Systémy s umělou inteligencí

2. Informované metody prohledávání stavového prostoru

Jiří Kubalík
Katedra kybernetiky, ČVUT-FEL

http://cw.felk.cvut.cz/doku.php/courses/A7B33SUI/start

Informované metody prohledávání

Ohodnocovací funkce – udává míru, nakolik je dobré expandovat daný uzel.

Best-first search – prohledávání, kde v každém kroku expandujeme uzel, který se jeví jako nejlepší vzhledem k dané ohodnocovací funkci.

- "Nejlepší" – pokud by to byl skutečně nejlepší možný uzel v daném kroku, tak by to znamenalo ...
- **Jakou ohodnocovací funkci jsme používali minulou hodinu? Je vhodná pro efektivní směřování prohledávání směrem k cílovému stavu?**

Informované metody prohledávání

Ohodnocovací funkce – udává míru, nakolik je dobré expandovat daný uzel.

Best-first search – prohledávání, kde v každém kroku expandujeme uzel, který se jeví jako nejlepší vzhledem k dané ohodnocovací funkci.

- "Nejlepší" – pokud by to byl skutečně nejlepší možný uzel v daném kroku, tak by to znamenalo ...
- **Jakou ohodnocovací funkci jsme používali minulou hodinu? Je vhodná pro efektivní směřování prohledávání směrem k cílovému stavu?**

Dva základní přístupy informovaného prohledávání

- **Minimalizace odhadnuté ceny do cílového stavu (hladové prohledávání)** – expanduje uzel, který je nejbliže některému cílovému stavu,
- **Minimalizace ceny celkové cesty (algoritmus A*)** – expanduje uzel, který leží na cestě k cíli s nejnižší cenou.
  - pro většinu problémů, cenu neumíme určit přesně, můžeme ji pouze odhadnout,
  - $h^*(n) =$ odhad ceny nejlevnější cesty ze stavu v uzlu n do cílového stavu,
  - $h^*(n) = 0$, pro cílový stav n.

Procedura Best-first search

begin

$open := [Start]$, $closed := []$

while $open \neq []$ do

odstran ze seznamu $open$ uzel x s nejlepší hodnotou $h(x)$

if$(x = goal)$ **return** cesta ze $Start$ do x

else generuj potomky $[x_i]$ uzlu x a pro kazdeho potomka x_i

if(x_i není v $open$ ani $closed$)

spočítej $h(x_i)$

přidej x_i do $open$

if(x_i je v $open$)

přiřaď stavu x_i hodnotu $h(x_i)$, pokud je lepší než ta stávající

if(x_i je v $closed$)

pokud byl stav x_i nalezen po kratší cestě,

tak odstraň x_i z $closed$ a dej ho do $open$ s novou hodnotou $h(x_i)$

dej x do seznamu $closed$

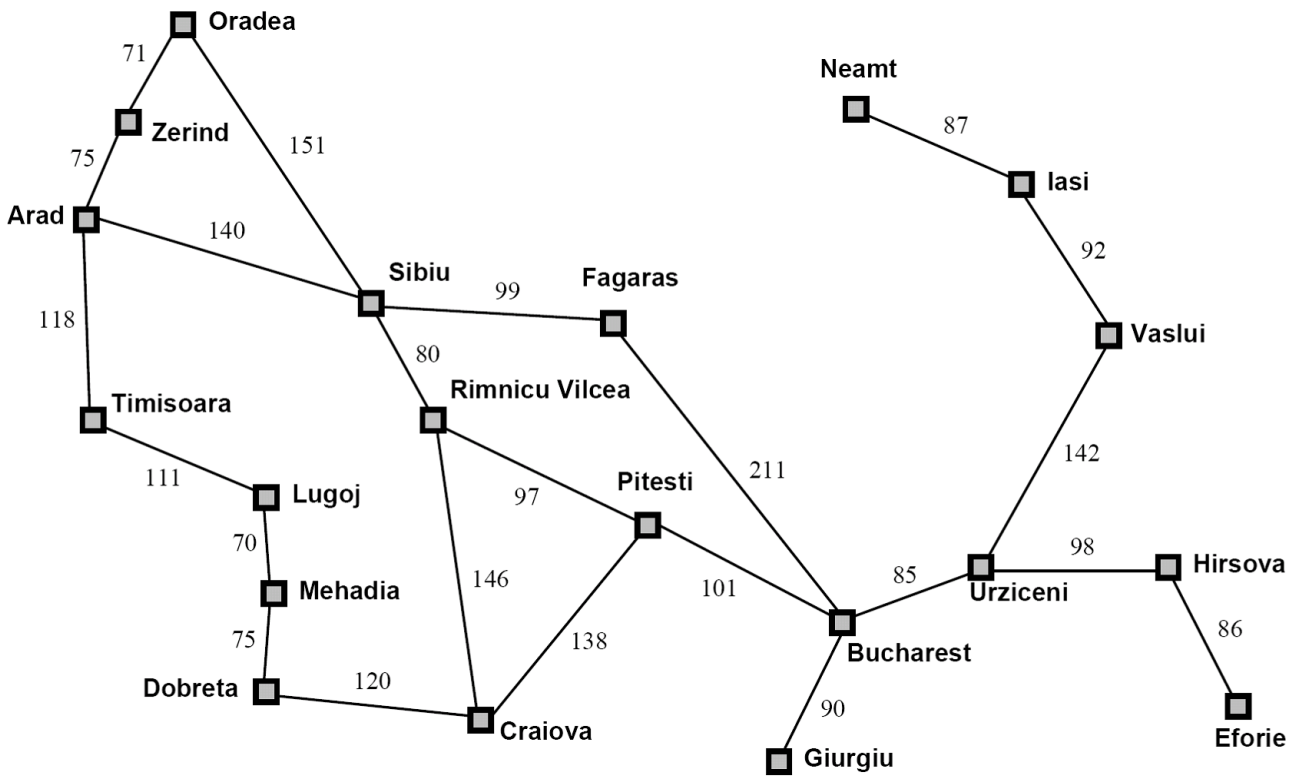
uspořádej uzly v $open$ podle dané heuristiky

return chyba

Heuristická funkce: Příklad

Problém: Hledání optimální cesty mezi městy *Arad* a *Bucharest*.

- **heuristická funkce** h_{SLD} – přímá vzdálenost z města do Bukurešti (**S**traight-**L**ine **D**istance).



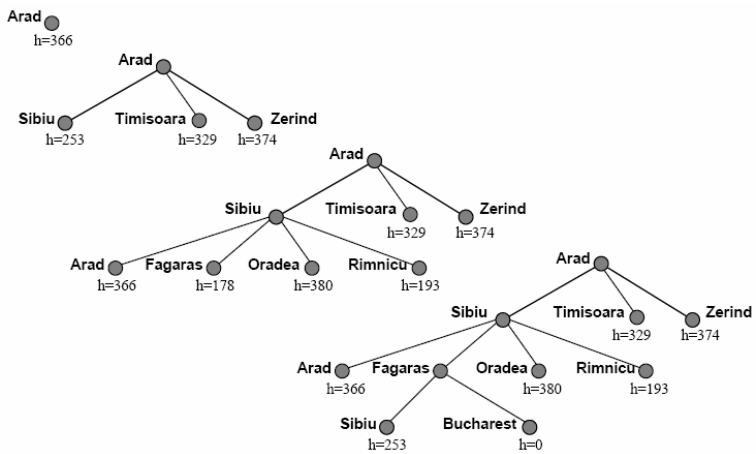
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Hladové prohledávání s heuristikou h_{SLD}

Fáze hladového prohledávání s heuristikou h_{SLD}.

- uzly jsou expandovány v pořadí: Arad, Sibiu, Fagaras,
- pouze uzly na cestě vedoucí k řešení jsou generovány, ale řešení není optimální.



©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Závěr: Hladové prohledávání

- najde rychle řešení, které však nemusí být optimální,
- je citlivé na výběr počátečního uzlu **(Jak by vypadalo hledání cesty z Iasi do Fagarse?)**,
- nemusí také najít žádné řešení (osciluje mezi opakovaně generovanými stavy),
- má časovou i paměťovou složitost $O(b^m)$, kde m je max. hloubka prohledávaného prostoru. S dobrou heuristikou se to dá výrazně omezit.

Čím se hladové prohledávání liší od prohledávání s jednotnou cenou?

Hladové prohledávání vs. prohledávání s jednotnou cenou

Hladové prohledávání minimalizuje odhad ceny cesty k cíli $h(n)$

- zvyšuje efektivnost prohledávání (redukuje počet generovaných uzlů),
- ale není úplné ani optimální.

Prohledávání s jednotnou cenou minimalizuje cenu dosavadní cesty $g(n)$

- úplné a optimální,
- ale může být značně neefektivní.

Co kdybychom to dali nějak dohromady?

Algoritmus A*

Hladové prohledávání minimalizuje odhad ceny cesty k cíli $h^*(n)$

- zvyšuje efektivnost prohledávání (redukuje počet generovaných uzlů),
- ale není úplné ani optimální.

Prohledávání s jednotnou cenou minimalizuje cenu dosavadní cesty $g(n)$

- úplné a optimální,
- ale může být značně neefektivní.

Co kdybychom to dali nějak dohromady?

- $f(n) = g(n) + h^*(n)$, odhad ceny nejlevnějšího řešení procházejícího uzlem n,
- best-first prohledávání s ohodnocovací funkcí $f(n)$ je úplné a optimální (za jistých podmínek!).

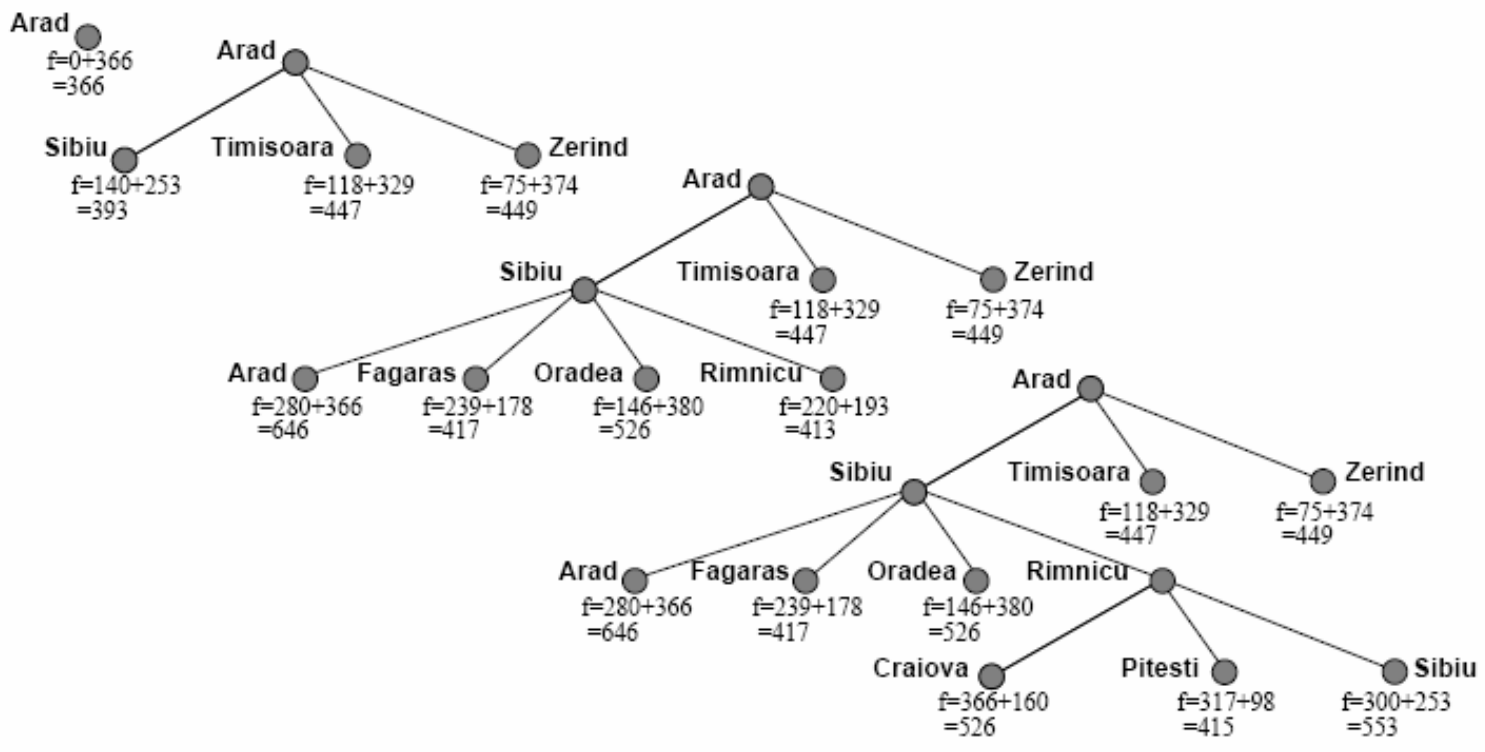
Přípustná heuristika (admissible heuristic) – optimistická; odhad ceny cesty z daného uzlu do cíle nikdy není větší, než je skutečná cena.

Algoritmus prohledávání A* – best-first prohledávání, používající ohodnocovací funkci $f(n)$ s přípustnou heuristickou funkcí $h^*(n)$.

Algoritmus A*: Příklad

Fáze prohledávání A* s heuristikou h_{SLD}.

- h_{SLD} je přípustná,
- uzly jsou expandovány v pořadí: Arad, Sibiu, Rimnicu, (?).

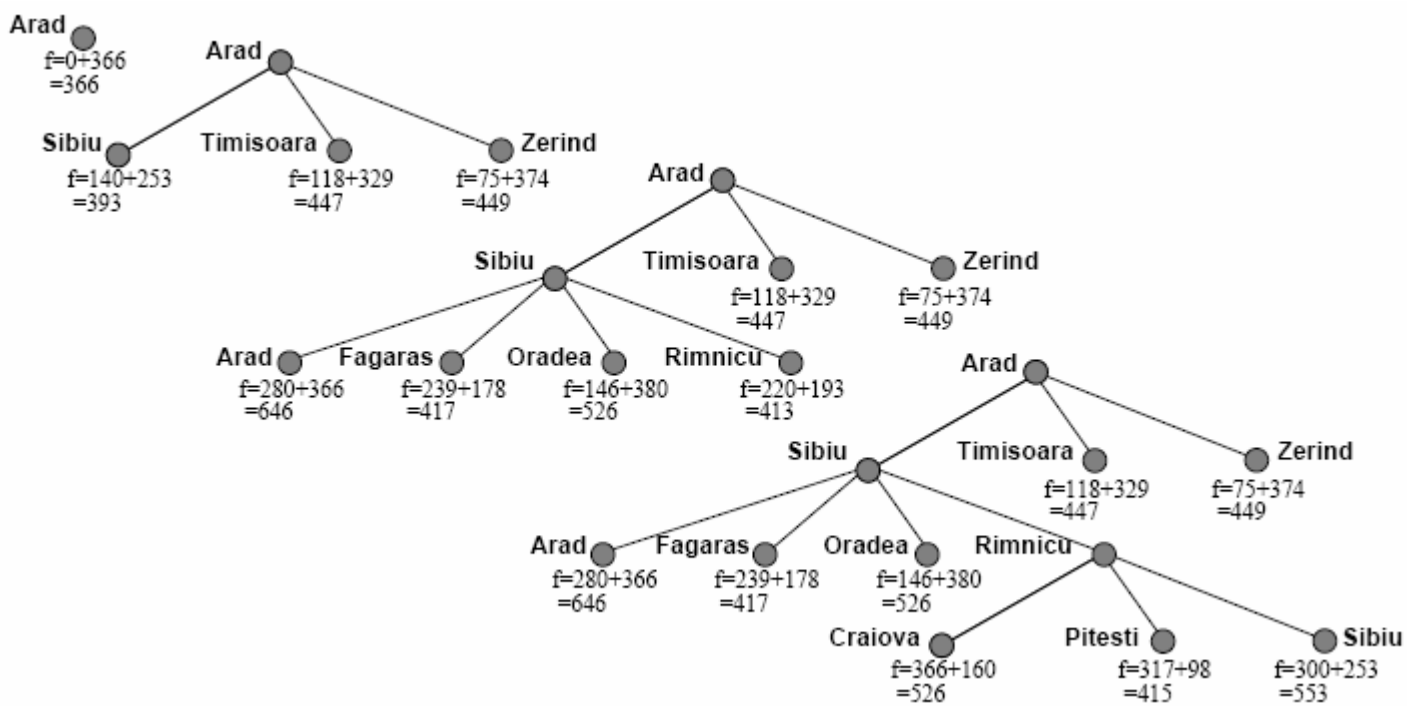


©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Algoritmus A*: Monotónnost

Monotónnost

- Podél každé cesty od kořenového uzlu dolů, hodnota funkce $f(n)$ nikdy neklesá.
- **Otázka: Co v případě, že to neplatí?**



©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Algoritmus A*: Monotónnost

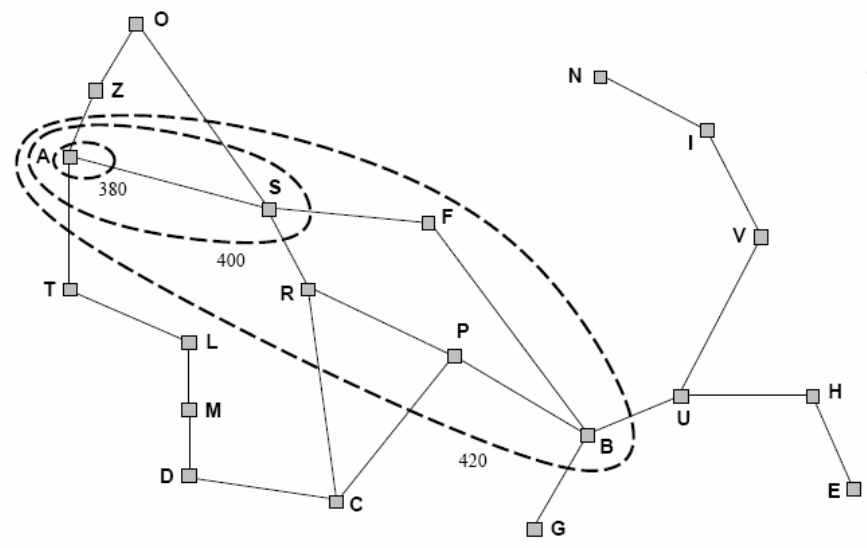
Monotónnost

- Podél každé cesty od kořenového uzlu dolů, hodnota funkce $f(n)$ nikdy neklesá.
- Pokud heuristika není monotónní, můžeme to napravit jednoduchou korekcí
  $f(n) = max(f(parent(n)), g(n) + h^*(n))$,
  kde $f(parent(n))$ je hodnota ohodnocovací funkce rodičovského uzlu.
  Příklad
  - Uvažujme uzly n a n', kde n je rodičem n',
    dále předpokládejme, že $g(n) = 3$ a $h^*(n) = 4$, takže $f(n) = 7$
    a $g(n') = 4$ a $h^*(n') = 2$, takže $f(n') = 6$.
    Je zřejmé, že když n je rodičem n', potom skutečná cena cesty, procházející uzlem n, je minimálně 7. Takže přiřadíme $f(n') = 7$.

Algoritmus A*: Vrstevnice funkce $f(n)$

Vrstevnice

- pro všechny uzly ohraničené vrstevnicí o hodnotě v platí $f(n) \leq v$,
- prohledávání prostoru probíhá od kořenového uzlu generováním dalších uzlů v soustředných vrstevnicích, roztažených podél optimální cesty k cílovému stavu,
- **Jak by vypadaly vrstevnice pro A* používající $h^*(n) = 0$?**



©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Závěry

- **A* expanduje všechny uzly s** $f(n) < f_{opt}$,
  kde f_{opt} je cena optimální cesty,
- **A* je optimální** – první nalezené řešení musí být i optimální (POZOR: Co je první nalezené řešení?),
- **A* je uplné**.

Algoritmus A*: Důkaz optimality

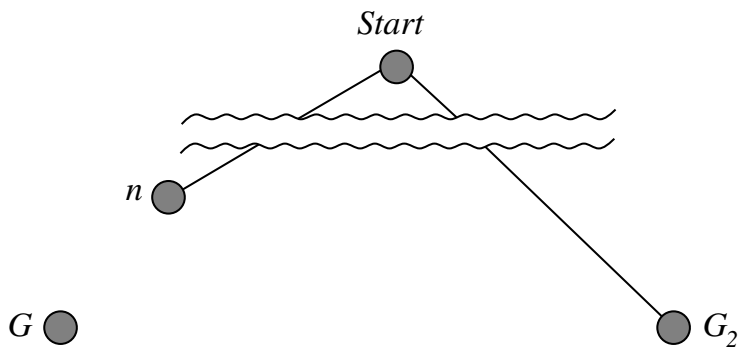
Předpoklad: Nechť G resp. G_2 je optimální cílový stav s cenou f^* resp. suboptimální cílový stav s cenou cesty $g(G_2) > f^*$. Předpokládejme, že A* vybral ze seznamu otevřených uzlů uzel G_2. Protože G_2 je cílový stav, prohledávání bude ukončeno (Toto ovšem nemůže nastat !!!).

Algoritmus A*: Důkaz optimality

Předpoklad: Nechť G resp. G_2 je optimální cílový stav s cenou f^* resp. suboptimální cílový stav s cenou cesty $g(G_2) > f^*$. Předpokládejme, že A* vybral ze seznamu otevřených uzlů uzel G_2. Protože G_2 je cílový stav, prohledávání bude ukončeno (Toto ovšem nemůže nastat !!!).

Důkaz:

1. Uvažujme aktuální koncový uzel n na cestě k G. Pro jeho $f(n)$ tedy platí $f(n) \leq f^*$.
2. A protože nebyl vybrán pro expanzi před G_2 musí zároveň platit $f(n) \geq f(G_2)$.
3. Ovšem potom by muselo platit i $f^* \geq f(G_2)$.
4. Jelikož $h^*(G_2) = 0 \implies f(G_2) = g(G_2)$, potom by muselo platit $f^* \geq g(G_2)$.



©Russel, Norvig: Artificial Intelligence: A Modern Approach.

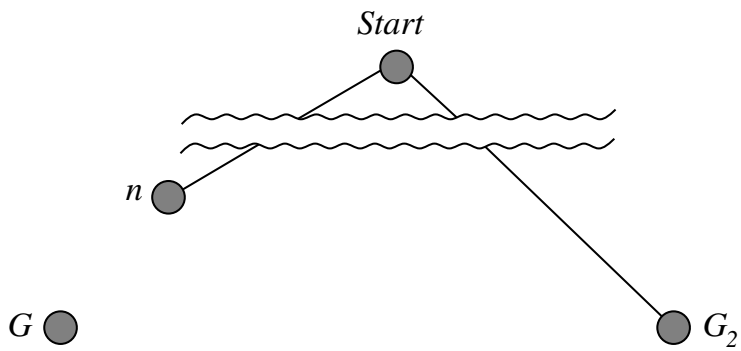
Ale to je rozpor s původním předpokladem ($g(G_2) > f^*$)!!!

Algoritmus A*: Důkaz optimality

Předpoklad: Nechť G resp. G_2 je optimální cílový stav s cenou f^* resp. suboptimální cílový stav s cenou cesty $g(G_2) > f^*$. Předpokládejme, že A* vybral ze seznamu otevřených uzlů uzel G_2. Protože G_2 je cílový stav, prohledávání bude ukončeno (Toto ovšem nemůže nastat !!!).

Důkaz:

1. Uvažujme aktuální koncový uzel n na cestě k G. Pro jeho $f(n)$ tedy platí $f(n) \leq f^*$.
2. A protože nebyl vybrán pro expanzi před G_2 musí zároveň platit $f(n) \geq f(G_2)$.
3. Ovšem potom by muselo platit i $f^* \geq f(G_2)$.
4. Jelikož $h^*(G_2) = 0 \implies f(G_2) = g(G_2)$, potom by muselo platit $f^* \geq g(G_2)$.



©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Ale to je rozpor s původním předpokladem ($g(G_2) > f^*$)!!!

Závěr:

- **A* nikdy nemůže vybrat suboptimální cílový stav pro expanzi.**
- Protože prohledávání končí v momentě, kdy je cílový stav vybrán pro expanzi, **musí být A* optimální**.

Algoritmus A*: Důkaz úplnosti

Důkaz: A* expanduje uzly v pořadí rostoucí hodnoty f, takže jednou musí dojít k expanzi cílového stavu, pokud cílový stav existuje a pokud neexistuje nekonečně mnoho uzlů s $f(n) < f^*$.

Podmínky:

- **lokálně konečný graf** – konečný větvící faktor b,
- existuje nějaké kladné δ takové, že **cena všech expanzních operátorů je minimálně** δ.

  Zenův paradox – nekonečně mnoho kroků s konečnou celkovou cenou. Trajektorie kamene vrženého na cíl se rozdělí na úseky, kde každý následující je polovinou toho předchozího. Kámen tedy nikdy nedorazí k cíli (nekonečně mnoho úseků), přičemž celková cena řešení je konečná (délka trajektorie se limitně blíží ke konečné hodnotě).

Algoritmus A*: Časová a paměťová složitost

Časová složitost: Pro většinu problémů, roste počet uzlů, ležících v oblasti ohraničené vrstevnicí s hodnotou ceny cílového stavu A*, exponenciálně s délkou optimální cesty. Lepší než exponenciální složitost je pouze v případech, kdy chyba heuristické funkce h^* neroste rychleji než logaritmus skutečné ceny

$$|h(n) - h^*(n)| \leq O(\log h(n)),$$

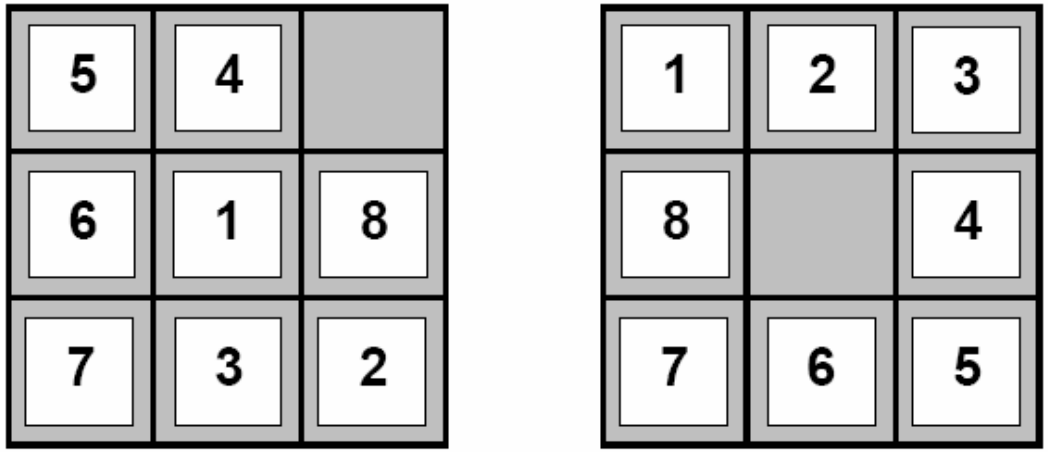
kde $h(n)$ je skutečná cena cesty z uzlu n do cílového stavu.

Paměťová náročnost: A* uchovává všechny vygenerované uzly v paměti $\Longrightarrow$ exponenciální nároky na paměť.

Ukázky heuristik

Dvě přípustné heuristiky pro problém Lišák

- h_1 = počet kamenů, které neleží na správných místech. 7 z 8 kamenů na obrázku jsou na špatné pozici.
- h_2 = součet vzdáleností všech kamenů od jejich správných pozic (Manhattan metrika). Hodnota h_2 pro konfiguraci z obrázku je 18.
- **Tak která heuristika je lepší?**



Start State Goal State

©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Algoritmus A*: Přesnost heuristiky

Porovnání počtu expandovaných uzlů u algoritmů A* s h_1, A* s h_2 a neinformovaného prohledávání s iterativním prohlubováním (IDA).

- pro každou hloubku řešení d bylo vygenerováno 100 počátečních konfigurací na úloze 8-queen.

	Search Cost		
d	IDS	A*(h_1)	A*(h_2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113
16	–	1301	211
18	–	3056	363
20	–	7276	676
22	–	18094	1219
24	–	39135	1641

©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Závěr: h_2 je mnohem lepší než h_1. **A platí to vždy?**

Algoritmus A*: Přesnost heuristiky 2

h_2 **dominuje** h_1 – pro každý uzel n platí $h_2(n) \geq h_1(n) \Longrightarrow$ A* používající h_2 expanduje v průměru méně uzlů než A* používající h_1.

Zdůvodnění:

1. již víme, že A* expanduje všechny uzly s $f(n) < f^*$.

   Takže každý uzel s $h(n) < f^* - g(n)$ bude expandován.

2. Protože pro všechny uzly je h_2 minimálně stejně velká jako h_1, bude každý uzel expandovaný A* používající h_2 také expandován A* používající h_1, a h_1 může navíc expandovat ještě další uzly.

Závěr: Čím přesnější je heuristika (čím vyšší je $h^*(n)$ spodním odhadem skutečné ceny cesty z n do cílového stavu), tím efektivnější je A* prohledávání.

Algoritmus A* s iterativním prohlubováním

A* s iterativním prohlubováním (IDA*)

- strategie iterativního prohlubování zlepšuje paměťovou náročnost A*,
- používá limit daný maximální cenou f (vrstevnicí s hodnotou f_{max}) namísto maximální hloubky d. V každé iteraci se expandují uzly s maximální hodnotou $f(n) \leq f_{max}$.
- pro další iteraci je vybrána nová hodnota f_{max} na základě vygenerovaných uzlů v dané iteraci tak, že

  $f_{max} = min\{f(i) : f(i) > f_{max}\}$.

Algoritmus A* s iterativním prohlubováním

A* s iterativním prohlubováním (IDA*)

- strategie iterativního prohlubování zlepšuje paměťovou náročnost A*,
- používá limit daný maximální cenou f (vrstevnicí s hodnotou f_{max}) namísto maximální hloubky d. V každé iteraci se expandují uzly s maximální hodnotou $f(n) \leq f_{max}$.
- pro další iteraci je vybrána nová hodnota f_{max} na základě vygenerovaných uzlů v dané iteraci tak, že

  $f_{max} = min\{f(i) : f(i) > f_{max}\}$.

Vlastnosti IDA*

- je optimální a úplné,
- paměťová náročnost je úměrná nejdelší prozkoumané cestě,
- časová složitost závisí na počtu hodnot k, kterých může funkce $h^*(n)$ nabývat,
  - pro malé k je efektivní,
  - pro velké k je neefektivní (*Proč?*).

Algoritmus A* s iterativním prohlubováním

A* s iterativním prohlubováním (IDA*)

- strategie iterativního prohlubování zlepšuje paměťovou náročnost A*,
- používá limit daný maximální cenou f (vrstevnicí s hodnotou f_{max}) namísto maximální hloubky d. V každé iteraci se expandují uzly s maximální hodnotou $f(n) \leq f_{max}$.
- pro další iteraci je vybrána nová hodnota f_{max} na základě vygenerovaných uzlů v dané iteraci tak, že

  $f_{max} = min\{f(i) : f(i) > f_{max}\}$.

Vlastnosti IDA*

- je optimální a úplné,
- paměťová náročnost je úměrná nejdelší prozkoumané cestě,
- časová složitost závisí na počtu hodnot k, kterých může funkce $h^*(n)$ nabývat,
  - pro malé k je efektivní,
  - pro velké k je neefektivní (*Proč?*).

    ϵ-admissible A* – f_{max} se zvětšuje o konstantní hodnotu $\epsilon \Longrightarrow$ snížení počtu iterací a expandovaných uzlů, ale za cenu možnosti vygenerování řešení, které je o ϵ horší než optimální řešení.

IDA*

```
function IDA*(problem) returns a solution sequence
   inputs: problem, a problem
   static: f-limit, the current f-COST limit
           root, a node

   root ← MAKE-NODE(INITIAL-STATE[problem])
   f-limit ← f-COST(root)
   loop do
      solution, f-limit ← DFS-CONTOUR(root, f-limit)
      if solution is non-null then return solution
      if f-limit = ∞ then return failure; end
```

```
function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f-COST limit
   inputs: node, a node
           f-limit, the current f-COST limit
   static: next-f, the f-COST limit for the next contour, initially ∞

   if f-COST[node] > f-limit then return null, f-COST[node]
   if GOAL-TEST[problem](STATE[node]) then return node, f-limit
   for each node s in SUCCESSORS(node) do
      solution, new-f ← DFS-CONTOUR(s, f-limit)
      if solution is non-null then return solution, f-limit
      next-f ← MIN(next-f, new-f); end
   return null, next-f
```

Simplified memory-bounded A* (SMA*)

IDA* využívá příliš málo paměti

- přenáší minimum informace mezi iteracemi,
- nemá žádný nástroj pro uchování historie výpočtu,
- může sice zamezit generování stavů, opakujících se v aktuální cestě, ale neumí zamezit opakovanému generování stejných stavů různými cestami.

Simplified memory-bounded A* (SMA*)

IDA* využívá příliš málo paměti

- přenáší minimum informace mezi iteracemi,
- nemá žádný nástroj pro uchování historie výpočtu,
- může sice zamezit generování stavů, opakujících se v aktuální cestě, ale neumí zamezit opakovanému generování stejných stavů různými cestami.

SMA* narozdíl od IDA* využívá dostupnou paměť pro průběžné uložení nejdůležitější informace o již vygenerovaných stavech

- Zamezuje opakovanému generování uzlů, pokud paměť dovolí.
- **Zapomenuté stavy** – stavy, které musely být uvolněny z paměti, když bylo třeba vygenerovat následníka, ale už nebyla volná paměť. Uzly s nejvyšší hodnotou f jsou uvolněny z paměti.
- Průběžně aktualizuje hodnotu f podle informací o potomcích.
- Každý uzel si uchovává informaci o nejlepší cestě v "zapomenutém" podstromu.
- Je úplné, jestliže má k dispozici dostatek paměti pro uložení celé cesty k nejmělčímu řešení,
- Je optimální, jestliže má k dispozici dostatek paměti pro uložení celé cesty k nejmělčímu optimálnímu řešení.

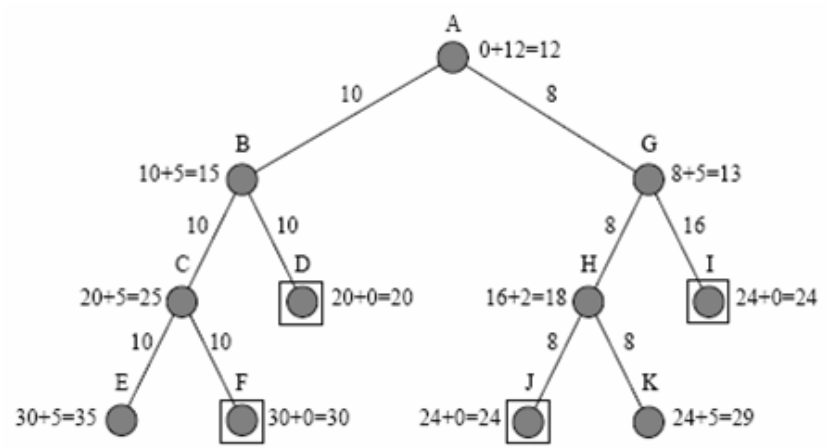
SMA*

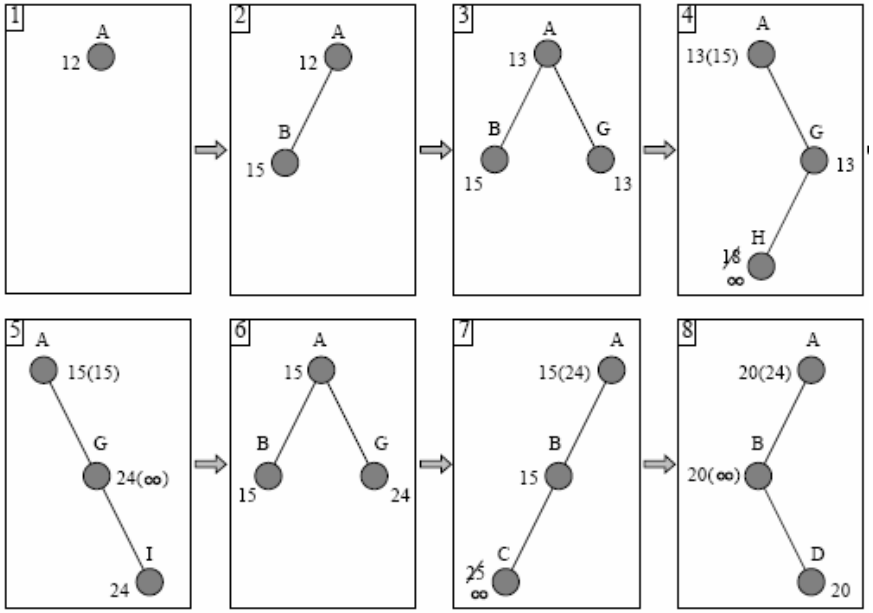
```
function SMA*(problem) returns a solution sequence
   inputs: problem, a problem
   static: Queue, a queue of nodes ordered by f-cost

   Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
   loop do
      if Queue is empty then return failure
      n ← deepest least-f-cost node in Queue
      if GOAL-TEST(n) then return success
      s ← NEXT-SUCCESSOR(n)
      if s is not a goal and is at maximum depth then
         f(s) ← ∞
      else
         f(s) ← MAX(f(n), g(s)+h(s))
      if all of n's successors have been generated then
         update n's f-cost and those of its ancestors if necessary
      if SUCCESSORS(n) all in memory then remove n from Queue
      if memory is full then
         delete shallowest, highest-f-cost node in Queue
         remove it from its parent's successor list
         insert its parent on Queue if necessary
      insert s on Queue
   end
```

Simplified memory-bounded A*: Příklad

Graf obsahuje 4 cílové uzly – D, F, I, J. Cílem je nalézt nejlevnější cílový uzel s dostupnou pamětí pro uložení tří uzlů.

- každý uzel je označen svojí cenou f, která je průběžně aktualizována (v závorce je cena nejlepšího zapomenutého následníka).





©Russel, Norvig: Artificial Intelligence: A Modern Approach.

Jak by to dopadlo, kdyby uzel J měl cenu 19 namísto 24? Jak se v SMA pozná, že nalezené řešení nemusí být optimální?*