2. Metody řešení úloh prohledáváním stavového prostoru.

1. Slepé metody (Blind Search Methods)

   Slepé metody jsou metody, které nevyužívají žádné informace, které by mohly usnadnit řešení úlohy. Jejich použití je proto oprávněné pouze v případech, kdy o řešených úlohách žádné informace skutečně nemáme.

2. Informované metody (Informed Search Methods)

   Informované metody jsou metody, které naopak využívají nějaké informace o řešené úloze – tyto informace pak usnadňují, resp. umožňují řešení této úlohy.

3. Metody lokálního prohledávání (Local Search Methods)

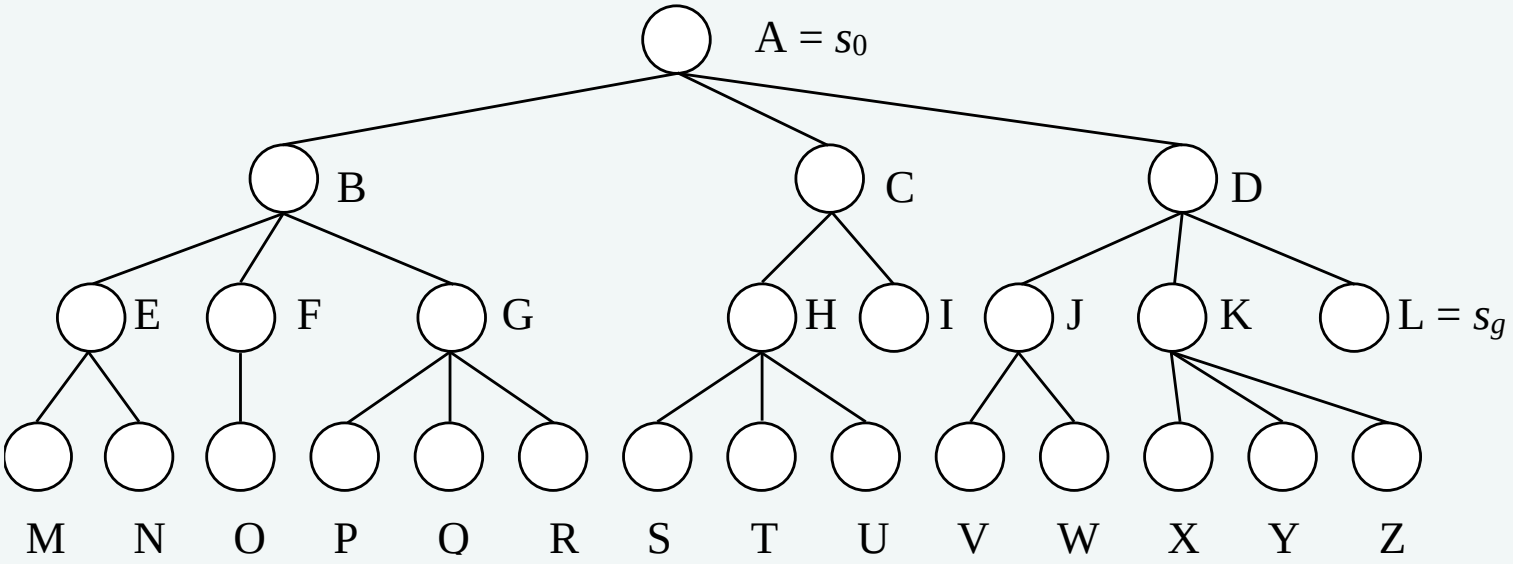
   Metody lokálního prohledávání jsou metody, které místo systematického prohledávání stavového prostoru prohledávají pouze okolí aktuálního stavu. Jsou vhodné především pro řešení optimalizačních problémů.

Hodnotící kritéria prohledávacích metod

- Úplnost metody – pokud nějaká řešení úlohy existují, tak úplná metoda jedno z nich musí nalézt.
- Optimálnost metody – pokud nějaká řešení úlohy existují, tak optimální metoda musí nalézt nejlepší z těchto řešení. Optimální metoda je proto vždy úplná.
- Časová složitost metody.
- Prostorová složitost metody.

Terminologie

Uvažujme jednoduchý stavový prostor (stav = uzel, hrany označující přechody mezi dvěma stavy jsou implicitně orientovány ve směru od počátečního stavu):



s_0 počáteční stav (Initial state)

s_g cílový stav (Goal state / Goal)

Terminologie

- uzel **A** je uzel kořenový,
- uzly **I**, **L**, **M, …, Z** jsou uzly listové,
- uzel **C** je bezprostředním předchůdcem uzlu **H**, apod.,
- uzly **A**, **D**, **J** jsou předchůdci uzlu **V**, apod.,
- uzel **K** je bezprostředním následníkem uzlu **D**, apod.,
- uzly **H**, **I**, **S**, **T, U** jsou následníci uzlu **C**, apod.,
- uzel **A** má hloubku 0, uzly **B**, **C**, **D** mají hloubku 1, apod.,
- expanzí uzlu se rozumí určení všech jeho bezprostředních následníků,
- generací uzlu se nazývá proces jeho vytvoření,
- ohodnocení uzlu (může být například dáno součtem cen přechodů z kořenového do ohodnocovaného uzlu),
- uzel **A** označuje počáteční stav úlohy (s_0),
- uzel **L** označuje cílový stav úlohy (s_g).

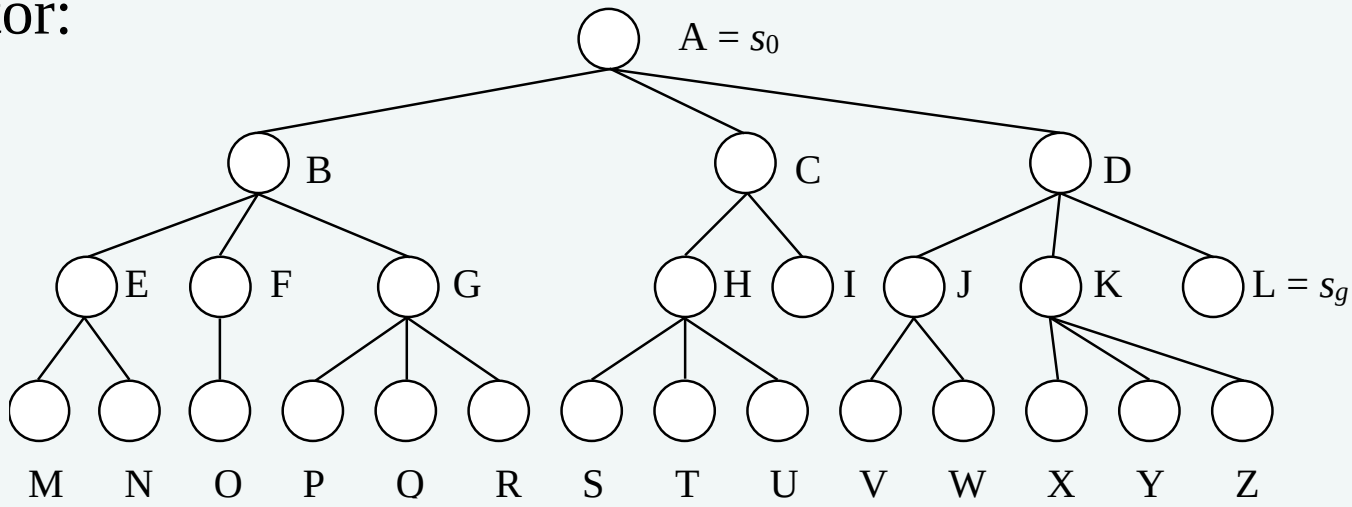
1. Slepé metody

- BFS – Breadth First Search (prohledávání do šířky)
- DFS – Depth First Search (prohledávání do hloubky)
- DLS – Depth Limited Search (prohledávání do omezené hloubky)
- IDS – Iterative Deepening Search (prohledávání do omezené hloubky s postupným zanořováním)
- BS – Bidirectional Search (obousměrné prohledávání)
- UCS – Uniform Cost Search (prohledávání do šířky s respektováním cen přechodů)
- Backtracking Search (prohledávání se zpětným navracením, vhodné i pro řešení CSP)
- Forward Checking Search (prohledávání s dopřednou kontrolou, pouze pro řešení CSP)
- Min-conflict Search (prohledávání s minimalizací konfliktů, pouze pro řešení CSP)

BFS – Breadth First Search (slepé prohledávání do šířky)

1. Sestrojte **frontu** OPEN (bude obsahovat všechny uzly určené k expanzi) a umístěte do ní počáteční uzel.
2. Je-li fronta OPEN prázdná, pak úloha nemá řešení, a proto ukončete prohledávání jako neúspěšné. Jinak pokračujte.
3. Vyberte z čela fronty OPEN první uzel.
4. Je-li vybraný uzel uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak pokračujte.
5. Vybraný uzel expandujte, všechny jeho bezprostřední následníky umístěte do fronty OPEN a vraťte se na bod 2.

Demonstrační stavový prostor:

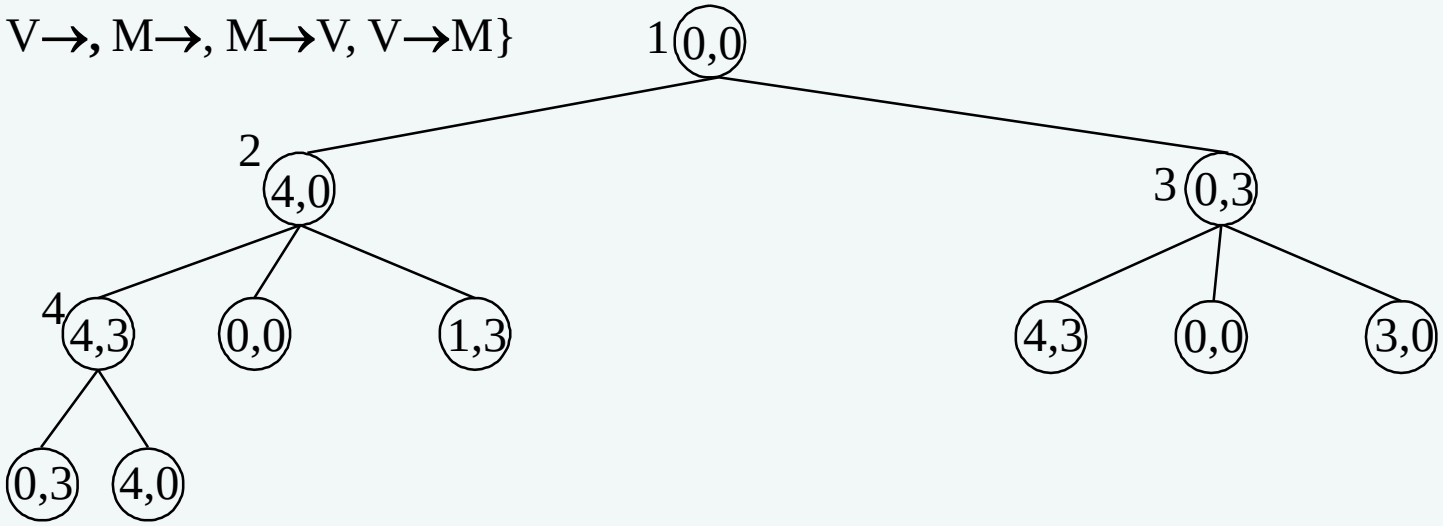


Krok	Fronta OPEN
0	[[A,nil]]
1	[[B,A,nil],[C,A,nil],[D,A,nil]]
2	[[C,A,nil],[D,A,nil],[E,B,A,nil],[F,B,A,nil],[G,B,A,nil]]
...	…
11	[[L,D,A,nil],[M,E,B,A,nil], …,[T,H,C,A,nil],…,[Z,K,D,A,nil]]
12	[[M,E,B,A,nil], …, [T,H,C,A,nil],…,[[Z,K,D,A,nil]] [L,D,A,nil] = Goal

Path (cesta, řešení úlohy): A $\rightarrow$ D $\rightarrow$ L

BFS – Úloha dvou džbánů,

$O = \{\rightarrow V, \rightarrow M, V\rightarrow, M\rightarrow, M\rightarrow V, V\rightarrow M\}$



Krok	Fronta OPEN
0	[[(0,0),nil]]
1	[[(4,0),(0,0),nil],[(0,3),(0,0),nil]]
2	[[(0,3),(0,0),nil],[(4,3),(4,0),(0,0),nil],[(0,0),(4,0),(0,0),nil], [(1,3),(4,0),(0,0),nil]]
3	[[(4,3),(4,0),(0,0),nil],[(0,0),(4,0),(0,0),nil],[(1,3),(4,0),(0,0),nil], [(4,3),(0,3),(0,0),nil],[(0,0) ,(0,3),(0,0),nil],[(3,0),(0,3),(0,0),nil]]
4	[[(0,0),(4,0),(0,0),nil],…,[(3,0),(0,3),(0,0),nil], [(0,3),(4,3),(4,0),(0,0),nil], [(4,0),(4,3),(4,0),(0,0),nil]]
5	...

Možné úpravy algoritmu BFS se týkají bodu 5 a zabraňují opakovanému expandování uzlů:

5. Vybraný uzel expandujte, do fronty OPEN umístěte všechny jeho bezprostřední následníky, kteří v ní ještě nejsou, a vraťte se na bod 2.

nebo

5. Vybraný uzel expandujte, do fronty OPEN umístěte všechny jeho bezprostřední následníky, kteří v ní ještě nejsou a kteří nejsou ani předky expandovaného uzlu, a vraťte se na bod 2.

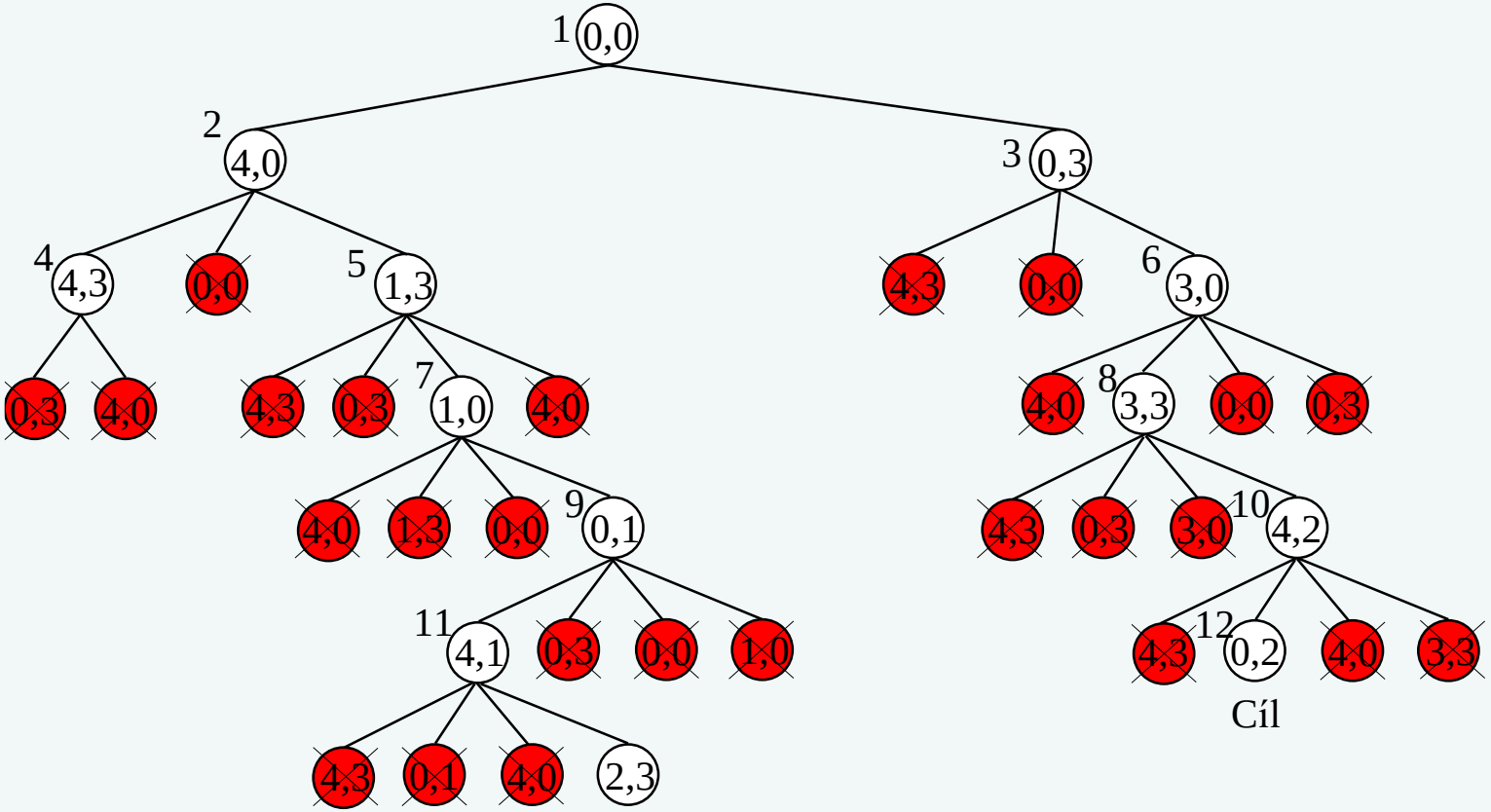
Obě tyto úpravy nejsou příliš významné, významná je však úprava se seznamem CLOSED uvedená na následujícím snímku.

BFS, verze se seznamem CLOSED

1. Sestrojte **frontu** OPEN (bude obsahovat všechny uzly určené k expanzi) a **seznam** CLOSED (bude obsahovat všechny již expandované uzly). Do fronty OPEN umístěte počáteční uzel.
2. Je-li fronta OPEN prázdná, pak úloha nemá řešení, a proto ukončete prohledávání jako neúspěšné. Jinak pokračujte.
3. Vyberte z čela fronty OPEN první uzel.
4. Je-li vybraný uzel uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak pokračujte.
5. Vybraný uzel expandujte, jeho bezprostřední následníky, kteří nejsou ani ve frontě OPEN, ani v seznamu CLOSED, umístěte do fronty OPEN, expandovaný uzel umístěte do seznamu CLOSED, a vraťte se na bod 2.

BFS – Úloha dvou džbánů, verze se seznamem CLOSED,

O = {→V, →M, V→, M→, M→V, V→M}



Pozn.: Úloha dvou džbánů s obsahy 4 a 3 litry má pouze 20 různých stavů. Ze stavu (0,0) není 6 stavů dosažitelných ((1,1), (1,2), (2,1), (2,2), (3,1), (3,2)), a tak jediným dosažitelným stavem, který není na obrázku ukázán, je stav (2,0).

Krok	OPEN	CLOSED
0	[[(0,0),nil]]	[ ]
1	[[(4,0),(0,0)], [(0,3),(0,0)]]	[[(0,0),nil]]
2	[[(0,3),(0,0)],[(4,3),(4,0)], [(1,3),(4,0)]]	[[(0,0),nil],[(4,0),(0,0)]]
3	[[(4,3),(4,0)],[(1,3),(4,0)] [(3,0),(0,3)]]	[[(0,0),nil],[(4,0),(0,0)],[(0,3),(0,0)]]
4	[[(1,3),(4,0)],[(3,0),(0,3)]]	[[(0,0),nil],[(4,0),(0,0)],[(0,3),(0,0)], [(4,3),(4,0)]]
5	[[(3,0),(0,3)],[(1,0),(1,3)]]	[[(0,0),nil],[(4,0),(0,0)],[(0,3),(0,0)], [(4,3),(4,0)],[(1,3),(4,0)]]
6	[[(1,0),(1,3)],[(3,3),(3,0)]]	[[(0,0),nil],[(4,0),(0,0)],[(0,3),(0,0)], [(4,3),(4,0)],[(1,3),(4,0)],[(3,0),(0,3)]]
7	[[(3,3),(3,0)],[(0,1),(1,0)]]	[[(0,0),nil],[(4,0),(0,0)],[(0,3),(0,0)], [(4,3),(4,0)],[(1,3),(4,0)],[(3,0),(0,3)], [(1,0),(1,3)]]

Krok	OPEN	CLOSED
8	[[(0,1),(1,0)],[(4,2),(3,3)]]	[[(0,0),nil],[(4,0),(0,0)],[(0,3),(0,0)], [(4,3),(4,0)],[(1,3),(4,0)],[(3,0),(0,3)], [(1,0),(1,3)],[(3,3),(3,0)]]
9	[[(4,2),(3,3)],[(4,1),(0,1)]]	[[(0,0),nil],[(4,0),(0,0)],[(0,3),(0,0)], [(4,3),(4,0)],[(1,3),(4,0)],[(3,0),(0,3)], [(1,0),(1,3)],[(3,3),(3,0)],[(0,1),(1,0)]]
10	[[(4,1),(0,1)],[(0,2),(4,2)]]	[[(0,0),nil],[(4,0),(0,0)],[(0,3),(0,0)], [(4,3),(4,0)],[(1,3),(4,0)],[(3,0),(0,3)], [(1,0),(1,3)],[(3,3),(3,0)], [(0,1),(1,0)], [(4,2),(3,3)]]
11	[[(0,2),(4,2)],[(2,3),(4,1)]]	[[(0,0),nil],[(4,0),(0,0)], [(0,3),(0,0)], [(4,3),(4,0)],[(1,3),(4,0)],[(3,0),(0,3)], [(1,0),(1,3)],[(3,3),(3,0)],[(0,1),(1,0)], [(4,2),(3,3)],[(4,1),(0,1)]]
12	[[(2,3),(4,1)]] [(0,2),(4,2)] = Goal	

Back path (zpětná cesta): (0,2) → (4,2) → (3,3) → (3,0) → (0,3) → (0,0)

Path (cesta, řešení úlohy): (0,0) → (0,3) → (3,0) → (3,3) →(4,2) → (0,2)

Poznámka:

Metodu BFS lze modifikovat testováním cílového stavu již při vygenerování uzlu. Pak se body 4 a 5 algoritmu u všech variant metody sloučí v bod jediný:

4. Vybraný uzel expandujte. Pokud je některý z jeho bezprostředních následníků uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak … a vraťte se na bod 2.

Takto modifikovaná metoda má poněkud nižší časovou i paměťovou náročnost, není však „kompatibilní“ s dále uváděnými metodami.

Hodnocení metody BFS

Metoda je **úplná** a **optimální**.

Označí-li se symbolem *b* maximální počet bezprostředních následníků všech expandovaných uzlů (tzv. faktor větvení) a symbolem *d* hloubka stromu, ve které se nachází řešení, pak maximální počty generovaných uzlů *N* jsou dány vztahy

$$N_{\mathrm{BFS}} = b^1 + b^2 + b^3 + (b^{\mathrm{d}+1} - b)$$
$$N_{\mathrm{BFSmodif}} = b^1 + b^2 + b^3 + b^d$$

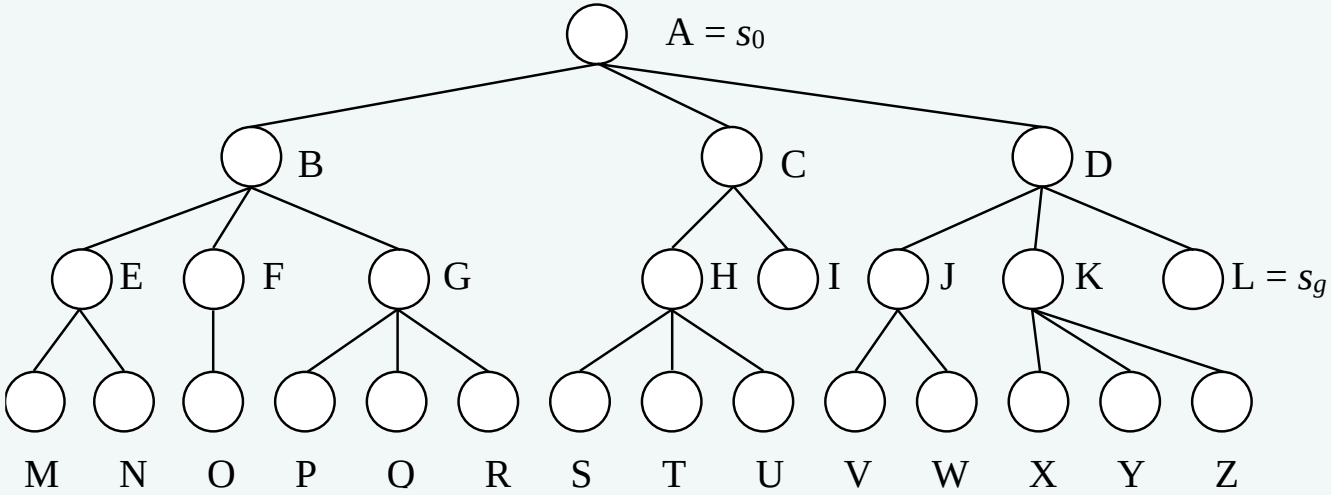
Protože všechny stavy zůstávají v paměti, tak časové i prostorové složitosti jsou stejné:

$$\boldsymbol{O_{\mathbf{BFS}}(b^{d+1})}$$
$$\boldsymbol{O_{\mathbf{BFSmodif}}(b^{d})}$$

DFS – Depth First Search (slepé prohledávání do hloubky)

1. Sestrojte **zásobník** OPEN (bude obsahovat všechny uzly určené k expanzi) a umístěte do něj počáteční uzel.
2. Je-li zásobník OPEN prázdný, pak úloha nemá řešení, a proto ukončete prohledávání jako neúspěšné. Jinak pokračujte.
3. Vyberte z vrcholu zásobníku OPEN první uzel.
4. Je-li vybraný uzel uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak pokračujte.
5. Vybraný uzel expandujte, všechny jeho bezprostřední následníky umístěte do zásobníku OPEN a vraťte se na bod 2.

Demonstrační stavový prostor

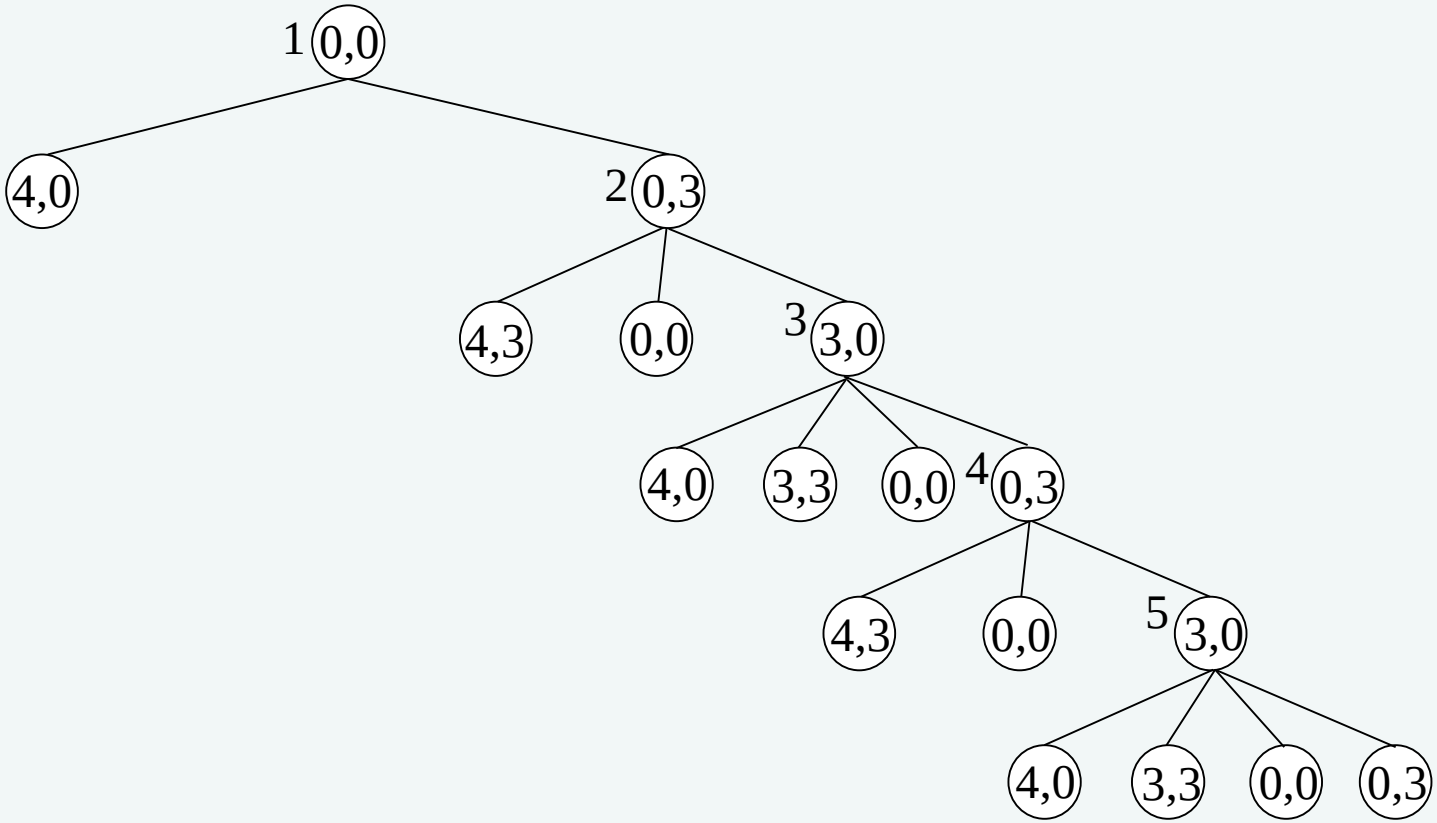


Krok	OPEN
0	[[A,nil]]
1	[[D,A,nil],[C,A,nil],[B,A,nil]]
2	[[L,D,A,nil],[K,D,A,nil],[J,D,A,nil],[C,A,nil],[B,A,nil]]
3	[[K,D,A,nil],[J,D,A,nil],[C,A,nil],[B,A,nil]]
	[L,D,A,nil] = Goal
	Path: A $\rightarrow$ D $\rightarrow$ L

Metoda DFS se na první pohled zdá být výhodnější, než metoda BFS, ale z následujícího snímku bude patrné, že tomu tak není.

DFS – Úloha dvou džbánů,

O = {→V, →M, V→, M→, M→V, V→M}

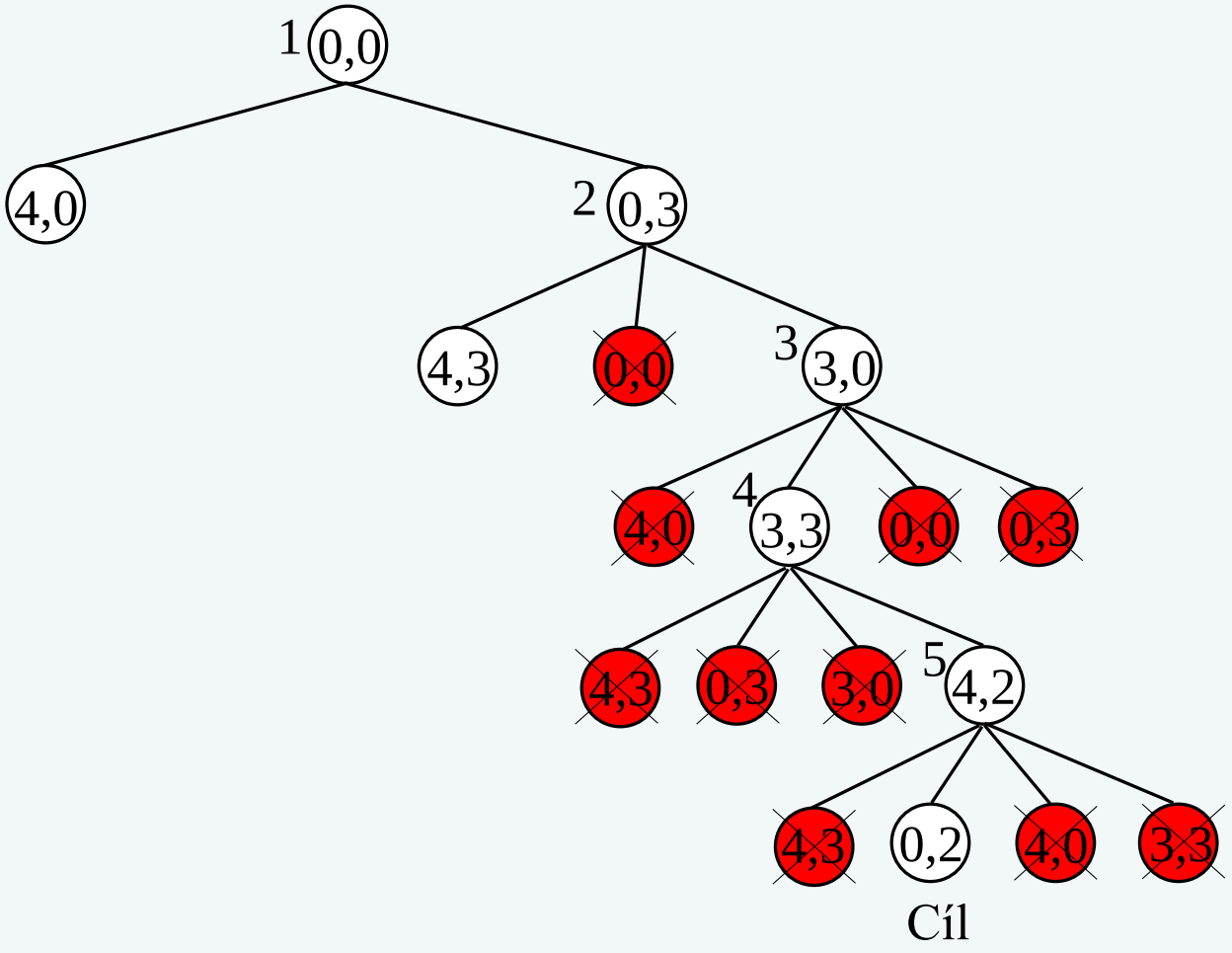


Prohledávání se zacyklí, metoda DFS nenalezne řešení!!!

Klasická metoda DFS není úplná (nemusí nalézt řešení, i když toto řešení existuje), a proto pro praktické použití je u této metody **nutná** podobná úprava, jako byla zmíněna u metody BFS – ta se opět týká bodu 5 algoritmu:

5. Vybraný uzel expandujte, do zásobníku OPEN umístěte všechny jeho bezprostřední následníky, kteří v tomto zásobníku ještě nejsou a kteří nejsou ani předky generovaného uzlu, a vraťte se na bod 2.

DFS s eliminací stejných stavů a předků v Open – úloha dvou džbánů, O = {→V, →M, V→, M→, M→V, V→M}



Krok	OPEN
0	[[(0,0),nil]]
1	[[(0,3),(0,0),nil],[(4,0),(0,0),nil]]
2	[[(3,0),(0,3),(0,0),nil],[(4,3),(0,3),(0,0),nil],[(4,0),(0,0),nil]]
3	[[(3,3),(3,0),(0,3),(0,0),nil],[(4,3),(0,3),(0,0),nil],[(4,0),(0,0),nil]]
4	[[(4,2),(3,3),(3,0),(0,3),(0,0),nil],[(4,3),(0,3),(0,0),nil],[(4,0),(0,0),nil]]
5	[[(0,2),(4,2),(3,3),(3,0),(0,3),(0,0),nil],[(4,3),(0,3),(0,0),nil], [(4,0),(0,0),nil]]
6	[[(4,3),(0,3),(0,0),nil],[(4,0),(0,0),nil]]
	[(0,2),(4,2),(3,3),(3,0),(0,3),(0,0),nil] = Goal

Pozn.: Ověřte si, že záměna prvních dvou operátorů
O = {→M, →V, V→, M→, M→V, V→M}
vede k řešení, které není optimální (je v hloubce 7)!

Hodnocení metody DFS

Původní metoda **není úplná** a není tedy **ani optimální**.
Modifikovaná metoda (s eliminací stejných stavů a předků v Open) **je úplná**, ale **není optimální**.

Označíme-li maximální počet možných různých stavů symbolem *m*, pak platí:

pro prostorovou složitost:

$O_{DFS}(\infty)$, $O_{DFSmodif}(m)$

a pro časovou složitost:

$O_{DFS}(\infty)$, $O_{DFSmodif}(b^m)$

DLS – Depth Limited Search (slepé prohledávání do omezené hloubky)

1. Sestrojte **zásobník** OPEN (bude obsahovat všechny uzly určené k expanzi) a umístěte do něj počáteční uzel.
2. Je-li zásobník OPEN prázdný, pak úloha nemá řešení, a proto ukončete prohledávání jako neúspěšné. Jinak pokračujte.
3. Vyberte z vrcholu zásobníku OPEN první uzel.
4. Je-li vybraný uzel uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak pokračujte.
5. Je-li hloubka vybraného uzlu menší než zadaná maximální hloubka, tak tento uzel expandujte a do zásobníku OPEN umístěte všechny jeho bezprostřední následníky.
6. Vraťte se na bod 2.

Pozn.: bod 5 je opět možné doplnit: …do zásobníku OPEN umístěte všechny jeho bezprostřední následníky, kteří v tomto zásobníku ještě nejsou a kteří nejsou ani předky generovaného uzlu.

Hodnocení metody DLS

Metoda **není úplná a tedy ani optimální** (a to i s případnou eliminací stejných stavů i předků v Open).

Označíme-li maximální prohledávanou hloubku stromu (limit) hodnotou l, pak platí:

pro prostorovou složitost:

$$O_{DLS}(b \cdot l)$$

a pro časovou složitost:

$$O_{DLS}(b^l)$$

IDS – Iterative Deepening Search (slepé prohledávání do omezené hloubky s postupným zanořováním)

1. Nastavte aktuální maximální hloubku prohledávání na hodnotu 1.
2. Zavolejte proceduru DLS s omezením na aktuální hloubku.
3. Skončí-li procedura DLS úspěchem, ukončete prohledávání jako úspěšné a vraťte cestu nalezenou procedurou DLS.
4. Skončí-li procedura DLS neúspěchem, pak
   a) pokud v proceduře DLS nebyl alespoň jeden uzel expandován z důvodu dosažení maximální hloubky (bod 5. procedury DLS) inkrementujte aktuální maximální hloubku prohledávání a vraťte se na bod 2.
   b) jinak ukončete prohledávání jako neúspěšné (úloha nemá řešení).

Hodnocení metody IDS

Metoda **je úplná i optimální.**

Počet generovaných uzlů metodou IDS je dán vztahem

$N_{\mathrm{IDS}} = d \cdot b^1 + (d-1) \cdot b^2 + (d-2) \cdot b^3 + \ldots + 1 \cdot b^{\mathrm{d}}$

a proto platí:

pro prostorovou složitost:

$\boldsymbol{O_{\mathrm{IDS}}(b \cdot d)}$

a pro časovou složitost:

$\boldsymbol{O_{\mathrm{IDS}}(b^d)}$

Porovnání BFS a IDS

Maximální počty generovaných uzlů N jsou dány vztahy:

$$N_{\mathrm{BFS}} = b^1 + b^2 + b^3 + (b^{d+1} - b)$$
$$N_{\mathrm{BFSmodif}} = b^1 + b^2 + b^3 + b^d$$
$$N_{\mathrm{IDS}} = d \cdot b^1 + (d-1) \cdot b^2 + (d-2) \cdot b^3 + \ldots + 1 \cdot b^d$$

Konkrétně, například pro $b = 10$ a $d = 5$:

$$N_{\mathrm{BFS}} = 1111100$$
$$N_{\mathrm{BFSmodif}} = 111110$$
$$N_{\mathrm{IDS}} = 123450$$

Praktická ukázka časové a paměťové náročnosti (b = 10, 10000 stavů(uzlů)/s, 100 B/stav), přibližné maximální hodnoty:

d = 6, l = 20	BFS/BFS_{modif}	DLS	IDS
Počet generovaných stavů	$10^7/10^6$	10^{20}	10^6
Potřebný čas	19/1.9 min	$3.2 \cdot 10^8$ roků	2.1 min
Potřebná paměť	1.1/0.11 GB	20 kB	6 kB

d = 8, l = 20	BFS/BFS_{modif}	DLS	IDS
Počet generovaných stavů	$10^9/10^8$	10^{20}	10^8
Potřebný čas	31/3.1 hod	$3.2 \cdot 10^8$ roků	3.4 hod
Potřebná paměť	111/11 GB	20 kB	8 kB

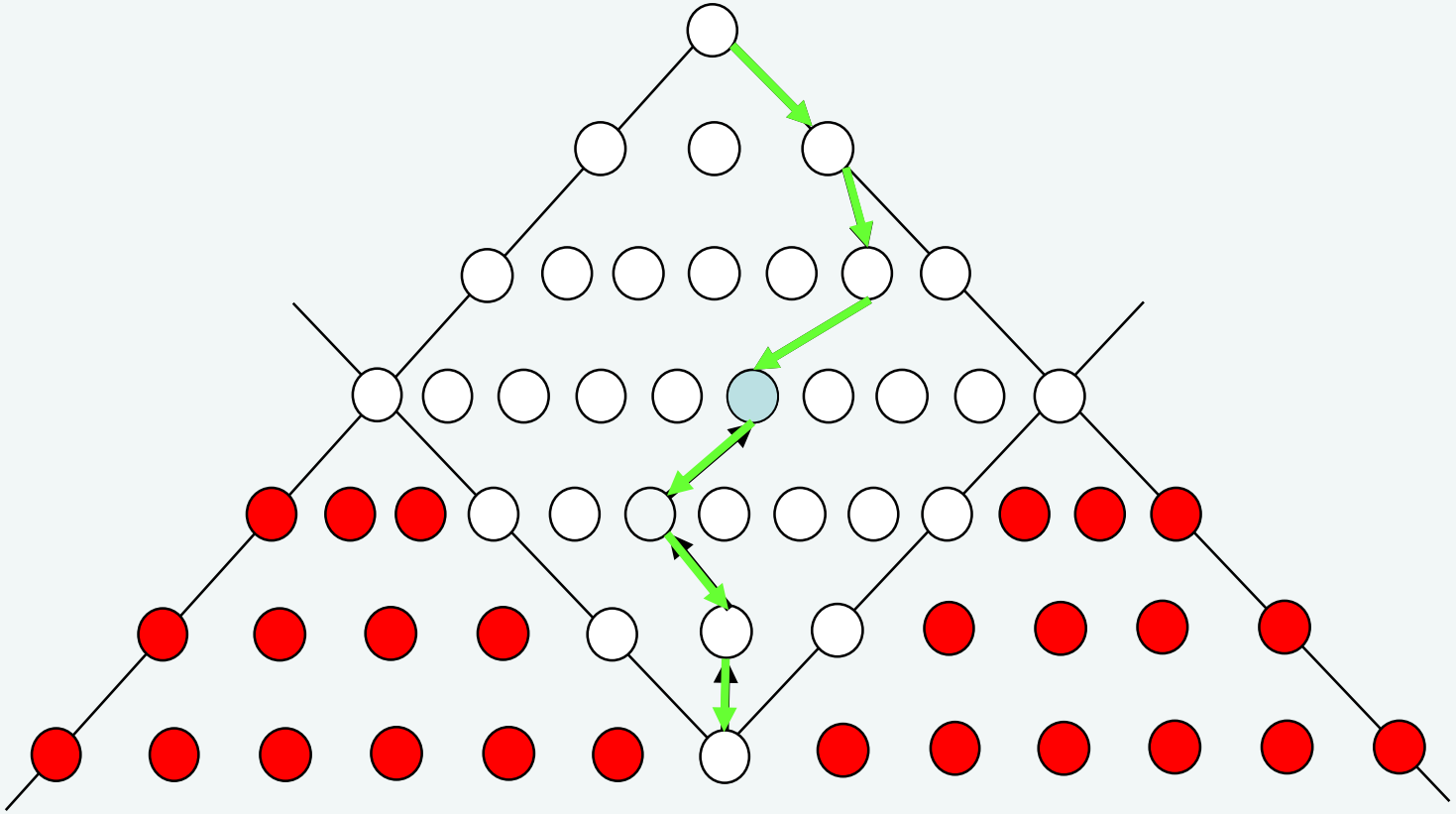
d = 10, l = 20	BFS/BFS_{modif}	DLS	IDS
Počet generovaných stavů	$10^{11}/10^{10}$	10^{20}	10^{10}
Potřebný čas	128/13 dnů	$3.2 \cdot 10^{8}$ roků	14.3 dnů
Potřebná paměť	11.1/1.1 TB	20 kB	10 kB

d = 12, l = 20	BFS/BFS_{modif}	DLS	IDS
Počet generovaných stavů	$10^{13}/10^{12}$	10^{20}	10^{12}
Potřebný čas	35/3.5 roku	$3.2 \cdot 10^{8}$ roků	3.9 roku
Potřebná paměť	1.1/0.11 PB	20 kB	12 kB

Pozn.: je důležité si uvědomit, že do časové náročnosti je navíc nutné zahrnout i časy porovnání expandovaných stavů s cílovým stavem!

BS - Bidirectional Search (obousměrné BFS)

Metodu lze použít pouze pro řešení úloh s reverzibilními operátory (lze ji například použít pro řešení úlohy Loydovy osmičky, ale nelze ji použít pro řešení úlohy dvou džbánů).

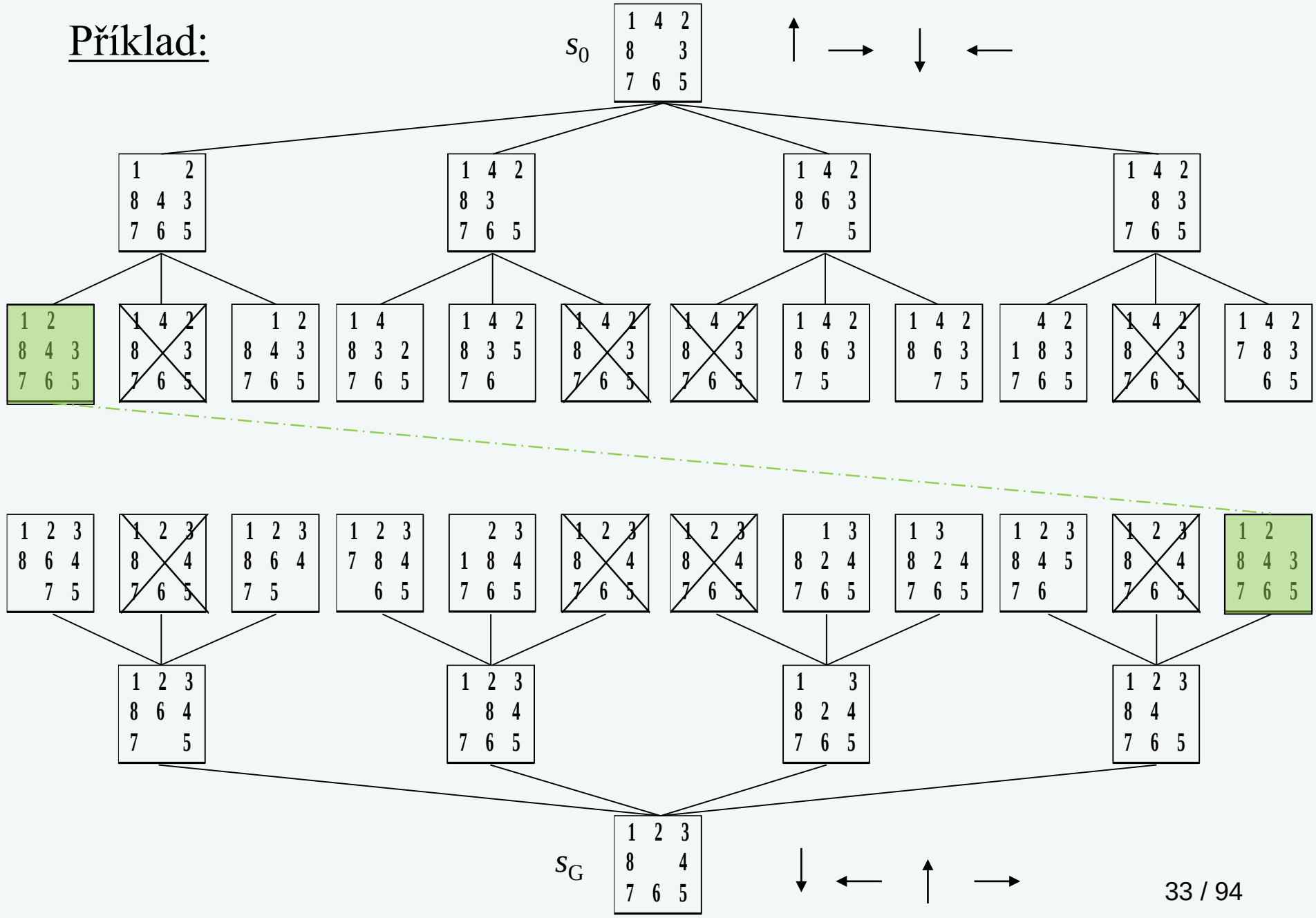


Metoda **je úplná i optimální** s prostorovou i časovou složitostí $\boldsymbol{O_{BS}(2 \cdot b^{d/2}) \sim O(b^{d/2})}$

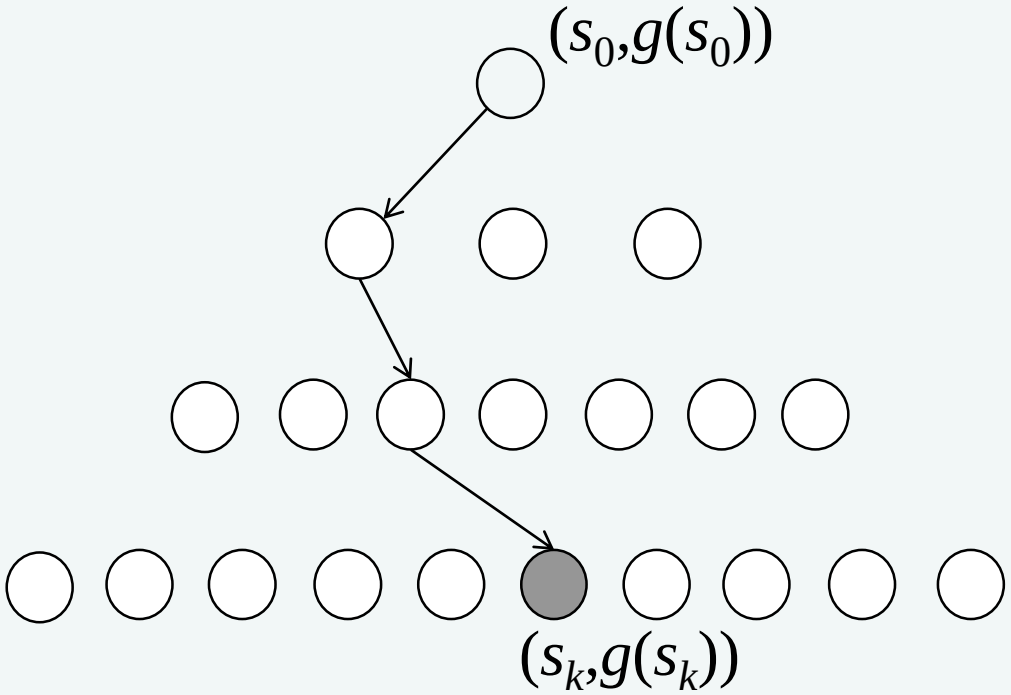
Konkrétní příklady složitosti:

	počet generovaných uzlů:	počet porovnání:
b=10; d=6		
BFS	11 111 100	1 111 111
BFS_{modif}	1 111 110	1 111 111
BS	2 220	1 111 111
b=10; d=7		
BFS	111 111 100	11 111 111
BFS_{modif}	11 111 110	11 111 111
BS	12 220	11 111 111

Příklad:



UCS – Uniform Cost Search (slepé prohledávání do šířky s respektováním cen přechodů



$g(s_k) > g(s_{k-1})$ … cena přechodu musí být kladná!

UCS

1. Sestrojte **seznam** OPEN (bude obsahovat všechny uzly určené k expanzi) a umístěte do něj počáteční uzel **včetně jeho (nulového) ohodnocení**.
2. Je-li seznam OPEN prázdný, pak úloha nemá řešení, a ukončete proto prohledávání jako neúspěšné. Jinak pokračujte.
3. Vyberte ze seznamu OPEN uzel s **nejnižším ohodnocením**.
4. Je-li vybraný uzel uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak pokračujte.
5. Vybraný uzel expandujte, všechny jeho bezprostřední následníky **včetně jejich ohodnocení** umístěte do seznamu OPEN a vraťte se na bod 2.

Dvě možné úpravy UCS jsou podobné variantám BFS a opět spočívají v úpravě bodu 5 algoritmu:

5. Vybraný uzel expandujte, všechny jeho bezprostřední následníky umístěte do seznamu OPEN, a to včetně jejich ohodnocení. Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponechte pouze uzel s nejlepším ohodnocením, ostatní ze seznamu OPEN vyškrtněte a vraťte se na bod 2.

resp.

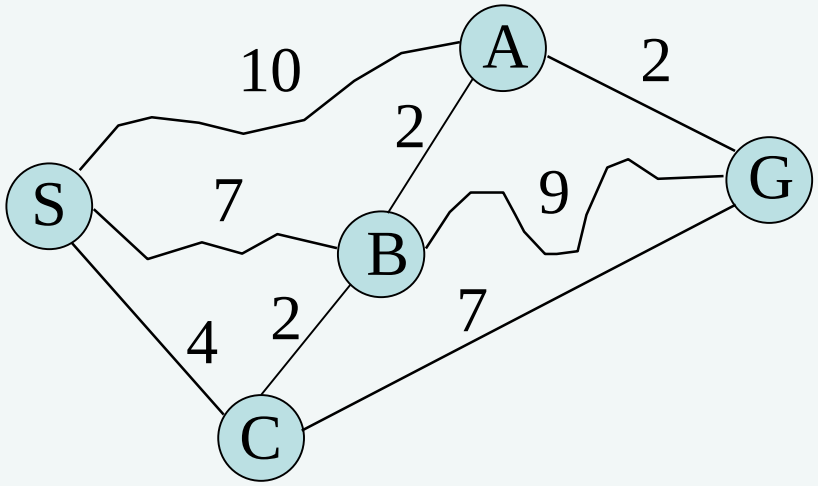
5. Vybraný uzel expandujte, všechny jeho bezprostřední následníky, kteří nejsou jeho předky, umístěte do seznamu OPEN, a to včetně jejich ohodnocení. Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponechte pouze uzel s nejlepším ohodnocením, ostatní ze seznamu OPEN vyškrtněte a vraťte se na bod 2.

Podobně jako u BFS nejsou tyto úpravy významné a prakticky se používá také jen verze se seznamem CLOSED.

UCS – verze se seznamem CLOSED

1. Sestrojte dva prázdné **seznamy**, OPEN (bude obsahovat uzly určené k expanzi) a CLOSED (bude obsahovat seznam expandovaných uzlů). Do seznamu OPEN umístěte počáteční uzel **včetně jeho ohodnocení**.
2. Je-li seznam OPEN prázdný, pak úloha nemá řešení, a proto ukončete prohledávání jako neúspěšné. Jinak pokračujte.
3. Vyberte ze seznamu OPEN uzel s **nejnižším ohodnocením**.
4. Je-li vybraný uzel uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak pokračujte.
5. Vybraný uzel expandujte a jeho bezprostřední následníky, kteří nejsou v seznamu CLOSED, umístěte do seznamu OPEN (**včetně jejich ohodnocení**). Expandovaný uzel umístěte do seznamu CLOSED. Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponechte pouze uzel s nejlepším ohodnocením, ostatní ze seznamu OPEN vyškrtněte a vraťte se na bod 2.

Příklad na UCS – nalezení nejkratší cesty mezi místy S a G:

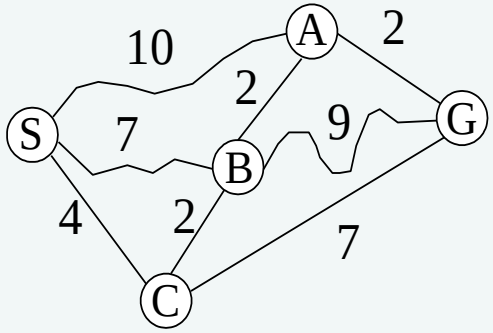


Vzdálenosti (ceny přechodů) v km nechť jsou následující:

S - A = 10, S - B = 7, S - C = 4, A - B = 2,

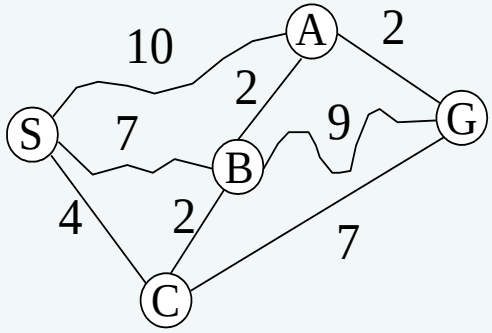
B - C = 2, A - G = 2, B - G = 9 C - G = 7

UCS základní varianta



Krok	OPEN
0	[[(S,nil),0]]
1	[[(A,S,nil),10],[(B,S,nil),7],[(C,S,nil),4]]
2	[[(A,S,nil),10],[(B,S,nil),7],[(S,C,S,nil),8],[(B,C,S,nil),6],[(G,C,S,nil),11]]
3	[[(A,S,nil),10],[(B,S,nil),7],[(S,C,S,nil),8],[(G,C,S,nil),11],[(S,B,C,S,nil),13], [(A,B,C,S,nil),8],[(C,B,C,S,nil),8],[(G,B,C,S,nil),15]]
4	[[(A,S,nil),10],[(S,C,S,nil),8],[(G,C,S,nil),11],[(S,B,C,S,nil),13], [(A,B,C,S,nil),8],[(C,B,C,S,nil),8],[(G,B,C,S,nil),15],[(S,B,S,nil),14], [(A,B,S,nil),9],[(C,B,S,nil),9], [(G,B,S,nil),16]]
5	[[(A,S,nil),10],[(G,C,S,nil),11],[(S,B,C,S,nil),13],[(A,B,C,S,nil),8], [(C,B,C,S,nil),8],[(G,B,C,S,nil),15],[(S,B,S,nil),14],[(A,B,S,nil),9], [(C,B,S,nil),9],[(G,B,S,nil),16],[(A,S,C,S,nil),18],[(B,S,C,S,nil),15], [(C,S,C,S,nil),12]]
6	…

UCS se seznamem CLOSED



Krok	OPEN	CLOSED
0	[[(S,nil),0]]	[ ]
1	[[(A,S),10],[(B,S),7],[(C,S),4]]	[[(S,nil),0]]
2	[[(A,S),10],~~[(B,S),7]~~,~~[(S,C),8]~~, [(B,C),6],[(G,C),11]]	[[(S,nil),0],[(C,S),4]]
3	[~~[(A,S),10]~~,[(G,C),11],~~[(S,B),13]~~, [(A,B),8],~~[(C,B),8]~~,~~[(G,B),15]~~]	[[(S,nil),0],[(C,S),4],[(B,C),6]]
4	[~~[(G,C),11]~~,~~[(S,A),18]~~,~~[(B,A),10]~~, [(G,A),10]]	[[(S,nil),0],[(C,S),4],[(B,C),6], [(A,B),8]]
5	[ ]	

$\Rightarrow$ [(G,A),10] = Goal

$\Rightarrow$ optimální cesta = S $\rightarrow$ C $\rightarrow$ B $\rightarrow$ A $\rightarrow$ G, vzdálenost = 10

Metoda UCS je **úplná a optimální**.

Časová i prostorová složitost metody je dána cenou optimálního řešení C^* dělenou nejmenším přírůstkem ceny mezi dvěma uzly Δc_{min}

$$O_{UCS}(b^{C^* / \Delta cmin})$$

Pozn.: Metoda je nevýhodná pro případy, kdy optimální řešení leží na málo cestách s vysokou cenou – prohledává se pak zbytečně mnoho cest s nízkými cenami (například při hledání cesty z Brna do Plzně po dálnici přes Prahu by se prohledávalo neúnosné množství kratších cest počínaje cestami Brno – Česká, Brno – Kuřim, Brno – Lelekovice, Brno – Jehnice, atd.).

Backtracking (slepé prohledávání se zpětným navracením)

1. Sestrojte **zásobník** OPEN (bude obsahovat uzly určené k expanzi) a umístěte do něj počáteční uzel.
2. Je-li zásobník OPEN prázdný, pak úloha nemá řešení, a proto ukončete prohledávání jako neúspěšné. Jinak pokračujte.
3. Jde-li na uzel na vršku zásobníku aplikovat první/další operátor, tak tento operátor aplikujte a pokračujte bodem 4, v opačném případě odstraňte testovaný uzel z vrcholu zásobníku a vraťte se na bod 2.
4. Je-li vygenerovaný uzel (tj. uzel vzniklý aplikací operátoru na uzel na vršku zásobníku) uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak uložte nový uzel na vršek zásobníku a vraťte se na bod 2.

Metoda zpětného navracení (Backtracking) je velmi podobná metodě DFS, ale místo expanze uzlu (tj. generování všech jeho následníků) generuje pouze jediného následníka - nejprve prvního následníka a při případných návratech pak postupně další následníky.

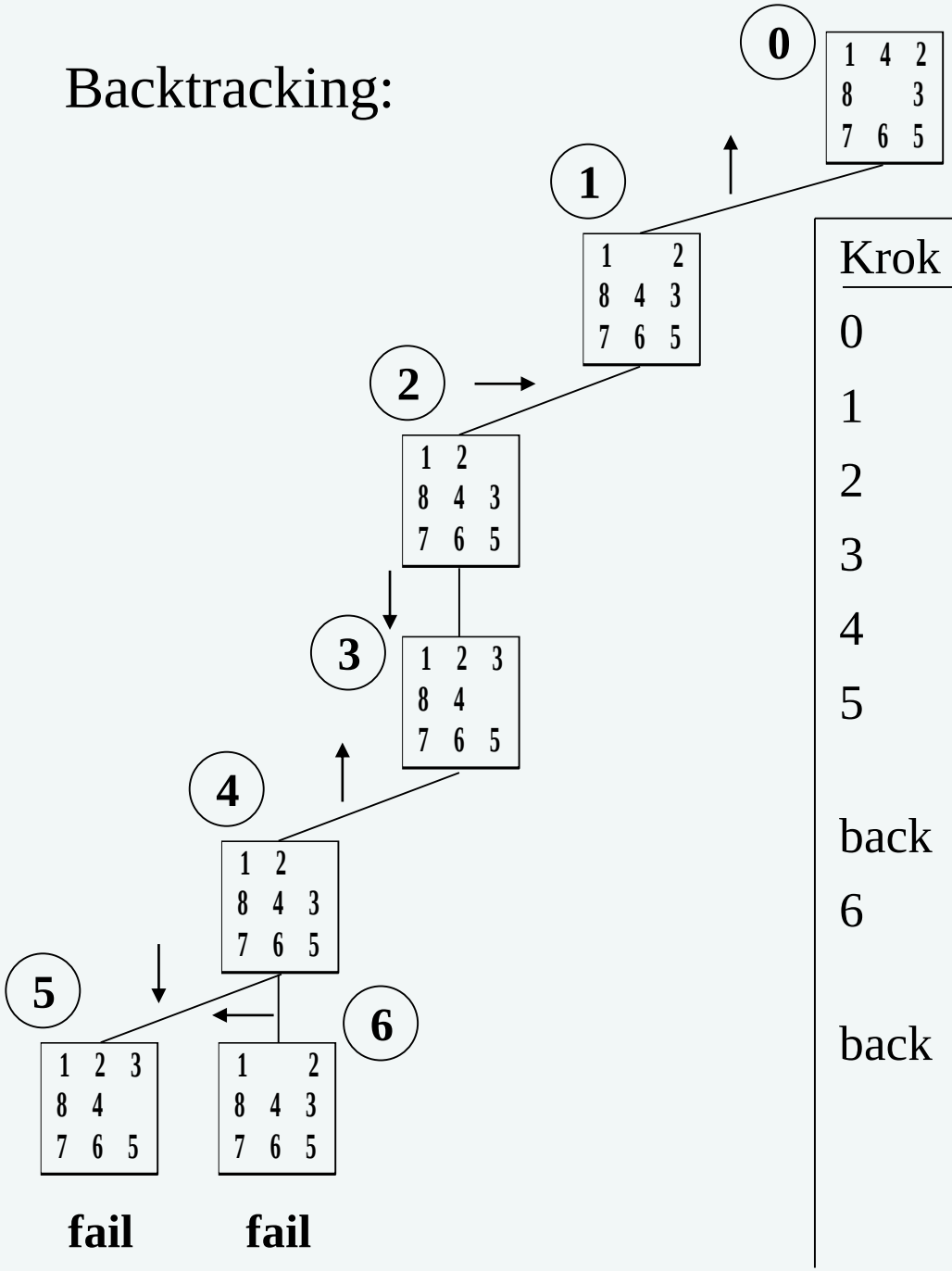
Modifikace metody spočívá v úpravě bodu 4. Nový uzel se do zásobníku OPEN neukládá, pokud se již v tomto zásobníku nachází (generovaný nový uzel je již předchůdcem právě expandovaného uzlu na vršku zásobníku).

Pro časovou složitost, prostorovou složitost, úplnost a optimálnost platí stejné závěry jako pro metodu DFS, resp. pro modifikovanou metodu DFS.

Metodu zpětného navracení lze dále upravit tak, že se podobně jako u metody DLS zadá maximální hloubka prohledávání (pokud uzel leží v maximální hloubce, pak v bodu 3 nelze aplikovat žádný operátor) a následně tím, že podobně jako u metody IDS se tato hloubka postupně zvyšuje. Pro hodnocení takto upravené metody pak platí všechny závěry uvedené u metod DLS a IDS.

Následující příklad uvažuje jak dříve zmíněnou modifikaci metody spočívající v úpravě bodu 4, tak i omezenou hloubku prohledávání.

Backtracking:

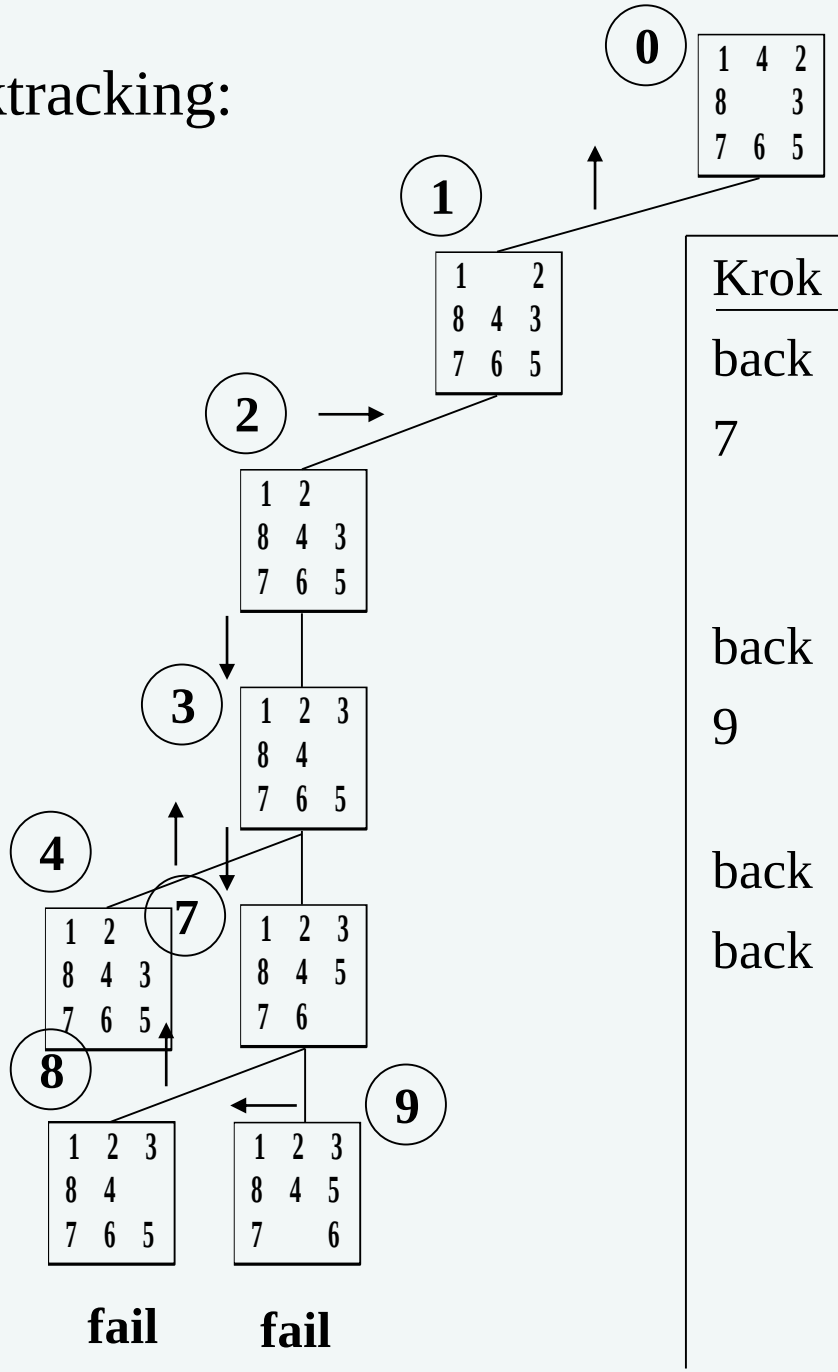


Max. hloubka = 5

Operátory: ↑ → ↓ ←

Krok	Open
0	[(0,nil)]
1	[(1,nil),(0,↑)]
2	[(2,nil),(1,→),(0,↑)]
3	[(3,nil),(2,↓),(1,→),(0,↑)]
4	[(4,nil),(3,↑),(2,↓),(1,→),(0,↑)]
5	[(5,nil),(4,↓),(3,↑),(2,↓),(1,→), (0,↑)] fail (max. hloubka)
back	[(4,↓),(3,↑),(2,↓),(1,→),(0,↑)]
6	[(6,nil),(4,←),(3,↑),(2,↓),(1,→), (0,↑)] fail (max. hloubka)
back	[(4,←),(3,↑),(2,↓),(1,→),(0,↑)]

Backtracking:

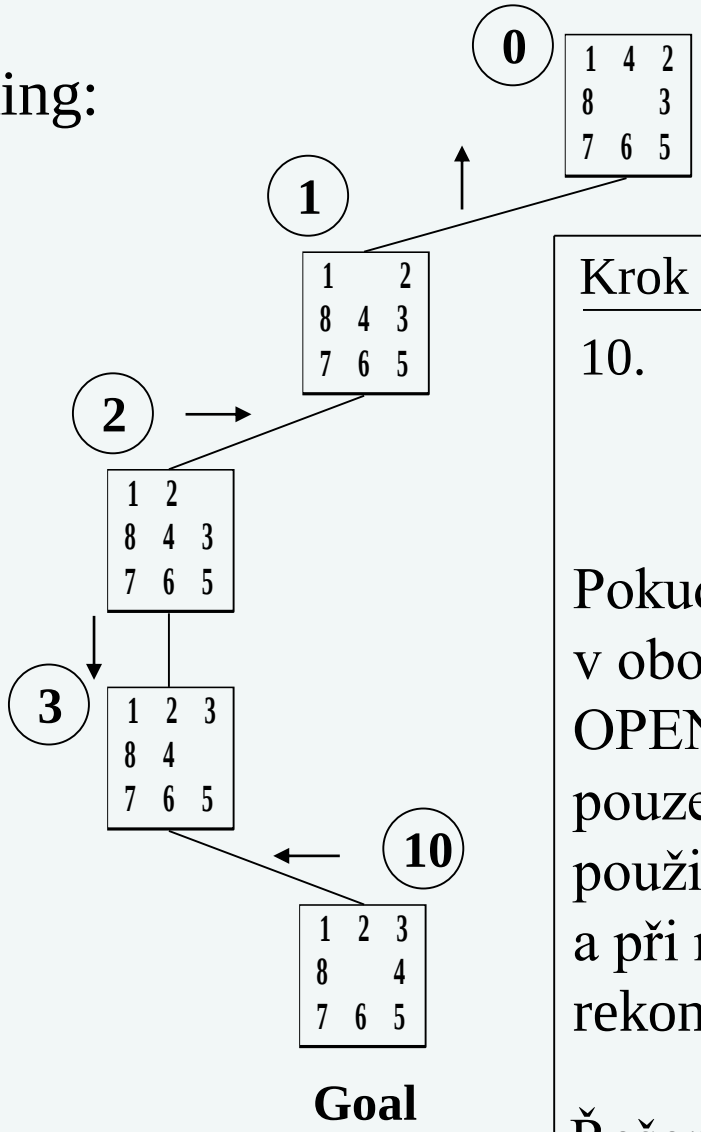


Max. hloubka = 5

Operátory: ↑ → ↓ ←

Krok	Open
back	[(3,↑),(2,↓),(1,→),(0,↑)]
7	[(7,nil),(3,↓),(2,↓),(1,→),(0,↑)]
	[(8,nil),(7,↑),(3,↓),(2,↓),(1,→), (0,↑)] fail (max. hloubka)
back	[(7,↑),(3,↓),(2,↓),(1,→),(0,↑)]
9	[(9,nil),(7,←),(3,↓),(2,↓),(1,→), (0,↑)] fail (max. hloubka)
back	[(7,←),(3,↓),(2,↓),(1,→),(0,↑)]
back	[(3,↓),(2,↓),(1,→),(0,↑)]

Backtracking:



Goal

Max. hloubka = 5

Operátory: ↑ → ↓ ←

Krok	Open
10.	[(10,nil),(3,←),(2,↓),(1,→), (0,↑)] cíl (goal)

Pokud jsou operátory aplikovatelné v obou směrech, není nutné ukládat do OPEN všechny stavy; stačí ukládat pouze aktuální stav se seznamem použitých operátorů ([10,←,↓,→,↑]) a při návratech předcházející stavy rekonstruovat.

Řešení úlohy se pak jednoduše popíše takto: [0,↑,→,↓,←]

Metody pro řešení CSP

- Backtracking
- Forward Checking
- Min-conflict

Backtracking pro CSP

Metodu zpětného navracení (Backtracking) lze k řešení CSP použít velmi snadno - pokud aplikace operátoru vede na stav porušující omezující podmínky, pak je tento operátor považován za neaplikovatelný (bod 3 algoritmu).

Metoda je **úplná** (a každá úplná metoda je pro CSP **optimální**). Označíme-li symbolem *n* počet proměnných a symbolem *m* maximální počet přiřaditelných hodnot, pak platí:

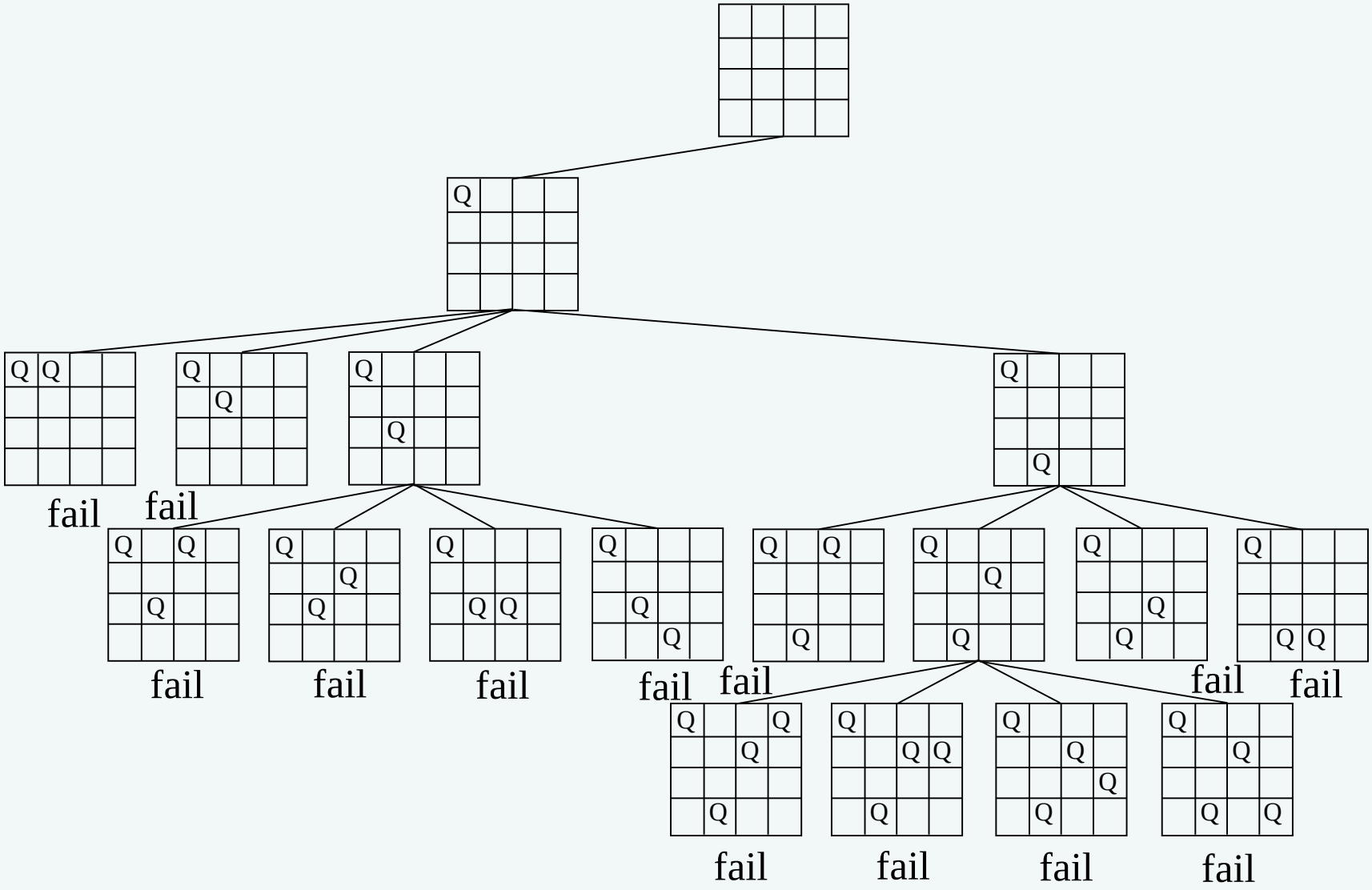
Pro prostorovou složitost:

$$O_{bck\text{-}CSP}(n)$$

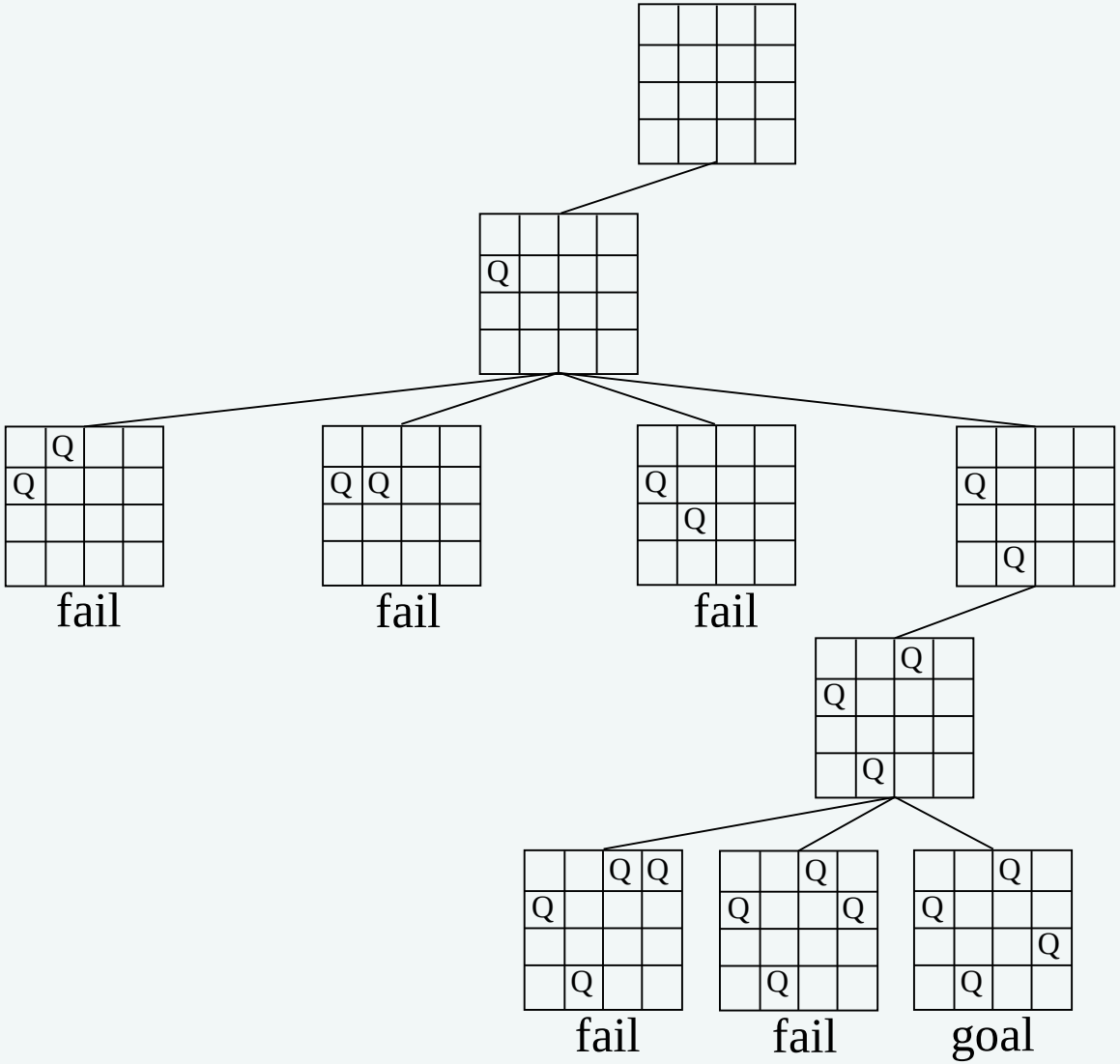
Pro časovou složitost:

$$O_{bck\text{-}CSP}(m^n)$$

Backtracking pro CSP - příklad 4 dam:



Backtracking pro CSP - příklad 4 dam:



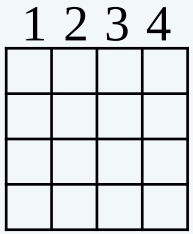
Forward Checking (pouze pro řešení CSP)

- Přiřaďte každé proměnné i ($i = 1, \ldots, n$) množinu přípustných hodnot S_i.
- Zavolejte proceduru Forward_checking(1).

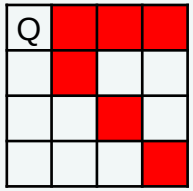
Procedura Forward_checking(int i)

1. Odstraňte první hodnotu z množiny S_i a přiřaďte tuto hodnotu proměnné x_i, nechť X je nový stav (je dán hodnotami všech proměnných x_i, $i = 1, \ldots, n$).
2. Je-li X cílovým stavem, ukončete proceduru úspěchem (vraťte stav X), jinak pokračujte.
3. Odstraňte z množin S_j ($j = i + 1, ..., n$) všechny hodnoty, které jsou v konfliktu s dosud přiřazenými hodnotami.
4. Jestliže je nějaká množina S_j ($j = i + 1, ..., n$) prázdná, obnovte původní stav množin S_j (tj. stav před bodem 3) a přejděte na bod 7. Jinak pokračujte.
5. Zavolejte proceduru Forward_checking ($i + 1$).
6. Skončí-li předchozí procedura úspěchem, skončete také úspěchem (vraťte vrácený stav). Jinak pokračujte.
7. Není-li množina S_i prázdná, vraťte se na bod 1. Jinak skončete neúspěchem.

Forward checking (příklad 4 dam):

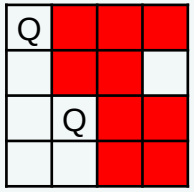


$S_1 = \{1,2,3,4\}, S_2 = \{1,2,3,4\}, S_3 = \{1,2,3,4\}, S_4 = \{1,2,3,4\}$



$x_1 = 1$

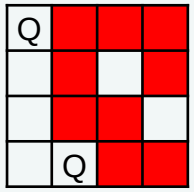
$S_1 = \{2,3,4\}, S_2 = \{3,4\}, S_3 = \{2,4\}, S_4 = \{2,3\}$



$x_1 = 1, x_2 = 3$

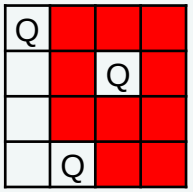
$S_1 = \{2,3,4\}, S_2 = \{4\}, S_3 = \{\ \}, S_4 = \{2\}$ fail !

$S_3 = \{2,4\}, S_4 = \{2,3\}$



$x_1 = 1, x_2 = 4$

$S_1 = \{2,3,4\}, S_2 = \{\}, S_3 = \{2\}, S_4 = \{3\}$

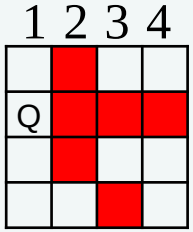


$x_1 = 1, x_2 = 4, x_3 = 2$

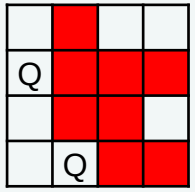
$S_1 = \{2,3,4\}, S_2 = \{\}, S_3 = \{\}, S_4 = \{\}$ fail !

$S_2 = \{1,2,3,4\}, S_3 = \{1,2,3,4\}, S_4 = \{1,2,3,4\}$

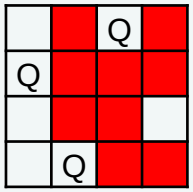
Forward checking (pokračování příkladu):



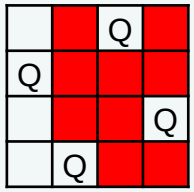
$x_1 = 2$,
$S_1 = \{3,4\}$, $S_2 = \{4\}$, $S_3 = \{1,3\}$, $S_4 = \{1,3,4\}$



$x_1 = 2$, $x_2 = 4$,
$S_1 = \{3,4\}$, $S_2 = \{\}$, $S_3 = \{1\}$, $S_4 = \{1,3\}$



$x_1 = 2$, $x_2 = 4$, $x_3 = 1$,
$S_1 = \{3,4\}$, $S_2 = \{\}$, $S_3 = \{\}$, $S_4 = \{3\}$



$x_1 = 2$, $x_2 = 4$, $x_3 = 1$, $x_4 = 3$,
$S_1 = \{3,4\}$, $S_2 = \{\}$, $S_3 = \{\}$, $S_4 = \{\}$
$X = (x_1, x_2, x_3, x_4) = (2, 4, 1, 3)$ Goal !

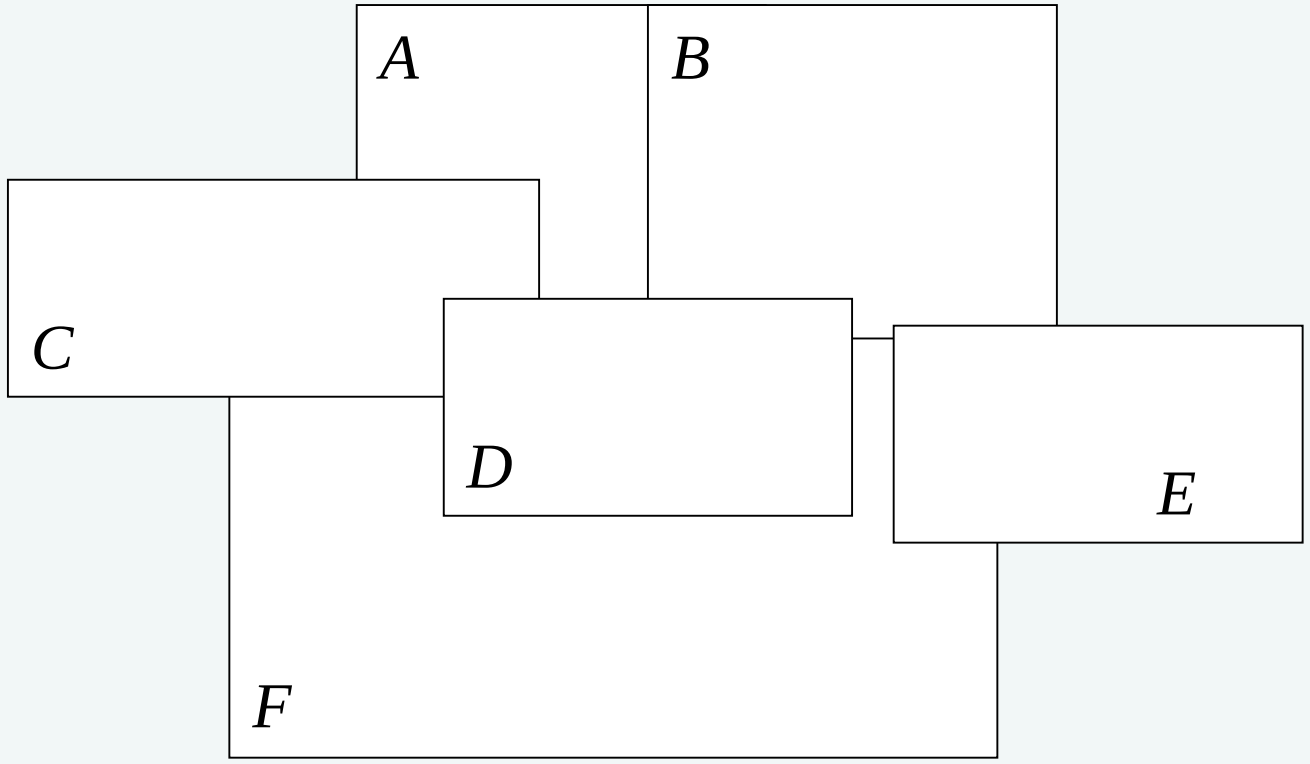
Heuristiky pro Forward checking:

- Kterou proměnnou z dosud volných proměnných vybrat pro přiřazení hodnoty?
  - Most-constrained-variable heuristic:
    Vyberte proměnnou s nejmenším počtem přípustných hodnot.
  - Most-constraining-variable heuristic:
    Vyberte proměnnou, která má největší vliv na omezení zbývajících volných proměnných.
- Kterou z přípustných hodnot vybrané proměnné přiřadit?
  - Least-constraining-value heuristic:
    Vybrané proměnné přiřaďte hodnotu, která vylučuje nejméně hodnot proměnných, které mají společná omezení s vybranou proměnnou.

Forward checking – příklad barvení mapy třemi barvami:

Největší vliv na omezení zbývajících proměnných má proměnná *D*. Výběr barvy proměnné *D* nemá na ostatní proměnné žádný vliv.

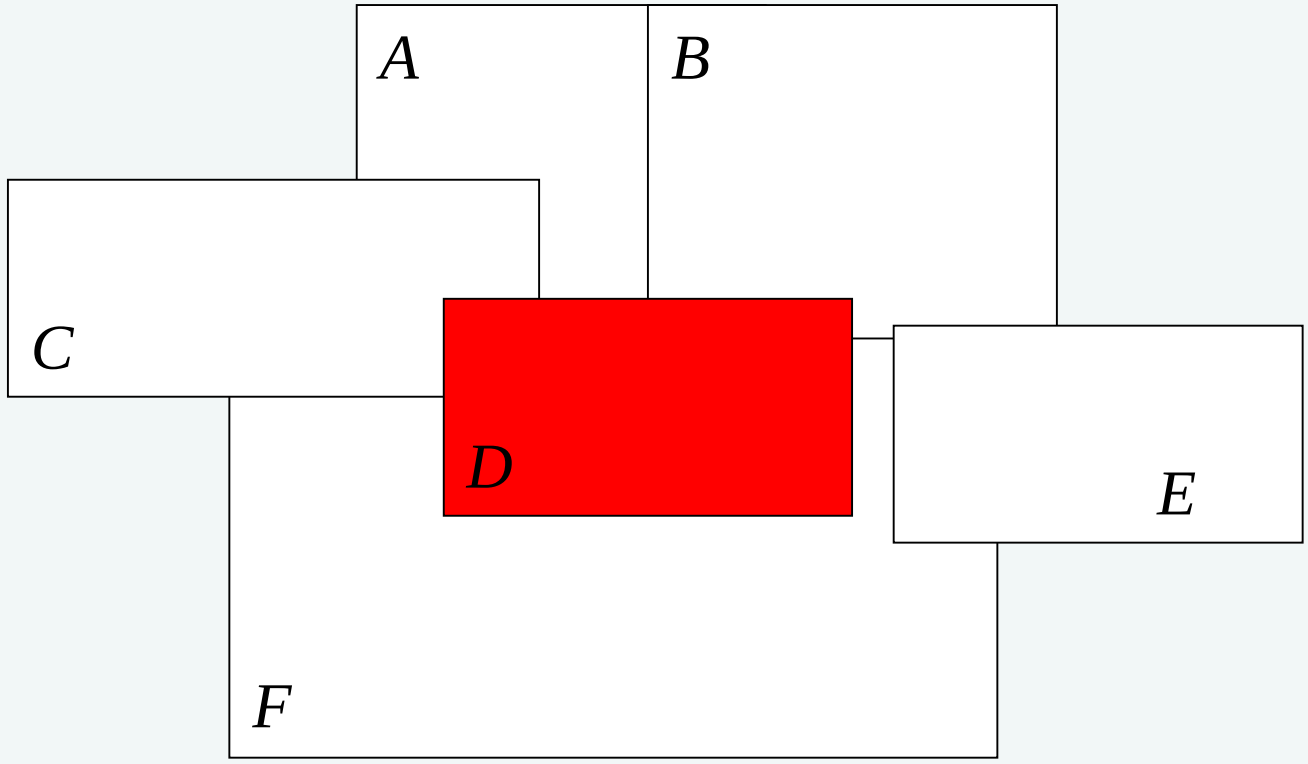
S_A = {R,G,B}
S_B = {R,G,B}
S_C = {R,G,B}
S_D = {R,G,B}
S_E = {R,G,B}
S_F = {R,G,B}



Forward checking – příklad barvení mapy třemi barvami:

Největší vliv na omezení zbývajících proměnných má nyní proměnná *F*. Výběr barvy proměnné *F* (G nebo B) nemá na zbývající proměnné žádný vliv.

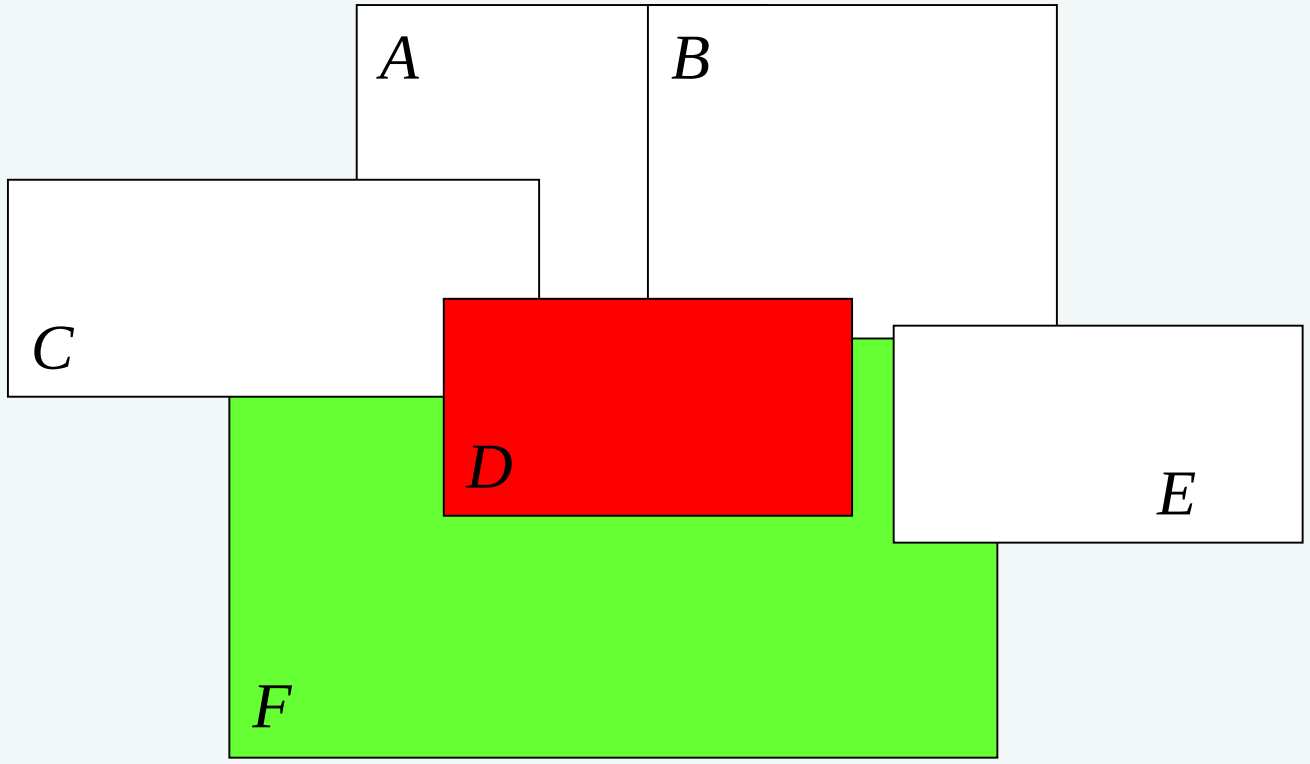
S_A = {G,B}
S_B = {G,B}
S_C = {G,B}
S_D = {G,B}
S_E = {R,G,B}
S_F = {G,B}



Forward checking – příklad barvení mapy třemi barvami:

Největší vliv na omezení zbývajících proměnných mají nyní proměnné *A* a *B*. *Z* těchto dvou proměnných má méně přípustných hodnot proměnná *B*.

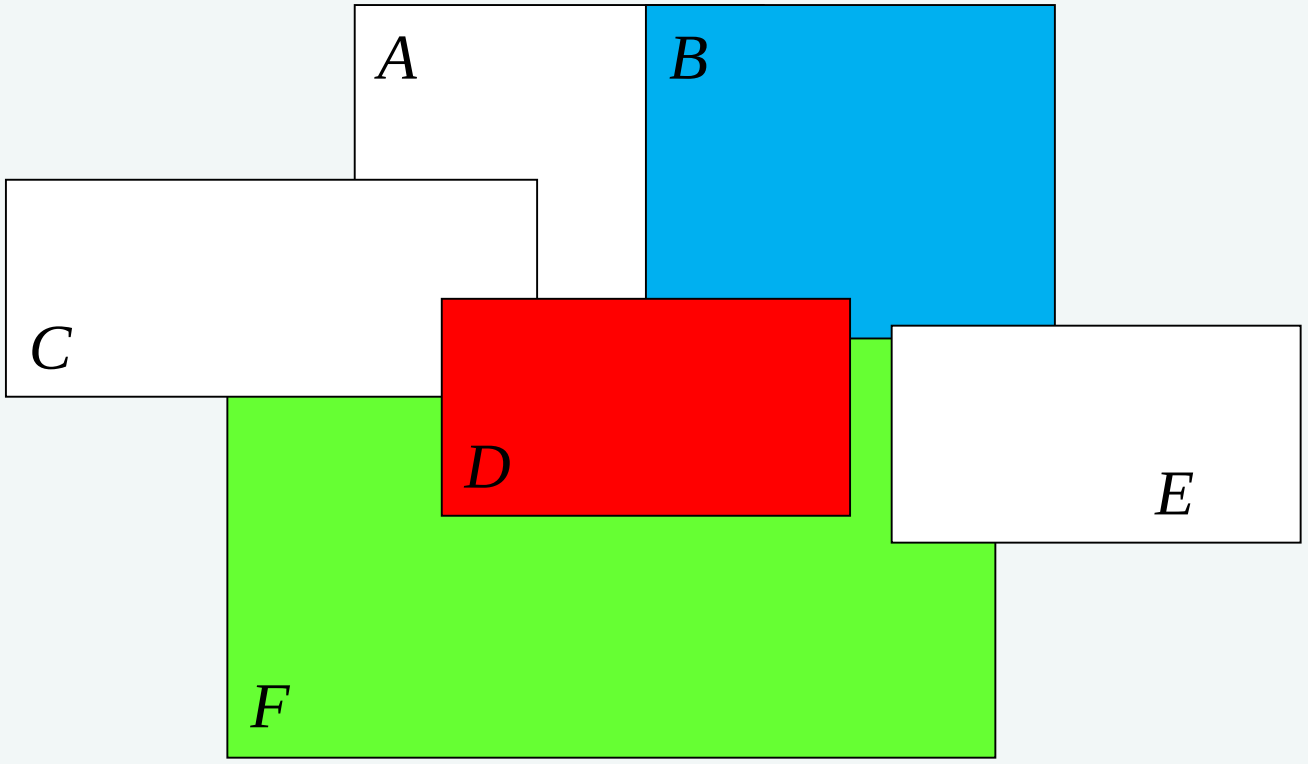
S_A = {G,B}
S_B = {B}
S_C = {B}
S_D = {G,B}
S_E = {R,B}
S_F = {B}



Forward checking – příklad barvení mapy třemi barvami:

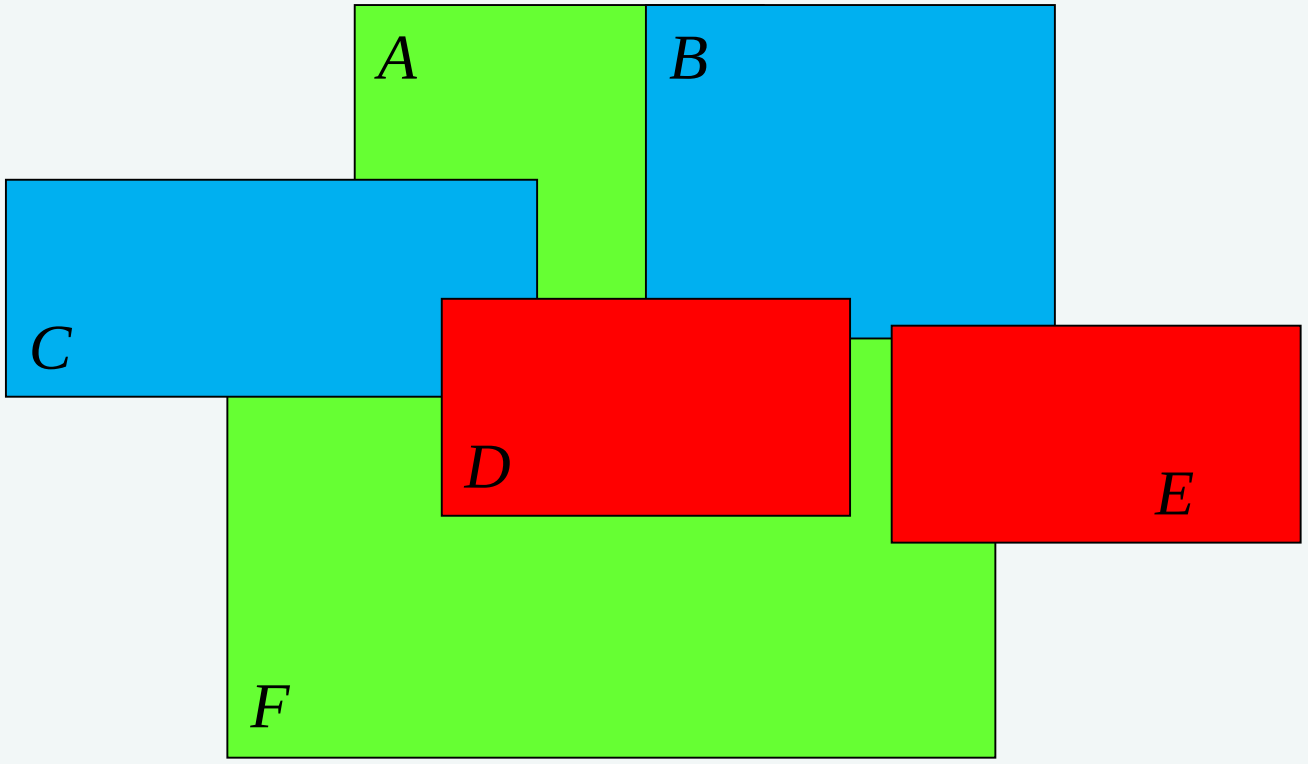
Zbývající proměnné mají přípustnou vždy jedinou hodnotu, a pro tyto hodnoty je celý problém řešitelný.

S_A = {G}
S_B = {}
S_C = {B}
S_D = {G,B}
S_E = {R}
S_F = {B}



Forward checking – příklad barvení mapy třemi barvami:

$S_A = \{\}$
$S_B = \{\}$
$S_C = \{\}$
$S_D = \{G,B\}$
$S_E = \{\}$
$S_F = \{G\}$



Pro hodnocení metody Forward checking platí stejné závěry, jako pro hodnocení metody Backtracking pro CSP.

Metoda je **úplná a optimální**, pro prostorovou složitost platí vztah

$$O_{F\text{-}checking}(n),$$

a pro časovou složitost vztah

$$O_{F\text{-}checking}(m^n).$$

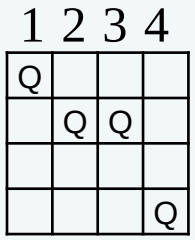
Min-conflict – pouze pro řešení CSP

1. Přiřaďte každé proměnné x_i ($i = 1, \ldots, n$) libovolnou hodnotu z množiny jejich přípustných hodnot. Nastavte pomocné proměnné i a j na hodnoty 1 ($i = j = 1$).
2. Pro každou hodnotu proměnné x_i spočítejte počet možných (včetně teoretický možných) konfliktů.
3. Pokud existuje pro jinou možnou hodnotu proměnné x_i počet konfliktů menší než počet konfliktů pro její aktuální hodnotu, změňte hodnotu této proměnné na hodnotu s minimálním počtem konfliktů. Pokud taková hodnota neexistuje, ale pokud existuje jiná hodnota proměnné x_i se stejným počtem konfliktů, tak změňte hodnotu proměnné x_i na tuto novou hodnotu. V obou uvedených případech nastavte hodnotu proměnné j na 1, jinak pouze inkrementujte hodnotu této proměnné.

Min-conflict – pokračování

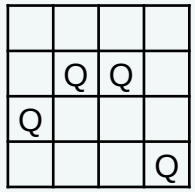
4. Pokud hodnota proměnné $j = n$, pak bylo nalezeno optimální řešení (tj. řešení s minimálním počtem konfliktů). Jinak inkrementujte hodnotu proměnné i a pokračujte.
5. Je-li hodnota proměnné $i > n$, pak změňte její hodnotu na hodnotu 1 ($i = 1$).
6. Vraťte se na bod 2.

Min-conflict (příklad 4 dam):

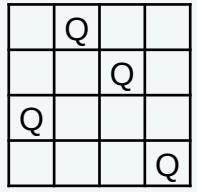


Počet ohrožení dámy *i* na jednotlivých řádcích sloupce *i*:

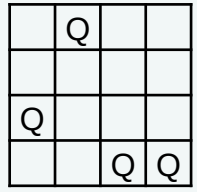
$Q_1 = \{2,2,1,2\}$



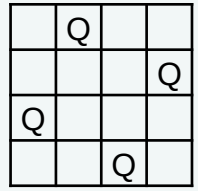
$Q_2 = \{1,3,2,2\}$



$Q_3 = \{2,1,2,1\}$



$Q_4 = \{1,0,3,1\}$

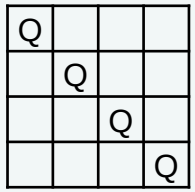


$Q_1 = \{1,3,0,1\}$, $Q_2 = \{0,2,2,3\}$, $Q_3 = \{3,2,2,0\}$, $Q_4 = \{1,0,3,1\}$

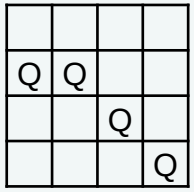
$\Rightarrow$ Goal

Příklad 4 dam, jiný výchozí stav:

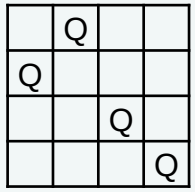
1 2 3 4



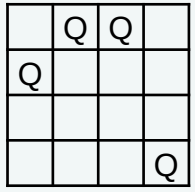
$Q_1 = \{3,1,2,1\}$



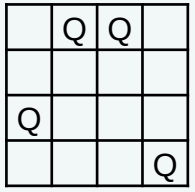
$Q_2 = \{1,3,2,2\}$



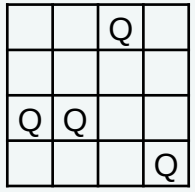
$Q_3 = \{1,2,1,2\}$



$Q_4 = \{2,2,1,0\}$
$Q_1 = \{3,1,1,1\}$

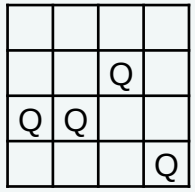


$Q_2 = \{1,3,1,2\}$

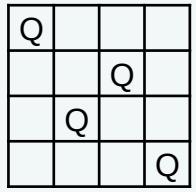


$Q_3 = \{1,1,3,2\}$

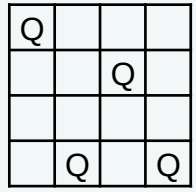
1 2 3 4



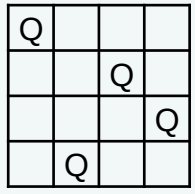
$Q_4 = \{2,1,3,0\}$
$Q_1 = \{1,2,1,3\}$



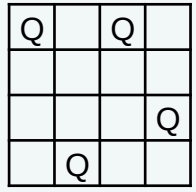
$Q_2 = \{2,3,1,1\}$



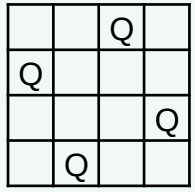
$Q_3 = \{1,0,3,2\}$
$Q_4 = \{2,2,1,2\}$



$Q_1 = \{0,1,2,2\}$
$Q_2 = \{3,2,2,0\}$
$Q_3 = \{1,1,3,2\}$



$Q_4 = \{2,2,0,2\}$
$Q_1 = \{1,0,3,1\}$



Goal

Metoda Min-conflict má extrémně malou prostorovou složitost, protože v paměti se udržuje pouze jediný stav s několika málo dodatečnými informacemi

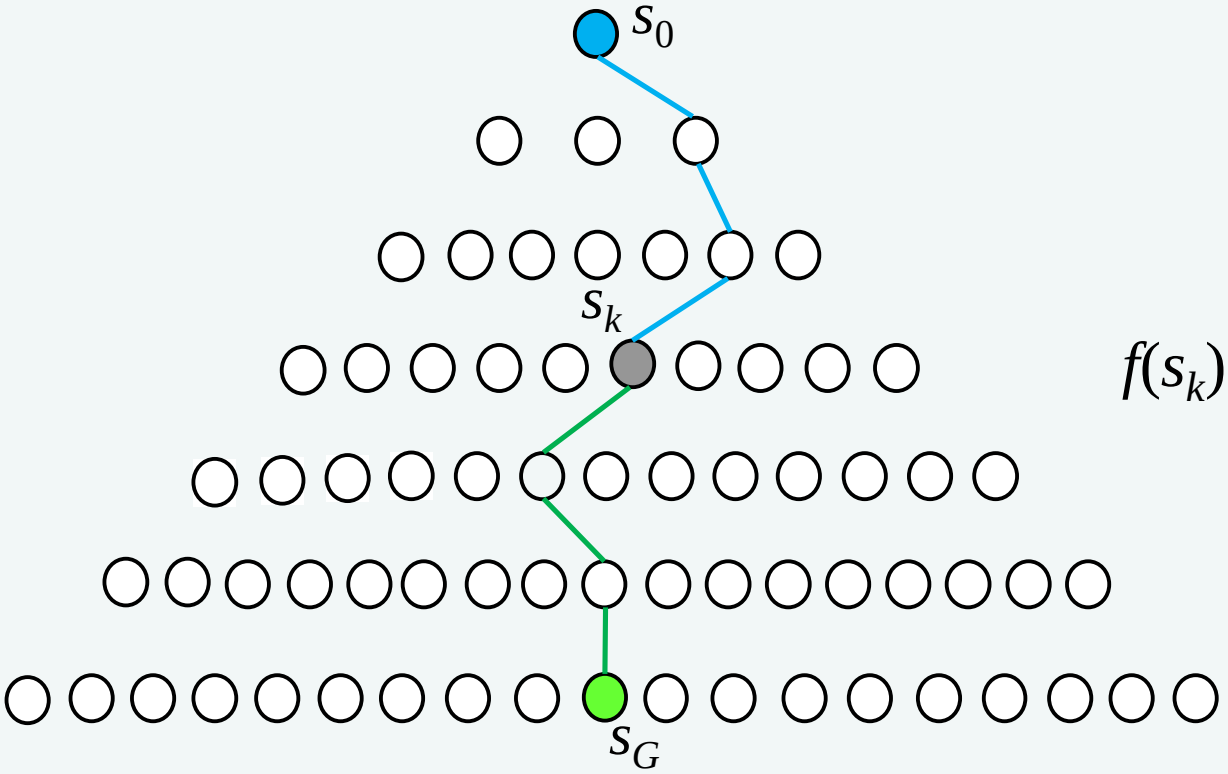
$$O_{Min\text{-}conflict}(1)$$

Zatím však zřejmě není znám důkaz konvergence této metody do bezkonfliktního stavu a tedy ani důkaz její úplnosti a optimálnosti a s tím související odhad časové složitosti.

2. Informované metody

- BestFS – Best First Search (prohledávání od nejlepšího)
  - GS – Greedy Search (chamtivé/lačné/hltavé prohledávání)
  - A^* Search (prohledávání „A s hvězdičkou“ (A - star))

Informované metody



$$f(s_k) = g(s_k) + h(s_k)$$

Hodnotou funkce $f(s_k)$ je celkové ohodnocení stavu/uzlu s_k.

Hodnotou funkce $g(s_k)$ je cena cesty od počátečního stavu ke stavu s_k.

Hodnotou (heuristické) funkce $h(s_k)$ je odhadovaná cena cesty od stavu s_k k cílovému stavu.

BestFS

1. Sestrojte **seznam** OPEN (bude obsahovat všechny uzly určené k expanzi) a vložte do něj počáteční uzel včetně jeho ohodnocení.
2. Je-li seznam OPEN prázdný, pak úloha nemá řešení, a ukončete proto prohledávání jako neúspěšné. Jinak pokračujte.
3. Vyberte ze seznamu OPEN uzel s nejlepším (nejnižším) ohodnocením.
4. Je-li vybraný uzel uzlem cílovým, ukončete prohledávání jako úspěšné a vraťte cestu od kořenového uzlu k uzlu cílovému. Jinak pokračujte.
5. Vybraný uzel expandujte. Všechny jeho bezprostřední následníky, kteří nejsou jeho předky, umístěte do seznamu OPEN včetně jejich ohodnocení. Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponechte pouze uzel s nejlepším ohodnocením, ostatní ze seznamu OPEN vyškrtněte a vraťte se na bod 2.

Pozn.: Ke kontrole, zda generovaný následník není předkem expandovaného uzlu, se může opět použít seznam CLOSED.

Extrémní metody BestFS:

UCS – Uniform Cost Search (slepá metoda!):

$f(s_k) = g(s_k) + h(s_k) = g(s_k) \quad (h(s_k) = 0)$

$f(s_0) = g(s_0) = 0$

GS – Greedy Search:

$f(s_k) = g(s_k) + h(s_k) = h(s_k) \quad (g(s_k) = 0)$

$f(s_G) = h(s_G) = 0$

Greedy Search

Metoda je **úplná**, ale **není optimální**.

Časová i prostorová složitost metody je stejná:

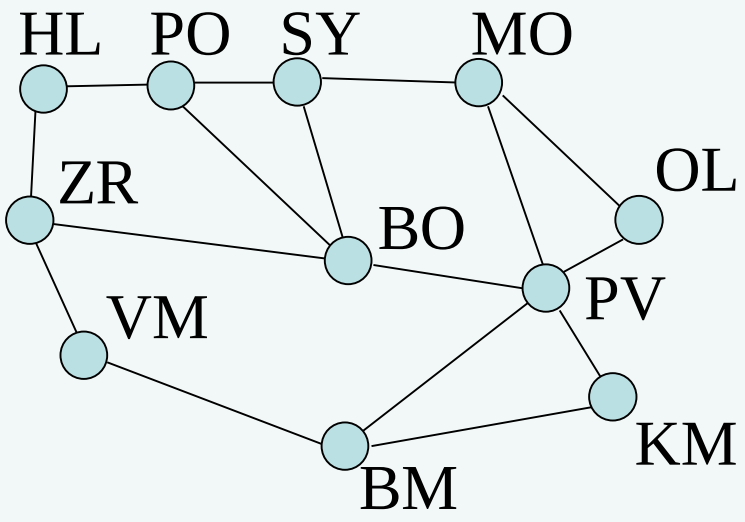
$\boldsymbol{O_{greedy}(b^d)}$

Dobrá heuristika může skutečnou složitost výrazně zmenšit!

Poznámka: Pokud se do seznamu OPEN ukládají všichni bezprostřední následníci expandovaného uzlu, tj. i jeho předci (v bodu 5 algoritmu se vypustí kontrola „kteří nejsou jeho předky“), pak metoda Greedy search **není** ani **úplná**!

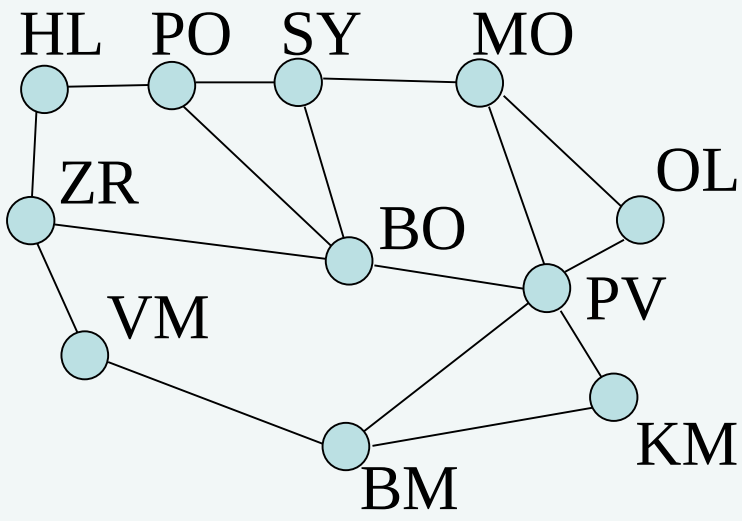
Greedy Search - příklad: Výchozí místo Žďár n/S, cílové místo Brno, přímé vzdálenosti z různých míst do Brna [km]:

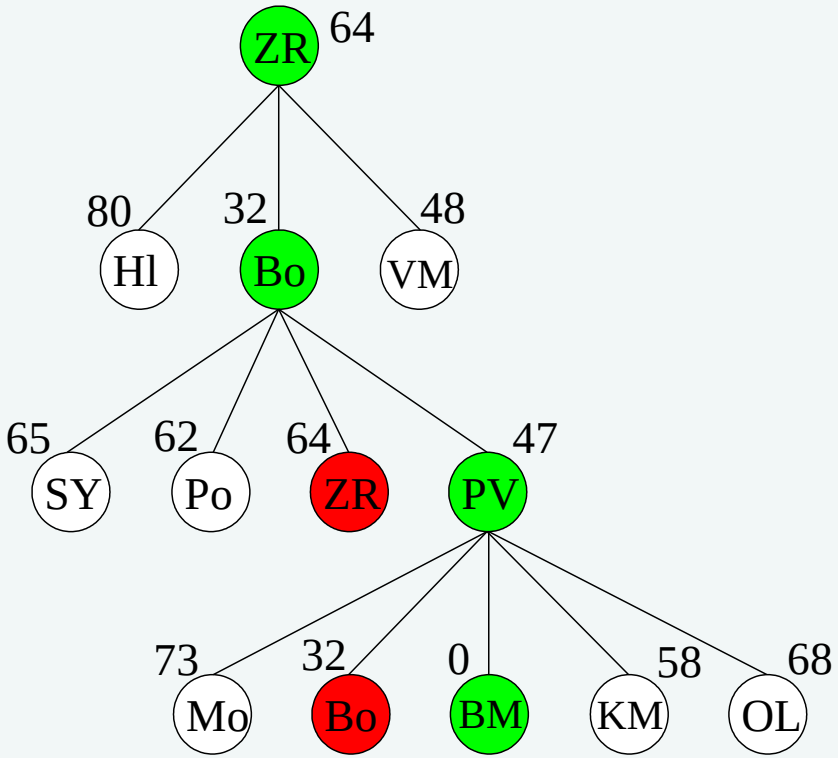
Žďár nad Sázavou	64
Hlinsko	80
Velké Meziříčí	48
Polička	62
Boskovice	32
Olomouc	68
Svitavy	65
Mohelnice	73
Prostějov	47
Kroměříž	58



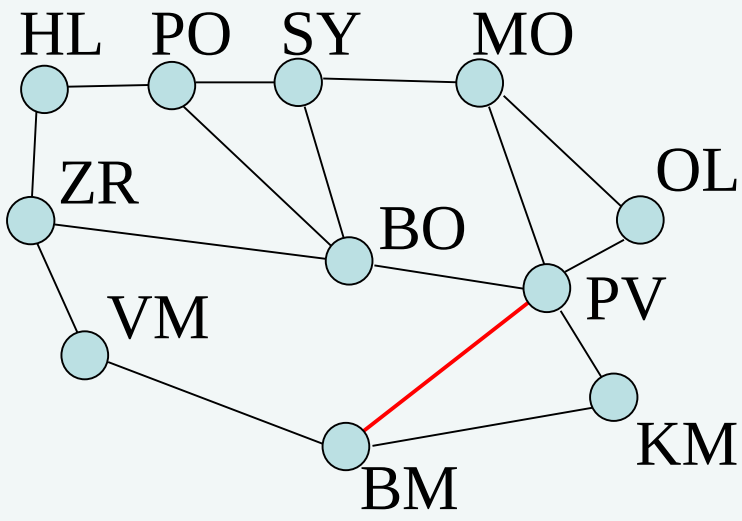
Krok	Open
0	[[(ZR,nil),64]]
1	[[(HL,ZR,nil),80],[(BO,ZR,nil),32],[(VM,ZR,nil),48)]
2	[[(HL,ZR,nil),80],[(VM,ZR,nil),48],[(SY,BO,ZR,nil),65], [(PO,BO,ZR,nil),62],[(ZR,Bo,ZR,nil),64],[(PV,Bo,ZR,nil),47]]
3	[[(HL,ZR,nil),80],[(VM,ZR,nil),48],[(SY,BO,ZR,nil),65], [(PO,BO,ZR,nil),62],[(MO,PV,BO,ZR,nil),73],[Bo,PV,Bo,ZR,nil),32], [(BM,PV,BO,ZR,nil),0],[(KM,PV,BO,ZR,nil),58],[(OL,PV,BO,ZR,nil),68]]
4	[[(Hl,ZR,nil),80],[(VM,ZR,nil),48],[(SY,BO,ZR,nil),65], [(PO,BO,ZR,nil),62],[(MO,PV,BO,ZR,nil),73],[(KM,PV,BO,ZR,nil),58], [(OL,PV,BO,ZR,nil),68]]

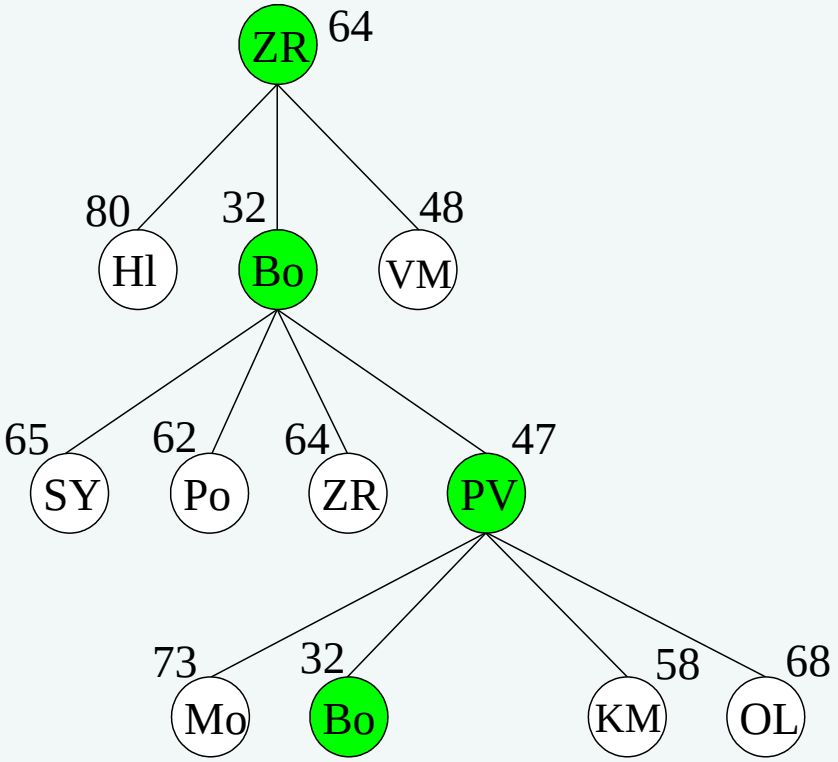
[(BM,PV,Bo,ZR,nil),0] $\Rightarrow$ Goal





Nalezená cesta není optimální.





a v případě ukládání všech generovaných uzlů do OPEN v bodu 5 algoritmu není ani úplná (například pro uvedený případ silniční uzávěry mezi PV a BM).

A* Search

A* je BestFS, u které je heuristická funkce $h(s_k)$ **spodním odhadem** skutečné ceny cesty $h^*(s_k)$ od ohodnocovaného uzlu k cíli – taková heuristika se pak nazývá přípustnou heuristikou.

Předpoklad odvození optimálnosti metody A*:

- A* je BestFS: $f(s_k) = g(s_k) + h(s_k)$, $g(s_0) = 0$, $h(s_G) = 0$
- *h* je přípustná heuristika: $h(s_k) \leq h^*(s_k)$
- *f* je monotónní funkce: $f(s_k) \geq f(s_{k-1})$

Pozn.: Pro přípustnou heuristiku lze zaručit monotónnost funkce *f* použitím vztahu $f(s_k) = \max(f(s_{k-1}), g(s_k) + h(s_k))$

Důkaz optimálnosti algoritmu A^*

1. Označme cenu optimální cesty do nejbližšího (optimálního) cílového uzlu G_{opt} jako $f_{opt}(s_{Gopt})$ a cenu jiné cesty do stejného nebo cesty do jiného cílového uzlu jako $f(s_G)$. Zřejmě musí platit

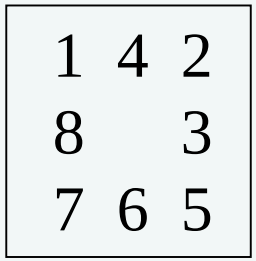
   $$f_{opt}(s_{Gopt}) < f(s_G).$$

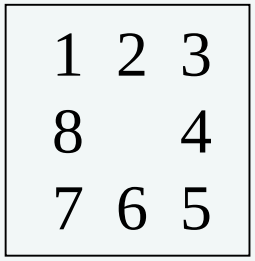
2. Nechť x je uzel na optimální cestě k optimálnímu cíli. Pro monotónní přípustnou heuristiku musí platit $f_{opt}(s_{Gopt}) \geq f(x)$.
3. Pokud by uzel x nebyl vybrán k expanzi, zatímco cílový uzel s_G s ohodnocením $f(s_G)$ ano, muselo by platit $f(x) \geq f(s_G)$.
4. Z bodů 2 a 3 vyplývá, že $f_{opt}(s_{Gopt}) \geq f(x) \geq f(s_G)$, tedy že $f_o(s_{Gopt}) \geq f(s_G)$, což je ve sporu se závěrem bodu 1.
5. Uzel x proto musí být vybrán k expanzi, stejně jako každý jiný uzel na optimální cestě k optimálnímu cíli, a proto metoda musí nalézt optimální cestu.

A* expanduje pouze uzly s_k, pro které platí $f(s_k) \leq f_{opt}$, kde f_{opt} je cena optimální cesty!

Platí-li pro dvě heuristiky podmínka $h^* \geq h_2 > h_1$, pak h_2 je vždy lepší heuristika než h_1!

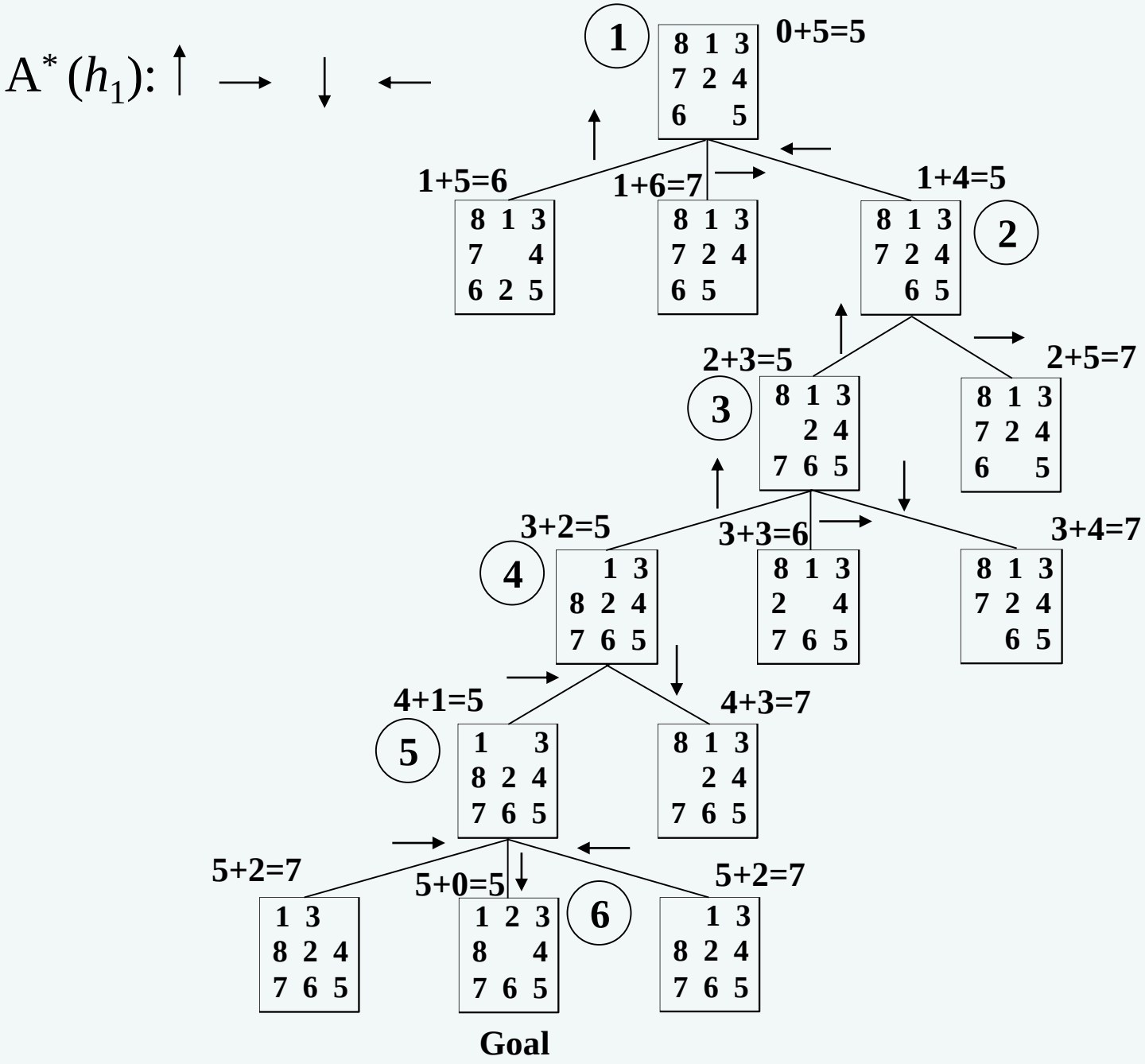
Heuristické funkce – příklad Loydova osmička (L8)

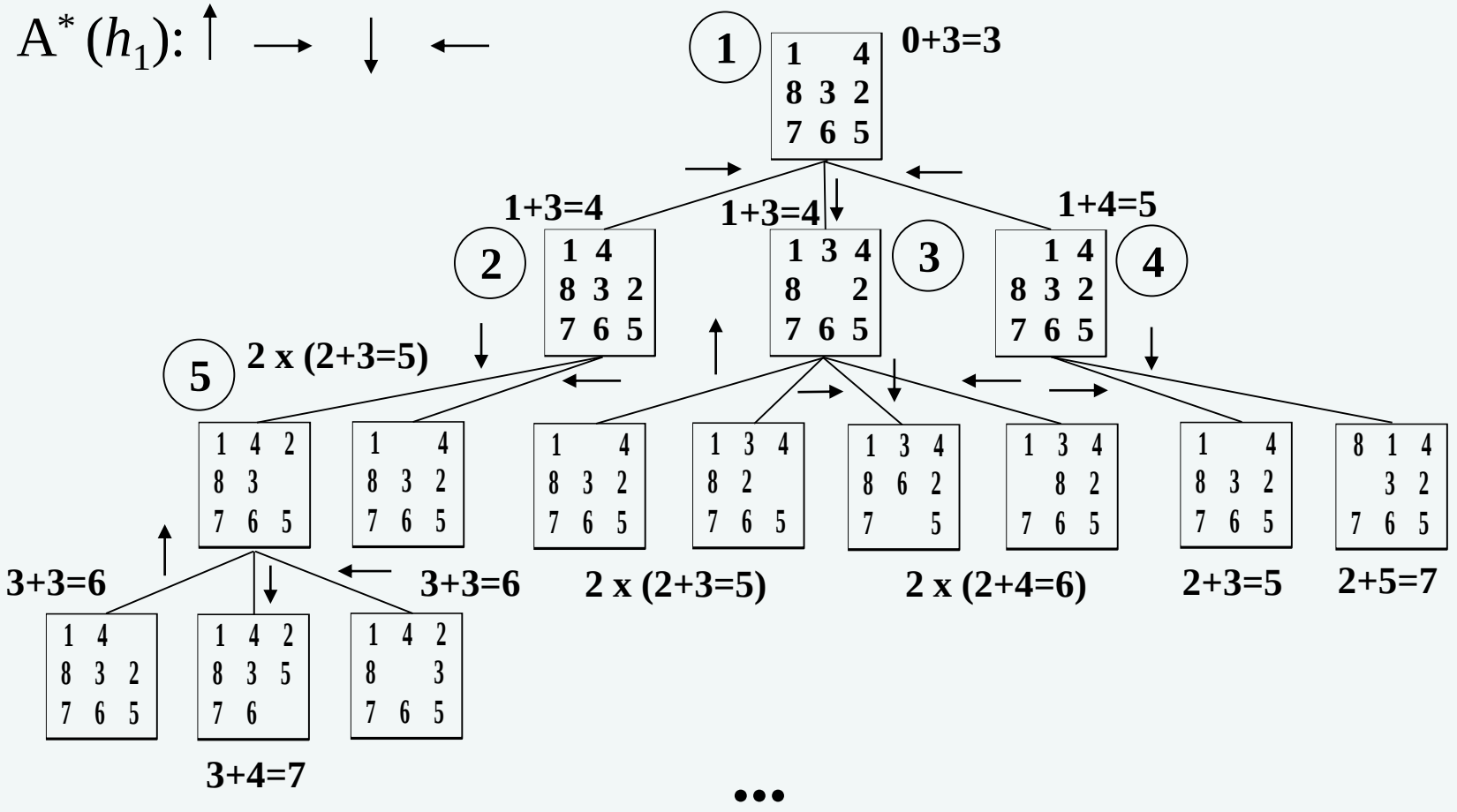




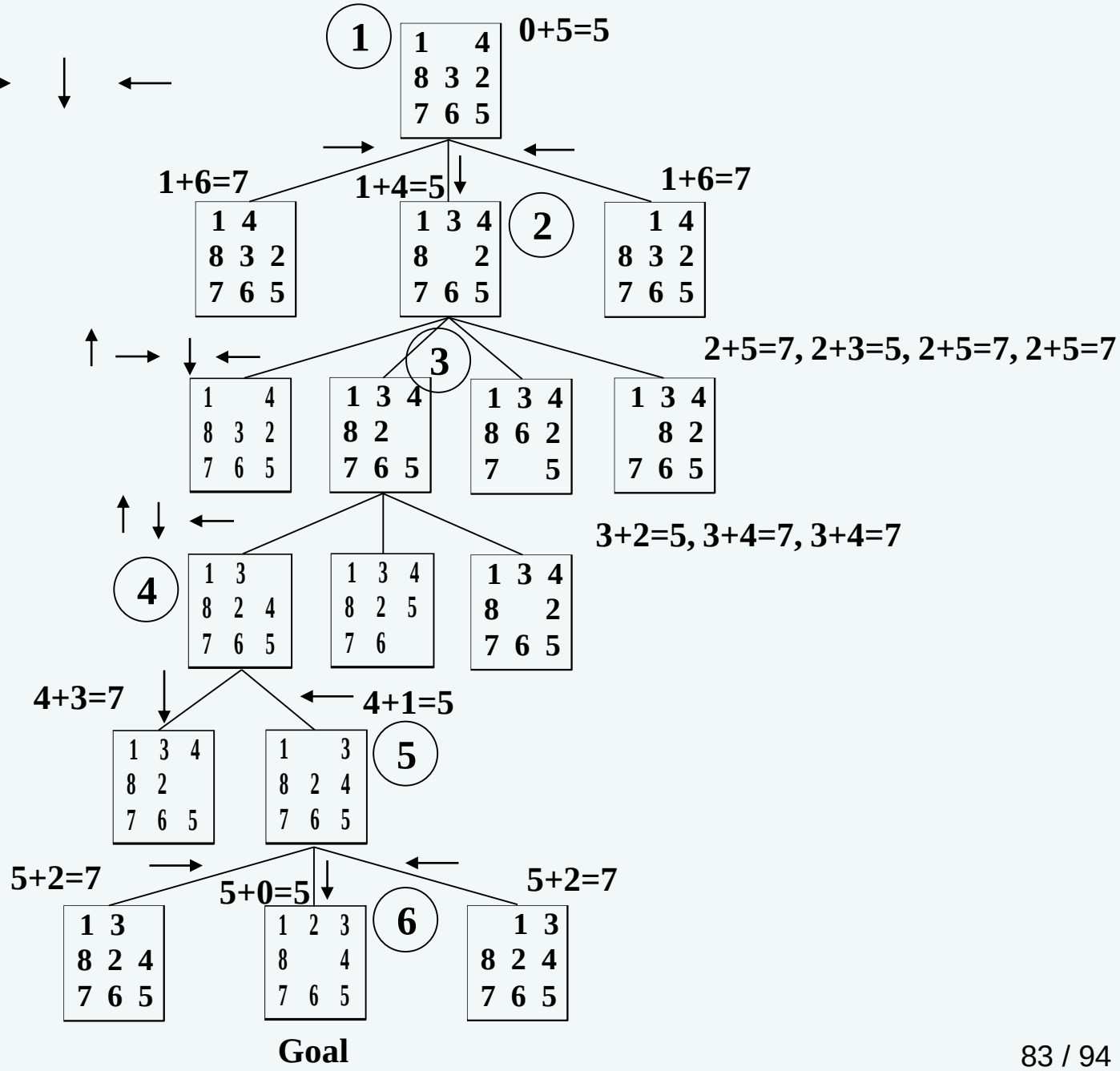
h_1 počet kamenů na nesprávných pozicích

h_2 suma (Manhattan) vzdáleností kamenů od jejich cílových pozic


A* (h1): ↑ → ↓ ←
1
0+5=5
1+5=6
1+6=7
1+4=5
2
2+3=5
2+5=7
3
3+2=5
3+3=6
3+4=7
4
4+1=5
4+3=7
5
5+2=7
5+0=5
5+2=7
6
Goal


A* (h1): ↑ → ↓ ←
1
0+3=3
1+3=4
1+3=4
1+4=5
2
3
4
5
2 x (2+3=5)
3+3=6
3+3=6
2 x (2+3=5)
2 x (2+4=6)
2+3=5
2+5=7
3+4=7
•••

$A^*(h_2)$: ↑ → ↓ ←


1
1 4
8 3 2
7 6 5
0+5=5
1+6=7
1+4=5
1+6=7
1 4
8 3 2
7 6 5
1 3 4
8 2
7 6 5
2
1 4
8 3 2
7 6 5
3
2+5=7, 2+3=5, 2+5=7, 2+5=7
1 4
8 3 2
7 6 5
1 3 4
8 2
7 6 5
1 3 4
8 6 2
7 5
1 3 4
8 2
7 6 5
4
3+2=5, 3+4=7, 3+4=7
1 3
8 2 4
7 6 5
1 3 4
8 2 5
7 6
1 3 4
8 2
7 6 5
4+3=7
4+1=5
1 3 4
8 2
7 6 5
1 3
8 2 4
7 6 5
5
5+2=7
5+0=5
5+2=7
1 3
8 2 4
7 6 5
1 2 3
8 4
7 6 5
6
1 3
8 2 4
7 6 5
Goal

Metoda A* je **úplná** a **optimální**. Časová i prostorová složitost je stejná a výrazně závisí na použité heuristické funkci. Pohybuje se od

$\boldsymbol{O_{A*}(b^d)}$ pro hodnoty h blízké nule (UCS)

do

$\boldsymbol{O_{A*}(d)}$ pro $h = h^*$

Hodnocení „kvality“ heuristických funkcí:

1. Efektivní faktor větvení b^*
2. Počet generovaných uzlů:

   $N = 1 + b^* + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$

L8	N			b^*		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.50	1.75	1.75
4	112	13	12	2.94	1.50	1.44
6	680	20	18	2.75	1.34	1.31
8	6384	39	25	2.83	1.34	1.24
10	47127	93	39	2.81	1.39	1.23
12	364404	227	73	2.80	1.42	1.26
14	3473941	539	113	2.84	1.44	1.25
16		1301	211		1.46	1.27
18		3056	363		1.47	1.27
20		7276	676		1.47	1.29
22		18094	1219		1.48	1.29
24		39135	1641		1.48	1.28

Volba (návrh) heuristických funkcí

Problémy s menší restrikcí (relaxed problems)

Např. pro Loydovu osmičku platí pravidlo

- Kamenem lze táhnout z pole A na pole B, jestliže pole A sousedí s polem B a pole B je prázdné

Pravidla pro problémy s menší restrikcí/omezením:

- Kamenem lze táhnout z A na B, jestliže A sousedí s B (h_2)
- Kamenem lze táhnout z A na B, jestliže B je prázdné (?)
- Kamenem lze táhnout z A na B (h_1)

3. Metody lokálního prohledávání

- Hill Climbing
- Simulated Annealing

Hill-climbing Search

1. Vytvořte uzel *Current* a uložte do něj počáteční stav s_0 spolu s jeho ohodnocením (obvykle je ohodnocení stavu dáno pouze hodnotou heuristické funkce *h*).
2. Expandujte uzel *Current*, ohodnoťte jeho bezprostřední následníky a vyberte z nich nejlépe ohodnoceného (nazvěme jej *Next*).
3. Je-li ohodnocení uzlu *Current* lepší než ohodnocení uzlu *Next*, ukončete řešení a vraťte jako výsledek uzel/stav *Current*. Jinak pokračujte.
4. Nahraďte uzel *Current* uzlem *Next* a vraťte se na bod 2.

Příklad (podobný, jako byl uveden u metody Greedy search), přímé vzdálenosti ze Žďáru nad Sázavou do cílového místa [km]:

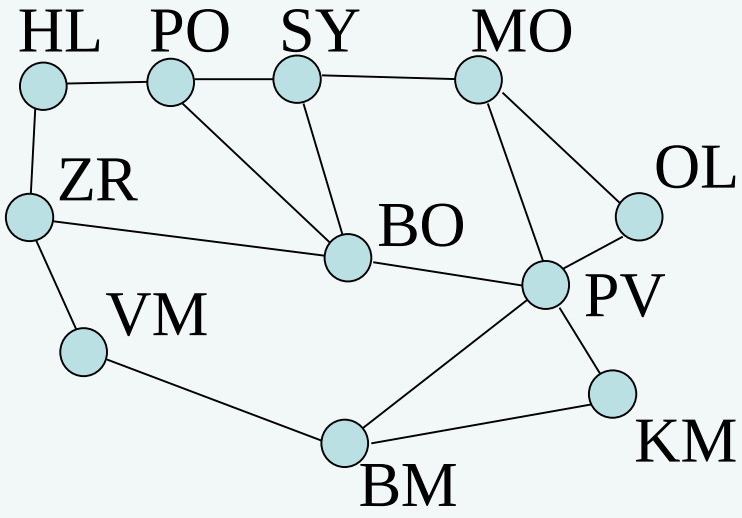
a) do Olomouce

Žďár nad Sázavou	96
Hlinsko	100
Velké Meziříčí	96
Polička	76
Boskovice	52
Brno	68
Svitavy	60
Mohelnice	35
Prostějov	16
Kroměříž	28

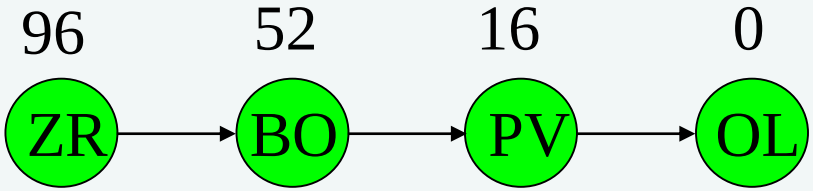
b) do Brna

Žďár nad Sázavou	64
Hlinsko	80
Velké Meziříčí	48
Polička	62
Boskovice	32
Olomouc	68
Svitavy	65
Mohelnice	73
Prostějov	47
Kroměříž	58

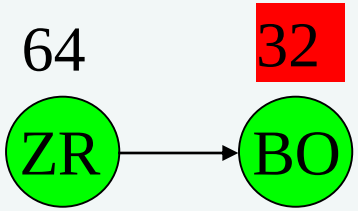
Výsledky získané algoritmem Hill-Climbing:



a) Při hledání cesty ze Žďáru n/S do Olomouce



b) Při hledání cesty ze Žďáru n/S do Brna



Metoda **není úplná ani optimální**.

Prostorová složitost je extrémně nízká

$\boldsymbol{O_{HillClimb}(1)}$

a velmi nízká je i časová složitost

$\boldsymbol{O_{HillClimb}(d)}$

Simulated Annealing

1. Vytvořte tabulku s předpisem pro klesání „teploty“ v závislosti na kroku výpočtu.
2. Vytvořte pracovní uzel *Current* a uložte do něj počáteční stav s_0 spolu s jeho ohodnocením (obvykle je ohodnocení stavu opět dáno pouze hodnotou heuristické funkce h). Nastavte krok výpočtu na nulu ($k = 0$).
3. Z tabulky zjistěte aktuální „teplotu“ T ($T = f(k)$). Je-li tato „teplota“ nulová ($T = 0$) ukončete řešení a vraťte jako výsledek uzel/stav *Current*. Jinak pokračujte.
4. Expandujte uzel *Current* a z jeho bezprostředních následníků vyberte náhodně jednoho z nich (nazvěme jej *Next*).
5. Vypočítejte rozdíl ohodnocení uzlů *Current* a *Next* ΔE = value(*Current*) – value(*Next*).

6. Jestliže $\Delta E > 0$, pak nahraďte uzel *Current* uzlem *Next*, jinak proveďte tuto náhradu s pravděpodobností $p = e^{\Delta E/T}$.
7. Inkrementujte krok výpočtu *k* a vraťte se na bod 3.

Algoritmus přechází do nového stavu nepodmíněně, pokud je ohodnocení tohoto stavu lepší (tj. nižší), než ohodnocení aktuálního stavu (nový stav je „blíže“ cílovému stavu). V opačném případě je přechod dán pravděpodobností, která je pro vysoké počáteční „teploty“ také vysoká, a která se snižující se „teplotou“ se také postupně snižuje. Algoritmus tak může opustit lokální extrémy, které jsou pro předchozí Hill climbing algoritmus nepřekonatelné, a může tak přispět k nalezení (optimálního) řešení.

Algoritmus má opět extrémně malou prostorovou složitost

$\boldsymbol{O_{SimAnn}(1)}$

ale jeho časová složitost je velmi vysoká a je závislá na rychlosti klesání „teploty“.

Pro reálné časové možnosti tak není zaručena **ani úplnost a tedy ani optimálnost** tohoto algoritmu.