze modelovat libovolný algoritmus, počítačový program.

Úvodní poznámka o umělé inteligenci

Symbolický funkcionlismus je založen na modelování dvou základních aspektů inteligentního uvažování:

- znalostí
- uvažování

Obojí lze modelovat na různých úrovních obecnosti. **Silné metody** umožňují obecné modely uvažování, zatímco **slabé metody** jsou specifické

:: Zjednodušená úloha umělé inteligence podle symbolického funkcionalismu tedy zní: jak reprezentovat ty správné znalosti a naprogramovat takové uvažovací mechanismy, které obohatí soubor znalostí o nové hypotézy.

Úvodní poznámka o umělé inteligenci

Mějme dva extrémní případy implementace umělé inteligence podle symbolického funkcionalismu:

- **silný** – znalosti jsou reprezentovány pomocí systému predikátové logiky a uvažování je reprezentováno výpočetním modelem dedukce
- **slabý** – znalosti jsou reprezentovány jako množina výroků a uvažování je reprezentováno pomocí souboru `if-then` pravidel

V obojím případě je množina nových znalostí (ať už vytvořených nebo hypotetických) veliká a je potřeba ji inteligentně vytvářet a prohledávat. Strategie prohledávání je součástí modelu uvažování. Prostor nových znalostí se nazývá stavový prostor. Při řešení problémů se skládá stavový prostor z mezířešení či pomocných hypotéz. Některá mezířešení jsou klasfikovaná jako cílová řešení.

Stavový prostor

Řešení problémů, jako jeden z projevů inteligentní uvažování, je chápáno jako problém nalezní (z počátečního stavu s_0) takového stavu s_n, který splňuje vlastnosti požadovaného řešení – $\texttt{goal}(s_n)$. Takovéto stavy nazýváme cílové – s_{goal}. V některých případech se definuje řešení problému jako problém nalezeni cesty z počátečního uzlu do uzlu cílového. Zde neprohledáváme prostor uzlů ale prostor cest.

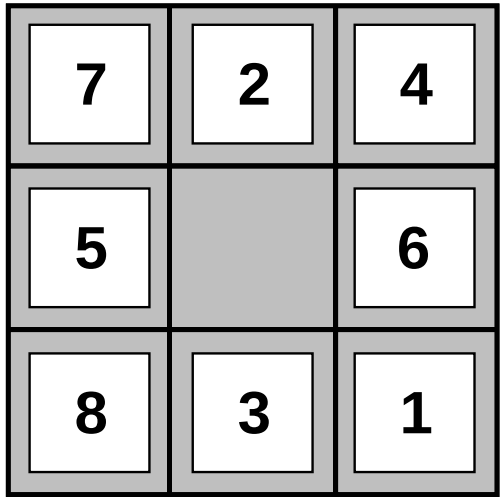
Problém prohledávání stavového prostoru je tedy definován pomocí

- počátečního stavu - s_0
- vlastností cílových stavů – $\texttt{goal}(s_n)$
- množiny stavových operátorů,
- objektivní funkce, ohodnocení ceny použití stavových operátoru

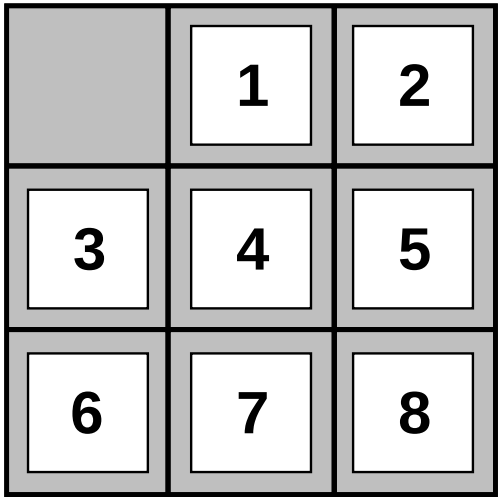
Příklady: problém 8-královen, kryptoaritmetika, šachy, lišák (hra 8-puzzle), dokazování v matematice, porozumění přirozenému jazyku, plánování a rozvrhování, robotická navigace

Situace se vážně komplikuje v případě, že se jedná o dynamicky se měnící stavový prostor – například, pří řešení problému v dynamickém prostředí nebo hře s kompetitivním oponentem.

Příklad: Hra Lišák

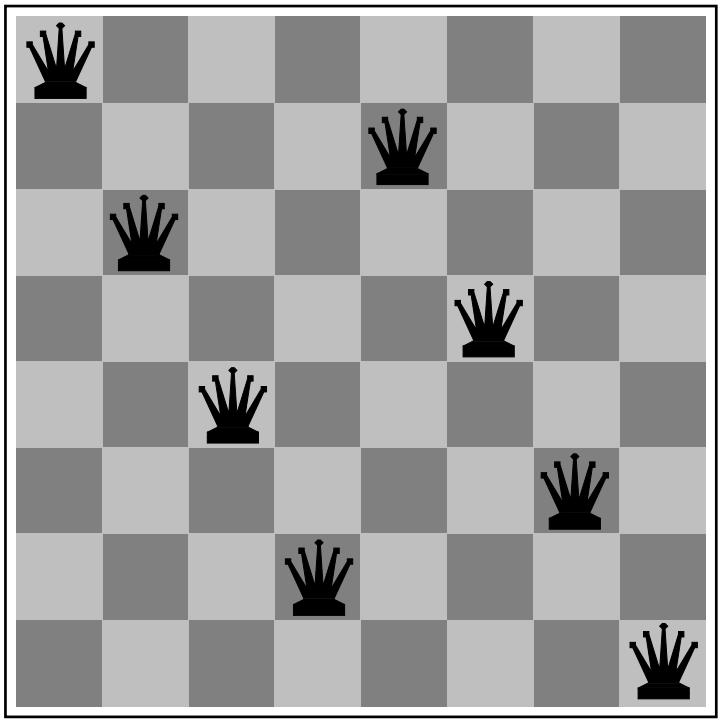


Start State



Goal State

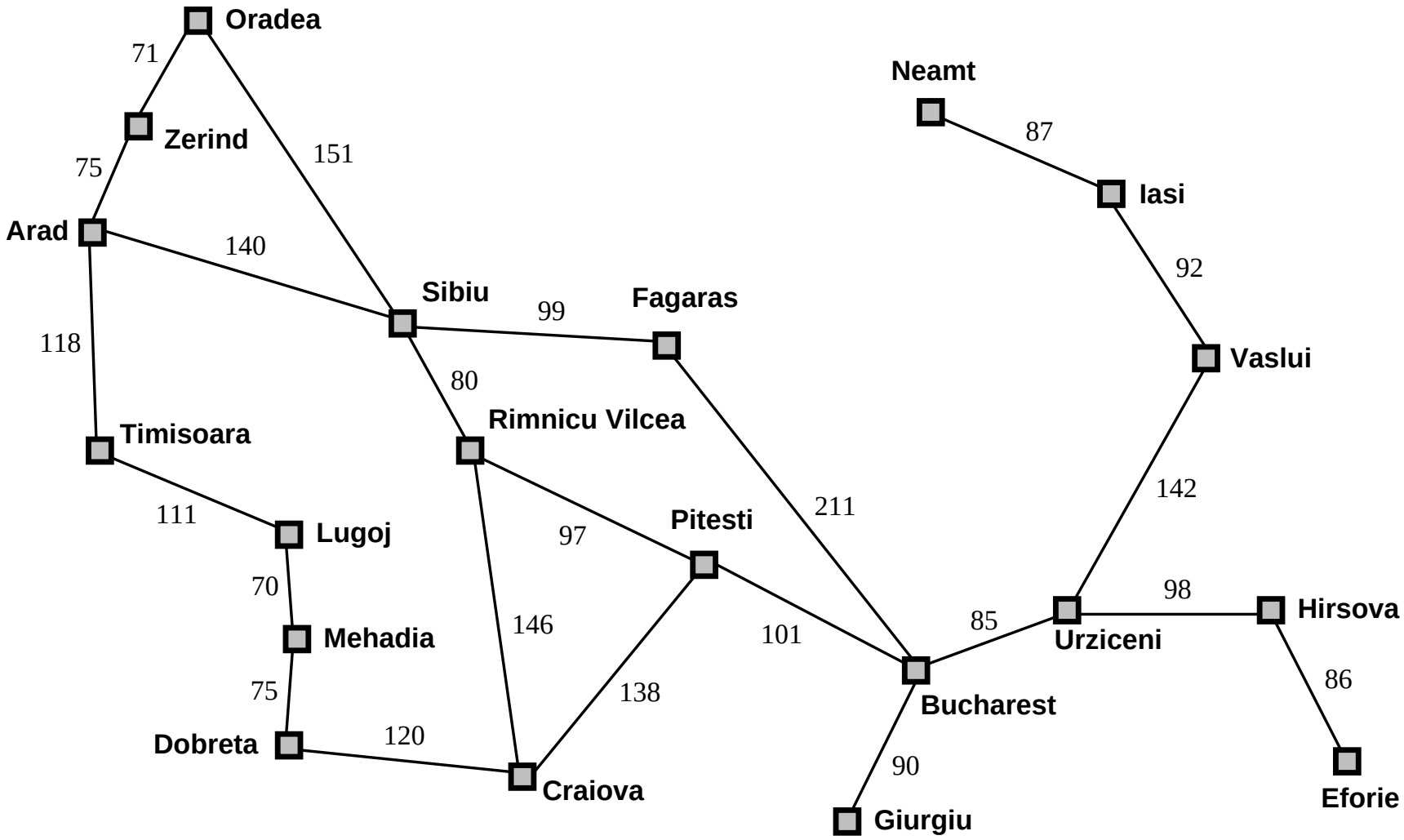
Příklad: 8-královen



Příklad: Kryptoaritmetika

```
    forty       solution:   19786       e.g. f=2, o=9, r=7, etc.
    + ten                   + 850
    + ten                   + 850
---------                 ---------
    sixty                   31486
```

Příklad: Obchodní cestující



SEARCH: Prohledávání Stavového Prostoru

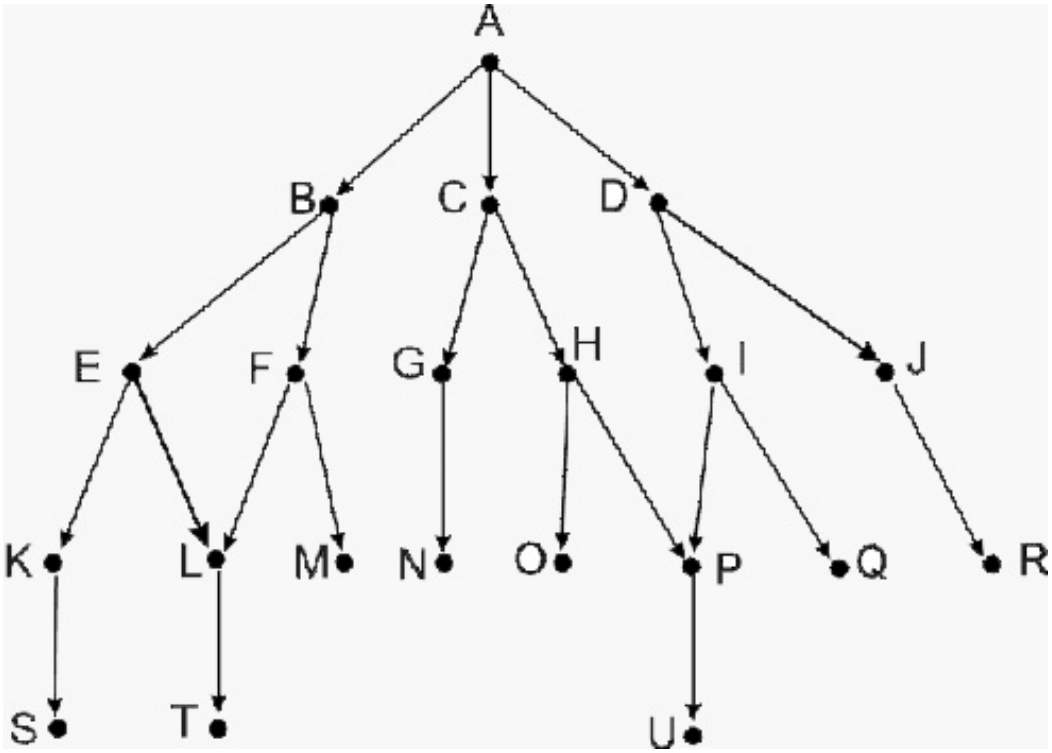
Stejně jako při modelování umělé inteligence tak při prohledávání stavového prostoru řešíme problémy s

- **reprezentací stavového prostoru** — implementace stavových operátorů (funkce `expand`), zabránění cyklům, ...
- **algoritmu prohledávání** — rozhodnutí, který operátor expandovat jako první, odhady, heuristiky

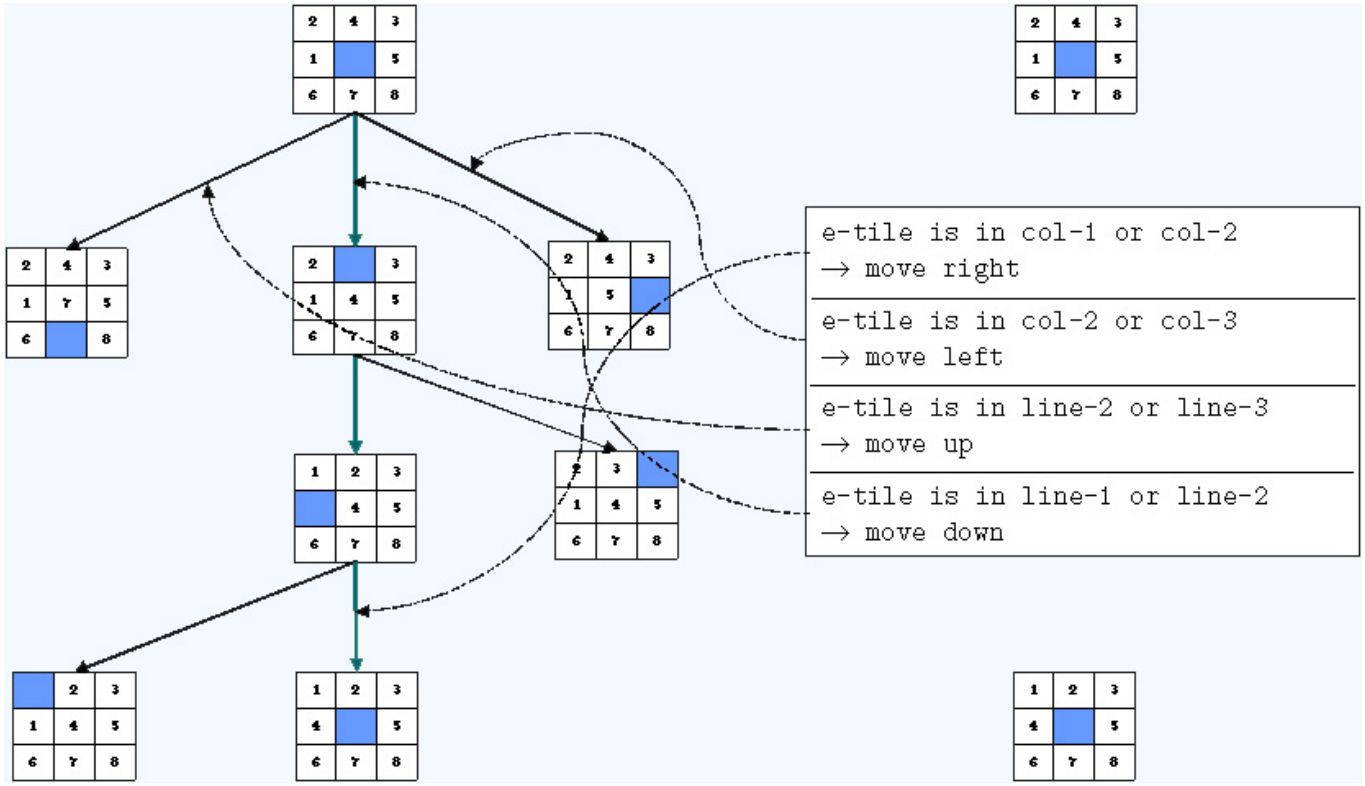
co požadujeme od úspěšného algoritmu?

- je zaručeno, že algoritmus najde řešení, prohledá celý stavový prostor? – **úplnost**
- ukončí se algoritmus?
- najde algoritmus vždy optimální řešení? – **optimalita**
- jaká je náročnost prohledávání? – **časová** and **paměťová** náročnost

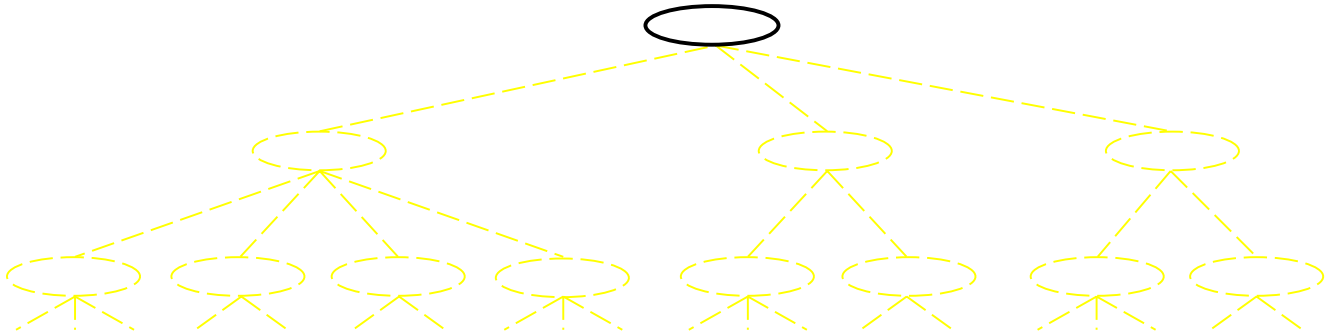
Prohledávání Stavového Prostoru



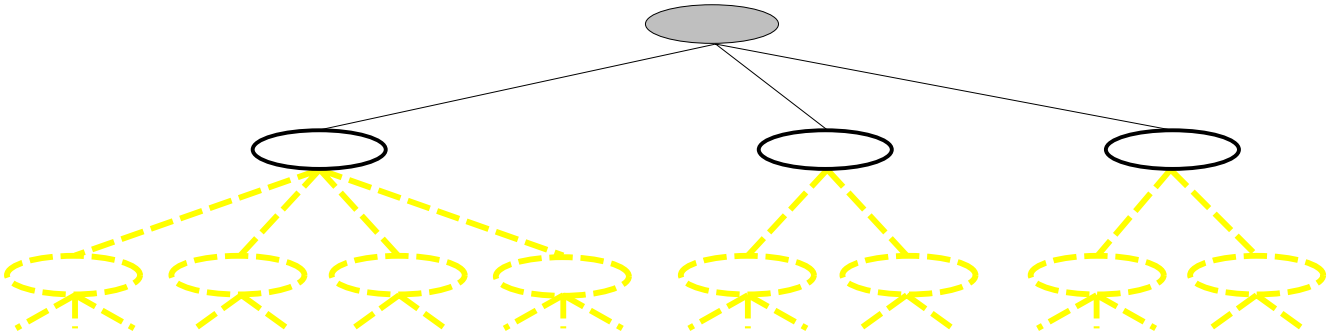
Prohledávání Stavového Prostoru



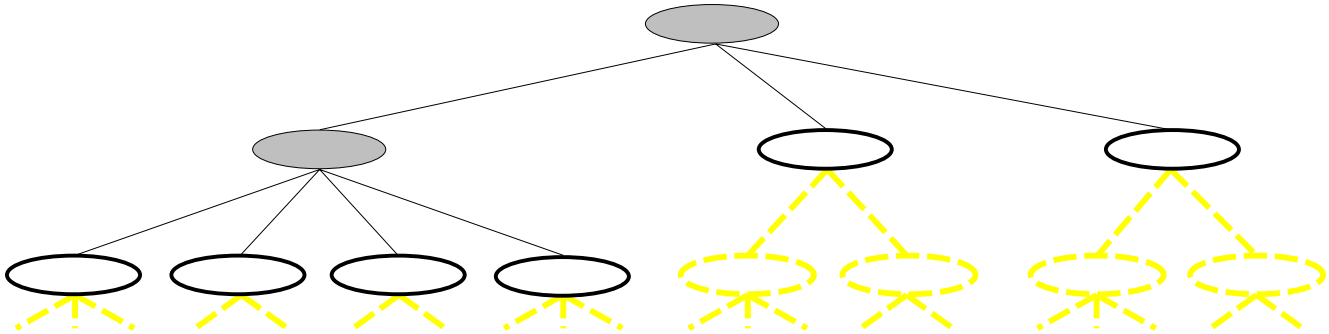
Prohledávání Stavového Prostoru



Prohledávání Stavového Prostoru



Prohledávání Stavového Prostoru



Strategie neinformovaného prohledávání stavového prostoru

dopředné řetězení – forward chaining

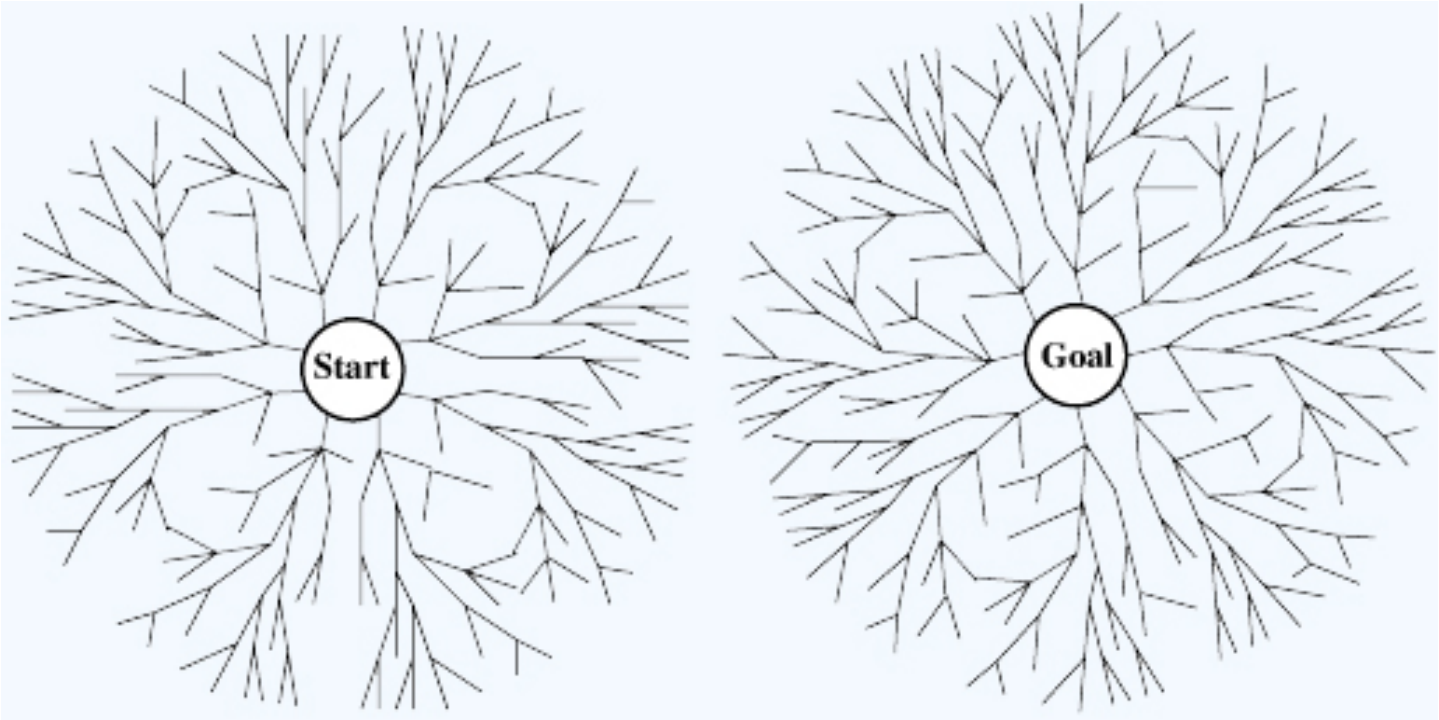
- prohledává prostor od počátku k cíli,
- aplikuje stavové operátory za účelem nalezení nových stavů,
- process iterativně pokračuje než je nalezeno řešení

zpětné řetězení – backward chaining

- prohledává prostor od cíle k počátku
- hledá stavové operátory, které generují aktuální stavy
- podmínky těchto operátorů generují nové cíle
- proces pokračuje až do stavu, který popisuje daný problém

Strategie neinformovaného prohledávání stavového prostoru

Alternativní strategie" **Obousměrné prohledávání** (Bidirectional search), prohledává stavový prostor z obou stran



Strategie neinformovaného prohledávání stavového prostoru

algoritmy prohledávání se dělí také podle toho v jakém pořadí jsou aplikovány aplikovatelné stavové operátory

- **prohledávání do hloubky** vždy aplikuje stavový operátor na co nejčerstvěji rozbalený stav, v případě selhání aplikuje *backtracking*
- **prohledávání do šířky** nejprve prohledá všechny stavy, které jsou stejně daleko od počátečního stavu před tím než expanduje o další úroveň

Při prohledávání stavového prostoru pracujeme s:

1. dynamicky se generovaným stavovým prostorem ve formě orientovaného grafu
2. s *datovými strukturami*: seznamy, které se používají pro prohledávání stavového prostoru :
   - `open list` – seznam otevřených stavů, slouží k řízení stavové expanze
   - `closed list` – seznam prohledaných uzlů, slouží k zabránění zacyklení

Prohledávání do šířky – Breadth First Search (BFS)

```
begin
    open := [Start]
    while (open <> []) do begin
        X := first(_open)
        open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                open :=  open + E
            end
    end
    return(failure)
end.
```

znak <> znamená nerovno,

operátor – znamená odebrání prvku/prvků ze seznamu a

operátor + znamená přiřazení prvku/prvků nakonec seznamu

Vsuvka - analýza algoritmů

```
function SUM(eq)
    sum <= 0
    for i : 1 to length(seq)
        sum = sum + seq(i)
return sum
```

- doba trvání algoritmu? (zjednodušení: doba trvání úměrná počtu operací)

Vsuvka - analýza algoritmů

```
function SUM(eq)
    sum <= 0
    for i : 1 to length(seq)
        sum = sum + seq(i)
return sum
```

- doba trvání algoritmu? (zjednodušení: doba trvání úměrná počtu operací)
  - pro `length(seq)` $= n$, $T(n) = 2n + 2$
  - pro různá n můžeme pracovat s $T(n)_{avg}$ a $T(n)_{worst}$

Vsuvka - analýza algoritmů

```
function SUM(eq)
    sum <= 0
    for i : 1 to length(seq)
        sum = sum + seq(i)
return sum
```

- doba trvání algoritmu? (zjednodušení: doba trvání úměrná počtu operací)
  - pro `length(seq)` $= n$, $T(n) = 2n + 2$
  - pro různá n můžeme pracovat s $T(n)_{avg}$ a $T(n)_{worst}$
- asymptotická analýza algoritmu?
  - $T(n) \approx O(f(n))$ `if` $T(n) \leq kf(n) + c \; \forall n$

Vsuvka - analýza algoritmů

- třídy problémů:
  - P problémy - (např.: $O(n^a)$,$O(\log n)$)
  - NP problémy - nedeterministické P problémy, na deterministickém Turingově stroji exponenciální složitost
  - NP těžké - nejtěžší ze třídy NP, tj. každý problém z NP lze převést na řešení NP-těžkého problému
  - NP-úplné problémy - třída NP problémů, které jsou NP těžké a v NP

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné** ?

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: ANO (je-li b konečné)

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: ANO (je-li b konečné)
- **čas** ?

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: ANO (je-li b konečné)
- **čas**: $1 + b + b^2 + b^3 + ... + b^d + b(b^d - 1) = O(b^{d+1})$ – podle algoritmu uvedeného na slidu 21, počítá se počet expandovaných uzlů (maximální počet uzlů na open seznamu), platí jen v případě, že $m > d$ (jinak je $O(b^d)$).

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: ANO (je-li b konečné)
- **čas**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$ – lze implementovat v případě, že se testuje je-li expandovaný uzel řešení ihned po expanzi (viz algoritmus na následujícím slidu) – z tohoto algoritmu vycházíme při studování komplexit v následujícím výkladu.

Vlastnosti BFS

```
begin
     if Start = GOAL then return(SUCCESS)
    while (_open <> []) do begin
        X := first(open)
        open := open - [X]
        else begin
                    E := expand(X)
                    if for any Y in E: Y = GOAL then return(SUCCESS)
                        else open := open + E
                end
    end
    return(failure)
end.
```

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: ANO (je-li b konečné)
- **čas**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **paměť** ?

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: ANO (je-li b konečné)
- **čas**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **paměť**: $O(b^d)$

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: ANO (je-li b konečné)
- **čas**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **paměť**: $O(b^d)$
- **optimální** ?

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: ANO (je-li b konečné)
- **čas**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **paměť**: $O(b^d)$
- **optimální**: ano, optimalizuje-li se hloubka

Vlastnosti BFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

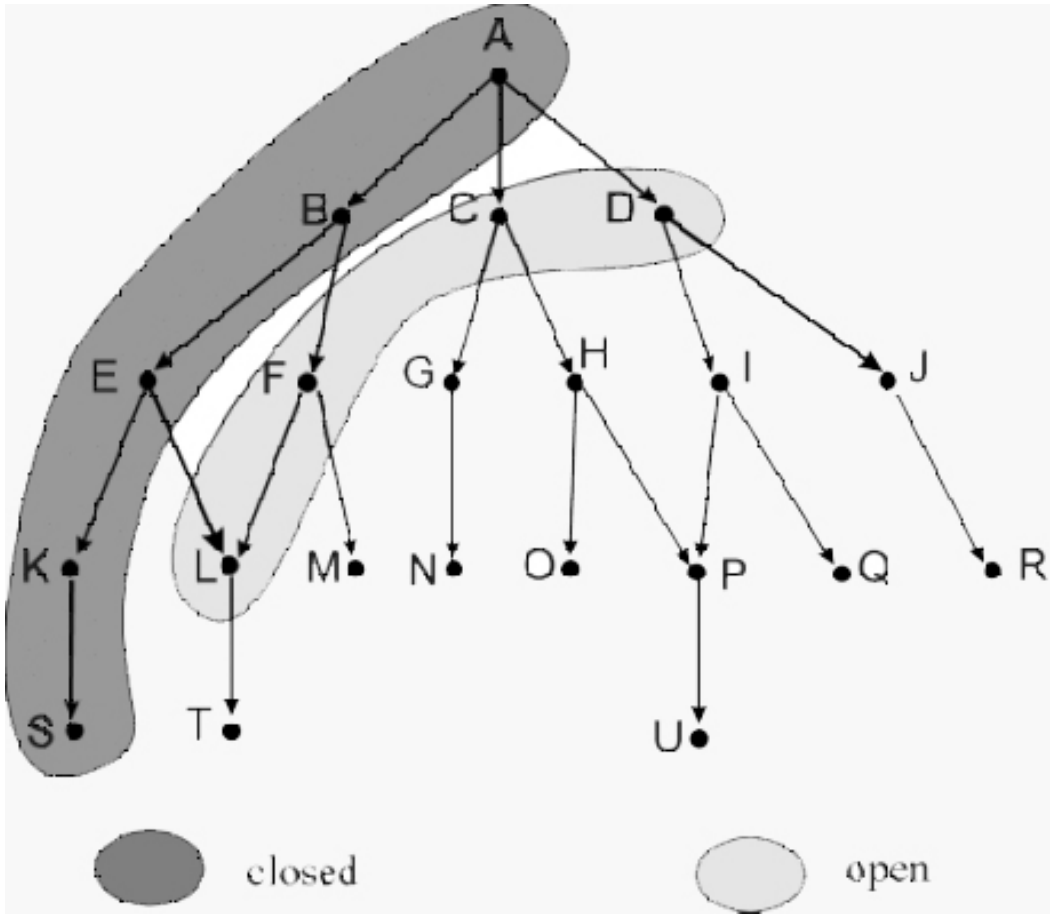
- **úplné**: ANO (je-li b konečné)
- **čas**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **paměť**: $O(b^d)$
- **optimální**: ano, optimalizuje-li se hloubka

Prostor je největší problém - lehce lze generovat 100MB/sec

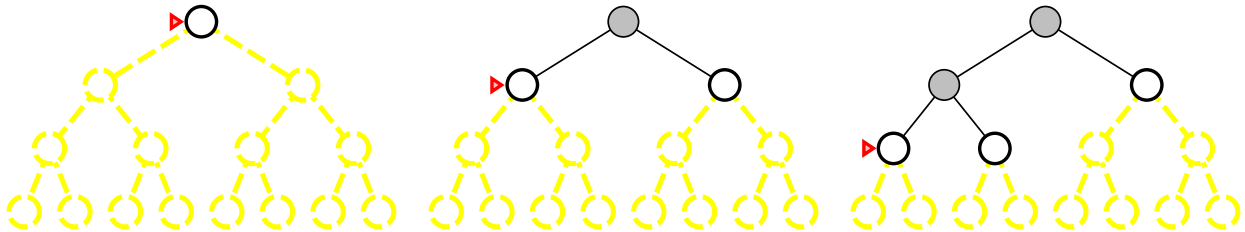
Prohledávání do šířky – Breadth First Search (BFS)

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

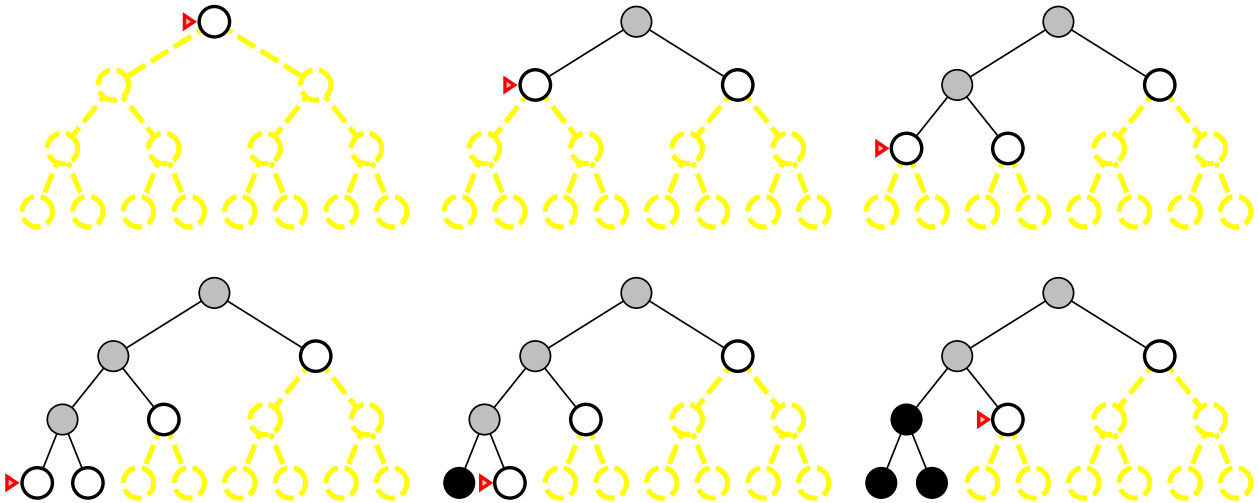
Prohledávání do hloubky – Depth First Search (DFS)



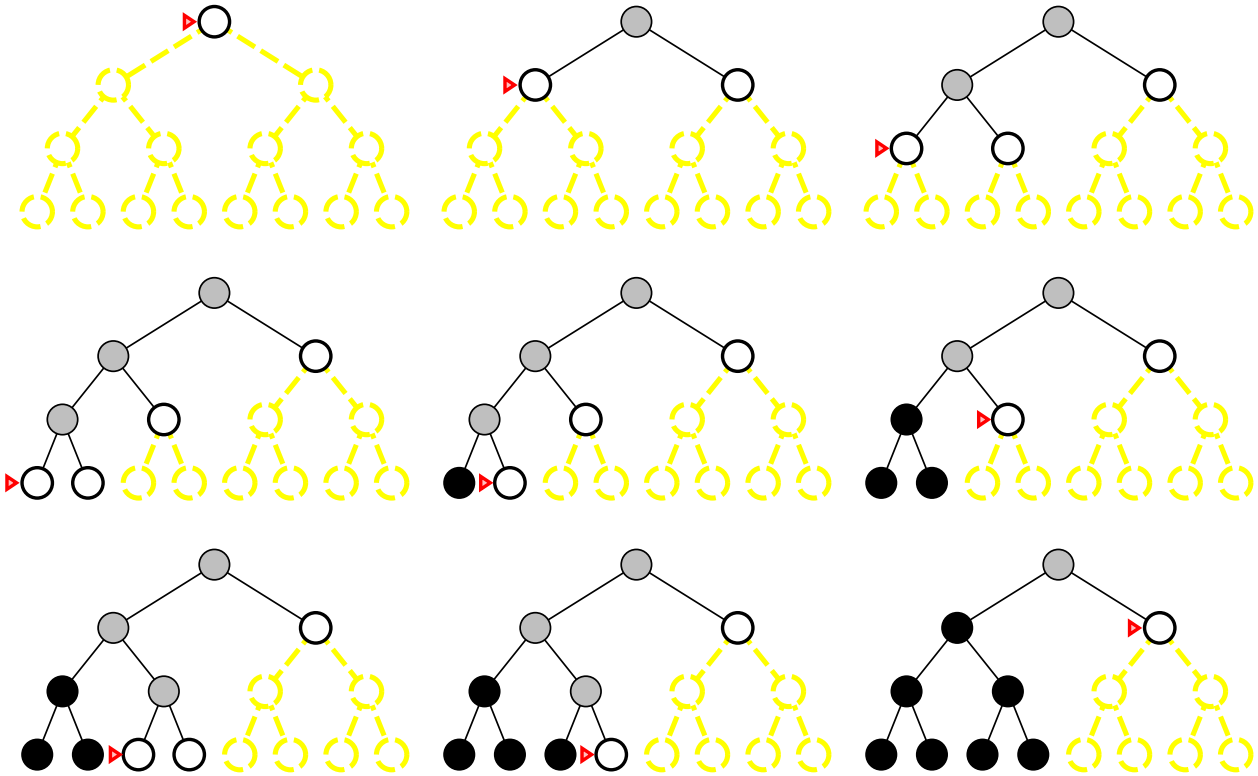
Prohledávání do hloubky – Depth First Search (DFS)



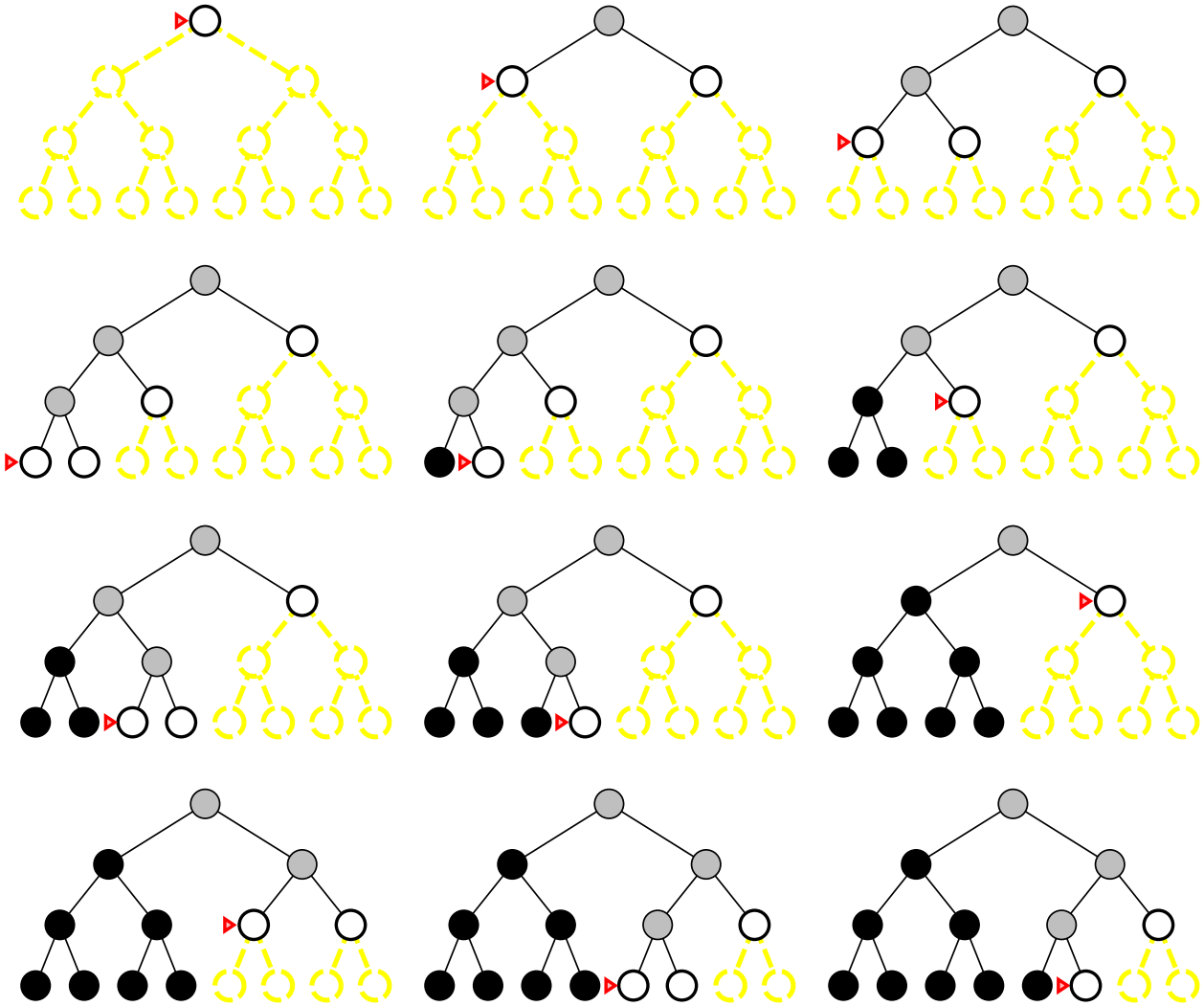
Prohledávání do hloubky – Depth First Search (DFS)



Prohledávání do hloubky – Depth First Search (DFS)



Prohledávání do hloubky – Depth First Search (DFS)



Vlastnosti DFS

Algoritmus, který neřeší možnost zacyklení:

```
begin
    open := [Start]
    while (open <> []) do begin
        X := first(open)
        open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                open := open + E
            end
    end
    return(failure)
end.
```

Vlastnosti DFS

Algoritmus, který zabraňuje zacyklení za použití sezamu `closed`:

```
begin
    open := [Start], closed := []
    while (open <> []) do begin
        X := first(open)
        closed := closed + [X], open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                E := E - closed
                open := open + E
            end
    end
    return(failure)
end.
```

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné** ?

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: NE (i když je b konečné, z důvodu možné existence smyček)

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: NE (i když je b konečné, z důvodu možné existence smyček)
- **čas** ?

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: NE (i když je b konečné, z důvodu možné existence smyček)
- **čas**: b^m – tzn. exponenciálně podle m, problémy, je-li m výrazně větší než d.

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: NE (i když je b konečné, z důvodu možné existence smyček)
- **čas**: b^m – tzn. exponenciálně podle m, problémy, je-li m výrazně větší než d.
- **paměť** ?

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: NE (i když je b konečné, z důvodu možné existence smyček)
- **čas**: b^m – tzn. exponenciálně podle m, problémy, je-li m výrazně větší než d.
- **paměť**: $O(bm)$

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: NE (i když je b konečné, z důvodu možné existence smyček)
- **čas**: b^m – tzn. exponenciálně podle m, problémy, je-li m výrazně větší než d.
- **paměť**: $O(bm)$
- **optimální** ?

Vlastnosti DFS

Mějme b *maximální faktor větvení* (největší počet hran jdoucích z libovolného uzlu) daného stromu, d - nejmenší hloubka stromu, kde se nachází řešení a m maximální hloubka stromu - může být ∞.

- **úplné**: NE (i když je b konečné, z důvodu možné existence smyček)
- **čas**: b^m – tzn. exponenciálně podle m, problémy, je-li m výrazně větší než d.
- **paměť**: $O(bm)$
- **optimální**: ne

Alternativní strategie

DL-DFS (Depth-limited) search:
prohledávání do hloubky s omezenou hloubkou prohledávání l

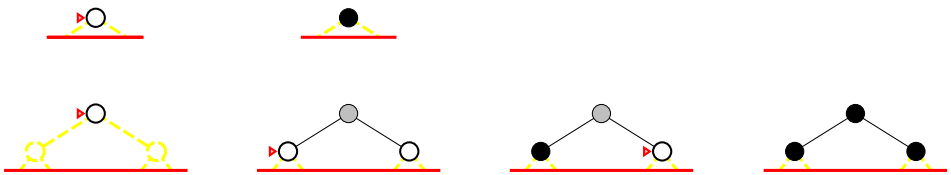
ID-DFS (Iterative deepening) search:
prohledávání do hloubky s iterativní se zvyšující hloubkou prohledávání l

Algoritmus:

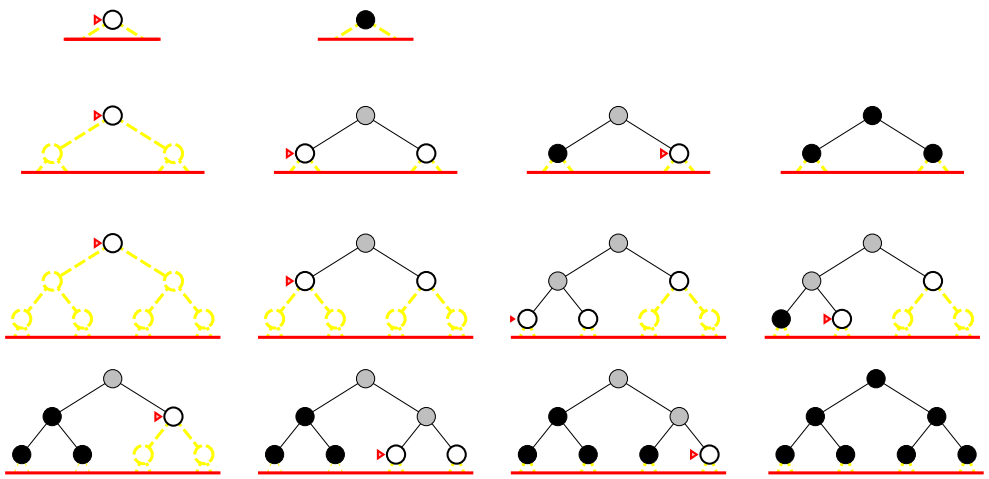
1. $l = 1$
2. `proveď DL-DFS s hloubkou` l
3. `if řešení nalezeno konec`
   `jinak` $l = l + 1$ `a jdi na` 2

Algoritmus IDDFS prohledávání

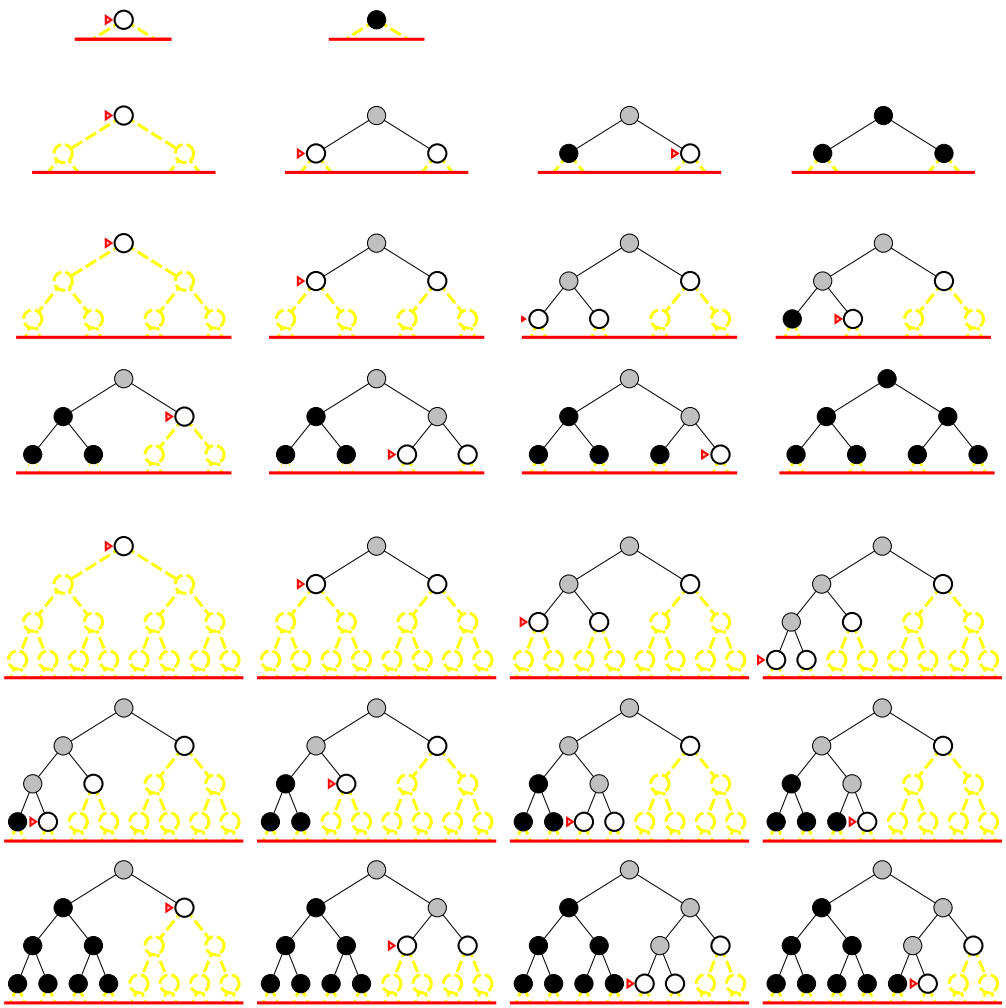
Algoritmus IDDFS prohledávání



Algoritmus IDDFS prohledávání



Algoritmus IDDFS prohledávání



Algoritmus ID-DFS prohledávání

- **úplné** ?

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)
- **čas** ?

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)
- **čas**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
  předpokládáme-li, že každé jedno prohledávání je realizováno algoritmem o komplexitě $O(b^l)$

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)
- **čas**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **paměť** ?

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)
- **čas**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **paměť**: $O(bd)$

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)
- **čas**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **paměť**: $O(bd)$
- **optimální** ?

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)
- **čas**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **paměť**: $O(bd)$
- **optimální**: ano, optimalizuje-li se hloubka

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)
- **čas**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **paměť**: $O(bd)$
- **optimální**: ano, optimalizuje-li se hloubka

:: **porovnání:** pro $b = 10$ a $d = 5$ v nejhorším případě:

Algoritmus ID-DFS prohledávání

- **úplné**: ANO (je-li b konečné)
- **čas**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **paměť**: $O(bd)$
- **optimální**: ano, optimalizuje-li se hloubka

:: **porovnání:** pro $b = 10$ a $d = 5$ je počet expandovaných uzlů nejhorším případě:

- $N(\text{id-dfs}) = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$
- $N(\text{bfs}) = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,100$

ID-DFS expanduje pouze o cca 11% uzlů, což se díky výrazným úsporám paměti vyplatí.

Porovnání Strategií

kritérium/algoritmus	BFS	DFS	Dlim	ID-DFS	BiDir
čas	b^d	b^m	b^l	b^d	$b^{\frac{d}{2}}$
paměť	b^d	bm	bl	bd	$b^{\frac{d}{2}}$
optimalita	ano	ne	ne	ano	ano
úplnost	ano	ne	ano (pro $l \geq d$)	ano	ano

kde b je faktor větvení, d je hloubka ve které se nachází nejmělčí řešení, m je maximální hloubka stromu, l je mez prohledávání.