TEORIE GRAFŮ – PROHLEDÁVÁNÍ

Kurz: **Datové struktury a algoritmy / Teoretická informatika**

Lektor: Doc. Ing. Radim Burget, Ph.D.

Autor: Doc. Ing. Radim Burget, Ph.D.

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Vytvoření této videopřednášky bylo podpořeno projektem č. CZ.1.07/2.2.00/28.0098 Evropského sociálního fondu a státním rozpočtem České republiky.

Cíl přednášky

I. Hledání nejkratší cesty v grafu

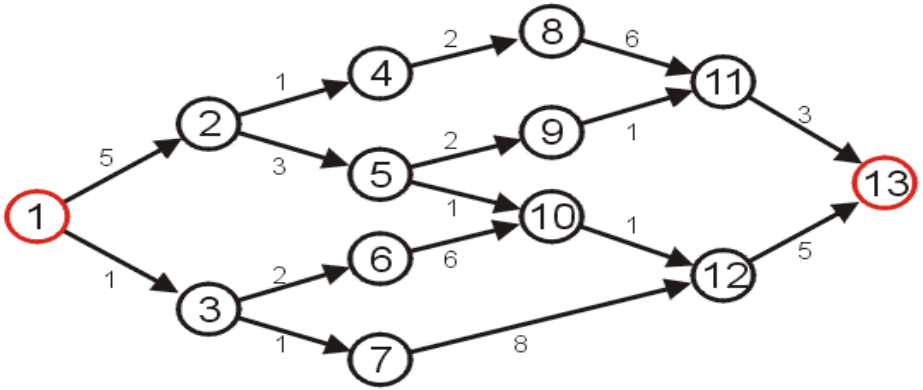
II. Algoritmy průchodu grafem
- Slepé metody
- Informované metody

III. Příklady použití

Nejkratší cesta v grafu

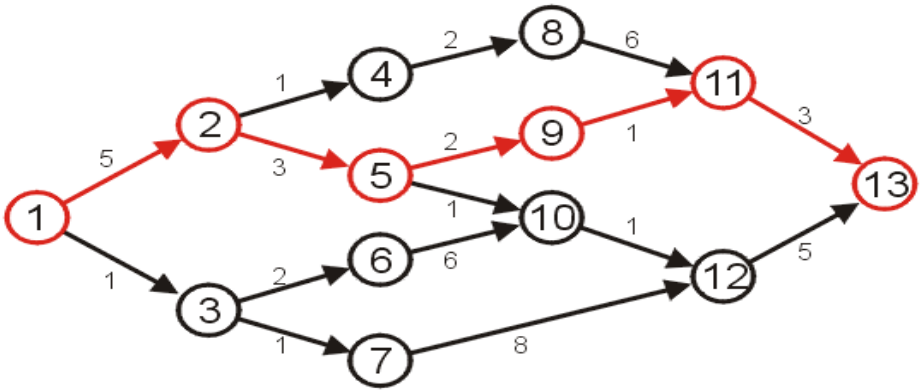
Nejkratší cesta v grafu

- Mějme graf zadaný níže a předpokládejme, že chceme nalézt nejkratší cestu z vrcholu 1 do vrcholu 13



Nejkratší cesta grafem

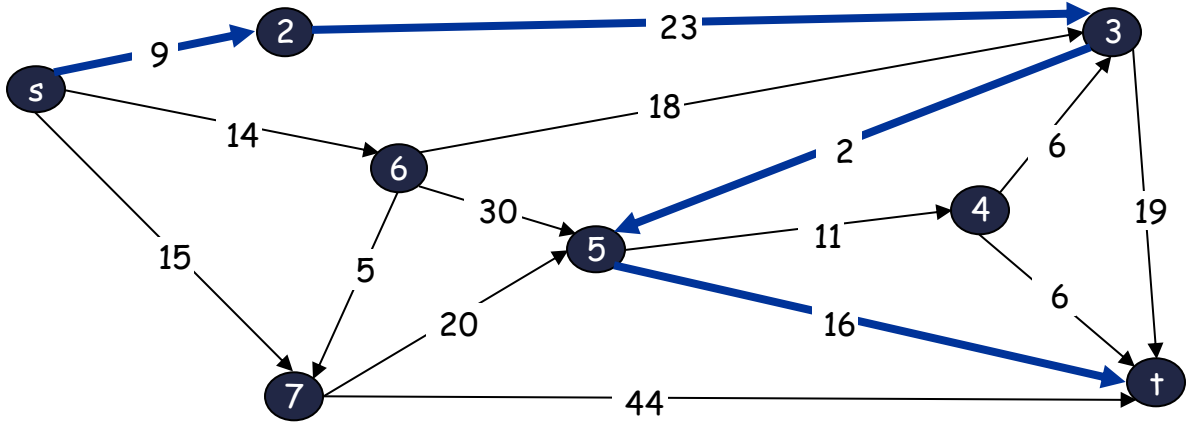
- Po nějaké době prohledávání jsme schopni určit, že nejkratší cesta grafem je následující (s celkovou délkou 14).



- Existují i jiné cesty grafem, jsou ale delší
- Všechny možnosti = **stavový prostor**

Nejkratší cesta grafem

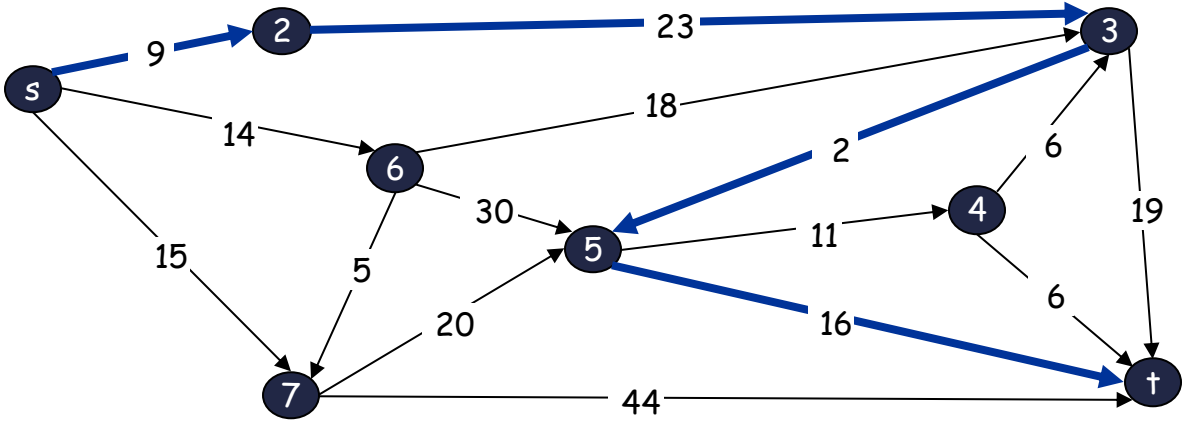
- Jak zobecňujeme problém nalezení nejkratší cesty v grafu?
- Vstup:
  - Vážený orientovaný graf G = (V, E, w).
  - Počáteční uzel *s*, cíl *t*.
- Výstup: jednotlivé uzly cesty z s do t.
- Nalezněte nejkratší orientovanou cestu *s*-*t*.



Cena cesty s-2-3-5-t
= 9 + 23 + 2 + 16
= 50.

K zamyšlení

- Kolik existuje možných cest z *s* do *t*?
- Můžeme ignorovat cykly? Pokud ano, jak?
- Nějaké návrhy jak redukovat množství variant?



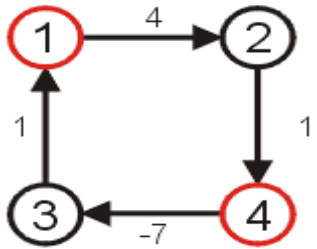
Závěry

- Pokud nejkratší cesta obsahuje vrchol *v*, potom:
  - Bude vrchol *v* obsahovat jen jedno spojení, cykly by přidaly délku a tedy i cenu cesty.
  - Cesta z *s* do *v* musí být současně nejkratší do *v* z *s*.
  - Cesta z *v* do *t* musí být nejkratší cesta do *t* z *v*.
- Z toho plyne, pokud jsme schopni určit nejkratší cestu všech ostatních vrcholů, které jsou předchůdci do cílového vrcholu, můžeme snadno spočíst nejkratší cestu.
  - Problém tedy lze rozložit na skupinu podproblémů a výpočet provádět v jednotlivých krocích.

Negativní cykly

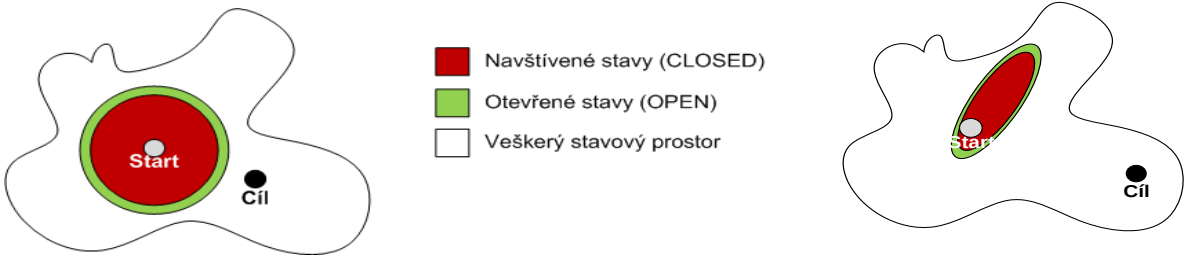
- Ohodnocení hran může být obecně zobrazením do množiny celých (reálných?) čísel.
- V takovém případě ale každý průchod hranou snižuje výslednou cenu, snižuje celkovou cenu grafu.
- Tj. cyklus v grafu ve výsledku snižuje celkovou cenu průchodu grafem

- Proto pro takový graf, kde jsou záporná ohodnocení hran, je cena cesty nedefinovaná.
- Uvažujeme nejkratší cenu cesty z 1 do 4...
- Uvažujeme pouze ne-negativní váhy hran.

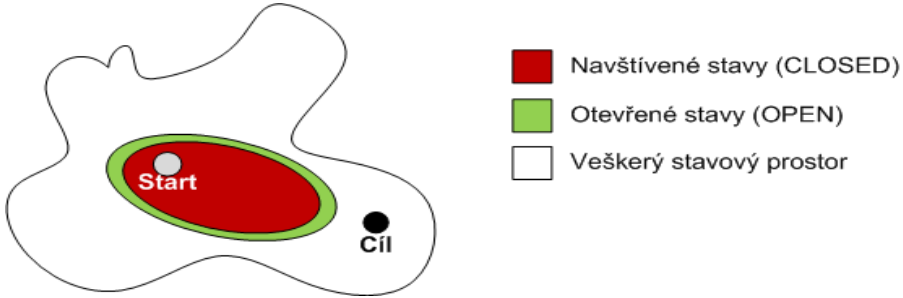


Členění algoritmů prohledávání grafů

- **Slepé prohledávání** – vyhledávají „náhodně – bez přemýšlení“



- **Informované metody** – snaží se odhadnout, kudy pokračovat ve vyhledávání, aby co nejdříve nalezli cíl

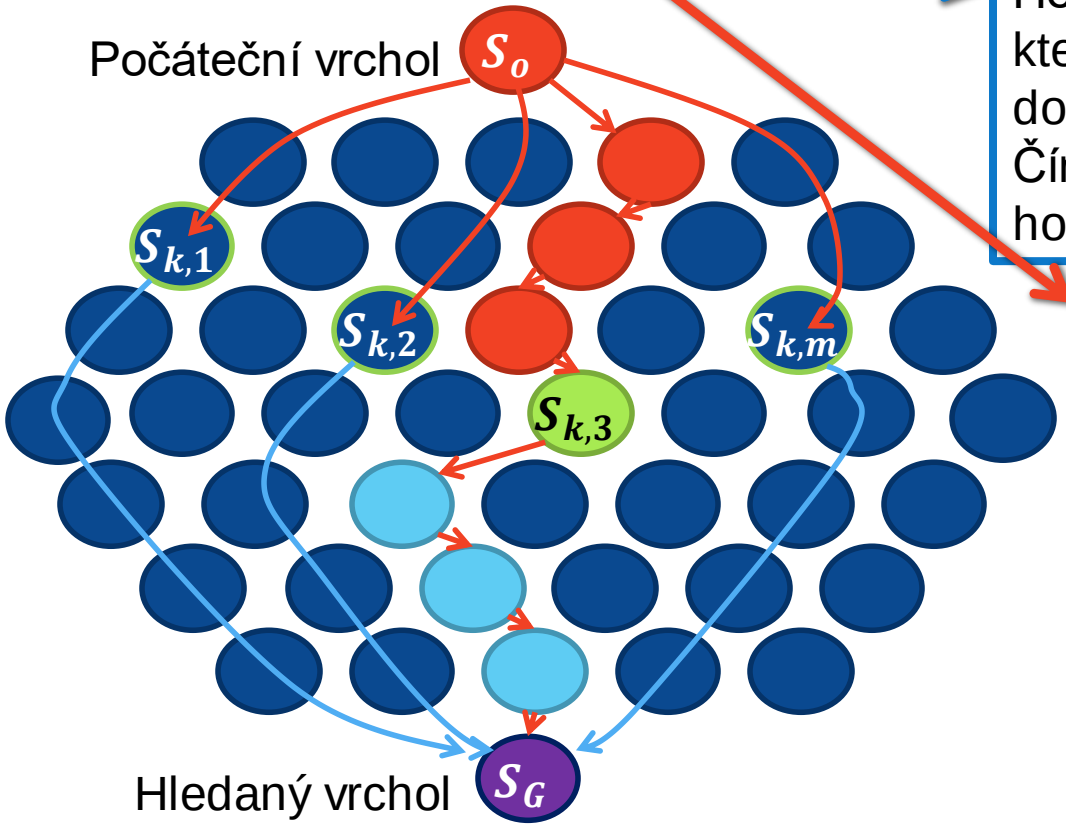


Prohledávací strategie

- Strategie vyhledávání se odvíjí od způsobu výběru pořadí expanze uzlu
- Strategie jsou hodnoceny podle následujících kritérií:
  - **Úplnost**: vždy najde řešení, pokud existuje?
  - **Optimálnost**: vždy najde řešení s nejnižšími náklady?
  - **Časová složitost**: počet generovaných uzlů
  - **Prostorová složitost**: maximální počet uzlů v paměti
- Časová a prostorová složitost se měří pomocí
  - *b*: maximální stupeň větvení stromu prohledávání
  - *d*: hloubka optimálního stromu řešení
  - *m*: maximální délka jakékoli cesty ve stavovém prostoru (může být i nekonečná)

Metody hledání cesty v grafu - zobecnění

$$F(S_k) = g(S_k) + h(S_k) = g(S_k - S_0) + h(S_G - S_k)$$



Heuristická funkce předpovídá, kterou cestou se nejrychleji dostanu ze stavu S_k do stavu S_G. Čím blíže je S_k k řešení S_G, tím je hodnota funkce $h(S_k)$ nižší.

Funkce ceny cesty, která byla vynaložena z počátečního vrcholu grafu S_0 do současného vrcholu grafu S_k.
Čím dále je S_k od počátku S_0, tím je hodnota funkce $g(S_k)$ vyšší.

Slepé prohledávání

- Nepoužívají heuristiku, tj. $h(S_k) = 0$

- Prohledávání do šířky (BFS)
- Prohledávání do hloubky (DFS)
- Prohledávání do hloubky s omezenou hloubkou (DLS)
- Obousměrné vyhledávání
- Iterativní prohledávání do hloubky s omezenou hloubkou, IDLS
- Dijkstrův algoritmus (Uniform-cost search, UCS)

a další…

$$F(S_k) = g(S_k) + h(S_k) = g(S_k - S_0) + h(S_G - S_k)$$

Informované prohledávání

- Best First Search (BestFS či Greedy Search)

$$F(S_k) = g(S_k) + h(S_k) = g(S_k - S_0) + h(S_G - S_k)$$

- Algoritmus A* (čti: A-star)

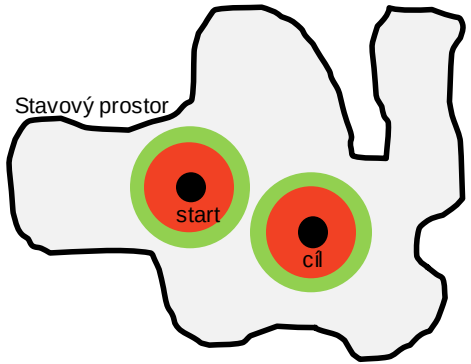
$$F(S_k) = g(S_k) + h(S_k) = g(S_k - S_0) + h(S_G - S_k)$$

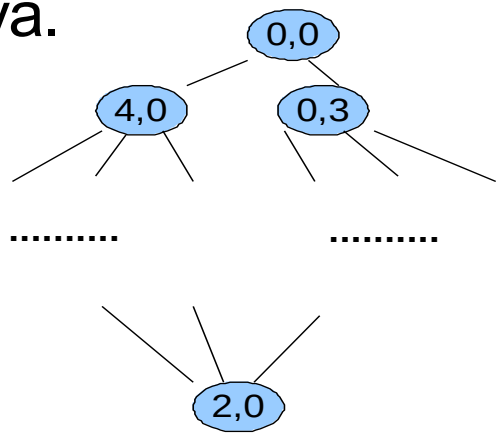
a další…

Metody slepého prohledávání

Obousměrné vyhledávání

- Prochází stavový prostor od počátku a současně i od požadovaného stavu a hledá se cesta mezi těmito dvěma stavy.
- Co algoritmus velice zatěžuje, je nutnost kontroly ekvivalence stavů v jednotlivých krocích. Díky tomu je časová složitost algoritmu ne příliš příznivá.
- V literatuře se uvádí, ale nepoužívá se





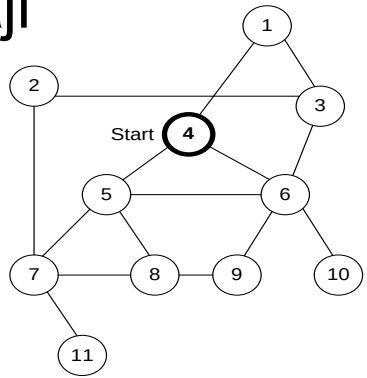
Obousměrné vyhledávání – vlastnosti

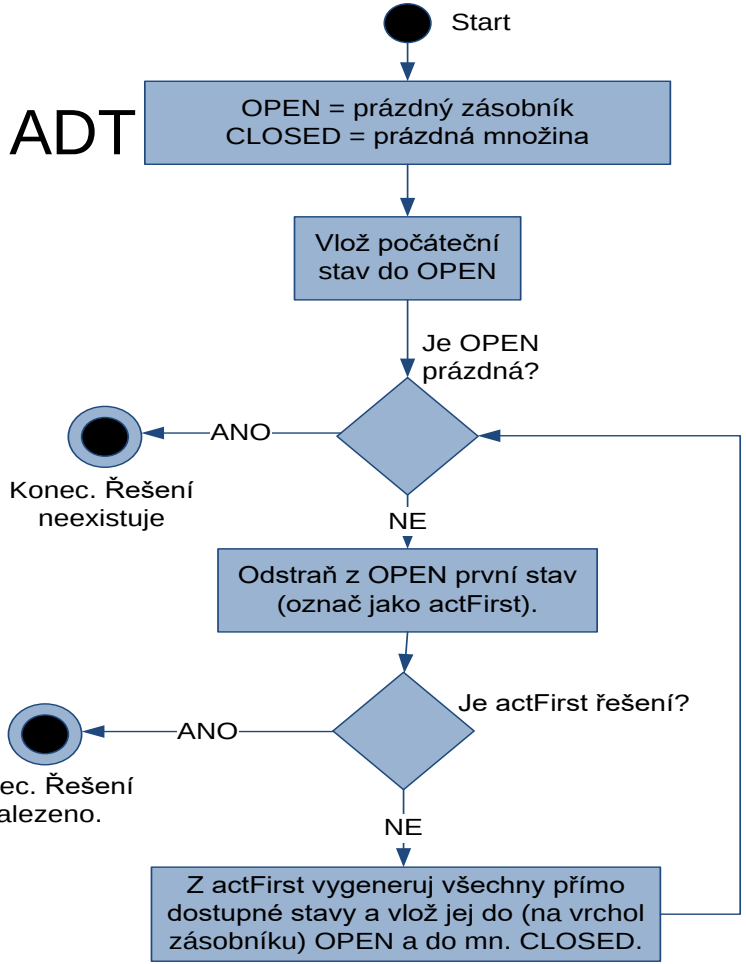
- **Úplnost** – algoritmus je úplný.
- **Optimálnost** – algoritmus je optimální, je nalezeno takové řešení, ke kterému vede nejmenší počet kroků.
- **Prostorová složitost** – BFS/2.
- **Časová složitost** – velký problém je porovnávání množin OPEN z obou směrů. V takovém případě se dostáváme na $O(m*2^n)$, kde *m* je počet prvků z množiny 1 a *n* je počet prvků z množiny 2.

Prohledávání do hloubky

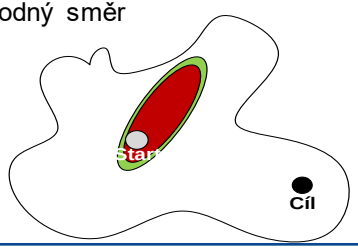
Prohledávání do hloubky

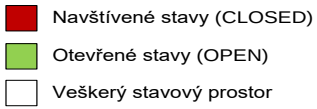
- Vychází z algoritmu známého z ADT
- Je třeba eliminovat cykly
- Chodí „po okraji“





Slepé prohledávání
= náhodný směr



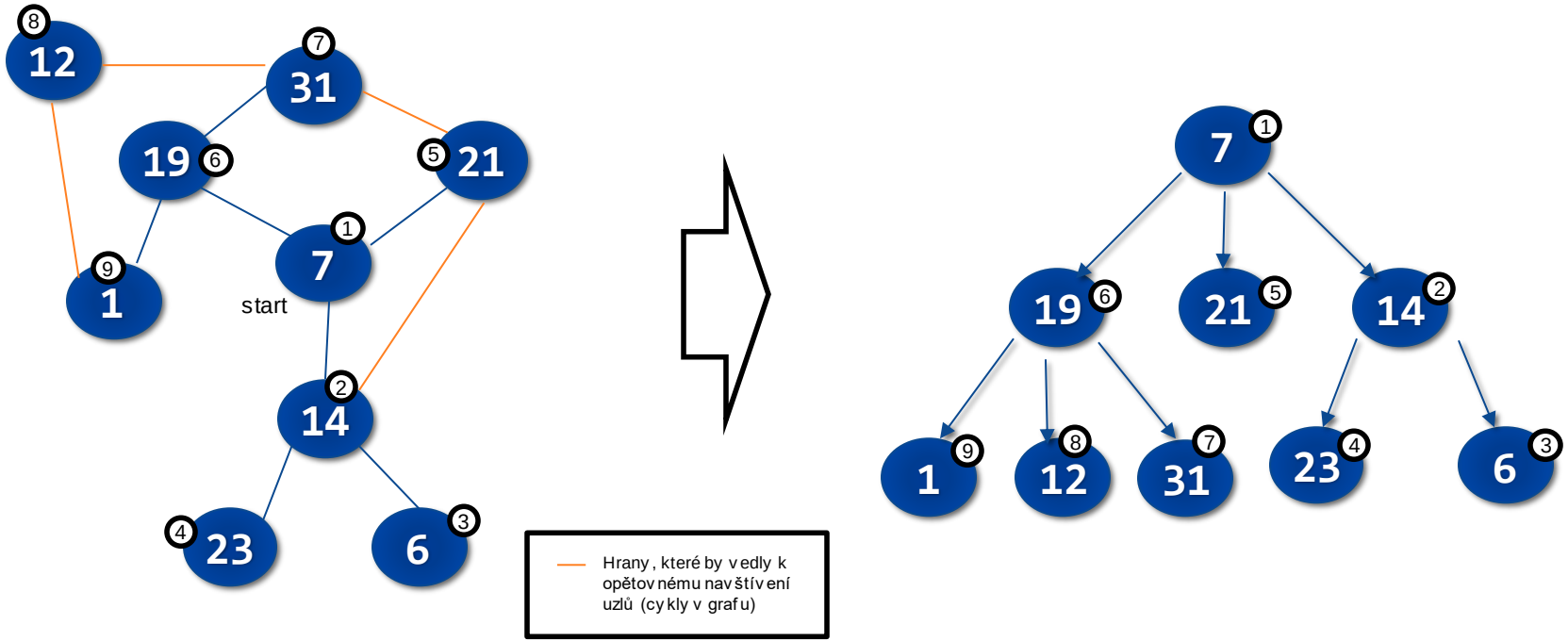


Prohledávání do hloubky – vlastnosti

- **Úplnost** – algoritmus je úplný (tj. vždy nalezne řešení, pokud existuje).
- **Optimálnost** – algoritmus DLS není optimální, upřednostňuje jednu větev oproti druhé. Z toho důvodu je velice pravděpodobné, že bude nalezeno řešení, ke kterému je velký počet kroků i když existuje řešení bližší.
- **Prostorová složitost** – o mnoho lepší než je BFS – O(*bm*), kde d je hloubka a b je stupeň stromu.
- **Časová složitost** – obdobně jako BFS. O(b^d)

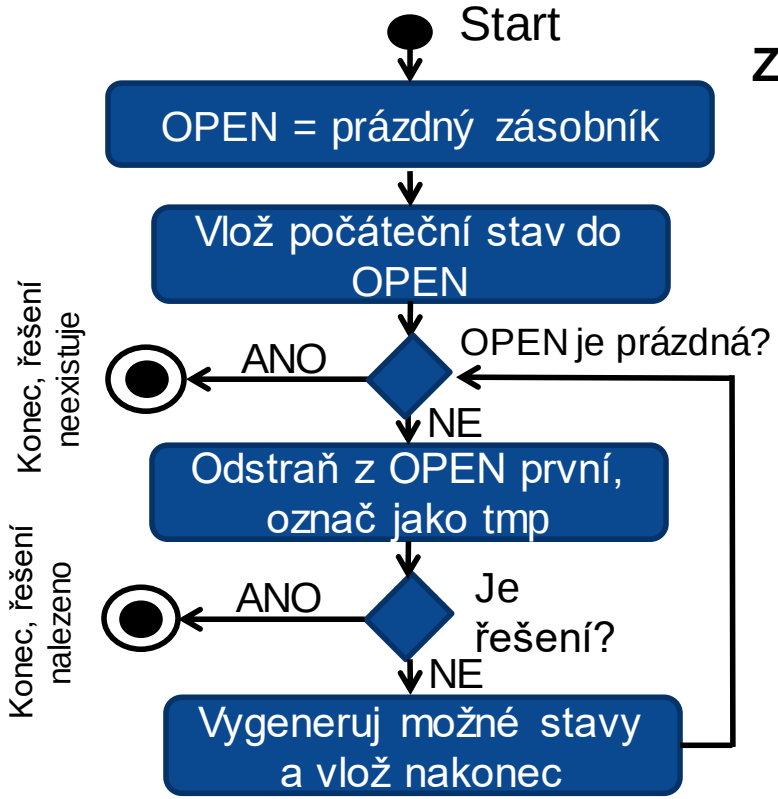
Procházení grafem do hloubky

- Průchodem grafu při odstranění cyklů získáváme strom
- Do hloubky se snaží (náhodným směrem) maximálně zanořit a vynořuje se, jakmile již není se zanořovat kam

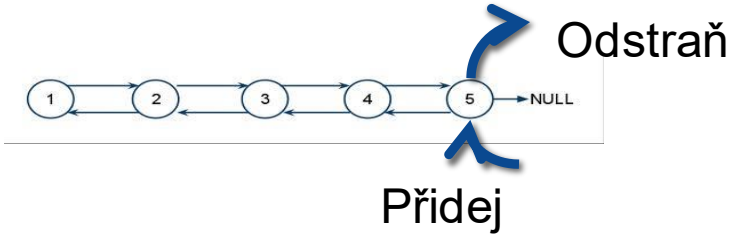


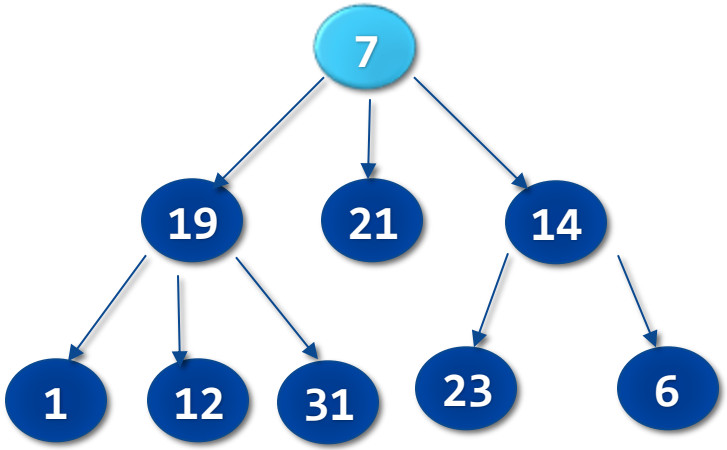
Prohledávání do hloubky – animace

Jak se dostanu ze stavu 7 do stavu **6 s nejmenším počtem tahů** ?



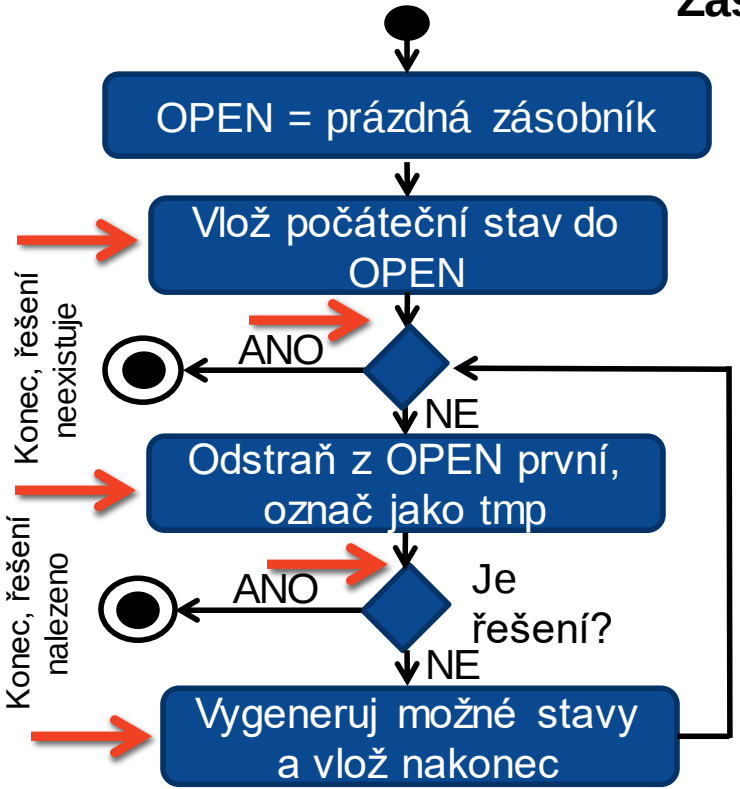
Zásobník OPEN:



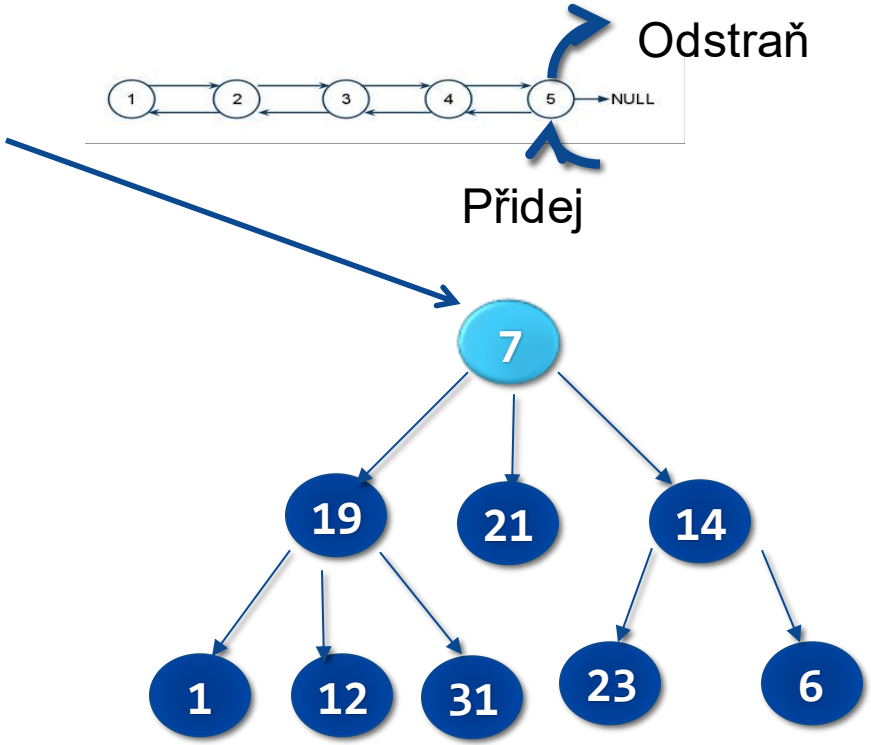


Prohledávání do hloubky – animace

Zásobník OPEN: ~~X~~ 19 21 14

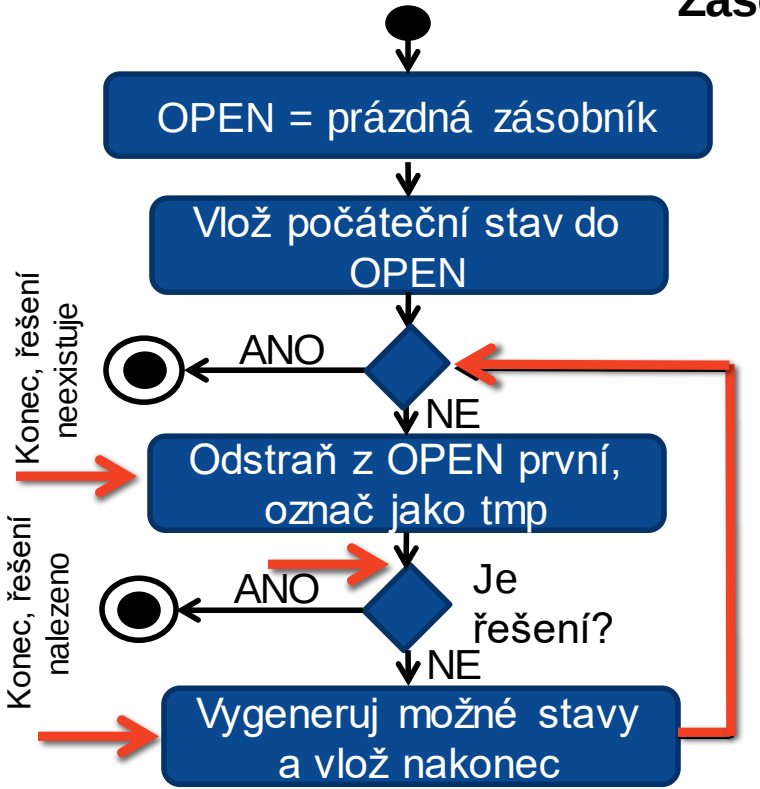


tmp:

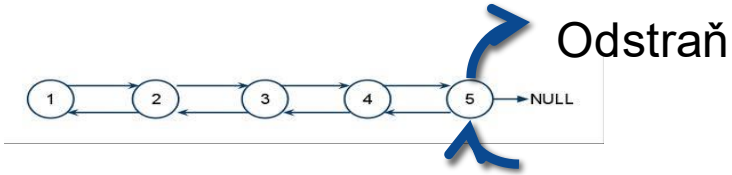


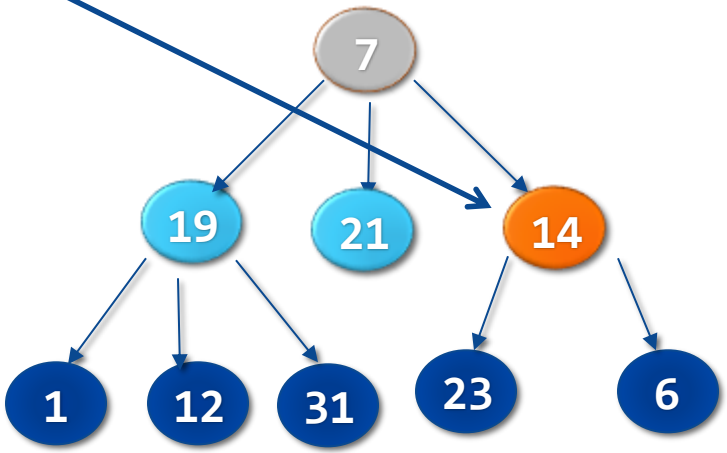
Prohledávání do hloubky – animace

Zásobník OPEN: ~~7~~ 19 21 ~~14~~ 23 6



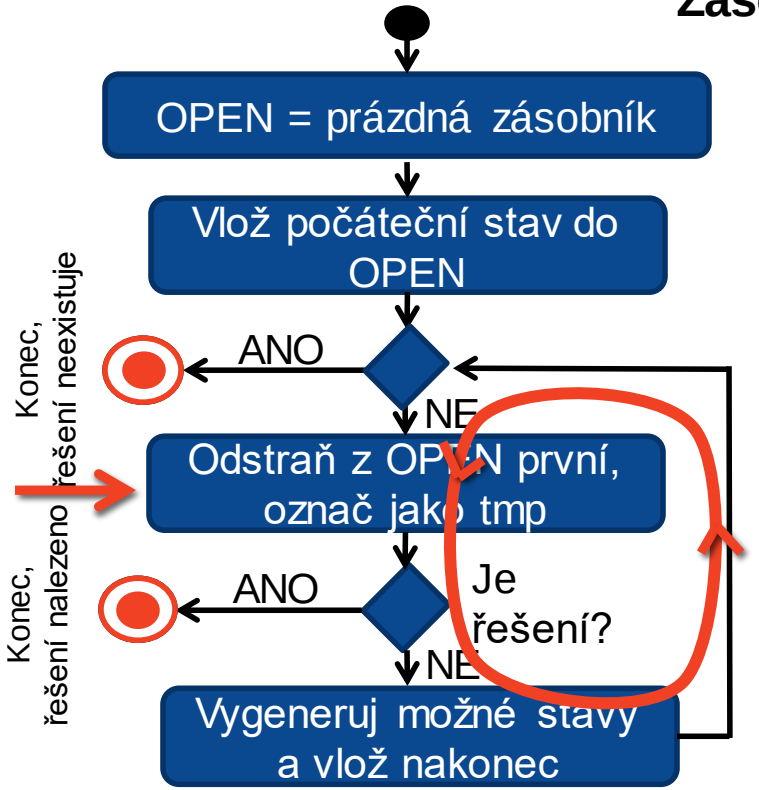
tmp:



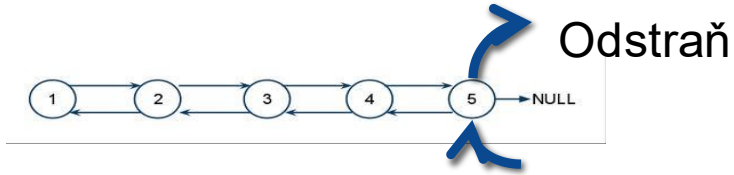


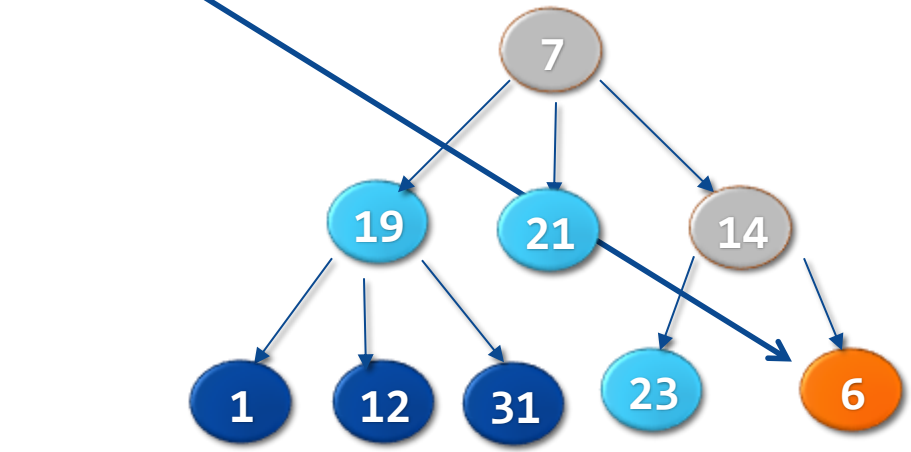
Prohledávání do hloubky – animace

Zásobník OPEN: ~~7~~ 19 21 ~~14~~ 23 ~~6~~



tmp:





Prohledávání do šířky

Prohledávání do šířky – vlastnosti

- **Úplnost** – algoritmus je úplný, tj. jestliže existuje řešení, BFS jej nalezne. Jedná-li se o nekonečný graf, algoritmus bude konvergovat k řešení (v praxi ale dříve nebo později dojde k vyčerpání paměťových prostředků, které jsou vždy konečné).
- **Optimálnost** – algoritmus je optimální, tj. vybere cestu s nejmenším počtem kroků.
- **Prostorová složitost** – $O(B^M)$, kde B je max. počet větvení, M je maximální hloubka v grafu od počátečního uzlu.
- **Časová složitost** – $O(b^m)$

Prohledávání do šířky (BFS) – příklad složitosti

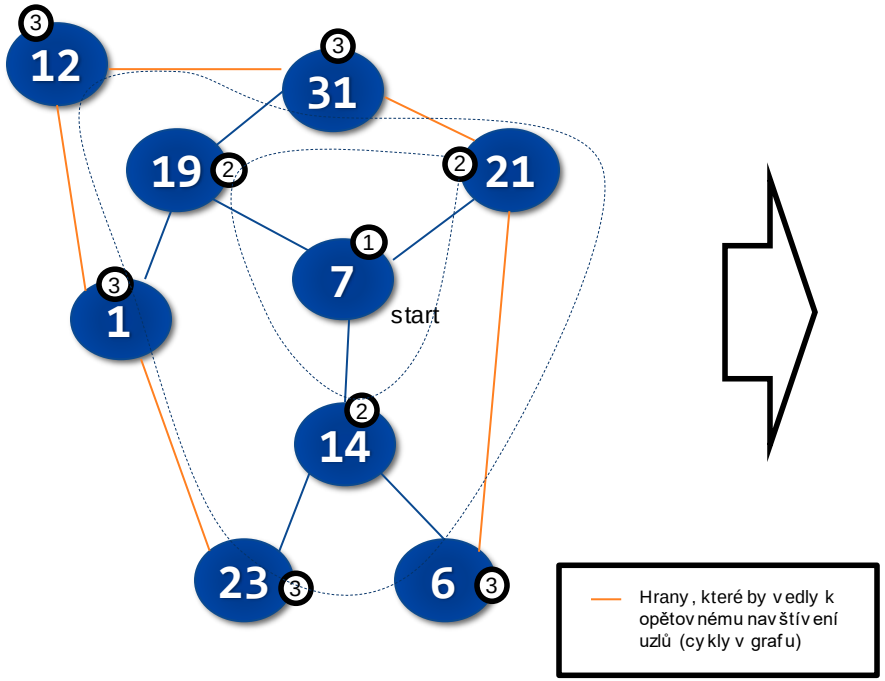
- Prostorová složitost $O(b^d)$ b... Stupeň větvení, d…hloubka stromu
- Časová složitost $O(b^d)$

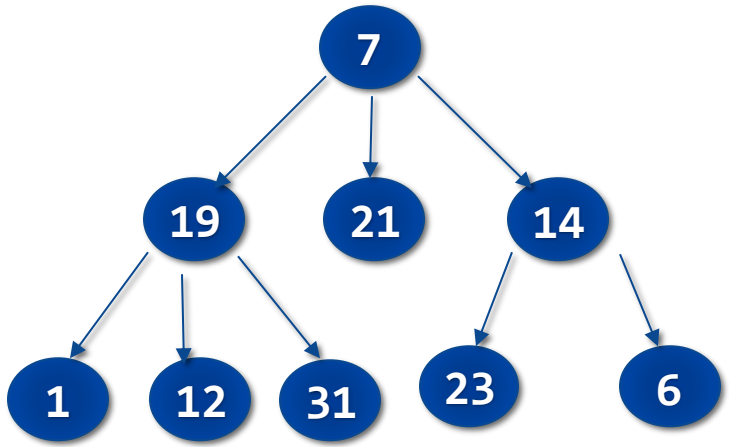
Hloubka	Uzlů	Čas	Paměť
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

b = 10, rychlost zpracování 1 milion uzlů za sekundu

Procházení grafem do šířky

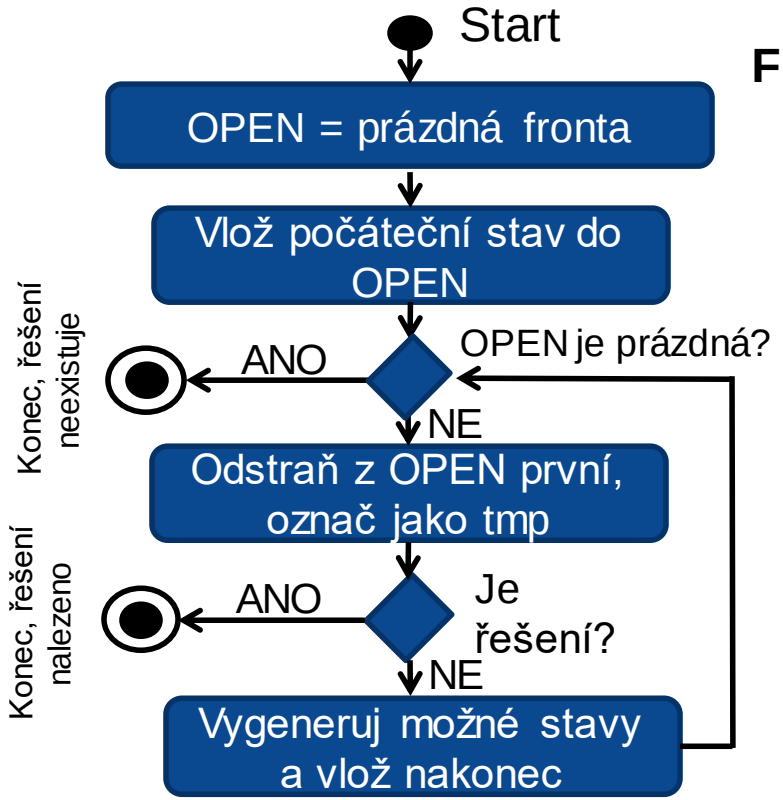
- Průchodem grafu při odstranění cyklů získáváme strom
- „Do šířky“ opisuje kružnice od „start“ uzlu



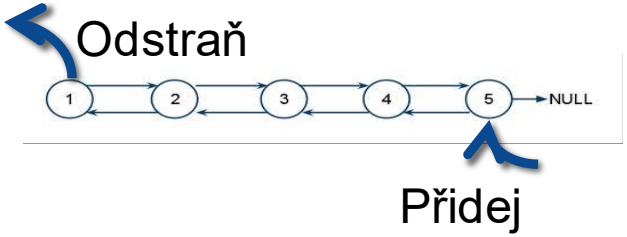


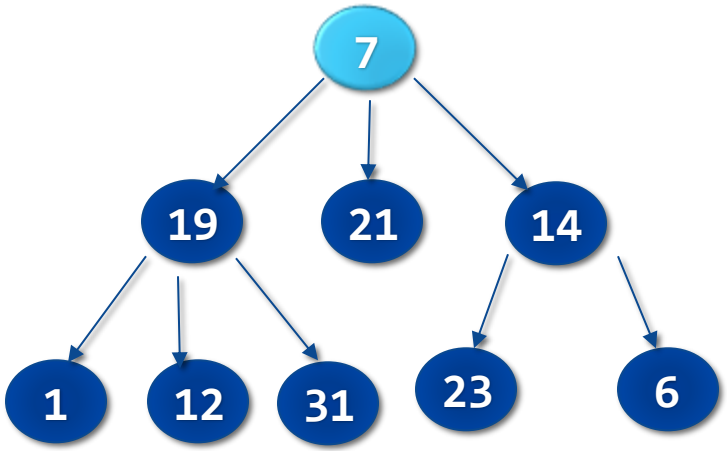
Prohledávání do šířky – animace

Jak se dostanu ze stavu 7 do stavu **6 s nejmenším počtem tahů** ?



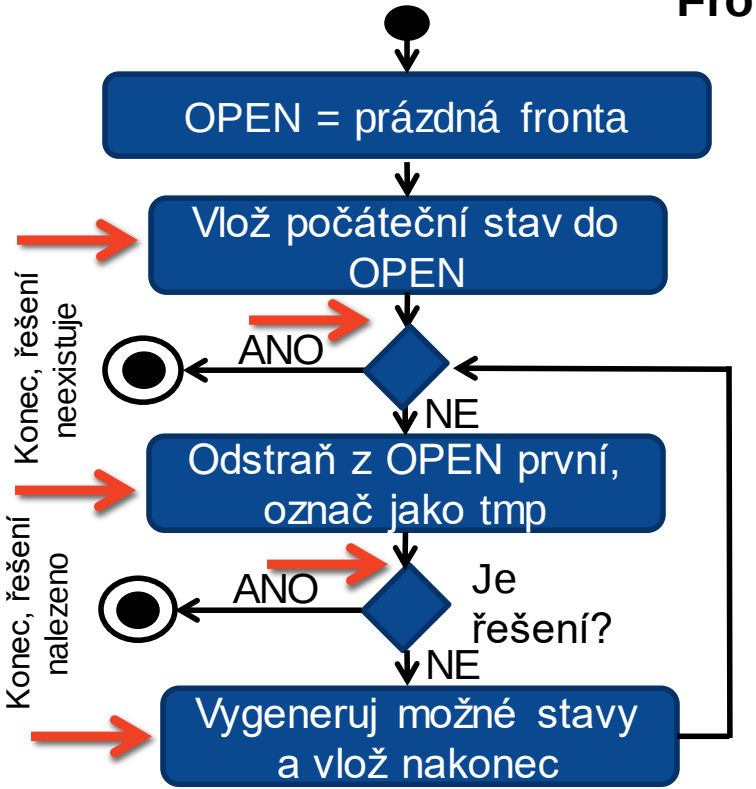
Fronta OPEN:



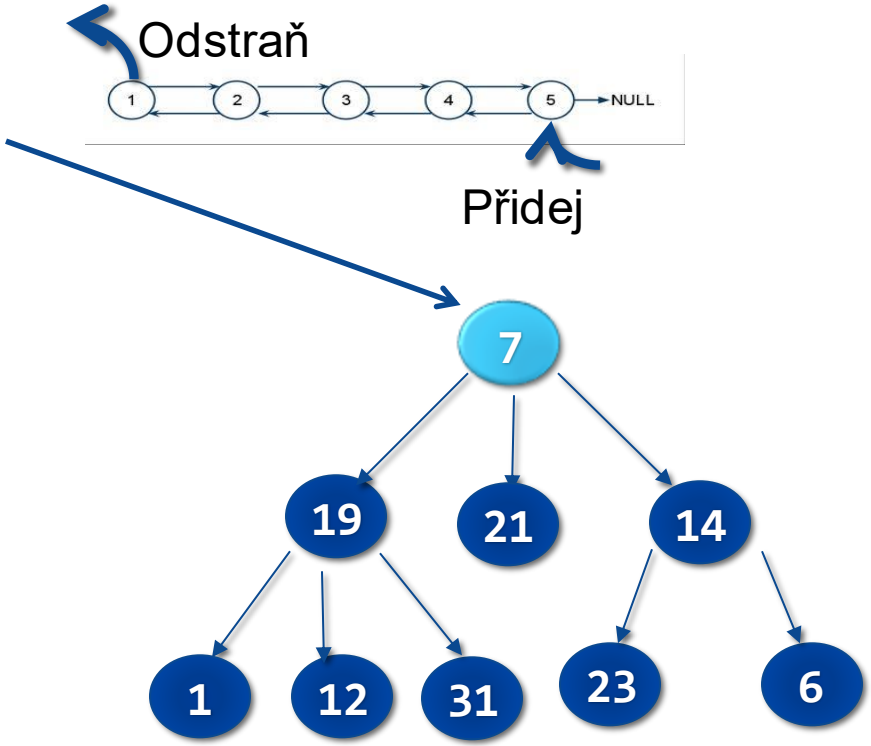


Prohledávání do šířky – animace

Fronta OPEN: ~~X~~ 19 21 14

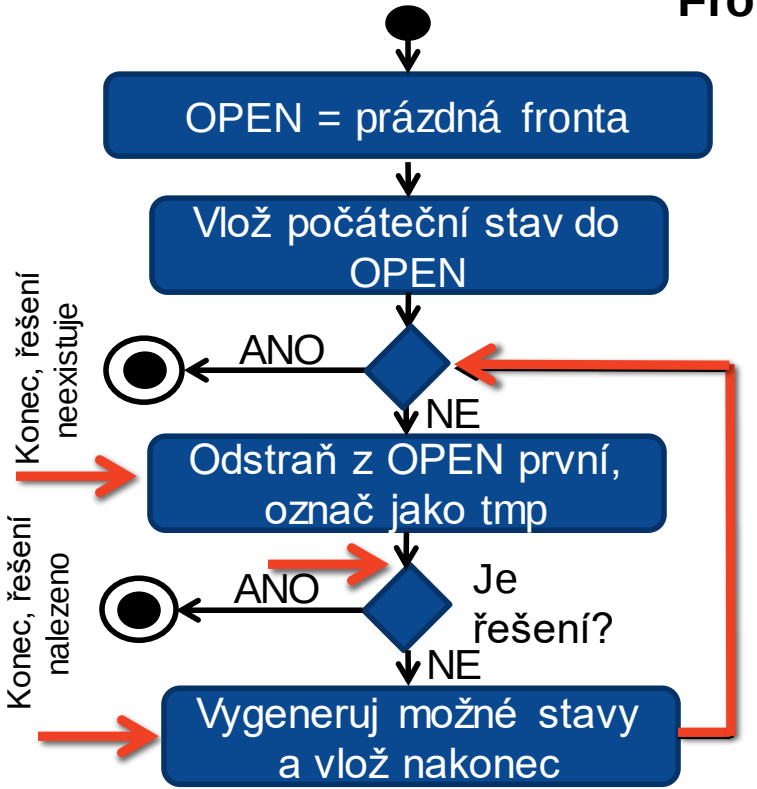


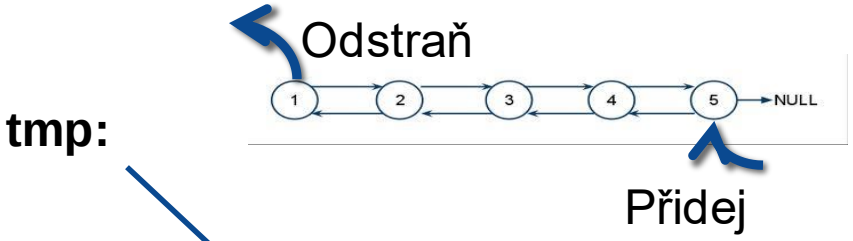
tmp:

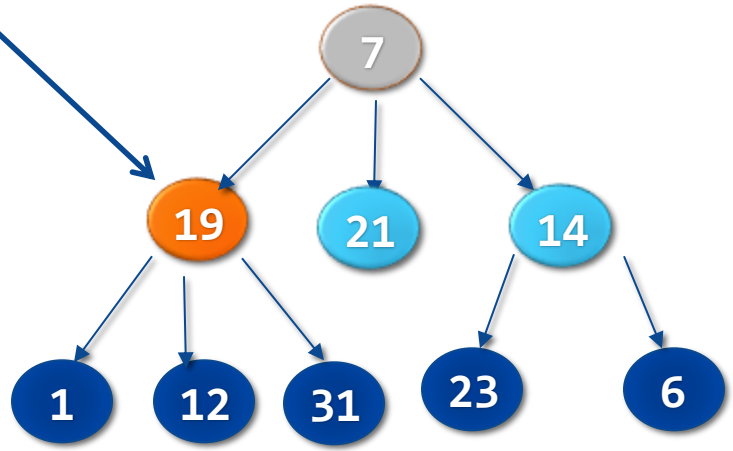


Prohledávání do šířky – animace

Fronta OPEN: ~~X~~ ~~19~~ 21 14 1 12 31

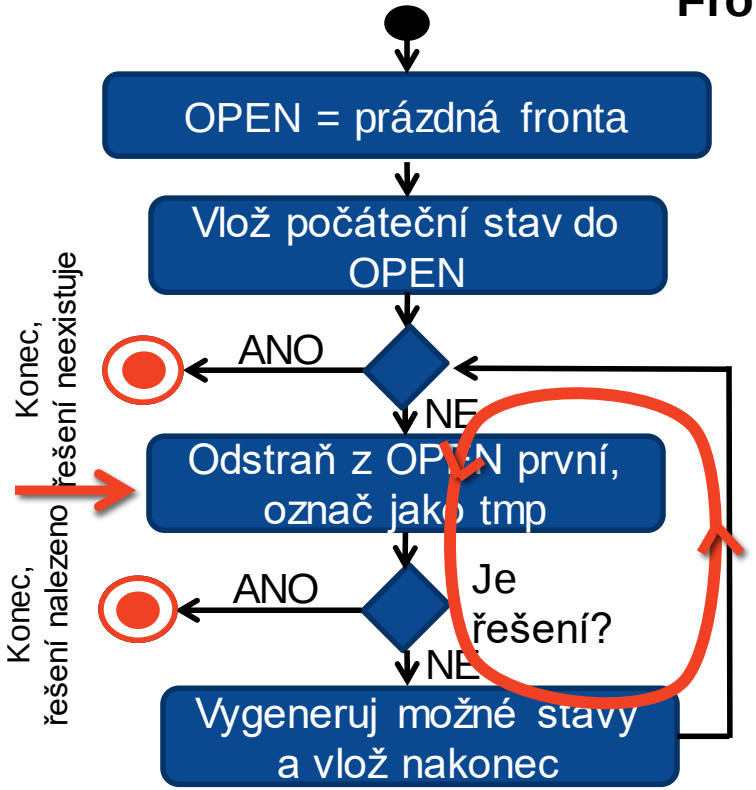


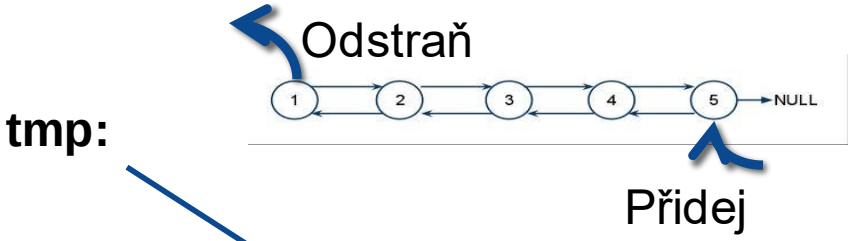


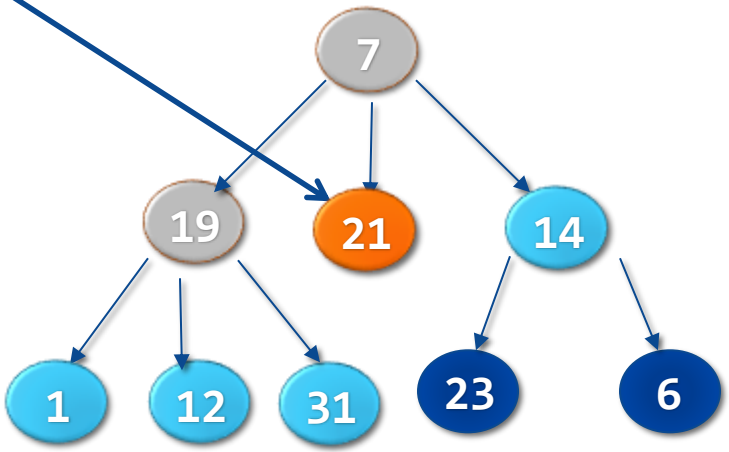


Prohledávání do šířky – animace

Fronta OPEN: ~~7~~ ~~19~~ ~~21~~ 14 1 12 31







Srovnání BFS a DFS

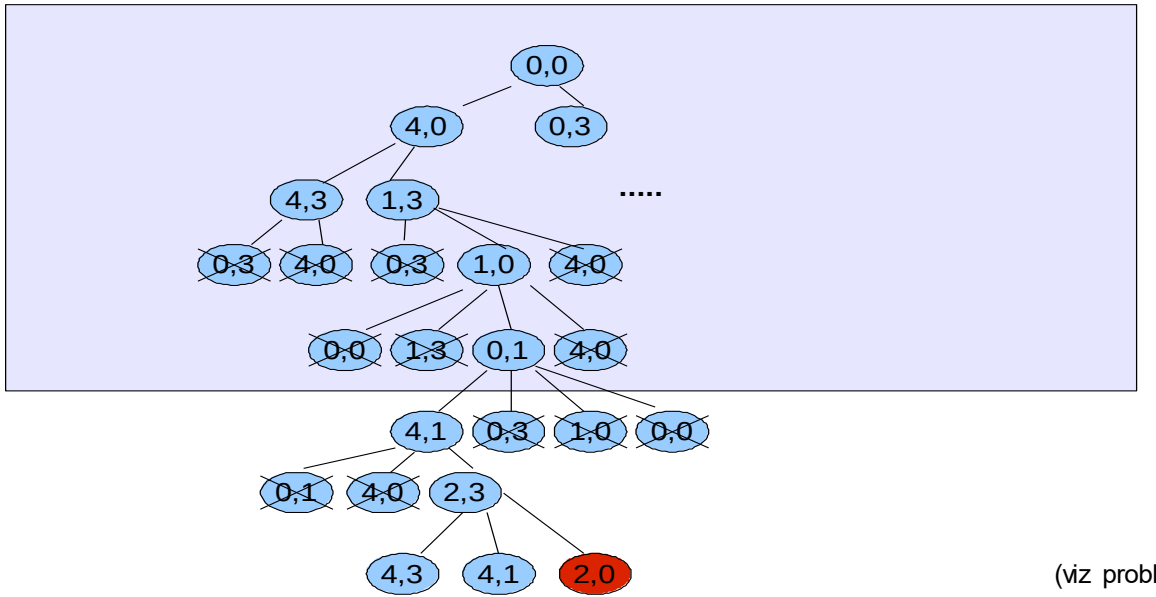
- DFS výrazně nižší paměťové nároky než BFS
- BFS je optimální

- Kompromis mezi BFS a DFS
  - Iterativní prohledávání do hloubky s omezenou hloubkou
  - Prohledávání s omezenou hloubkou (pokud znám hloubku, ve které se řešení nalézá)

  - Současně optimální a současně nízké paměťové nároky

Prohledávání s omezenou hloubkou

- Vychází z algoritmu DFS, ale max. hloubka je omezena
- Výhodné, znám-li počet potřebných tahů (úspora paměti i průměrného času řešení)

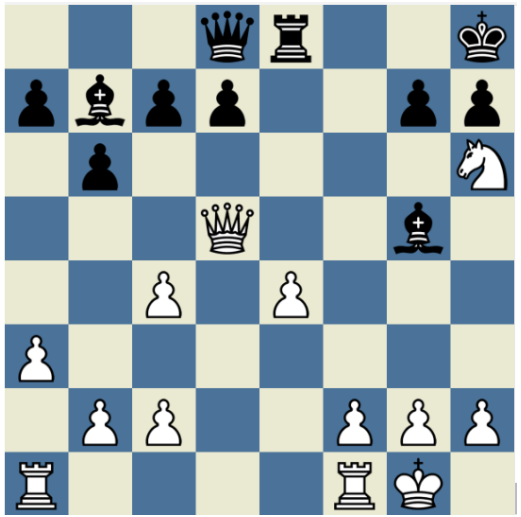


(viz problém dvou džbánů později)

Prohledávání s omezenou hloubkou

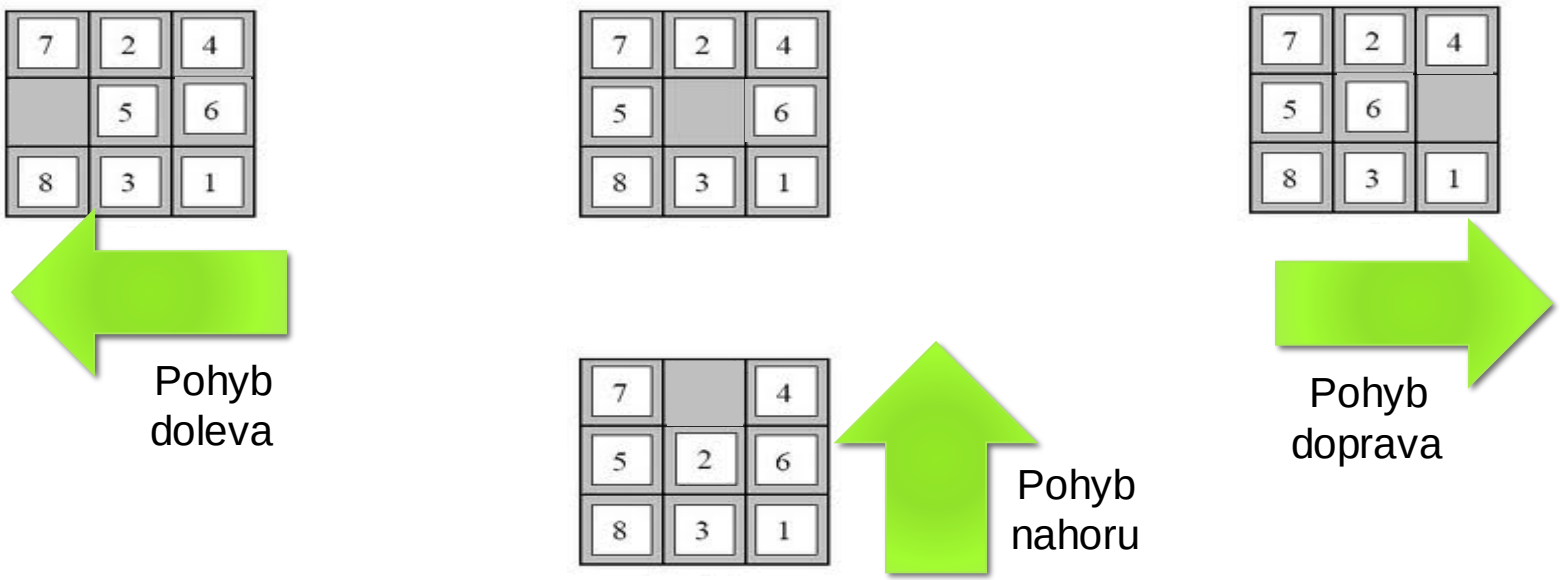
- Ideální v případě, kdy známe hloubku, ve které se řešení nachází (v takovém případě by byl optimální)
- Nízká spotřeba paměti (DFS)

Nalezněte všechny možnosti jak dát mat



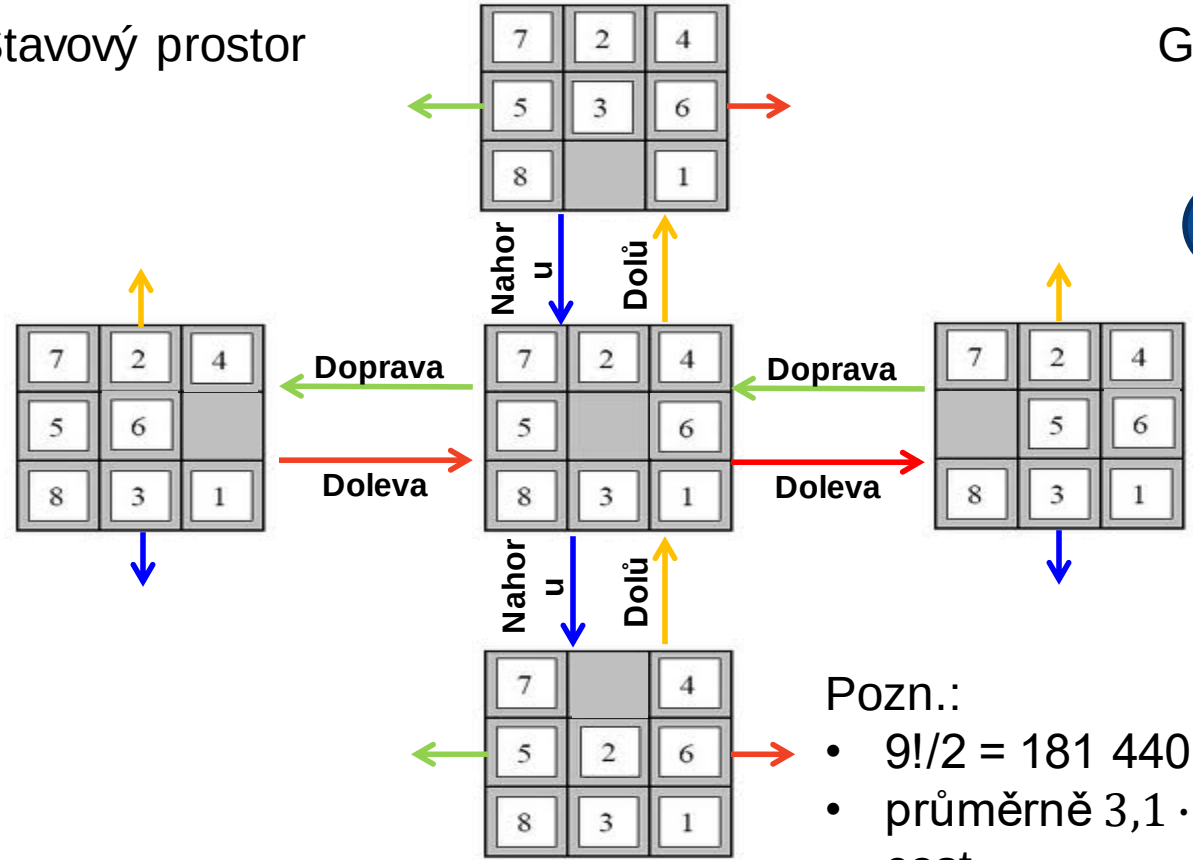
Příklad: osmička – zobecnění řešení

- Zjednodušení pro budoucí algoritmy:
  - nepohybuji fyzickou kostkou, ale jakoby „**prázdným políčkem**“

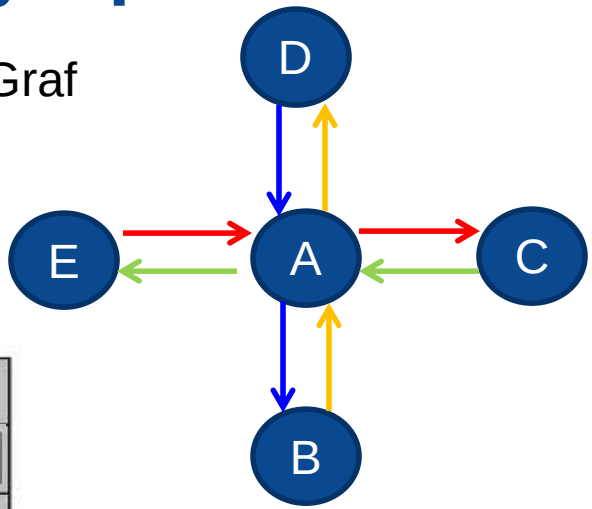


Souvislost mezi grafem a stavovým prostorem

Stavový prostor



Graf

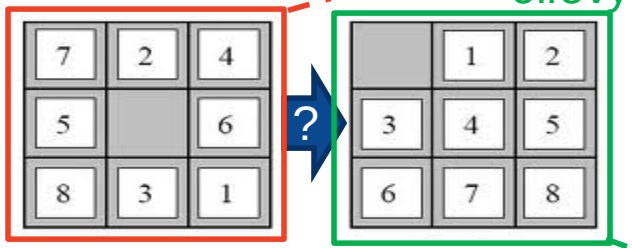


Pozn.:

- 9!/2 = 181 440 možných stavů
- průměrně $3{,}1 \cdot 10^{10}$ = 31 miliard možných cest

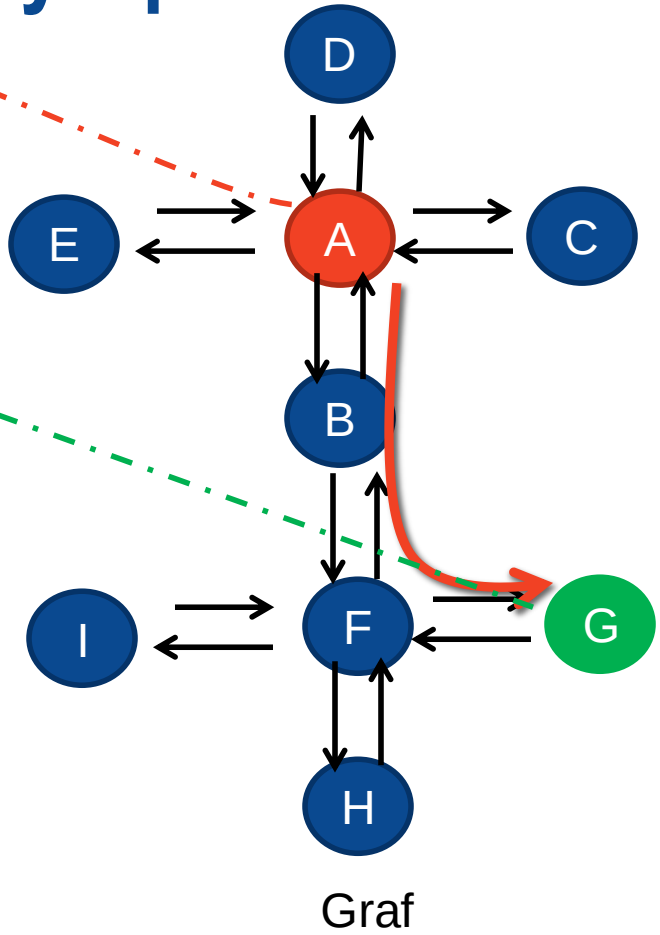
Souvislost mezi grafem a stavovým prostorem

počáteční stav



cílový stav

Stavový prostor



Graf

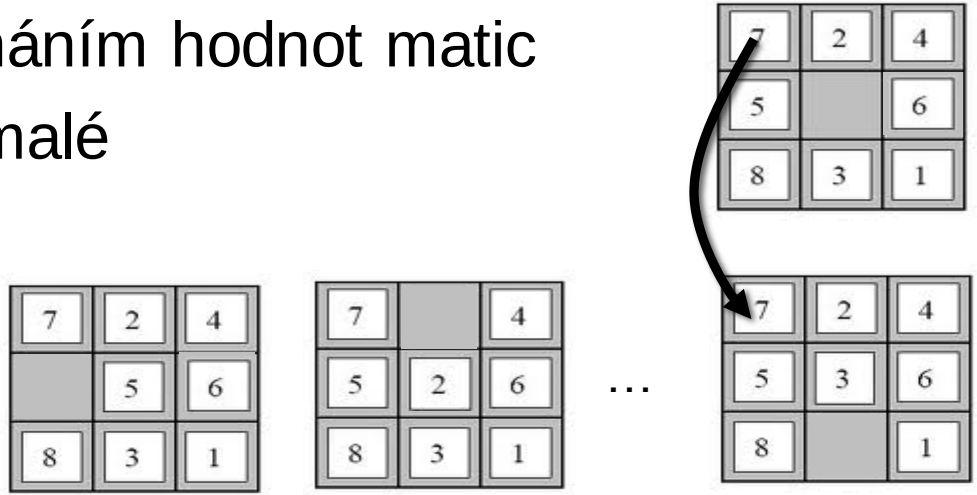
- Předpoklady:
  - znám počáteční stav,
  - umím vytvářet nové stavy,
  - znám cílový stav,

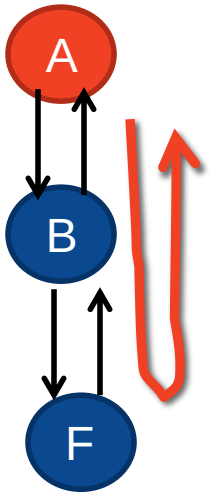
=> řešitelné jako problém hledání cesty v grafu

Problém odstranění duplicitních stavů

Jak zajistit, aby se neprohledávaly již navštívené stavy?

1. Porovnáním hodnot matic
   pomalé



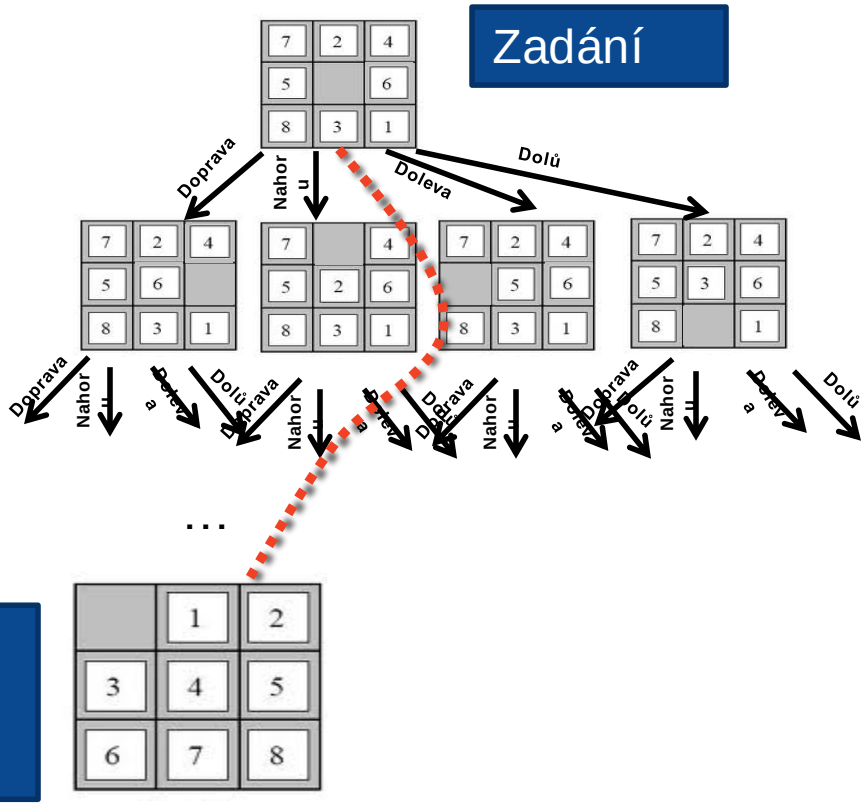


- 9!/2 = 181 440 možných stavů

Zjištění výsledného řešení?

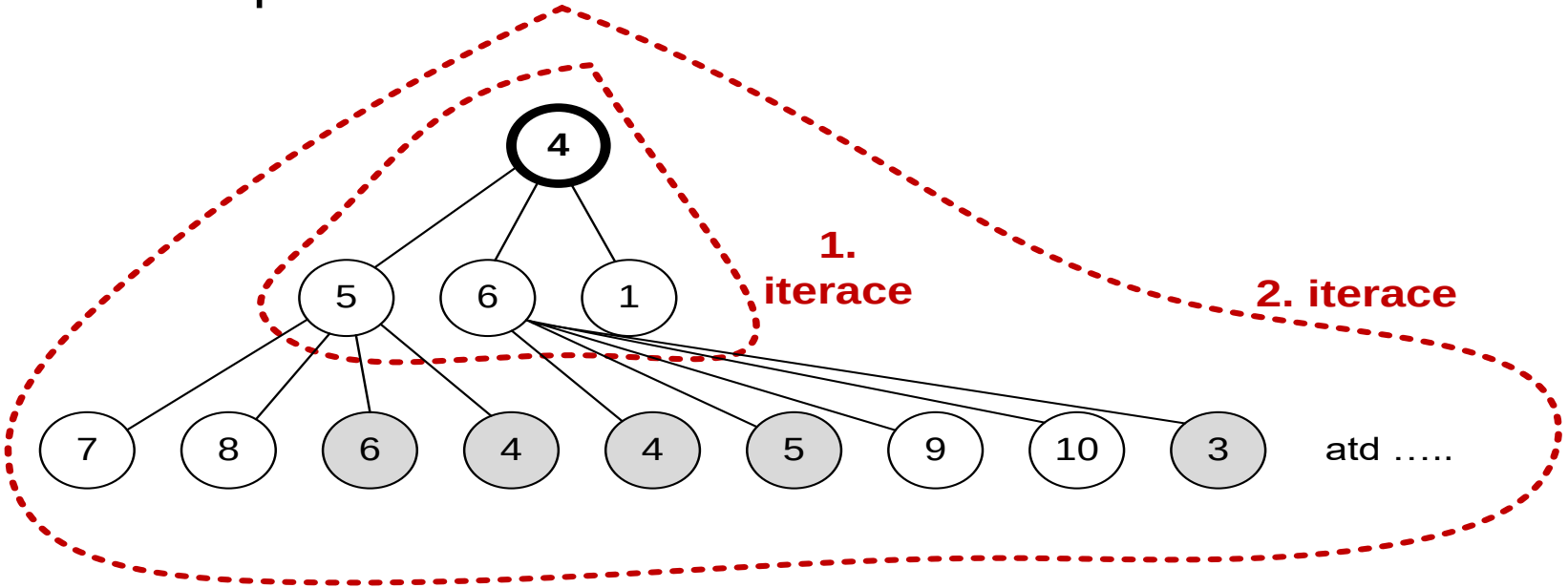
Vytvoření nového stavu
-> pamatovat způsob, jakým pohybem vznikl
+
->rodičovský stav:
(nahoru, dolů, doleva či doprava).

=> Jsem schopen vypsat cestu



Iterativní prohledávání s omezenou hloubkou

- Vychází z algoritmu DFS
- Kompromisem mezi DFS a BFS
- Šetří paměť na úkor času

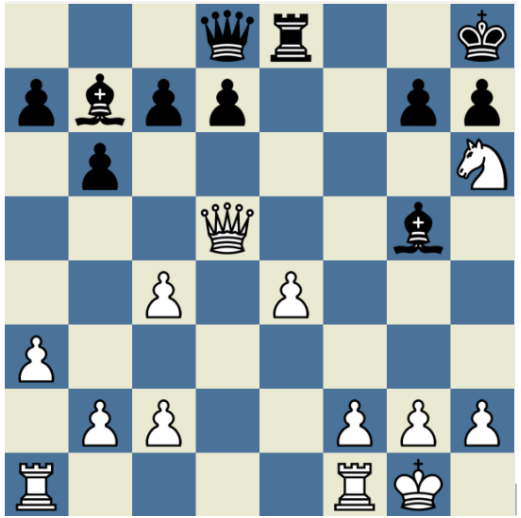


Iterativní prohledávání s omezenou hloubkou

- **Úplnost** – algoritmus je úplný.
- **Optimálnost** – algoritmus je optimální, je nalezeno takové řešení, ke kterému vede nejmenší počet kroků.
- **Prostorová složitost** – jako DFS.
- **Časová složitost** – suma jednotlivých iterací, kde v každé iteraci je použit algoritmus DFS.

Iterativní prohledávání s omezenou hloubkou

V jakém nejkratším tahu je možné vyhrát?
Prohledávání do hloubky hlásí vyčeprání paměti.

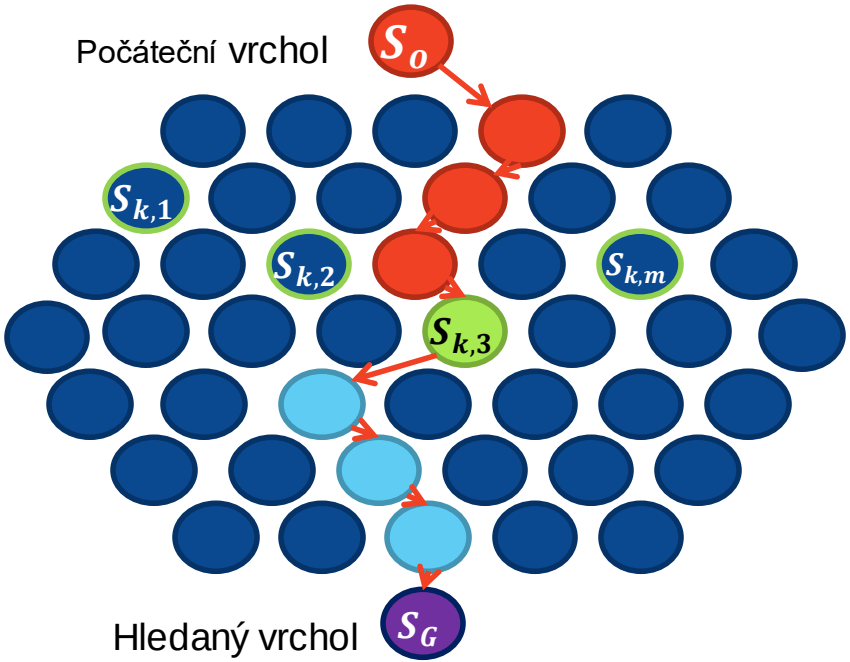


Srovnání – Dijkstrův algoritmus vs. DFS a BFS

- Algoritmus je identický
- Liší se ve způsobu výběru kandidátních cest
  - Do šířky (BFS) … fronta
  - Do hlobky (DFS)…zásobník
  - Dijkstra … prioritní fronta

$$F(S_k) = g(S_k) + h(S_k)$$
$$= g(S_k - S_0) + h(S_G - S_k)$$
$$h(S_k) = 0$$

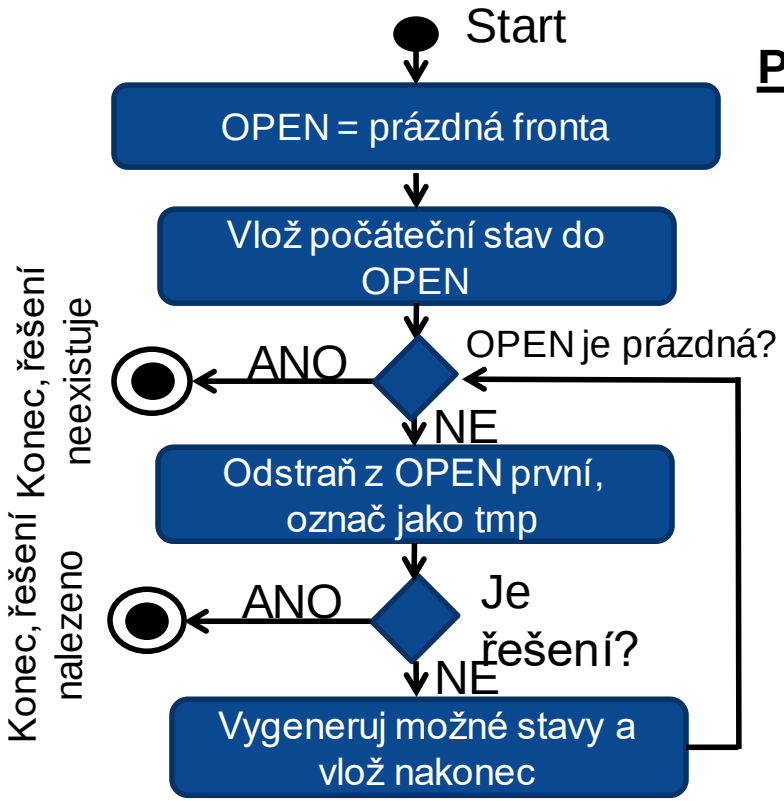
$g(S_k)$ ... cena cesty (suma vah hran)



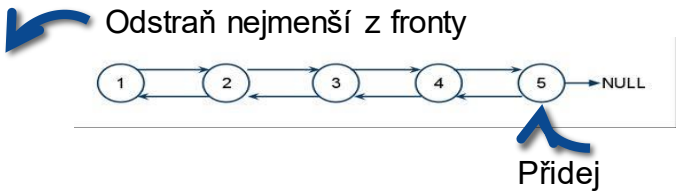
Dijkstrův algoritmus

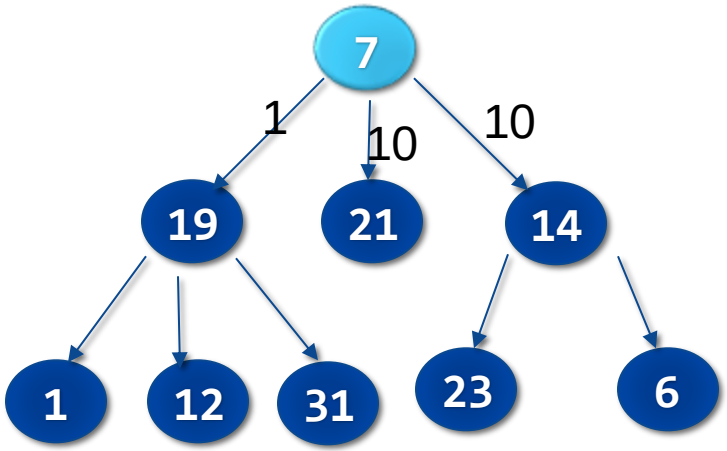
- Pracuje jen pokud všechny váhy jsou kladné.
- Poskytuje nejkratší cesty ze zdroje do všech dalších vrcholů v grafu
- Směrovací tabulka
  - IS-IS
  - OSPF
- Může být ukončen okamžitě po nalezení cesty do hledaného vrcholu *t*, pokud je to pro daný problém žádoucí.

Dijkstrův algoritmus

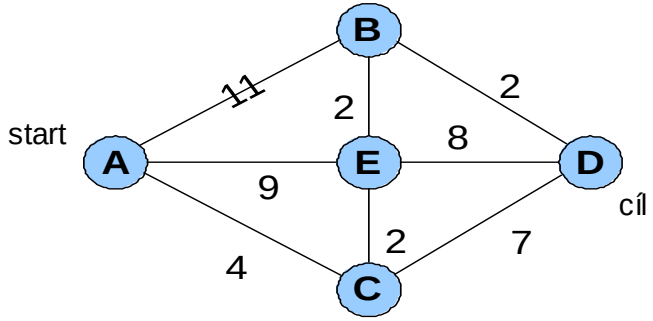


Prioritní fronta OPEN:



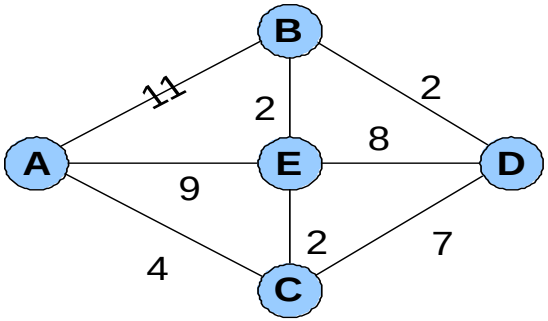


Dijkstrův algoritmus – příklad



- 0) **[((A),0)]**
- 1) [~~((A),0)~~] [((B,A),11), (([E,A]),9), **((C,A),4)**]
- 2) [((B,A),11), (([E,A]),9),((A,C,A),8),**((E,C,A),6)**,((D,C,A),11)]
- 3) [((B,A),11), (([E,A]),9), ((A,C,A),8), ~~((E,C,A),6),~~ ((B,E,C,A),8), ((A,E,C,A),15), ((C,E,C,A),8), ((D,E,C,A),14), ((D,C,A),11)]
- 4) [((B,A),11), (([E,A]),9), ~~((A,C,A),8)~~, ((B,A,C,A),19), ((E,A,C,A),17), ((C,A,C,A),12), ((B,E,C,A),8), ((A,E,C,A),15), **((C,E,C,A),8)**, ((D,E,C,A),14), ((D,C,A),11)]

Dijkstrův algoritmus – příklad



- 5) [((B,A),11), (([E,A]),9), ((B,A,C,A),19), ((E,A,C,A),17), ((C,A,C,A),12), **((B,E,C,A),8)**, ((A,E,C,A),15), ~~((C,E,C,A),8)~~, ((A,C,E,C,A),12), ((E,C,E,C,A),10), ((D,C,E,C,A),15), ((D,E,C,A),14), ((D,C,A),11)]
- 6) pokračuji (([E,A]),9)… a rozgeneruji
  7) [((B,A),11), (([E,A]),9), ((B,A,C,A),19), ((E,A,C,A),17), ((C,A,C,A),12), ~~((B,E,C,A),8)~~, ((A,B,E,C,A),19), ((E,B,E,C,A),10), **((D,B,E,C,A),10)**, ((A,E,C,A),15),, ((A,C,E,C,A),12), ((E,C,E,C,A),10), ((D,C,E,C,A),15), ((D,E,C,A),14), ((D,C,A),11)]
- 8) Konec – vybrán stav, který vede z A do D a má nejnižší cenu

Horolezecký algoritmus – Hill-climbing

- Podobný BFS, na rozdíl od BFS odstraňuje všechna starší řešení a ponechává pouze tyto poslední expandované stavy.
- Vybírá nejlepší a expanduje jej
- Není úplný (nemusí nalézt řešení)
- Není optimální
- Šetří paměť

Informované metody prohledávání

- Best First Search (BestFS, hladové vyhledávání = greedy search)
- A* (čti A-star)

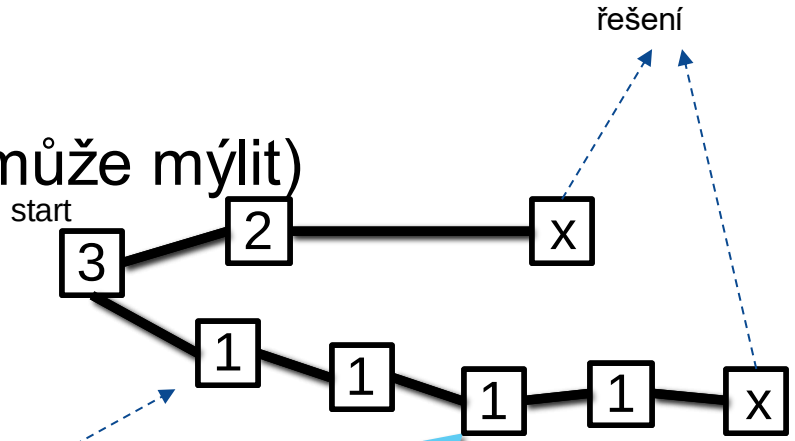
Best First Search (BestFS, Greedy Search)

- Zobecňuje BFS
- Namísto fronty používá prioritní frontu, kde se prvky řadí dle hodnoty $F(S_k)$
- $g(S_k)$ … funkce kolik mne stála cesta sem = 0, tj. neberu v potaz při výběru dalšího vrcholu grafu
- $h(S_k)$ … heuristická funkce, pokouší se odhadnout, jak jsem blízko ke hledanému stavu S_G. Čím blíže cíli S_g, tím je nižší (tj. klesající).

$$F(S_k) = g(S_k) + h(S_k) = g(S_k - S_0) + h(S_G - S_k)$$

Best First Search (BestFS, Greedy Search)

- Používá heuristickou hodnotu aby pomohla uhodnout nejlepší cestu.
- Je úplný
- Není optimální (heuristika se může mýlit)
- Časová složitost: $O(b^m)$
- Prostorová složitost: $O(b^m)$

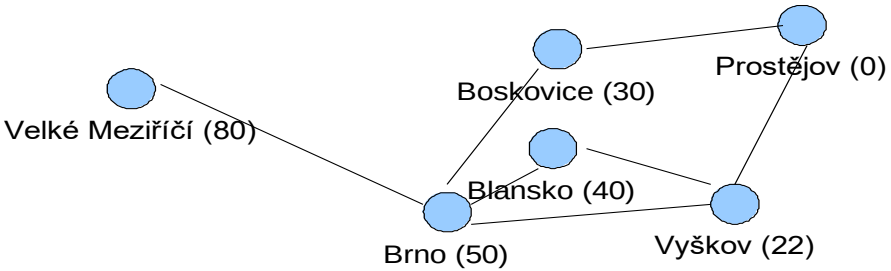


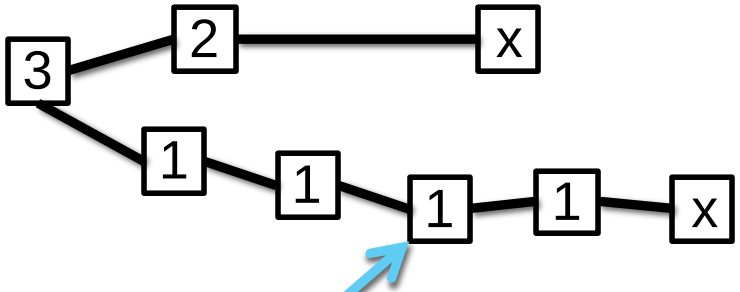
$h(S_G - S_k)$ … odhadovaná vzdálenost k cíli

V tomto bodě je výhodnější použít alternativní cestu, BestFS ale pokračuje dál

Best First Search (BestFS, Greedy Search)

- $g(S_k) = 0$, pro další kroky se rozhoduje výhradně na základě heuristiky
- Příklad – nalezněte cestu z Velkého Meziříčí do města Prostějov

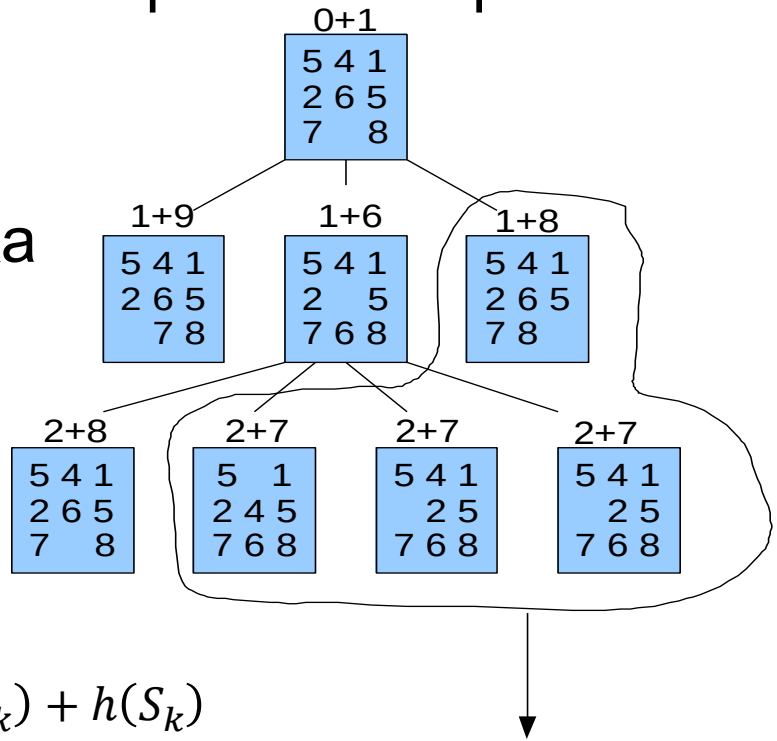




V tomto bodě je výhodnější použít alternativní cestu, BestFS ale pokračuje dál

A* (Čtěte A-star)

- Vychází z BestFS + navíc bere v potaz i kompletní cestu, kterou již urazil.
- Sčítá se minulost + heuristika
- Heuristika $h(S_k)$: např. kolik políček stále není na svém místě



$$F(S_k) = g(S_k) + h(S_k)$$

Náhodně jednou z nejlepších pokračuje.

A* (Čtěte A-star)

- Časová složitost algoritmu je $O(b^d)$
- Prostorová složitost je $O(b^d)$.

- Je úplný
- Je optimální

- *b* je stupeň stromu stavového prostoru a *d* je hloubka

Slepé metody prohledávání - srovnání

Algoritmus	Úplný	Optimální	Časová složitost	Prostorová složitost	Výběr uzlů
BFS	Ano	Pokud je cena hran stejná	$O(b^d)$	$O(b^d)$	Fronta
UCS (Dijkstra)	Ano	Ano	Počet uzlů s g(n) ≤ C*		Prioritní fronta $F(S_k) = g(S_k)$
DFS	Ano	Ne	$O(b^m)$	$O(bm)$	Zásobník
IDLS	Ano	Pokud je cena hran stejná	$O(b^d)$	$O(bd)$	Zásobník

b: max stupeň uzlu
d: hloubka optimálního řešení
m: max. délka cesty ve stavovém prostoru
C*: cena optimálního řešení

Veškeré metody prohledávání - srovnání

Algoritmus	Úplný	Optimální	Časová složitost	Prostorová složitost	Výběr uzlů
BFS	Ano	Pokud je cena hran stejná	$O(b^d)$	$O(b^d)$	Fronta
UCS (Dijkstra)	Ano	Ano	Počet uzlů $g(n) \leq C^*$		Prioritní fronta $F(S_k) = g(S_k)$
DFS	Ano	Ne	$O(b^m)$	$O(bm)$	Zásobník
IDLS	Ano	Pokud je cena hran stejná	$O(b^d)$	$O(bd)$	Zásobník
BestFS (Greedy)	Ne	Ne	Nejhorší: $O(b^m)$ Nejlepší: $\Omega(bd)$		Prioritní fronta $F(S_k) = h(S_k)$
A*	Ano	Ano	Počet uzlů: $g(n) + h(n) \leq C^*$		Prioritní fronta $F(S_k) = g(S_k) + h(S_k)$

Příklady použití A*

- An Efficient A* Search Algorithm For Statistical **Machine Translation**. 2001
  - Strojové překlady na základě statistiky
- The Generalized A* Architecture. Journal of Artificial Intelligence Research (2007)
  - Počítačové vidění … nový model pro nalezení nejvýznamnějších křivek
- Factored A*search for models over sequences and trees International Conference on AI. 2003….
  - Hlavní výzva pro A* je navrhnout heuristickou funkci, která je současně dostatečně rychlá, co nejpřesnější… časté použití je zpracování přirozeného jazyka, bio-informatice a zpracování obrazu

Příklady použití A* (pokračování)

- Aker, A., Cohn, T., Gaizauskas, R.: **Multi-document summarization** using A* search and discriminative training. Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing. ACL (2010)
  - Automatická sumarizace významu obsahu několika dokumentů

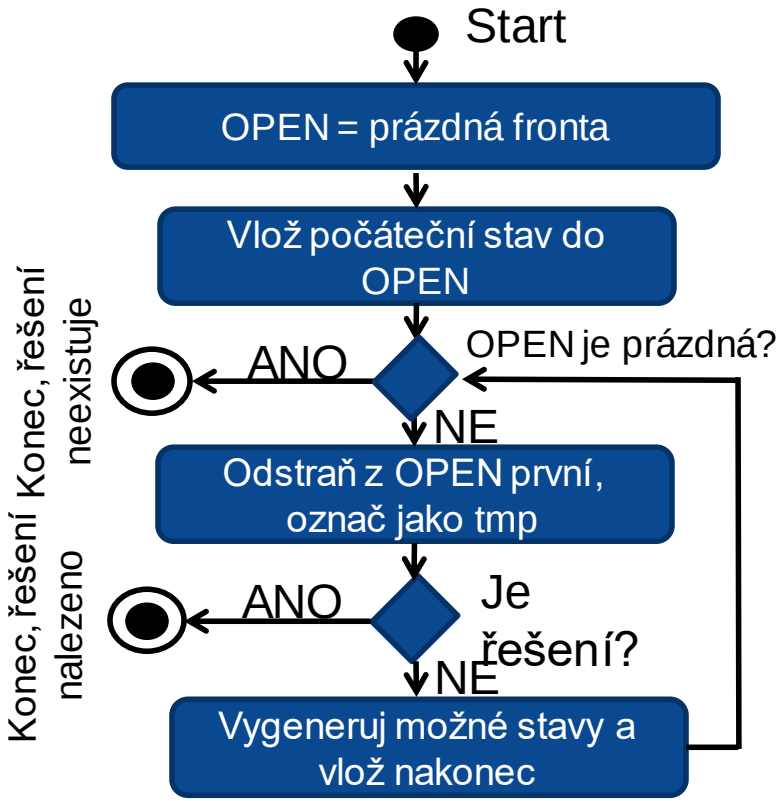
Prohledávání grafů – příklady použití

- Cílem je získat představu o uplatnění nabytých znalostí v praxi …
- A to nejen v oblasti sítí

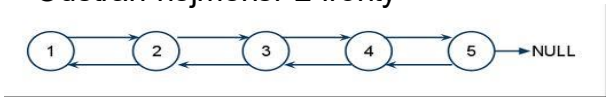
Diskstrův algoritmus - poznámka

- V prostředí sítí má každý směrovač samostatný procesor, tj. výpočet probíhá samostatně na každém z nich distribuovaným způsobem
- Distribuovaná varianta nás v tomto předmětu nebude zajímat

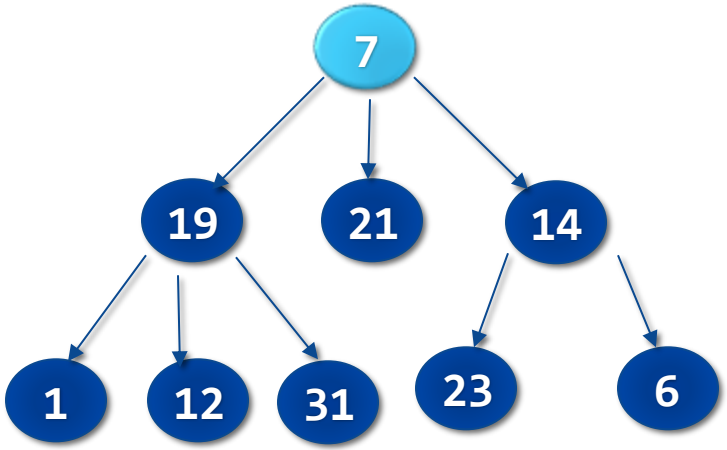
Shrnutí



Odstraň nejmenší z fronty

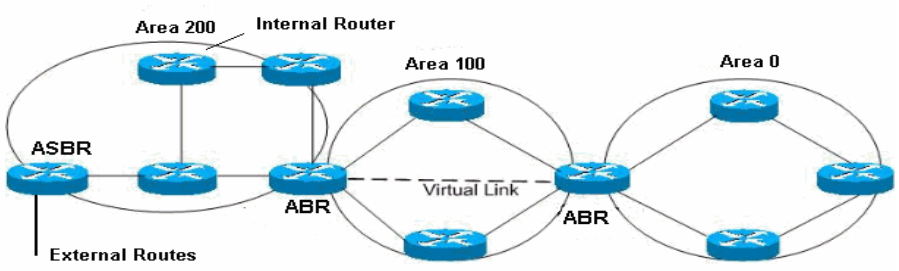


Konst. cena + zásobník	… Do hloubky
Konst. cena + fronta	… Do šířky
Cena + prio. fronta	… Dijkstra (UCS)
Heurist. + prio. fronta	… BestFS (Greedy)
Cena + heurist + prio. fronta	… A*



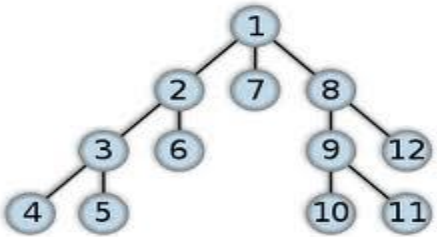
Příklad I.

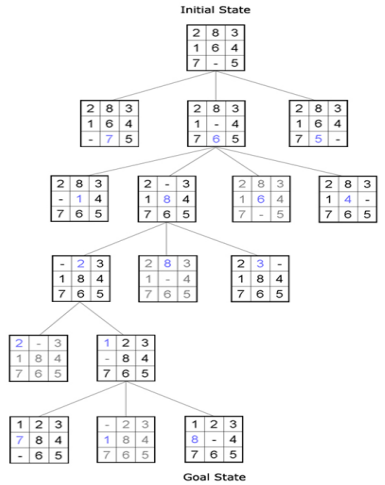
- Mějme směrovač, ve kterém chceme nalézt nejkratší cestu z lokálního uzlu do všech okolních stanic (OSPF, IS-IS).
  - Co musí směrovač vědět o svém okolí, než začne s výpočtem?
  - Jaký algoritmus by pro tyto potřeby měl být použit?



Příklad II.

- Aplikujte algoritmy prohledávání grafů na problematiku hry čísla.
  - Jaký algoritmus (ritmy) lze použít?
  - Jaká je analogie hry čísla a prohledávání stromů, kde jsou uzly čísla
    - isSolution() : boolean
    - Rozgenerovat následníky každého uzlu



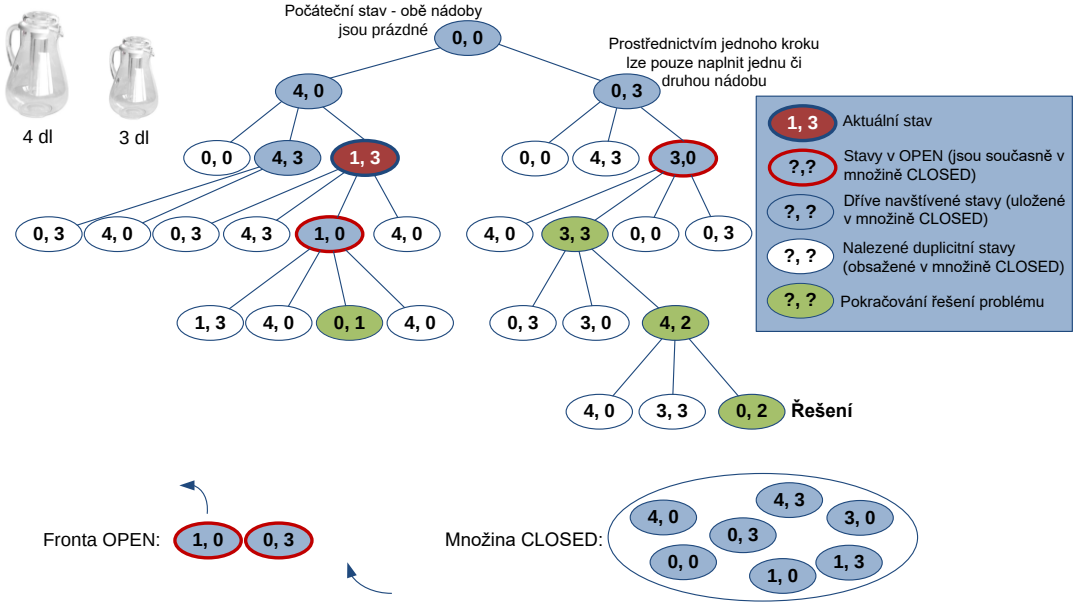


Příklad III.

- Mějme dvě sklenice, jedna má 0,4 litru, druhá 0,3 litru. Jak s jejich pomocí (přelévání) přesně odměřit 0,2 litru?
  - Jaký algoritmus lze pro tento problém použít?

Příklad III.

- Mějme dvě sklenice, jedna má 0,4 litru, druhá 0,3 litru. Jak s jejich pomocí (přelévání) přesně odměřit 0,2 litru?
  - Jaký algoritmus lze pro tento problém použít?



Otázka

- Mějme algoritmus A*, kde heuristika bude naprosto bezcenná a všechny hrany mají konstantní cenu, jaký algoritmus je ekvivalentní?
  - BFS
  - DFS
  - IDFS

Shrnutí

- Umíme projít všechny vrcholy grafu
  - (= vyhledat vrchol v grafu)
- Umíme najít nejkratší cestu v grafu
  - Které z algoritmů jsou optimální?
- Algoroitmy
  - Slepé metody
  - Informované metody
- Složitost je často exponenciální $O(b^d)$

Hloubka	Uzlů	Čas	Paměť
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

b = 10, rychlost zpracování 1 milionů uzlů za sekundu

Děkuji za pozornost