

Řešení problémů prohledáváním

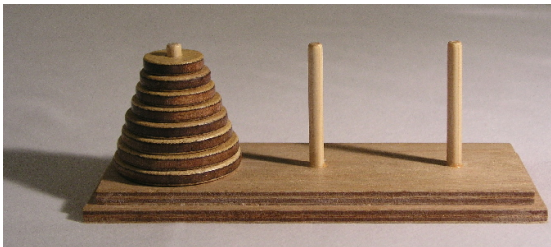
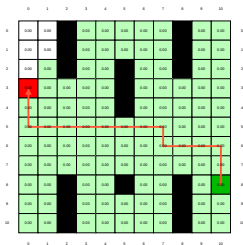
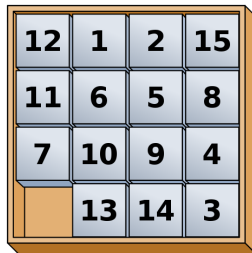
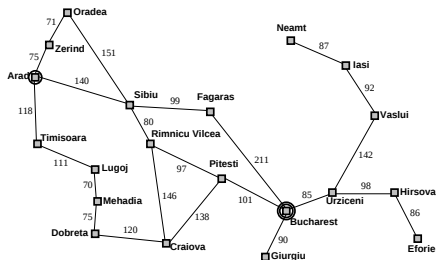
Hledání optimální sekvence stavů, rozhodnutí, akcí

Tomáš Svoboda, Petr Pošík

Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

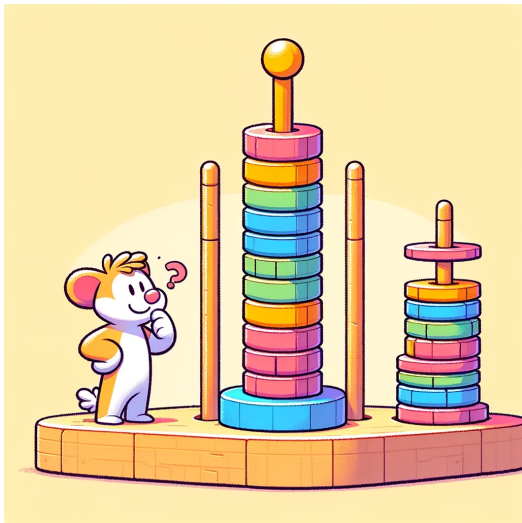
26. února 2025

Jaké problémy chceme řešit?



1

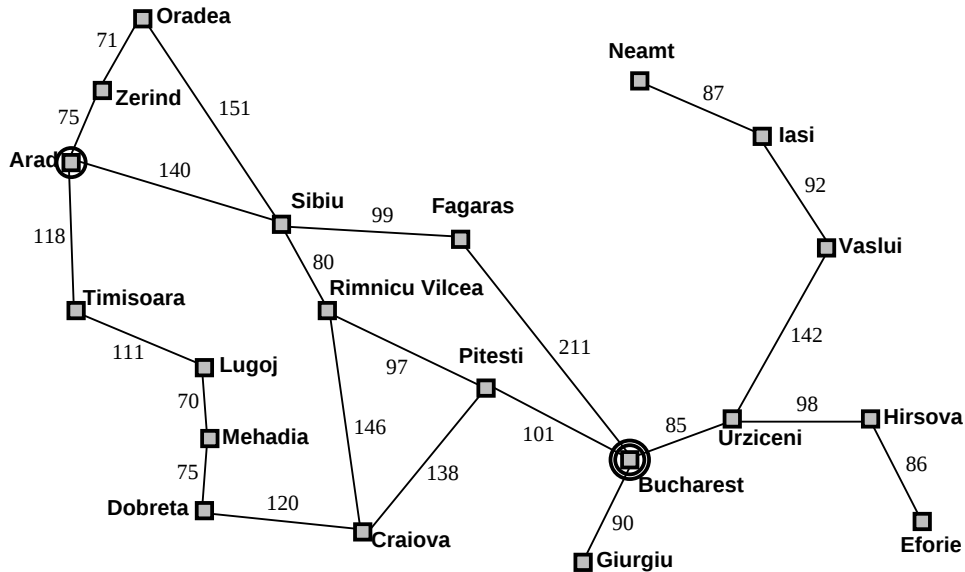
Klíčové je rozumět problému. DALL-E:



Obsah

- ▶ Problém prohledávání. *Co chceme řešit?*
- ▶ Graf stavového prostoru. *Jak problém formalizujeme/reprezentujeme? Abstrakce problému.*
- ▶ Prohledávací stromy. *Vizualizace běhu algoritmu.*
- ▶ Strategie: které větve stromu vybírat?
- ▶ Vlastnosti strategie/algoritmu. *Paměť, čas, ...*
- ▶ Jak to naprogramovat?

Příklad: Cestujeme po Rumunsku



Příklad: Stavy a akce

Cíl:

být v Bukurešti

Formulace problému:

stav: město (v jakém městě se nacházíte)

akce (rozhodnutí): volba cesty z města

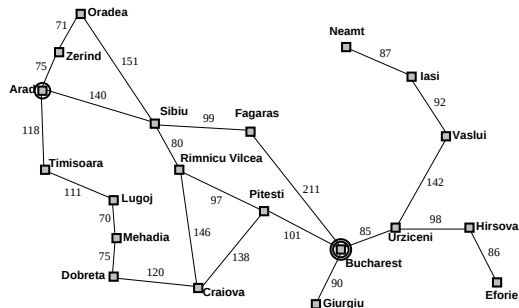
Řešení:

Posloupnost měst (cesta)

(posloupnost akcí/rozhodnutí [2])

Optimalita – náklady, ztráty, užitek, profit, ...:

Energie, čas, poplatky, ...



Příklad: Stavy a akce

Cíl:

být v Bukurešti

Formulace problému:

stav: město (v jakém městě se nacházíte)

akce (rozhodnutí): volba cesty z města

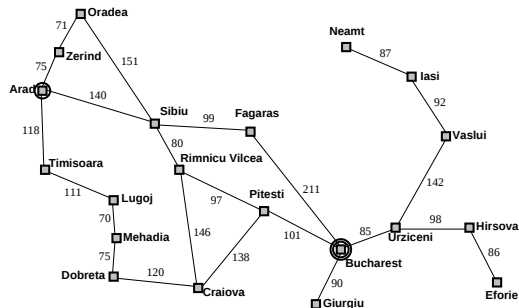
Řešení:

Posloupnost měst (cesta)

(posloupnost akcí/rozhodnutí [2])

Optimalita – náklady, ztráty, užitek, profit, ...:

Energie, čas, poplatky, ...



Příklad: Stavy a akce

Cíl:

být v Bukurešti

Formulace problému:

stav: město (v jakém městě se nacházíte)

akce (rozhodnutí): volba cesty z města

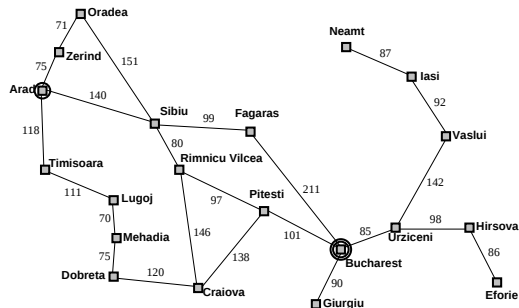
Řešení:

Posloupnost měst (cesta)

(posloupnost akcí/rozhodnutí [2])

Optimalita – náklady, ztráty, užitek, profit, ...:

Energie, čas, poplatky, ...



Příklad: Stavy a akce

Cíl:

být v Bukurešti

Formulace problému:

stav: město (v jakém městě se nacházíte)

akce (rozhodnutí): volba cesty z města

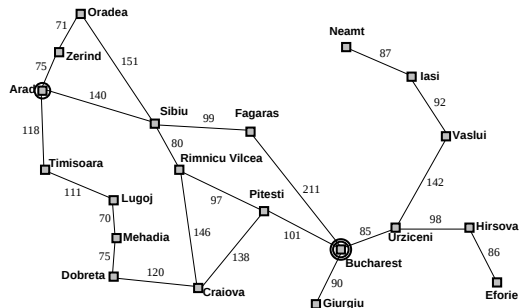
Řešení:

Posloupnost měst (cesta)

(posloupnost akcí/rozhodnutí [2])

Optimalita – náklady, ztráty, užitek, profit, ...:

Energie, čas, poplatky, ...



Příklad: Stavy a akce

Cíl:

být v Bukurešti

Formulace problému:

stav: město (v jakém městě se nacházíte)

akce (rozhodnutí): volba cesty z města

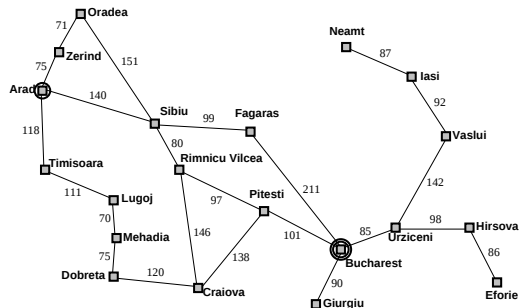
Řešení:

Posloupnost měst (cesta)

(posloupnost akcí/rozhodnutí [2])

Optimalita – náklady, ztráty, užitek, profit, ...:

Energie, čas, poplatky, ...



Příklad: Stavy a akce

Cíl:

být v Bukurešti

Formulace problému:

stav: město (v jakém městě se nacházíte)

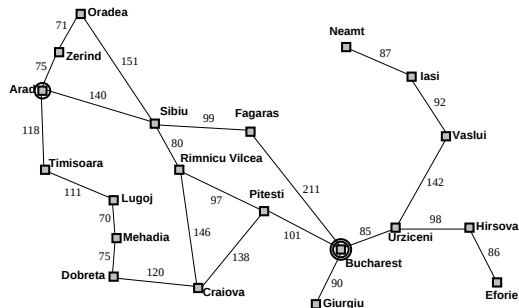
akce (rozhodnutí): volba cesty z města

Řešení:

Posloupnost měst (cesta)
(posloupnost akcí/rozhodnutí [2])

Optimalita – náklady, ztráty, užitek, profit, ...:

Energie, čas, poplatky, ...



Příklad: Skládačka

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

stavy?

akce?

řešení?

náklady/ztráty?

Úloha prohledávání (Search Problem)

- ▶ **Stavový prostor** (včetně počátečního stavu): pozice, konfigurace hrací plochy, ...
- ▶ Prostor/množina akcí : jed' do, Nahoru, Dolů, Doleva, ...
- ▶ Přechodový model : Pro daný stav a akci, vrať následný stav (a cenu nebo odměnu za přechod)
- ▶ Test dosažení cíle : Máme hotovo?

Úloha prohledávání (Search Problem)

- ▶ **Stavový prostor** (včetně počátečního stavu): pozice, konfigurace hrací plochy, ...
- ▶ **Prostor/množina akcí** : jed' do, Nahoru, Dolů, Doleva, ...
- ▶ **Přechodový model** : Pro daný stav a akci, vrať následný stav (a cenu nebo odměnu za přechod)
- ▶ **Test dosažení cíle** : Máme hotovo?

Úloha prohledávání (Search Problem)

- ▶ **Stavový prostor** (včetně počátečního stavu): pozice, konfigurace hrací plochy, ...
- ▶ **Prostor/množina akcí** : jed' do, Nahoru, Dolů, Doleva, ...
- ▶ **Přechodový model** : Pro daný stav a akci, vrať následný stav (a **cenu** nebo **odměnu** za přechod)
- ▶ **Test dosažení cíle** : Máme hotovo?

Úloha prohledávání (Search Problem)

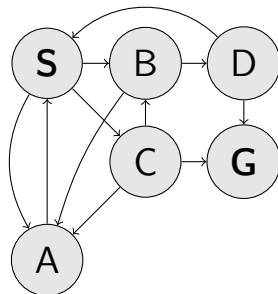
- ▶ **Stavový prostor** (včetně počátečního stavu): pozice, konfigurace hrací plochy, ...
- ▶ **Prostor/množina akcí** : jed' do, Nahoru, Dolů, Doleva, ...
- ▶ **Přechodový model** : Pro daný stav a akci, vrať následný stav (a **cenu** nebo **odměnu** za přechod)
- ▶ **Test dosažení cíle** : Máme hotovo?

Diskrétní stavový prostor

Graf stavového prostoru: reprezentace úlohy prohledávání

- ▶ **Stavy** $s \in \mathcal{S} = \{S, A, B, C, D, G\}$ (konečná množina)
- ▶ Hrany reprezentují **akce** a : pro každý stav s , $a \in \mathcal{A}(s)$ ($\mathcal{A}$ je také konečná množina)
- ▶ **Přechodová funkce** $s' = \text{result}(s, a)$
- ▶ **Počáteční (startovní) stav** $s_0 \in \mathcal{S}$, $s_0 = S$.
- ▶ **Množina cílových stavů** $\mathcal{S}_G \subset \mathcal{S}$.

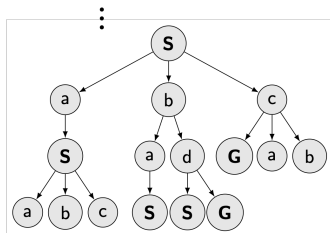
Každý stav se ve stavovém prostoru
vyskytuje pouze *jednou*.



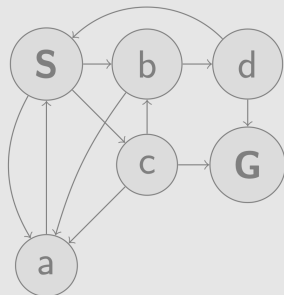
Dialog mezi agentem a prostředím (pohled programátora)

(search) agent

$Q.insert(s_0)$
 $s \leftarrow Q.pop_first()$



problem - env



$s_0, \mathcal{S}_G = \text{env.reset}()$

$\mathcal{A} = \text{env.get_actions}(s_0)$

$s' = \text{env.apply_transition}(s, a)$

⋮

Do šířky (BFS): Q je FIFO (fronta)

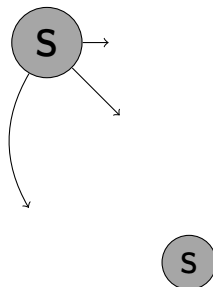
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (_, S)
visited: S

Do šířky (BFS): Q je FIFO (fronta)

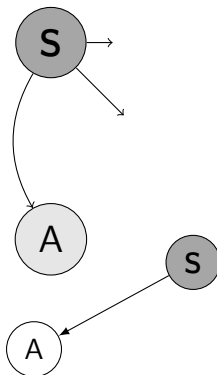
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q:
visited: S

Do šířky (BFS): Q je FIFO (fronta)

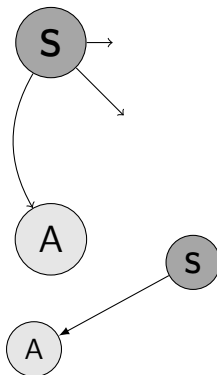
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q:
visited: S

Do šířky (BFS): Q je FIFO (fronta)

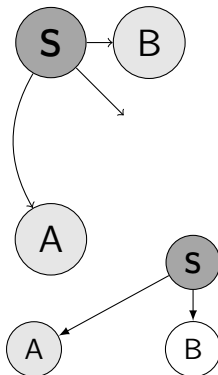
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A)
visited: S A

Do šířky (BFS): Q je FIFO (fronta)

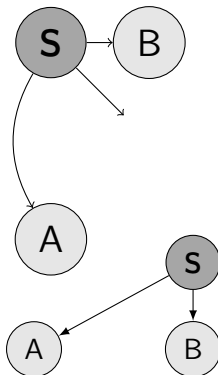
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A)
visited: S A

Do šířky (BFS): Q je FIFO (fronta)

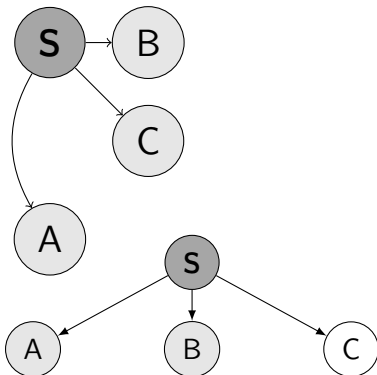
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B)
visited: S A B

Do šířky (BFS): Q je FIFO (fronta)

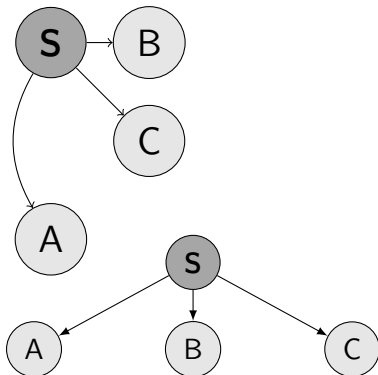
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B)
visited: S A B

Do šířky (BFS): Q je FIFO (fronta)

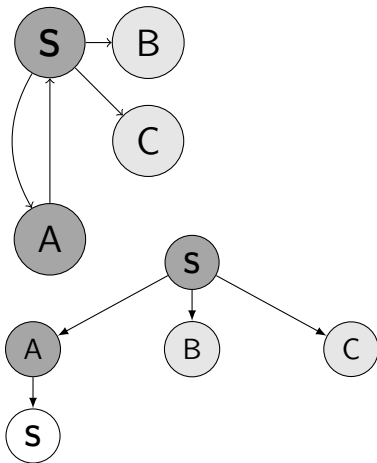
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B) (S,C)
visited: S A B C

Do šířky (BFS): Q je FIFO (fronta)

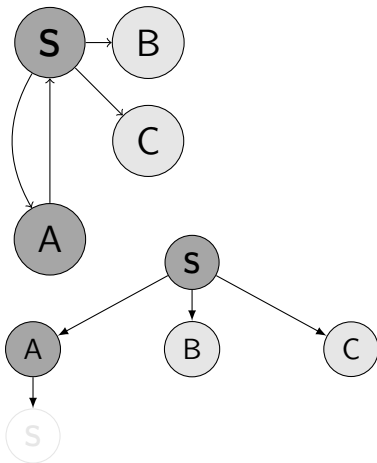
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,B) (S,C)
visited: S A B C

Do šířky (BFS): Q je FIFO (fronta)

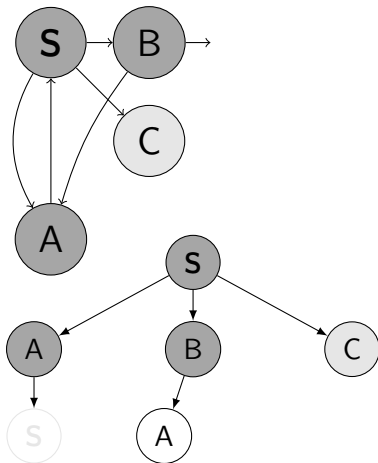
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,B) (S,C)
visited: S A B C

Do šířky (BFS): Q je FIFO (fronta)

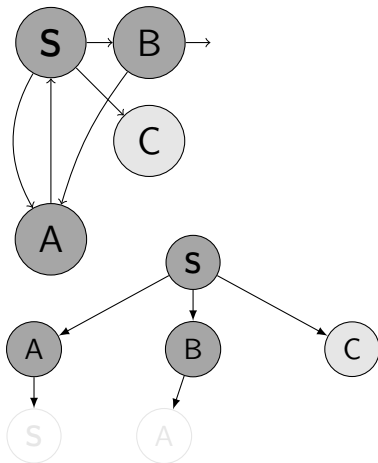
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,C)
visited: S A B C

Do šířky (BFS): Q je FIFO (fronta)

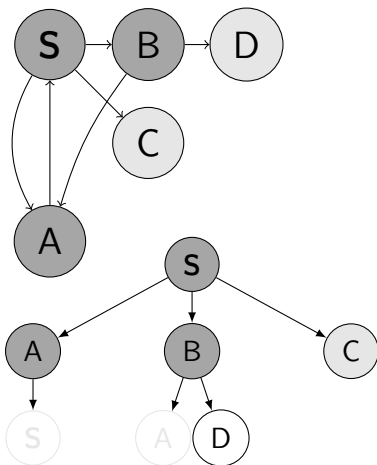
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,C)
visited: S A B C

Do šířky (BFS): Q je FIFO (fronta)

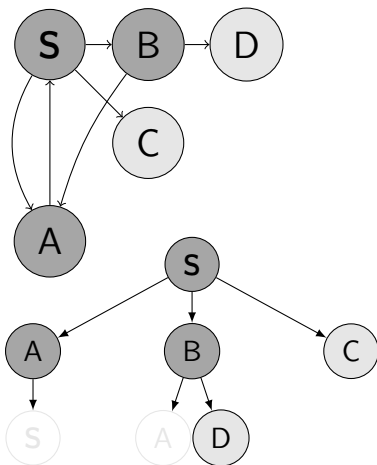
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,C)
visited: S A B C

Do šířky (BFS): Q je FIFO (fronta)

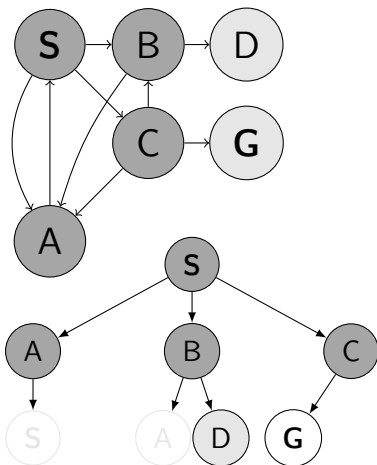
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,C) (B,D)
visited: S A B C D

Do šířky (BFS): Q je FIFO (fronta)

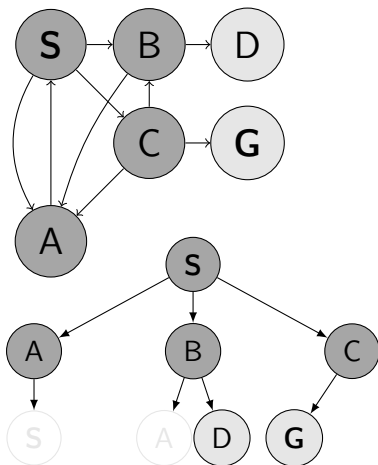
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (B,D)
visited: S A B C D

Do šířky (BFS): Q je FIFO (fronta)

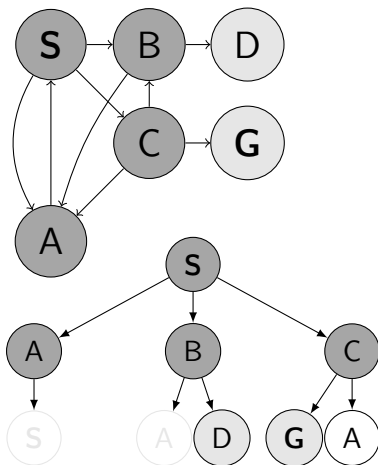
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (B,D) (C,G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

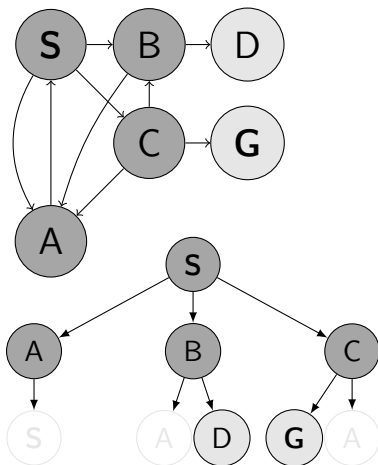
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (B,D) (C,G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

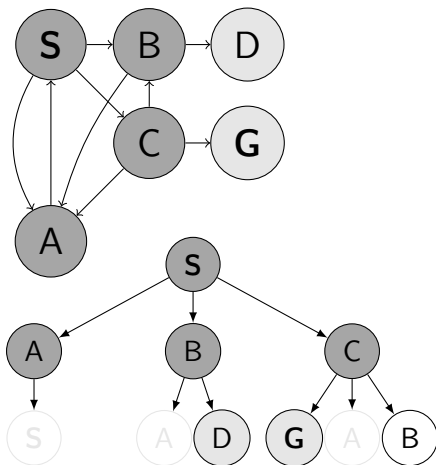
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (B,D) (C,G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

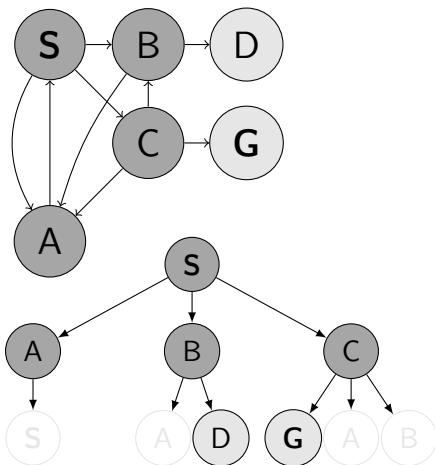
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (B,D) (C,G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

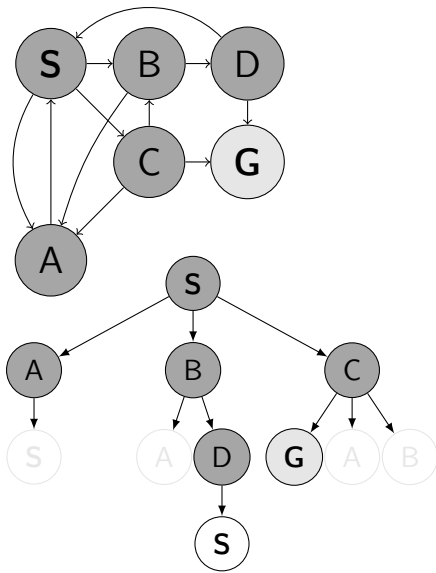
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (B,D) (C,G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

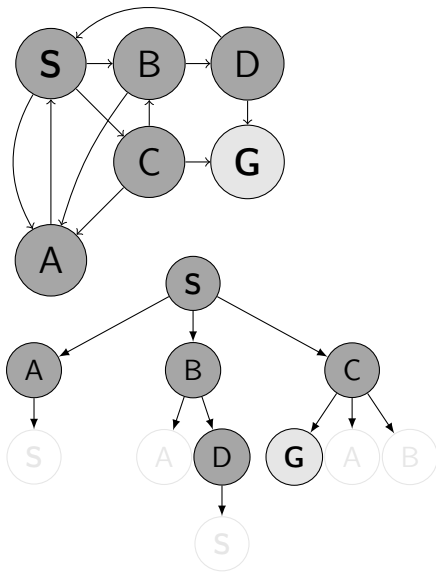
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (C,G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

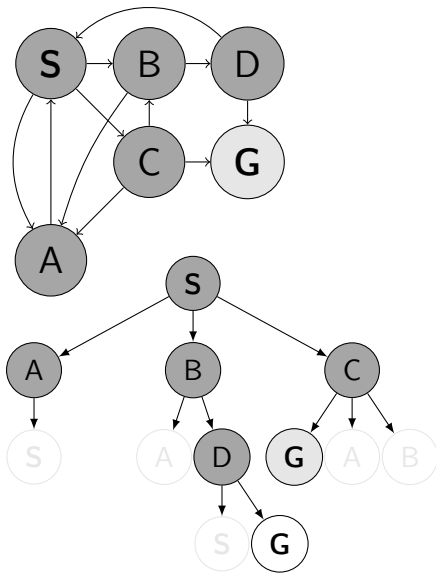
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (C,G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

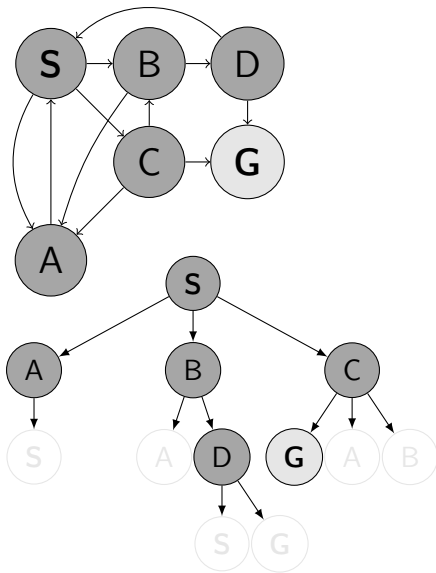
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (C, G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

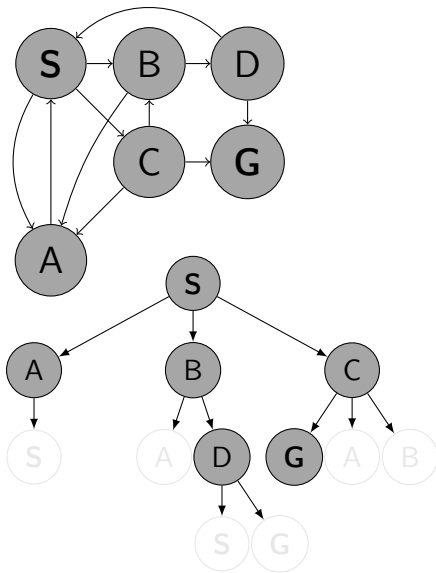
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (C,G)
visited: S A B C D G

Do šířky (BFS): Q je FIFO (fronta)

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q:

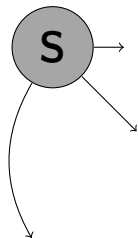
visited: S A B C D G

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



Najdete pro ně vhodná jména?

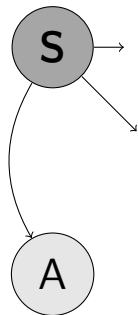
Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



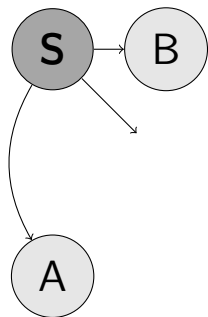
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny

Najdete pro ně vhodná jména?

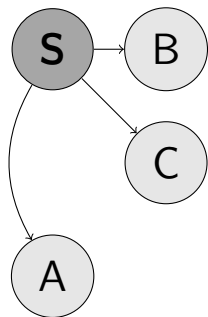


Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



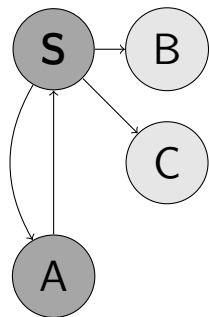
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



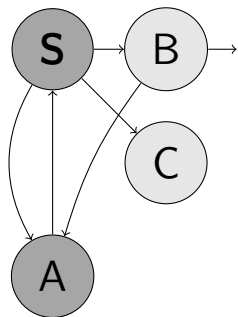
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



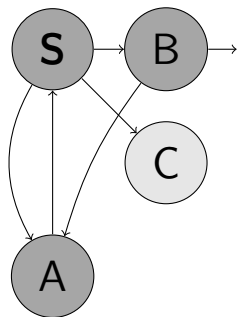
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



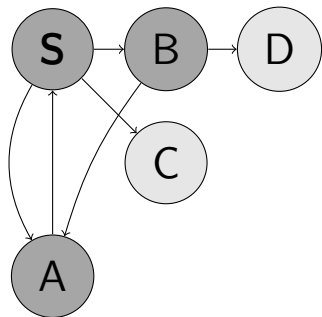
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



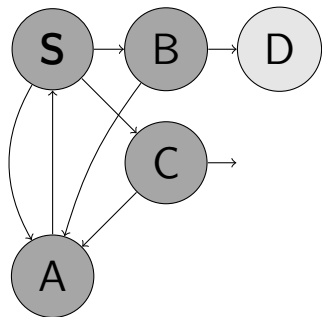
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



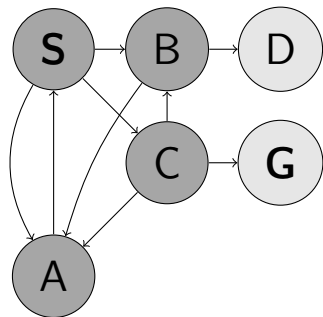
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



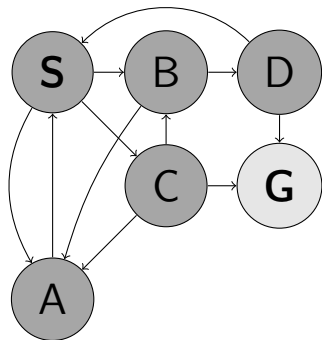
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



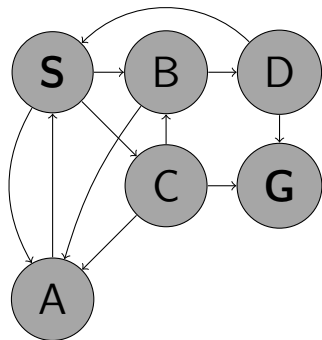
Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



Najdete pro ně vhodná jména?

Prohledávací algoritmus dělí stavový prostor na 3 disjunktní množiny



Najdete pro ně vhodná jména?

Vlastnosti prohledávání (prohledávacího algoritmu)

- ▶ Najde určité řešení (pokud existuje)? Úplný?
- ▶ Najde určité řešení s nejnižší cenou? Optimální?
- ▶ Kolik kroků (operací s uzlem) potřebuje? Časová složitost?
- ▶ Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

► Najde určité řešení (pokud existuje)? Úplný?

► Najde určité řešení s nejnižší cenou? Optimální?

► Kolik kroků (operací s uzlem) potřebuje? Časová složitost?

► Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

► Najde určité řešení (pokud existuje)? Úplný?

► Najde určité řešení s nejnižší cenou? Optimální?

► Kolik kroků (operací s uzlem) potřebuje? Časová složitost?

► Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

- ▶ Najde určité řešení (pokud existuje)? Úplný?
- ▶ Najde určité řešení s nejnižší cenou? Optimální?

▶ Kolik kroků (operací s uzlem) potřebuje? Časová složitost?

▶ Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

- ▶ Najde určité řešení (pokud existuje)? Úplný?
- ▶ Najde určité řešení s nejnižší cenou? Optimální?
- ▶ Kolik kroků (operací s uzlem) potřebuje? Časová složitost?
- ▶ Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

- ▶ Najde určité řešení (pokud existuje)? Úplný?
- ▶ Najde určité řešení s nejnižší cenou? Optimální?
- ▶ Kolik kroků (operací s uzlem) potřebuje? Časová složitost?
- ▶ Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

- ▶ Najde určitě řešení (pokud existuje)? Úplný?
- ▶ Najde určitě řešení s nejnižší cenou? Optimální?
- ▶ Kolik kroků (operací s uzlem) potřebuje? Časová složitost?
- ▶ Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

- ▶ Najde určitě řešení (pokud existuje)? Úplný?
- ▶ Najde určitě řešení s nejnižší cenou? Optimální?
- ▶ Kolik kroků (operací s uzlem) potřebuje? Časová složitost?
- ▶ Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

- ▶ Najde určitě řešení (pokud existuje)? Úplný?
- ▶ Najde určitě řešení s nejnižší cenou? Optimální?
- ▶ Kolik kroků (operací s uzlem) potřebuje? Časová složitost?
- ▶ Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Vlastnosti prohledávání (prohledávacího algoritmu)

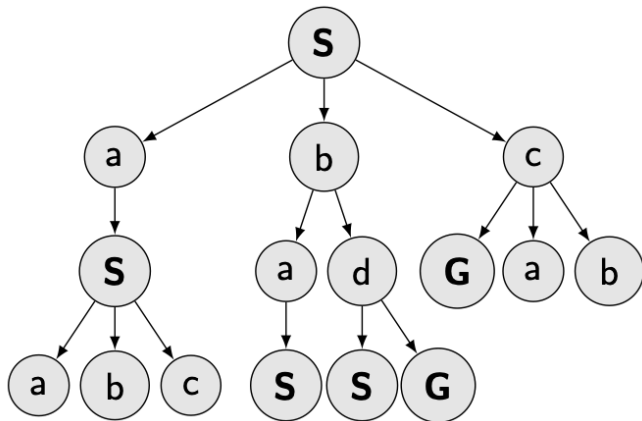
- ▶ Najde určité řešení (pokud existuje)? Úplný?
- ▶ Najde určité řešení s nejnižší cenou? Optimální?
- ▶ Kolik kroků (operací s uzlem) potřebuje? Časová složitost?
- ▶ Kolik uzlů si musí pamatovat? Prostorová/Paměťová složitost?

Kolik je uzlů v prohledávacím stromě? Jaké jsou parametry prohledávacího stromu?

Prohledávací strategie

Jak procházet/stavět prohledávací strom?

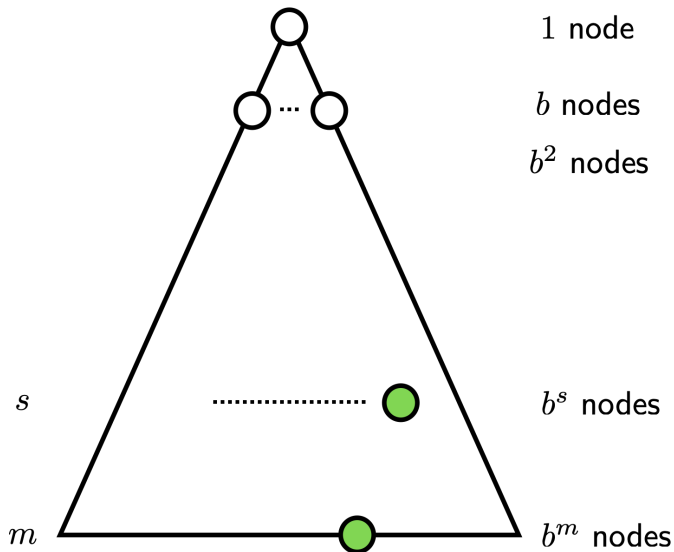
- ▶ **Hloubka** d uzlu ve stromě.
- ▶ **Maximální hloubka** stromu m .
Může být ∞ .
- ▶ (Průměrný) **faktor větvení** b .
- ▶ s je nejmenší hloubka, v níž leží některý z Cílů.
- ▶ Kolik je uzlů v celém stromě?



Prohledávací strategie

Jak procházet/stavět prohledávací strom?

- ▶ **Hloubka** d uzlu ve stromě.
- ▶ **Maximální hloubka** stromu m .
Může být ∞ .
- ▶ (Průměrný) **faktor větvení** b .
- ▶ s je nejmenší hloubka, v níž leží některý z Cílů.
- ▶ Kolik je uzlů v celém stromě?



Vlastnosti algoritmu prohl. do šířky (BFS)

Úplný?

Optimální?

Časová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$

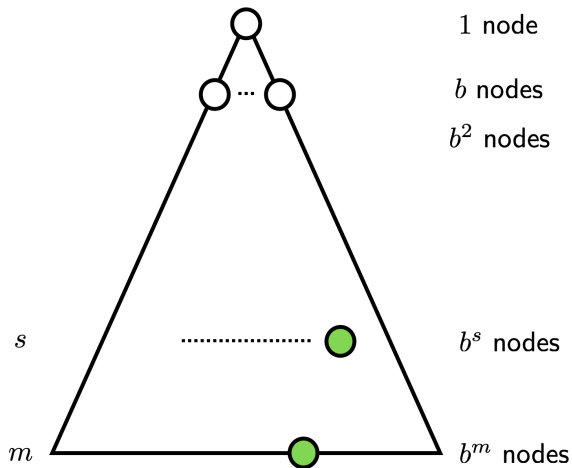
Prostorová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$



Vlastnosti algoritmu prohl. do šířky (BFS)

Úplný?

Optimální?

Časová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$

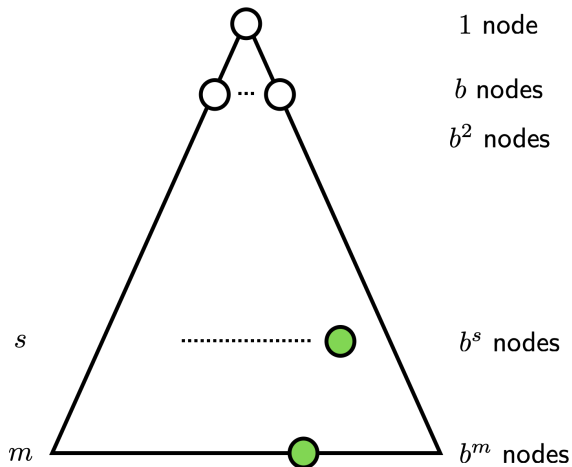
Prostorová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$



Vlastnosti algoritmu prohl. do šířky (BFS)

Úplný?

Optimální?

Časová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$

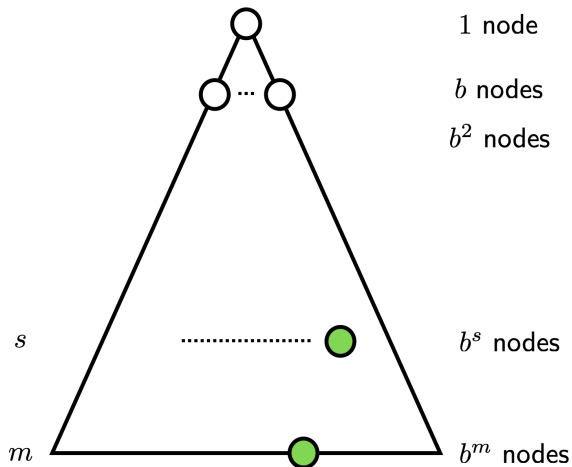
Prostorová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$



Vlastnosti algoritmu prohl. do šířky (BFS)

Úplný?

Optimální?

Časová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$

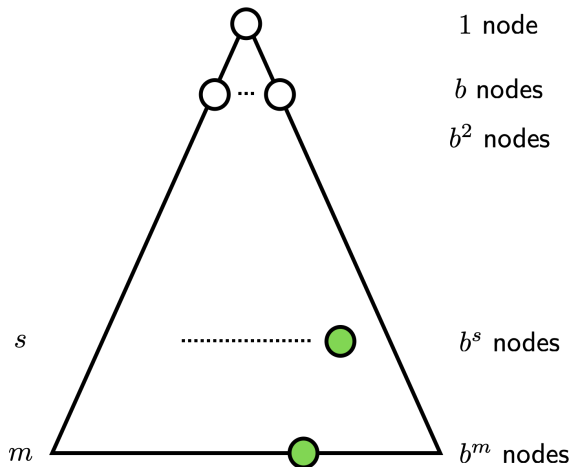
Prostorová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

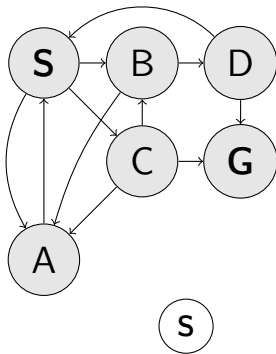
C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$



Do hloubky (DFS): Q je LIFO (zásobník)

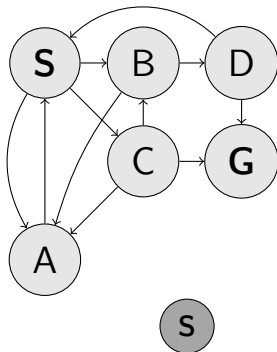
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (_, S)
visited: S

Do hloubky (DFS): Q je LIFO (zásobník)

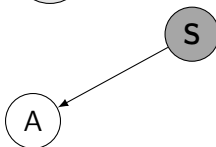
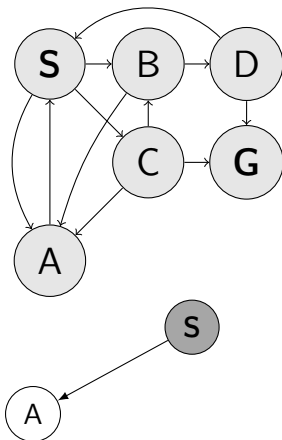
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q:
visited: S

Do hloubky (DFS): Q je LIFO (zásobník)

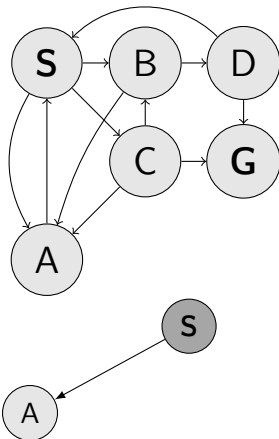
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q:
visited: S

Do hloubky (DFS): Q je LIFO (zásobník)

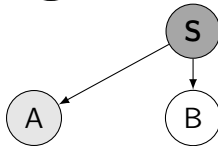
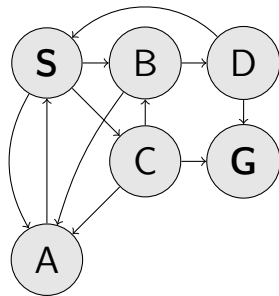
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s')
12:     else
13:       Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A)
visited: S A

Do hloubky (DFS): Q je LIFO (zásobník)

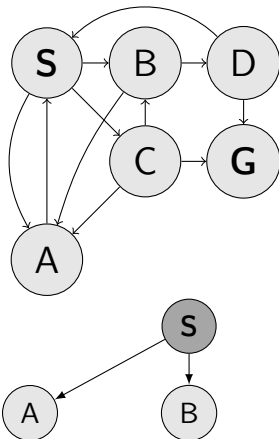
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s')
12:     else
13:       Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A)
visited: S A

Do hloubky (DFS): Q je LIFO (zásobník)

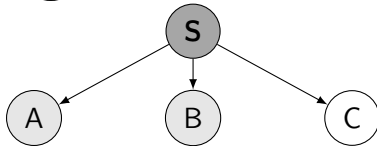
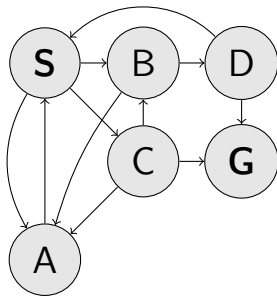
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B)
visited: S A B

Do hloubky (DFS): Q je LIFO (zásobník)

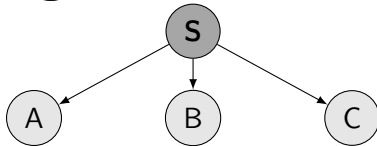
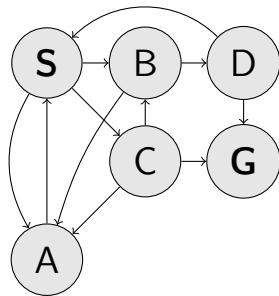
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B)
visited: S A B

Do hloubky (DFS): Q je LIFO (zásobník)

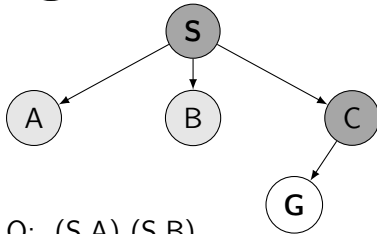
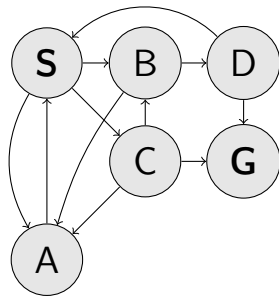
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s')
12:     else
13:       Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B) (S,C)
visited: S A B C

Do hloubky (DFS): Q je LIFO (zásobník)

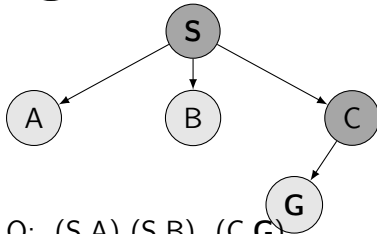
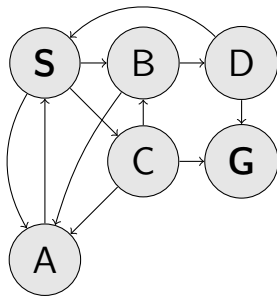
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s')
12:     else
13:       Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B)
visited: S A B C

Do hloubky (DFS): Q je LIFO (zásobník)

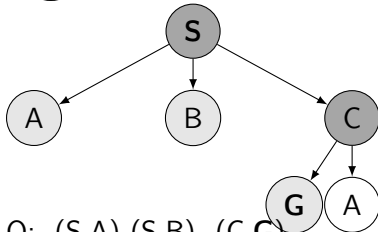
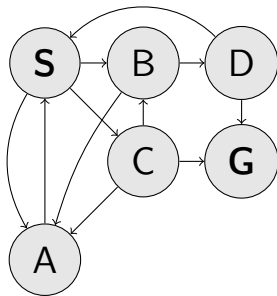
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B) (C,G)
visited: S A B C G

Do hloubky (DFS): Q je LIFO (zásobník)

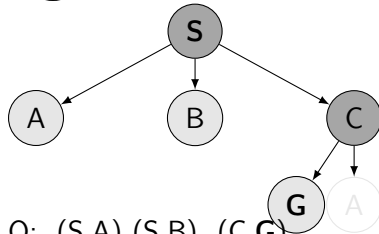
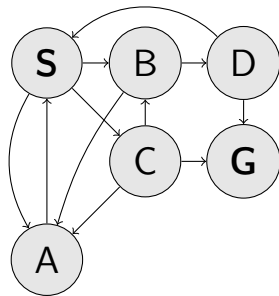
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:         Označ s' jako navštívený
11:         Q.insert(s, s')
12:       else
13:         Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B) (C,G)
visited: S A B C G

Do hloubky (DFS): Q je LIFO (zásobník)

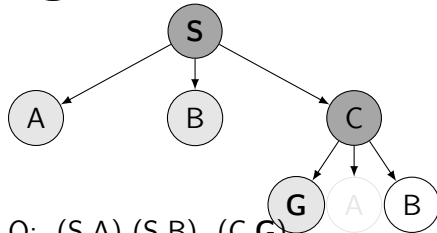
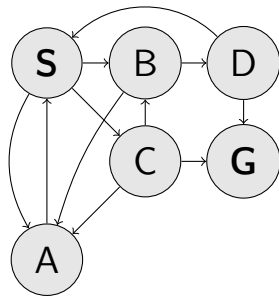
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s')
12:     else
13:       Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B) (C,G)
visited: S A B C G

Do hloubky (DFS): Q je LIFO (zásobník)

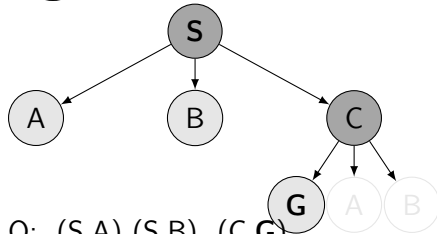
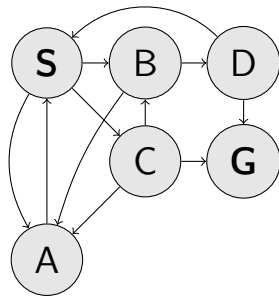
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s')
12:     else
13:       Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B) (C,G)
visited: S A B C G

Do hloubky (DFS): Q je LIFO (zásobník)

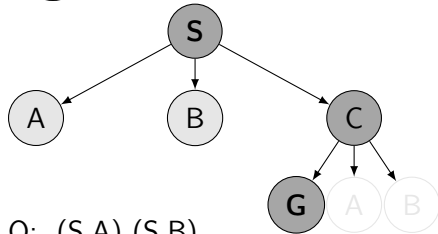
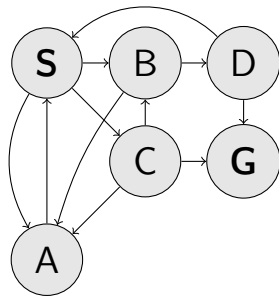
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s')
12:     else
13:       Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B) (C,G)
visited: S A B C G

Do hloubky (DFS): Q je LIFO (zásobník)

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s ← Q.pop()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s' ← result(s, a)
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s')
12:     else
13:       Vyřeš duplikaci s' v Q
return Failure
```



Q: (S,A) (S,B)
visited: S A B C G

Vlastnosti algoritmu prohl. do hloubky (DFS)

Úplný?

Optimální?

Časová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$

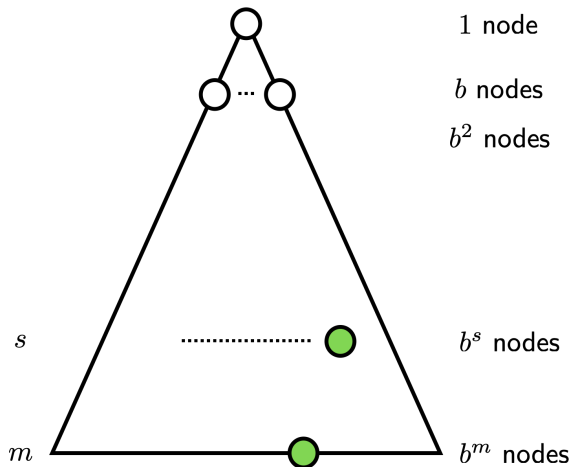
Prostorová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$



Vlastnosti algoritmu prohl. do hloubky (DFS)

Úplný?

Optimální?

Časová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$

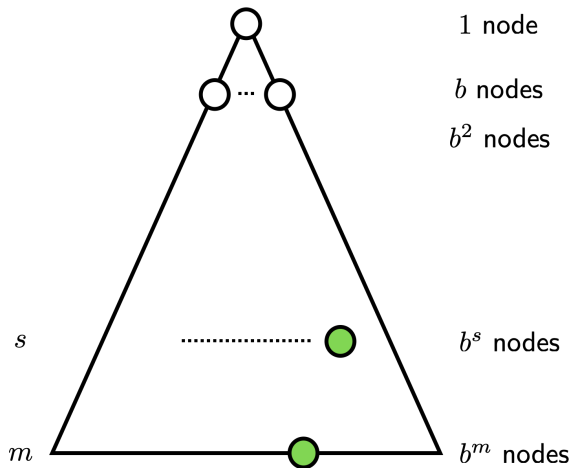
Prostorová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$



Vlastnosti algoritmu prohl. do hloubky (DFS)

Úplný?

Optimální?

Časová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$

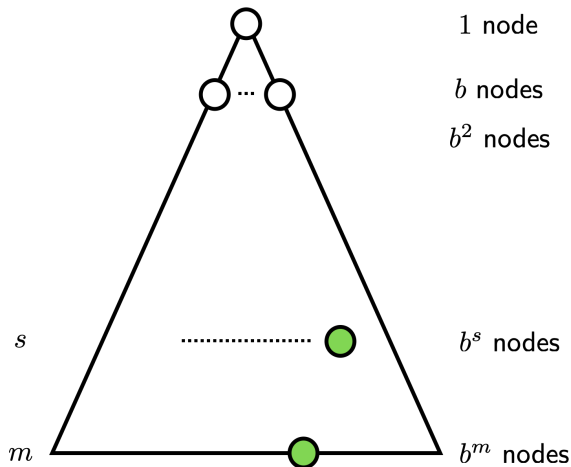
Prostorová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$



Vlastnosti algoritmu prohl. do hloubky (DFS)

Úplný?

Optimální?

Časová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$

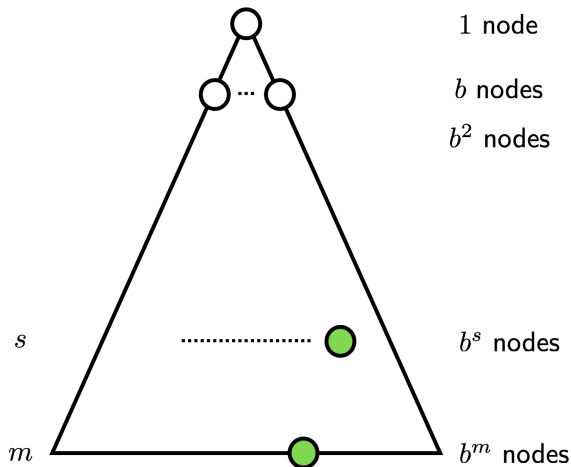
Prostorová složitost?

A $\mathcal{O}(bm)$

B $\mathcal{O}(b^m)$

C $\mathcal{O}(m^b)$

D $\mathcal{O}(b^s)$



DFS s iterativním prohlubováním (ID-DFS)

- ▶ Začněte s `maxdepth = 1`
- ▶ Spusťte DFS s omezenou hloubkou. Bylo nalezeno řešení?
- ▶ Pokud řešení nebylo nalezeno, vše zapomeňte, zvyšte `maxdepth` a jděte na předchozí krok.

Zapomínání všeho mezi jednotlivými kroky - není to obrovské plýtvání?

DFS s iterativním prohlubováním (ID-DFS)

- ▶ Začněte s `maxdepth = 1`
- ▶ Spusťte DFS s omezenou hloubkou. Bylo nalezeno řešení?
 - ▶ Pokud řešení nebylo nalezeno, vše zapomeňte, zvyšte `maxdepth` a jděte na předchozí krok.

Zapomínání všeho mezi jednotlivými kroky - není to obrovské plýtvání?

DFS s iterativním prohlubováním (ID-DFS)

- ▶ Začněte s `maxdepth = 1`
- ▶ Spusťte DFS s omezenou hloubkou. Bylo nalezeno řešení?
- ▶ Pokud řešení nebylo nalezeno, vše zapomeňte, zvyšte `maxdepth` a jděte na předchozí krok.

Zapomínání všeho mezi jednotlivými kroky - není to obrovské plýtvání?

DFS s iterativním prohlubováním (ID-DFS)

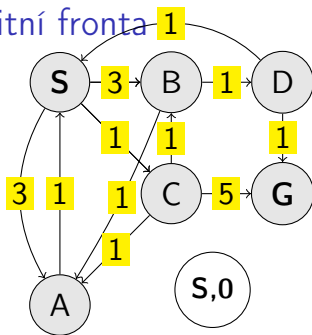
- ▶ Začněte s `maxdepth = 1`
- ▶ Spusťte DFS s omezenou hloubkou. Bylo nalezeno řešení?
- ▶ Pokud řešení nebylo nalezeno, vše zapomeňte, zvyšte `maxdepth` a jděte na předchozí krok.

Zapomínání všeho mezi jednotlivými kroky - není to obrovské plýtvání?

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

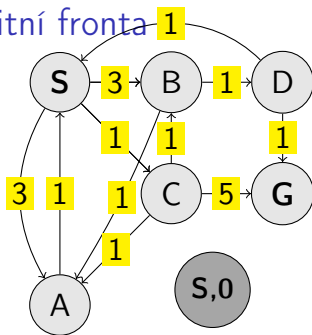


Q: (_, S, 0)
visited: S

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

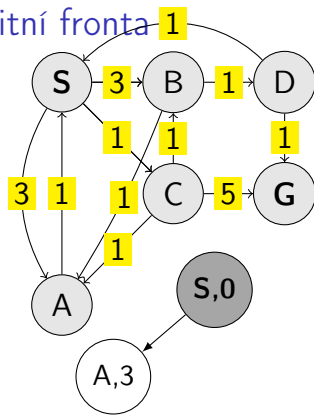


Q:
visited: S

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

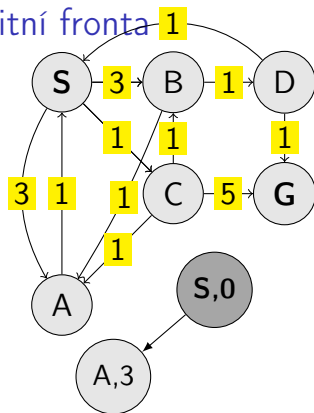


Q:
visited: S

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

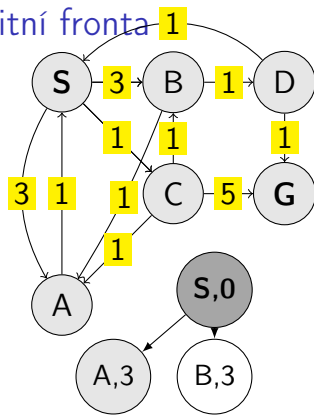


Q: (S,A,3)
visited: S A

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

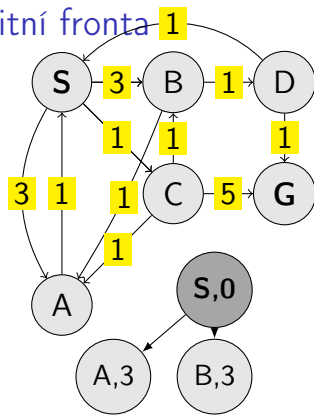


Q: (S,A,3)
visited: S A

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

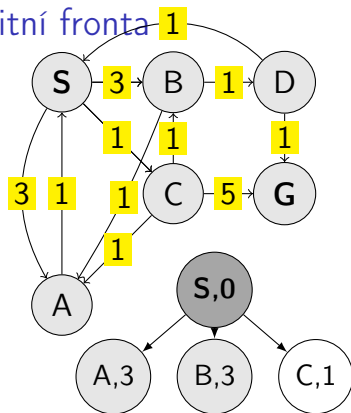


Q: (S,A,3) (S,B,3)
visited: S A B

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

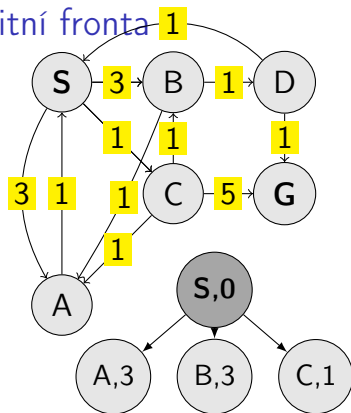


Q: (S,A,3) (S,B,3)
visited: S A B

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

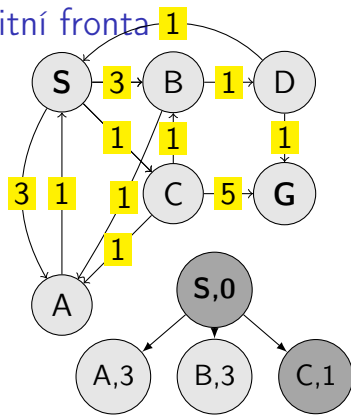


Q: (S,C,1) (S,A,3) (S,B,3)
visited: S A B C

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

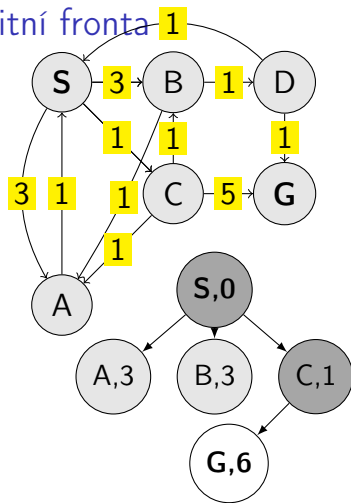


Q: (S,A,3) (S,B,3)
visited: S A B C

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

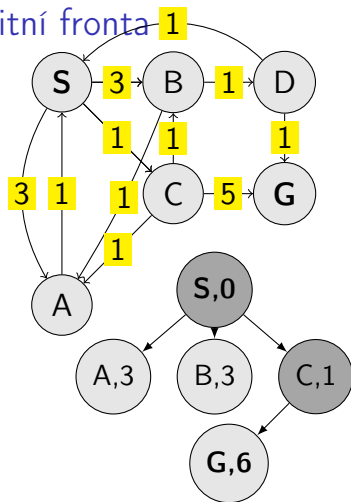


Q: (S,A,3) (S,B,3)
visited: S A B C

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

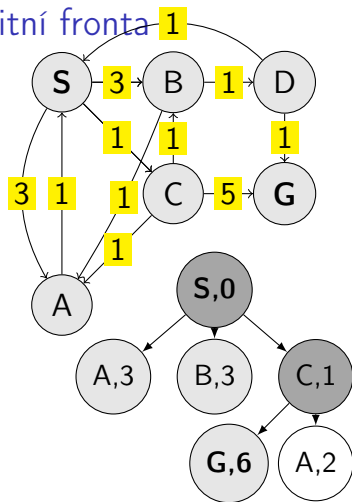


Q: (S,A,3) (S,B,3) (C,G,6)
visited: S A B C G

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

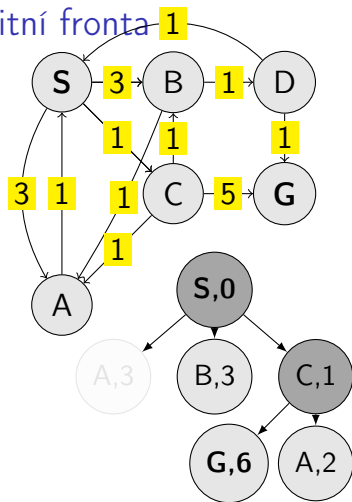


Q: (S,A,3) (S,B,3) (C,G,6)
visited: S A B C G

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

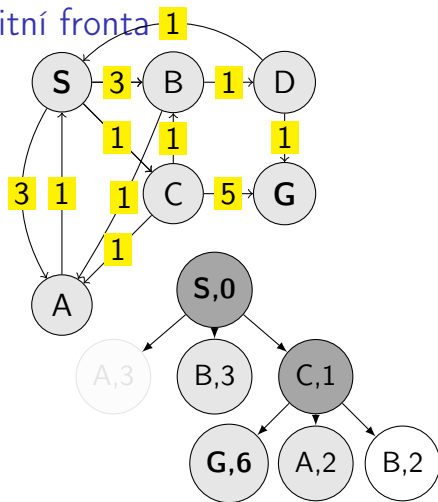


Q: (C,A,2) (S,B,3) (C,G,6)
visited: S A B C G

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

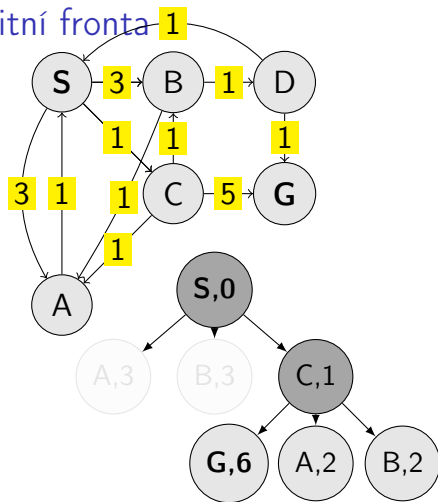


Q: (C,A,2) (S,B,3) (C,G,6)
visited: S A B C G

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

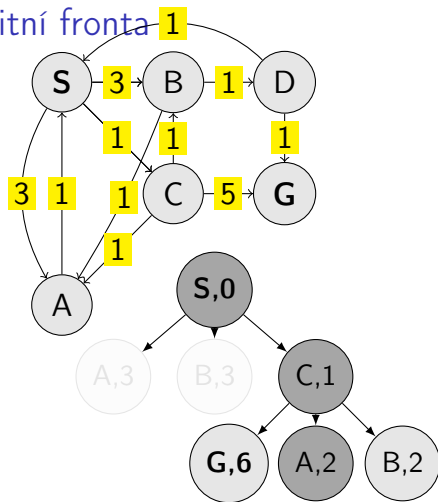
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

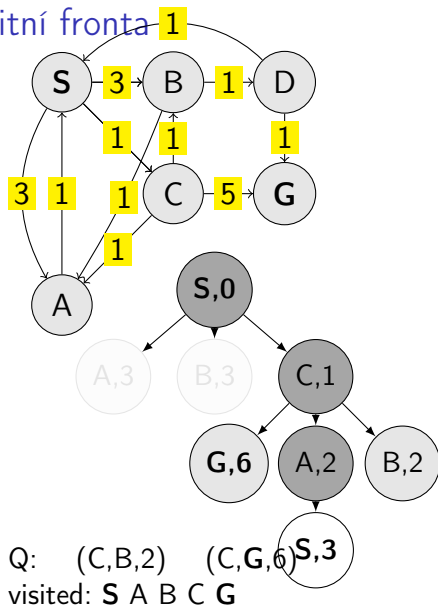


Q: (C,B,2) (C,G,6)
visited: S A B C G

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

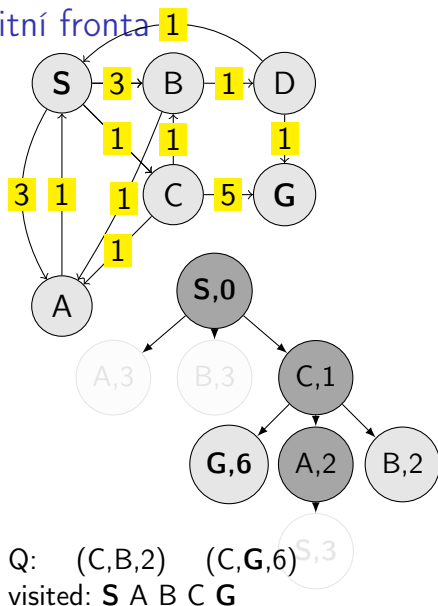
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

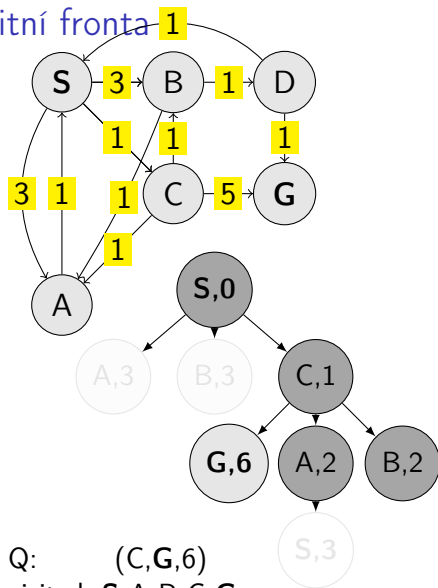
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```

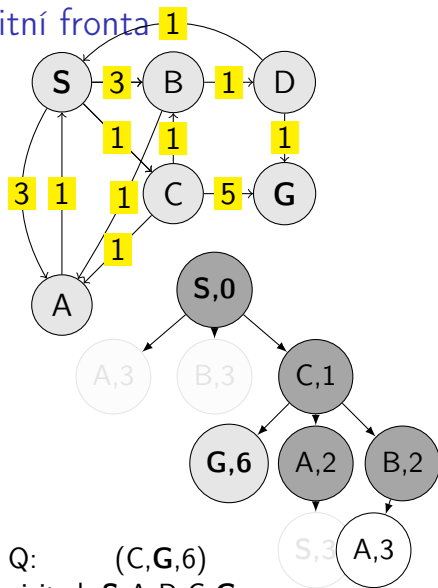


Q: (C,G,6)
visited: S A B C G

Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

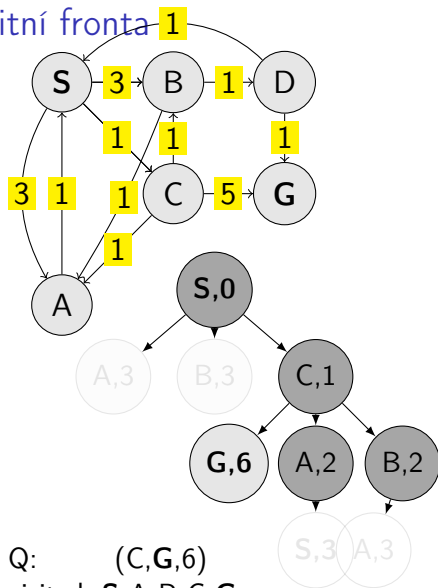
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

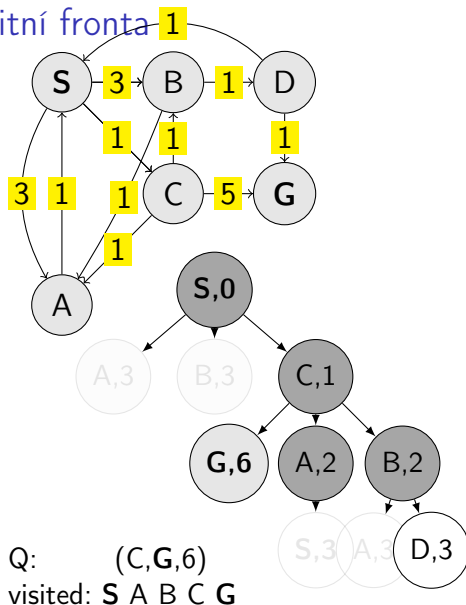
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

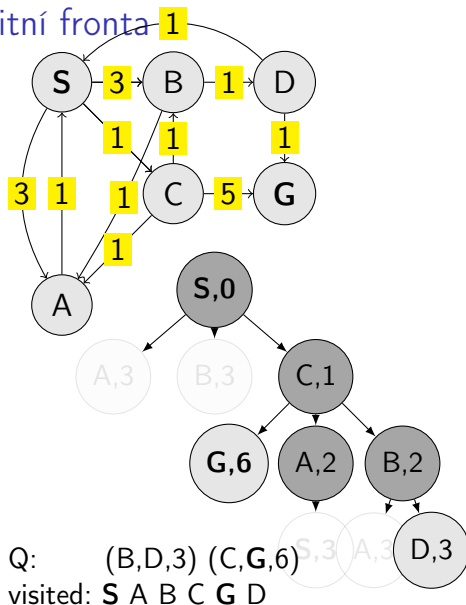
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

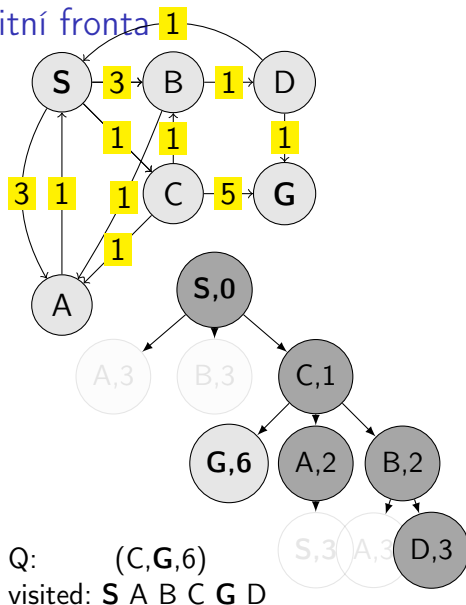
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

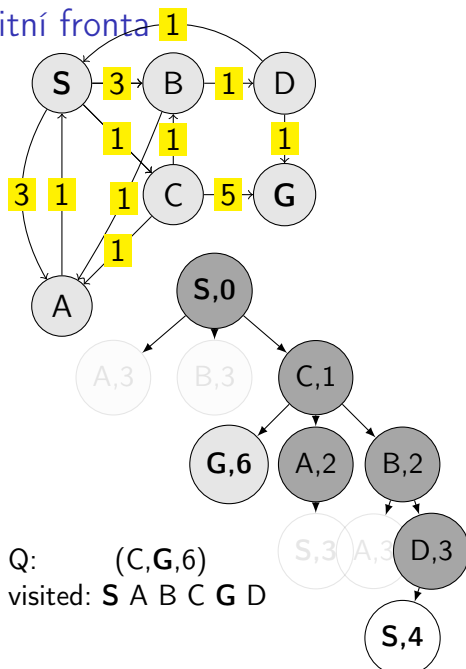
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

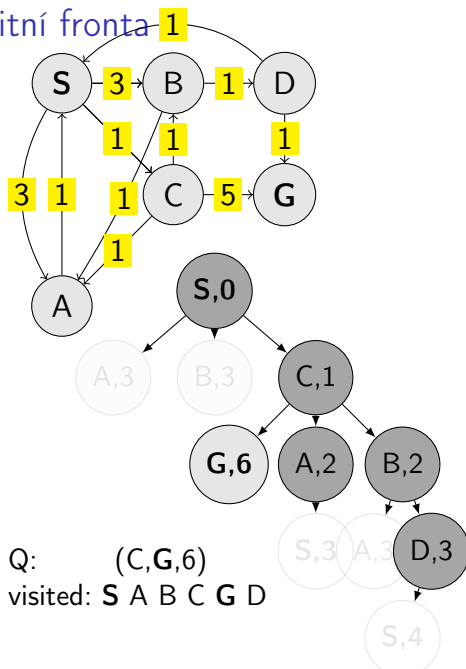
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

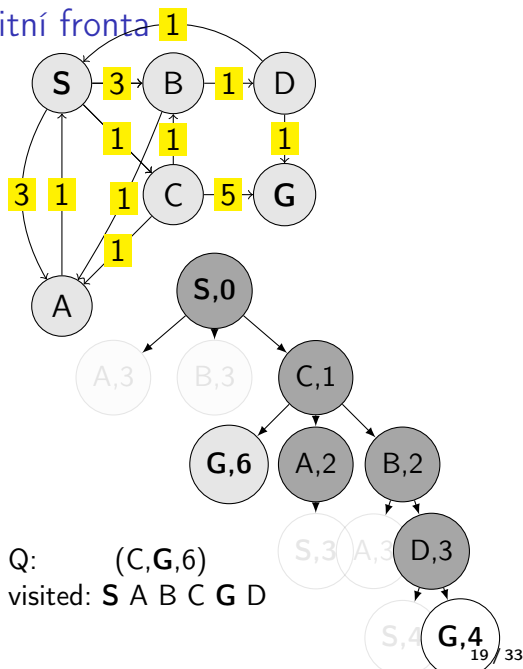
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

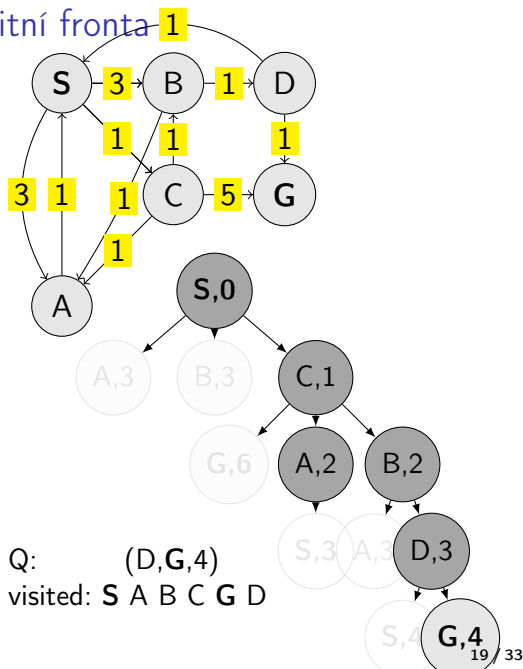
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

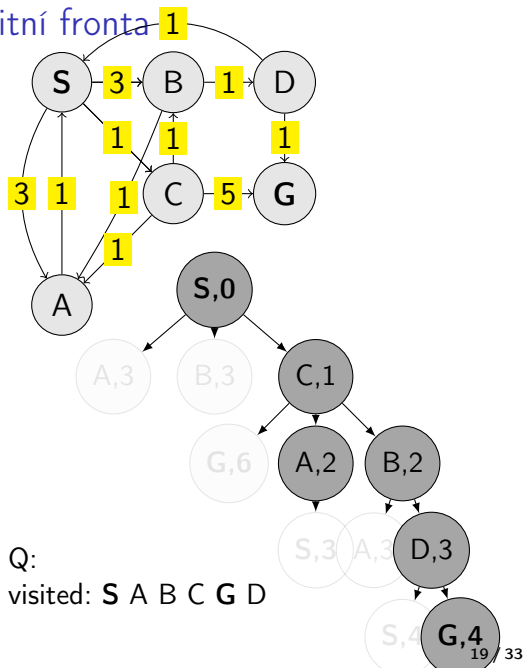
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



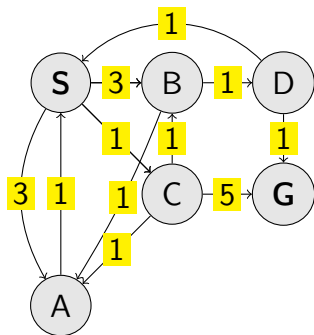
Uniform Cost Search (Dijkstra): Q je prioritní fronta

Prohledávání se stejnoměrnou cenou

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0) a označ s0 jako navštívený
3:   while Q není prázdná do
4:     p, s, _ ← Q.pop_first()
5:     parent[s] ← p
6:     if s ∈ SG then return Success
7:     for all a ∈ A(s) do
8:       s', c ← result(s, a)      ▷ c je cena
9:       if s' není navštívený then
10:        Označ s' jako navštívený
11:        Q.insert(s, s', cost_from_start)
12:       else
13:        Vyřeš duplikaci s' v Q
return Failure
```



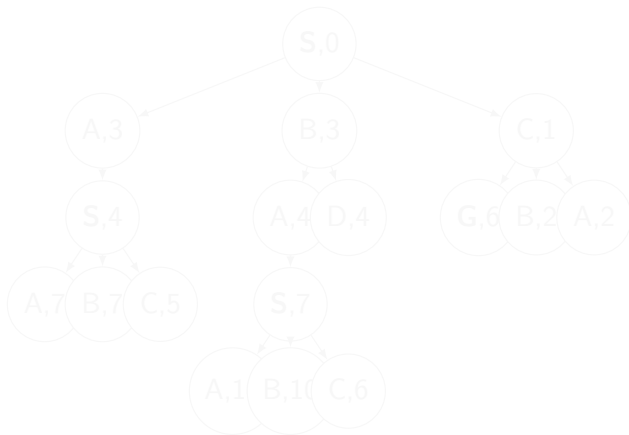
Vlastnosti algoritmu UCS



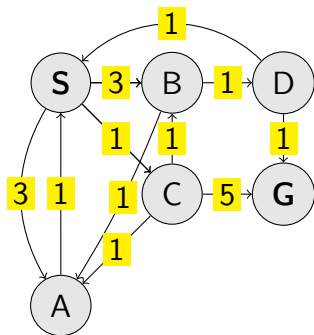
Úplný?

Optimální?

Časová a prostorová složitost?



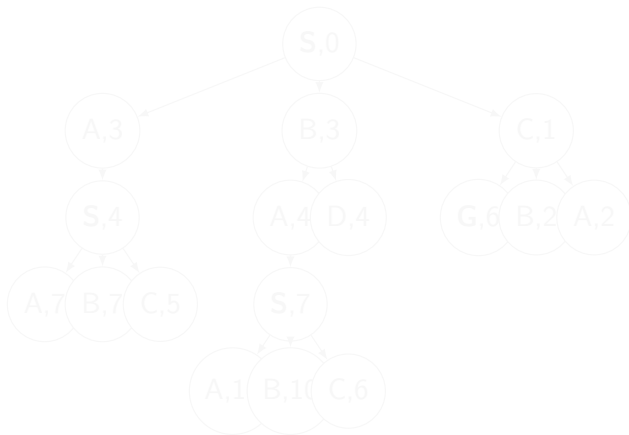
Vlastnosti algoritmu UCS



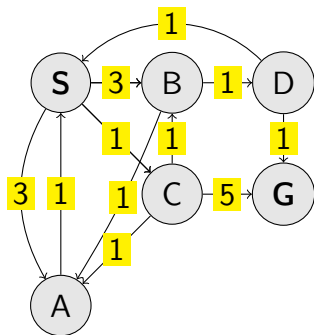
Úplný?

Optimální?

Časová a prostorová složitost?



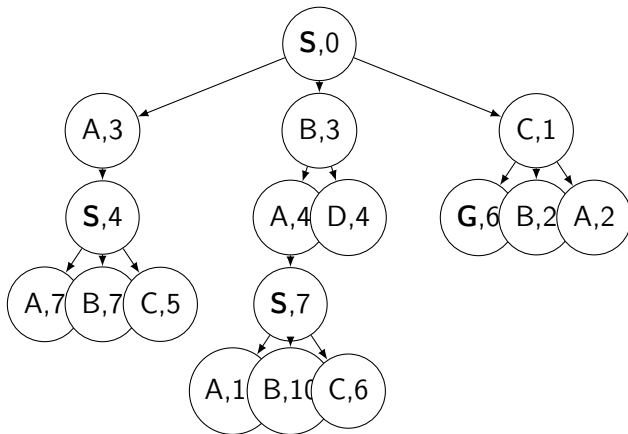
Vlastnosti algoritmu UCS



Úplný?

Optimální?

Časová a prostorová složitost?



Výběr uzlu pomocí argmin $f(n)$. Prohledávaný uzel: $n = (p, s, \text{cost_value})$

Výběr dalšího uzlu k prozkoumání (operace pop):

$$\text{node} \leftarrow \underset{n \in Q}{\operatorname{argmin}} f(n)$$

Jakou funkci $f(n)$ používají algoritmy DFS, BFS a UCS? Spárujte je!

- | | |
|--------|---------------------------------------|
| ▶ DFS: | ▶ $f(n) = n.\text{cost_from_start}$ |
| ▶ BFS: | ▶ $f(n) = n.\text{depth}$ |
| ▶ UCS: | ▶ $f(n) = -n.\text{depth}$ |

Výhoda: prioritní fronta jako univerzální datová struktura pro otevřené uzly (frontier)
(Přesto, zásobník(LIFO) a fronta (FIFO) jsou konceptuálně perfektní datové struktury pro DFS a BFS.)

Nevýhoda: všechny uvedené $f(n)$ odpovídají součtu cen ze startu do n , cost_from_start . Co nám schází?

Výběr uzlu pomocí $\text{argmin } f(n)$. Prohledávaný uzel: $n = (p, s, \text{cost_value})$

Výběr dalšího uzlu k prozkoumání (operace pop):

$$\text{node} \leftarrow \underset{n \in Q}{\text{argmin}} f(n)$$

Jakou funkci $f(n)$ používají algoritmy DFS, BFS a UCS? Spárujte je!

- | | |
|--------|---------------------------------------|
| ▶ DFS: | ▶ $f(n) = n.\text{cost_from_start}$ |
| ▶ BFS: | ▶ $f(n) = n.\text{depth}$ |
| ▶ UCS: | ▶ $f(n) = -n.\text{depth}$ |

Výhoda: prioritní fronta jako univerzální datová struktura pro otevřené uzly (frontier)
(Přesto, zásobník(LIFO) a fronta (FIFO) jsou konceptuálně perfektní datové struktury pro DFS a BFS.)

Nevýhoda: všechny uvedené $f(n)$ odpovídají součtu cen ze startu do n , cost_from_start . Co nám schází?

Výběr uzlu pomocí $\text{argmin } f(n)$. Prohledávaný uzel: $n = (p, s, \text{cost_value})$

Výběr dalšího uzlu k prozkoumání (operace pop):

$$\text{node} \leftarrow \underset{n \in Q}{\text{argmin}} f(n)$$

Jakou funkci $f(n)$ používají algoritmy DFS, BFS a UCS? Spárujte je!

- | | |
|--------|---------------------------------------|
| ▶ DFS: | ▶ $f(n) = n.\text{cost_from_start}$ |
| ▶ BFS: | ▶ $f(n) = n.\text{depth}$ |
| ▶ UCS: | ▶ $f(n) = -n.\text{depth}$ |

Výhoda: prioritní fronta jako univerzální datová struktura pro otevřené uzly (frontier)
(Přesto, zásobník(LIFO) a fronta (FIFO) jsou konceptuálně perfektní datové struktury pro DFS a BFS.)

Nevýhoda: všechny uvedené $f(n)$ odpovídají součtu cen ze startu do n , cost_from_start . Co nám schází?

Výběr uzlu pomocí $\text{argmin } f(n)$. Prohledávaný uzel: $n = (p, s, \text{cost_value})$

Výběr dalšího uzlu k prozkoumání (operace pop):

$$\text{node} \leftarrow \underset{n \in Q}{\text{argmin}} f(n)$$

Jakou funkci $f(n)$ používají algoritmy DFS, BFS a UCS? Spárujte je!

- | | |
|--------|---------------------------------------|
| ▶ DFS: | ▶ $f(n) = n.\text{cost_from_start}$ |
| ▶ BFS: | ▶ $f(n) = n.\text{depth}$ |
| ▶ UCS: | ▶ $f(n) = -n.\text{depth}$ |

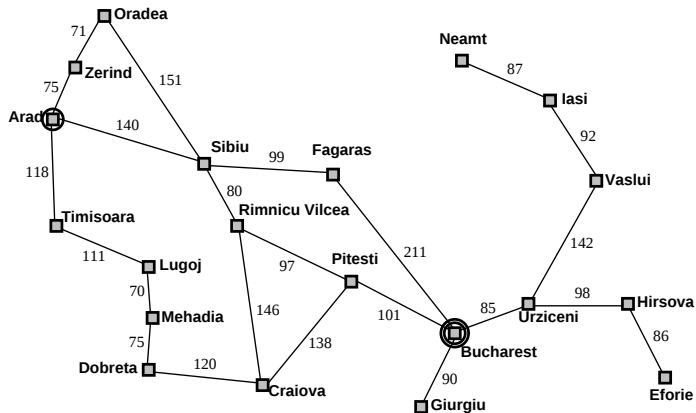
Výhoda: prioritní fronta jako univerzální datová struktura pro otevřené uzly (frontier)
(Přesto, zásobník(LIFO) a fronta (FIFO) jsou konceptuálně perfektní datové struktury pro DFS a BFS.)

Nevýhoda: všechny uvedené $f(n)$ odpovídají součtu cen ze startu do n , `cost_from_start` . Co nám schází?

Jak blízko jsme k cíli, **cost-to-go** ? – Heuristika

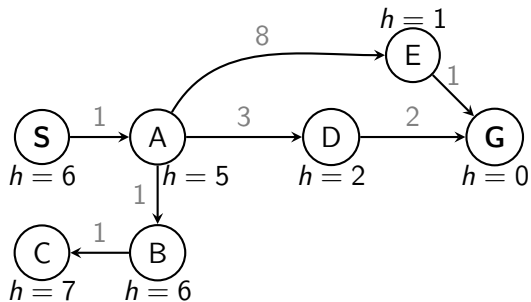
- ▶ Funkce, která odhaduje, jak blízko je *stav* od nejbližšího cíle.
(„Jakou cenu musíme ještě zaplatit, abychom se dostali do některého cíle?“)
- ▶ Navržena pro konkrétní problém.
- ▶ $h(s)$ – je to funkce stavu (její hodnota se stává atributem uzlu v prohledávacím stromu)
- ▶ Aplikována na *uzel* n , $h(n)$ je heuristická hodnota stavu uloženého v uzlu n .

Příklad heuristiky



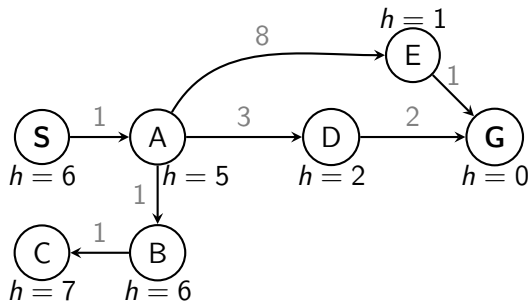
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Hladový alg., vyber $n^* = \operatorname{argmin}_{n \in Q} h(n)$



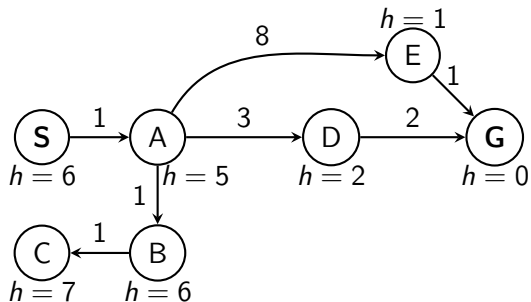
Co je špatné (a dobré) na hladovém algoritmu?

Hladový alg., vyber $n^* = \operatorname{argmin}_{n \in Q} h(n)$



Co je špatné (a dobré) na hladovém algoritmu?

A* kombinuje UCS a hladový algoritmus

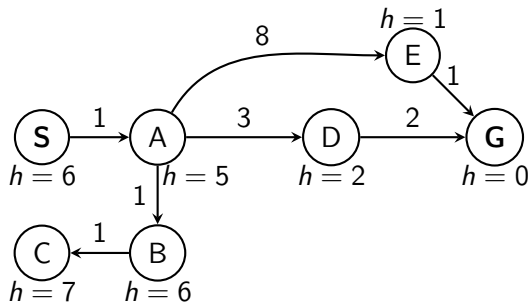


UCS řadí uzly podle $g(n)$ (cost_from_start)

Hladový alg. řadí uzly podle $h(n)$ (heuristika, cost_to_go)

A* řadí uzly podle: $f(n) = g(n) + h(n)$

A* kombinuje UCS a hladový algoritmus

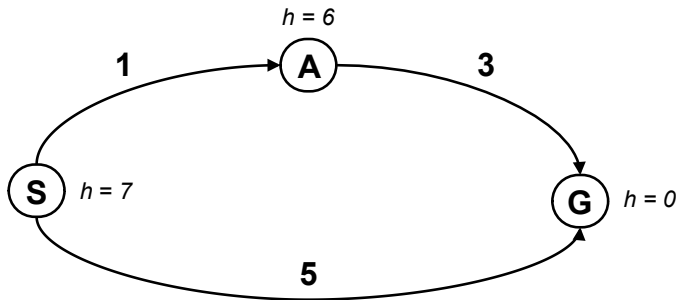


UCS řadí uzly podle $g(n)$ (cost_from_start)

Hladový alg. řadí uzly podle $h(n)$ (heuristika, cost_to_go)

A* řadí uzly podle: $f(n) = g(n) + h(n)$

Je A^* optimální?

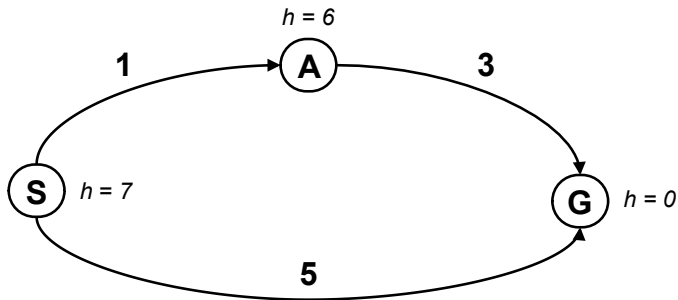


2

Co je tu za problém?

²Příklad grafu: Dan Klein a Pieter Abbeel

Je A^* optimální?



2

Co je tu za problém?

²Příklad grafu: Dan Klein a Pieter Abbeel

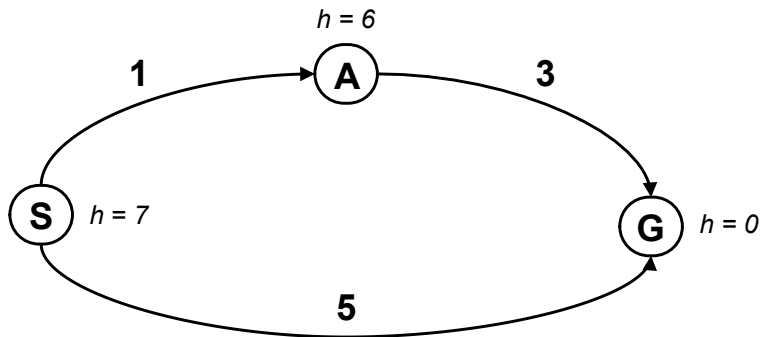
Jaká je správná hodnota $h(A)$?

A: $0 \leq h(A) \leq 4$

B: $h(A) \leq 3$

C: $0 \leq h(A) \leq 3$

D: $0 \leq h(A)$



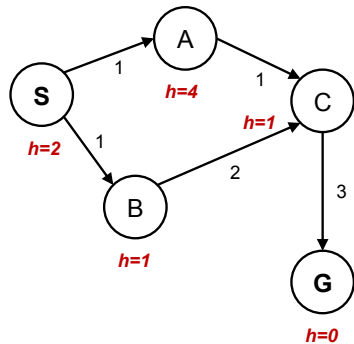
Přípustná heuristika

Heuristická funkce h je přípustná, pokud:

$$\begin{aligned}\forall \text{Goal} : h(n) &\leq \text{cost}(n.\text{state}, \text{Goal}) \\ h(\text{Goal}) &= 0\end{aligned}$$

Heuristika: Stačí přípustnost?

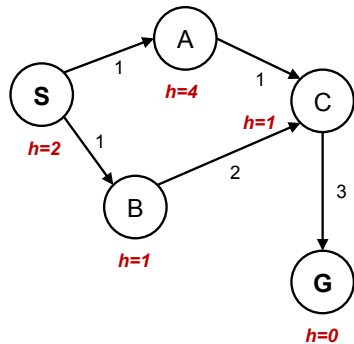
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



S,2

Heuristika: Stačí přípustnost?

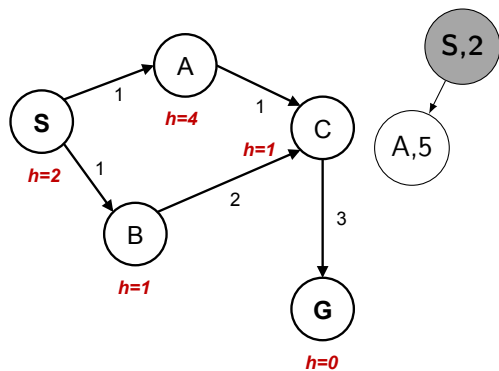
```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



S,2

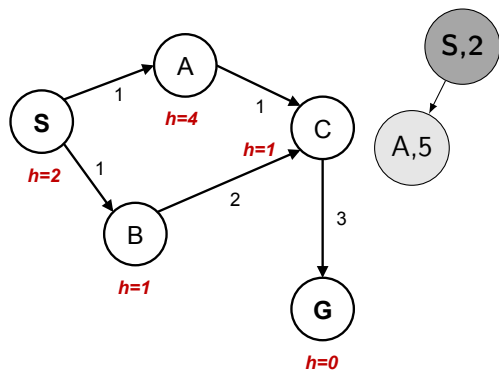
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



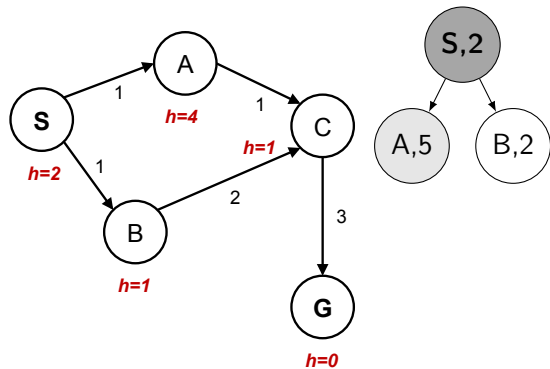
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



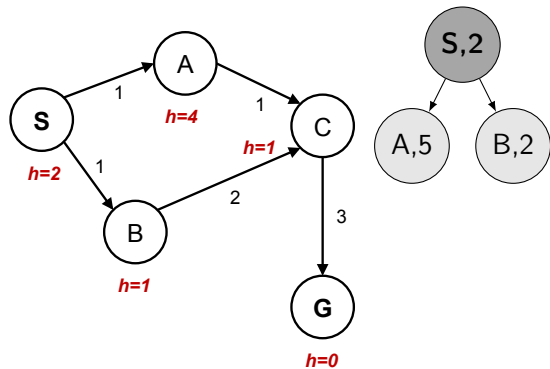
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



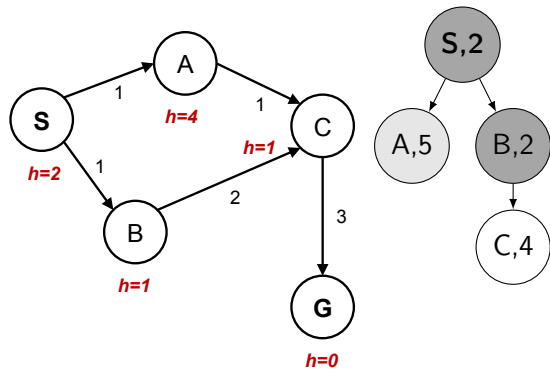
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



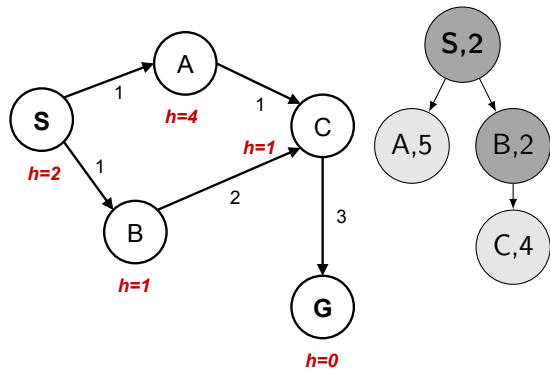
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



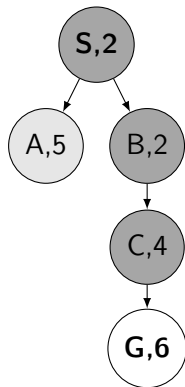
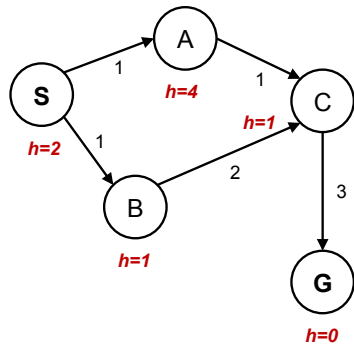
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



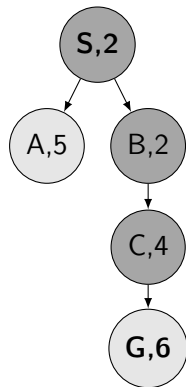
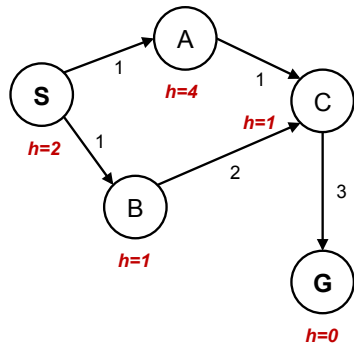
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



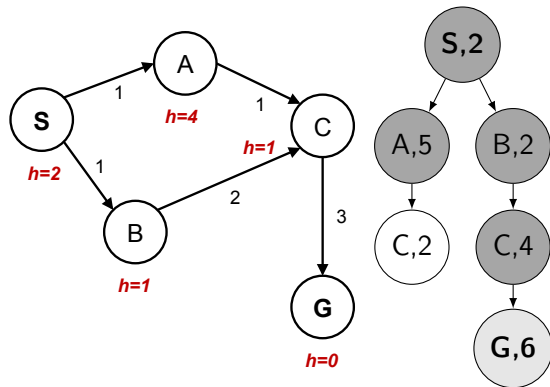
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



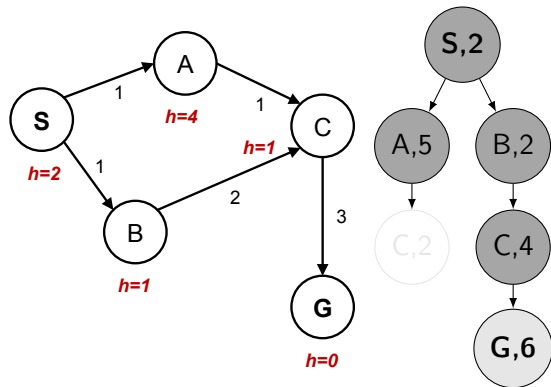
Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



Heuristika: Stačí přípustnost?

```
1: function FORWARD_SEARCH
2:   Q.insert(_, s0, 0, h(s0))
3:   Označ s0 jako navštívený
4:   while Q není prázdná do
5:     p, s, g, _ ← Q.pop()
6:     parent[s] ← p
7:     if s ∈ SG then return Success
8:     for all a ∈ A(s) do
9:       s', c ← result(s, a)
10:      if s' není navštívený then
11:        Označ s' jako navštívený
12:        Q.insert(s, s', g + c, g + c + h(s'))
13:      else
14:        Vyřeš duplikaci s' v Q
return Failure
```



Jaká je správná hodnota $h(A)$?

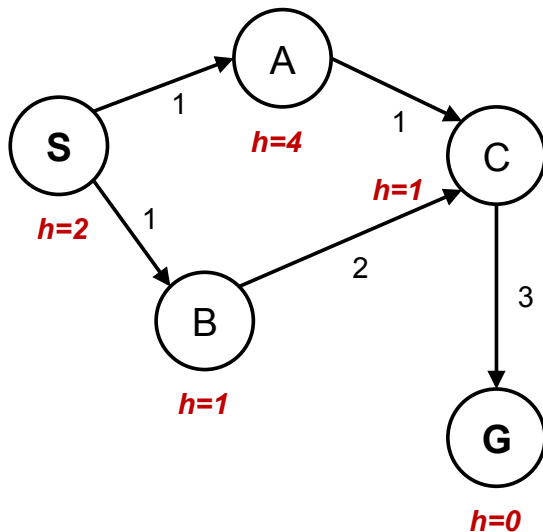
Předpokládejte, že ostatní $h(s)$ se nezmění.

A: $h(A) = 1$

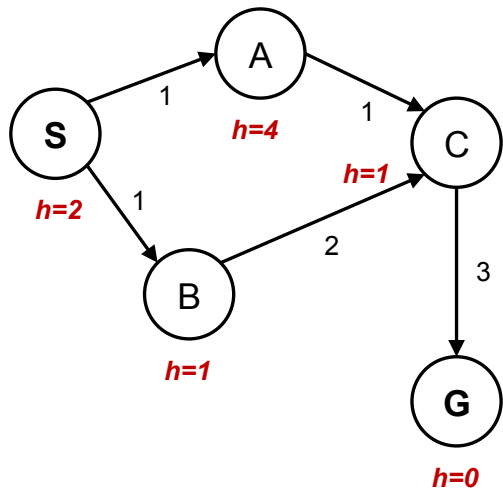
B: $h(A) = 2$

C: $1 \leq h(A) \leq 2$

D: $0 \leq h(A) \leq 1$



Konzistentní heuristika



Připustná h :

$h(A) \leq$ skutečná nejnižší cena $A \rightarrow G$

$h(\text{Cíl}) = 0$

Konzistentní h :

$h(\text{Cíl}) = 0$ pro všechny cíle

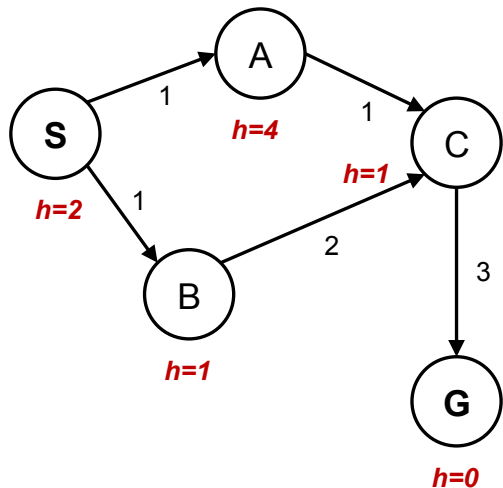
$h(A) - h(C) \leq$ skutečná nejnižší $A \rightarrow C$

obecně:

$h(p) - h(s) \leq$ skutečná nejnižší cena $p \rightarrow s$ pro každý pár rodiče p a jeho následníka s

$f(n) = g(n) + h(n)$ se podél cesty ze startu do cíle nikdy nesnižuje!

Konzistentní heuristika



Připustná h :

$h(A) \leq$ skutečná nejnižší cena $A \rightarrow G$

$h(\text{Cíl}) = 0$

Konzistentní h :

$h(\text{Cíl}) = 0$ pro všechny cíle

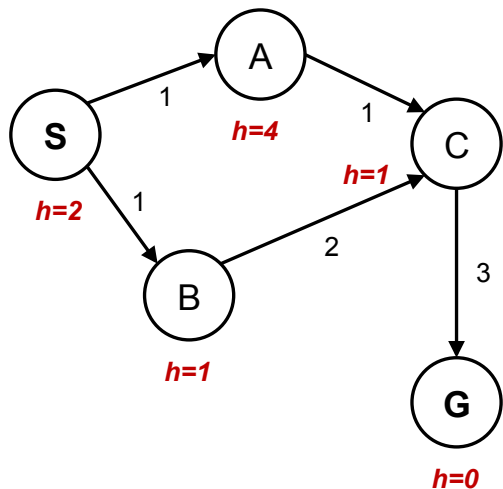
$h(A) - h(C) \leq$ skutečná nejnižší $A \rightarrow C$

obecně:

$h(p) - h(s) \leq$ skutečná nejnižší cena $p \rightarrow s$ pro každý pár rodiče p a jeho následníka s

$f(n) = g(n) + h(n)$ se podél cesty ze startu do cíle nikdy nesnižuje!

Konzistentní heuristika



Připustná h :

$h(A) \leq$ skutečná nejnižší cena $A \rightarrow G$

$h(\text{Cíl}) = 0$

Konzistentní h :

$h(\text{Cíl}) = 0$ pro všechny cíle

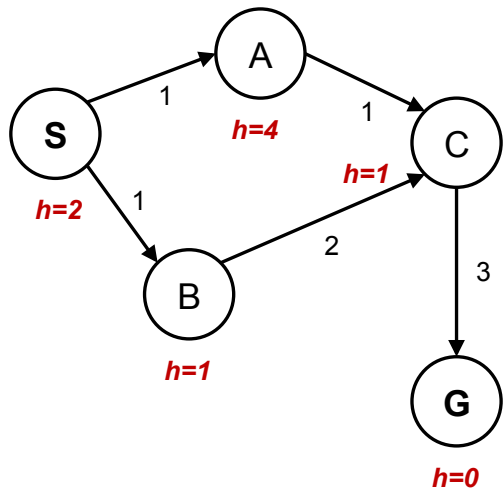
$h(A) - h(C) \leq$ skutečná nejnižší $A \rightarrow C$

obecně:

$h(p) - h(s) \leq$ skutečná nejnižší cena $p \rightarrow s$ pro každý pár rodiče p a jeho následníka s

$f(n) = g(n) + h(n)$ se podél cesty ze startu do cíle nikdy nesnižuje!

Konzistentní heuristika



Připustná h :

$h(A) \leq$ skutečná nejnižší cena $A \rightarrow G$

$h(\text{Cíl}) = 0$

Konzistentní h :

$h(\text{Cíl}) = 0$ pro všechny cíle

$h(A) - h(C) \leq$ skutečná nejnižší $A \rightarrow C$

obecně:

$h(p) - h(s) \leq$ skutečná nejnižší cena $p \rightarrow s$ pro každý pár rodiče p a jeho následníka s

$f(n) = g(n) + h(n)$ se podél cesty ze startu do cíle nikdy nesnižuje!

- ▶ Graf stavového prostoru vs. strom prohledávání
- ▶ Strategie prohledávání - vlastnosti, složitosti
- ▶ Ohodnocení stavů - `cost_from_start` a `cost_to_go`
- ▶ Efektivita – využití heuristických odhadů hodnot `cost_to_go`
- ▶ Ne všechny heuristiky jsou stejně dobré (přípustnost, konzistence, informovanost)

Odkazy, další čtení

Některé obrázky převzaty z [2]. Kapitola 2 v [1] poskytuje kompaktní/zhuštěný úvod do vyhledávacích algoritmů.

[1] Steven M. LaValle.

Planning Algorithms.

Cambridge, 1st edition, 2006.

Online version available at: <http://planning.cs.uiuc.edu>.

[2] Stuart Russell and Peter Norvig.

Artificial Intelligence: A Modern Approach.

Prentice Hall, 4th edition, 2021.

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