# Heuristiky, best-first search, A* search

**Aleš Horák**

**E-mail: `hales@fi.muni.cz`**

**`http://nlp.fi.muni.cz/uui/`**

Obsah:

- ➜ Informované prohledávání stavového prostoru
- ➜ Heuristické hledání nejlepší cesty
- ➜ Příklad – řešení posunovačky
- ➜ Jak najít dobrou heuristiku?
- ➜ Příklad – rozvrh práce procesorů

# Informované prohledávání stavového prostoru

**Neinformované prohledávání:**

- ➜ DFS, BFS a varianty
- ➜ nemá (téměř) žádné informace o pozici cíle – **slepé prohledávání**
- ➜ zná pouze:
  - počáteční/cílový stav
  - přechodovou funkci

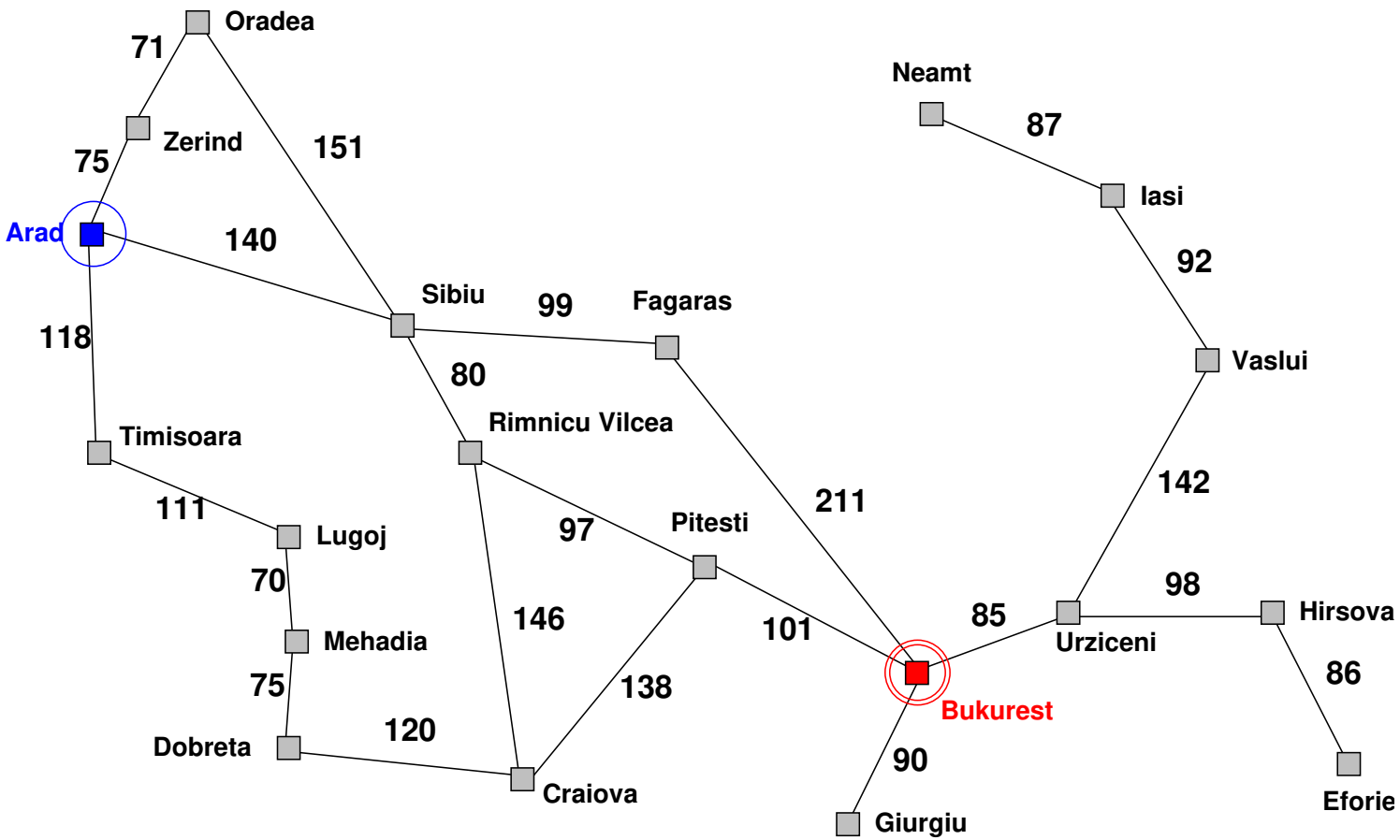
**Informované prohledávání:**

má navíc informaci o (odhadu) blízkosti stavu k cílovému stavu – **heuristická funkce** (heuristika)

# Heuristické hledání nejlepší cesty

- Best-first Search
- použití **ohodnocovací funkce** $f(n)$ pro každý uzel – výpočet **přínosu** daného uzlu
- udržujeme seznam uzlů uspořádaný (vzestupně) vzhledem k $f(n)$
- použití **heuristické funkce** $h(n)$ pro každý uzel – **odhad vzdálenosti** daného uzlu od cíle
- čím *menší* $h(n)$, tím blíže k cíli, $h(\mathrm{Goal}) = 0$.
- nejjednodušší varianta – **hladové heuristické hledání**, *Greedy best-first search*
  $f(n) = h(n)$

## Schéma rumunských měst

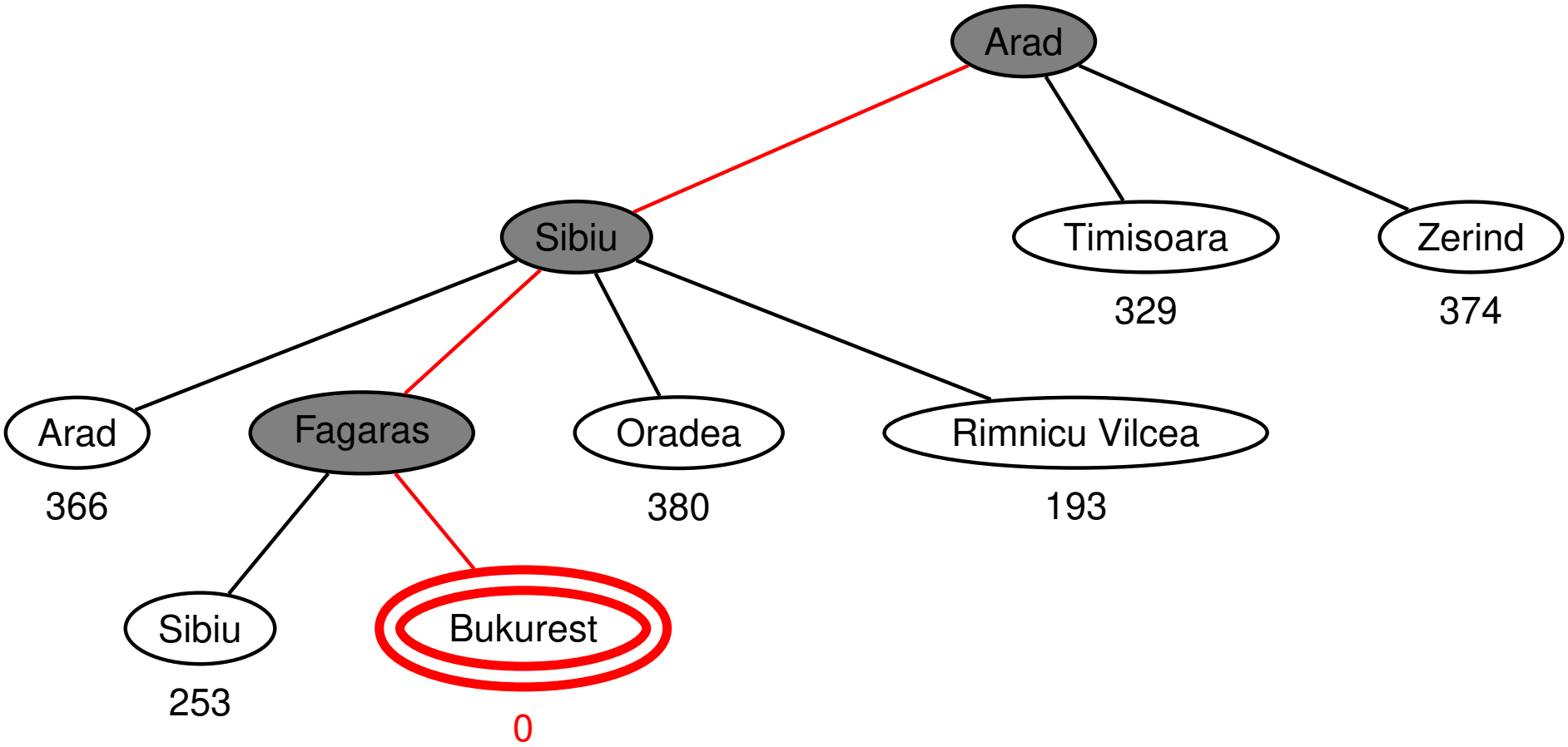



| | |
|---|---|
| **Arad** | **366** |
| **Bukurest** | **0** |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vilcea | 199 |
| Zerind | 374 |

## Hladové heuristické hledání – příklad

Hledání cesty z města *Arad* do města *Bukurest*

ohodnocovací funkce $f(n) = h(n) = h_{\mathrm{vzd_Buk}}(n)$, přímá vzdálenost z $n$ do Bukuresti



## Hladové heuristické hledání – vlastnosti

- expanduje vždy uzel, který **se zdá** nejblíže k cíli
- cesta nalezená v příkladu ($g(\mathrm{Arad} \longrightarrow \mathrm{Sibiu} \longrightarrow \mathrm{Fagaras} \longrightarrow \mathrm{Bukurest}) = 450$) je sice úspěšná, ale **není optimální** ($g(\mathrm{Arad} \longrightarrow \mathrm{Sibiu} \longrightarrow \mathrm{RimnicuVilcea} \longrightarrow \mathrm{Pitesti} \longrightarrow \mathrm{Bukurest}) = 418$)
- *úplnost* — obecně **není** úplný (nekonečný prostor, cykly)
  *optimálnost* — **není** optimální
  *časová složitost* — $O(b^m)$, hodně záleží na $h$
  *prostorová složitost* — $O(b^m)$, každý uzel v paměti

## Hledání nejlepší cesty – algoritmus A*

- některé zdroje označují tuto variantu jako Best-first Search
- ohodnocovací funkce – kombinace $g(n)$ a $h(n)$:

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

| | |
|---|---|
| $g(n)$ | je cena cesty do $n$ |
| $h(n)$ | je odhad ceny cesty z $n$ do cíle |
| $f(n)$ | je **odhad** ceny **nejlevnější cesty**, která vede přes $n$ |

- A* algoritmus vyžaduje tzv. **přípustnou** (*admissible*) heuristiku:

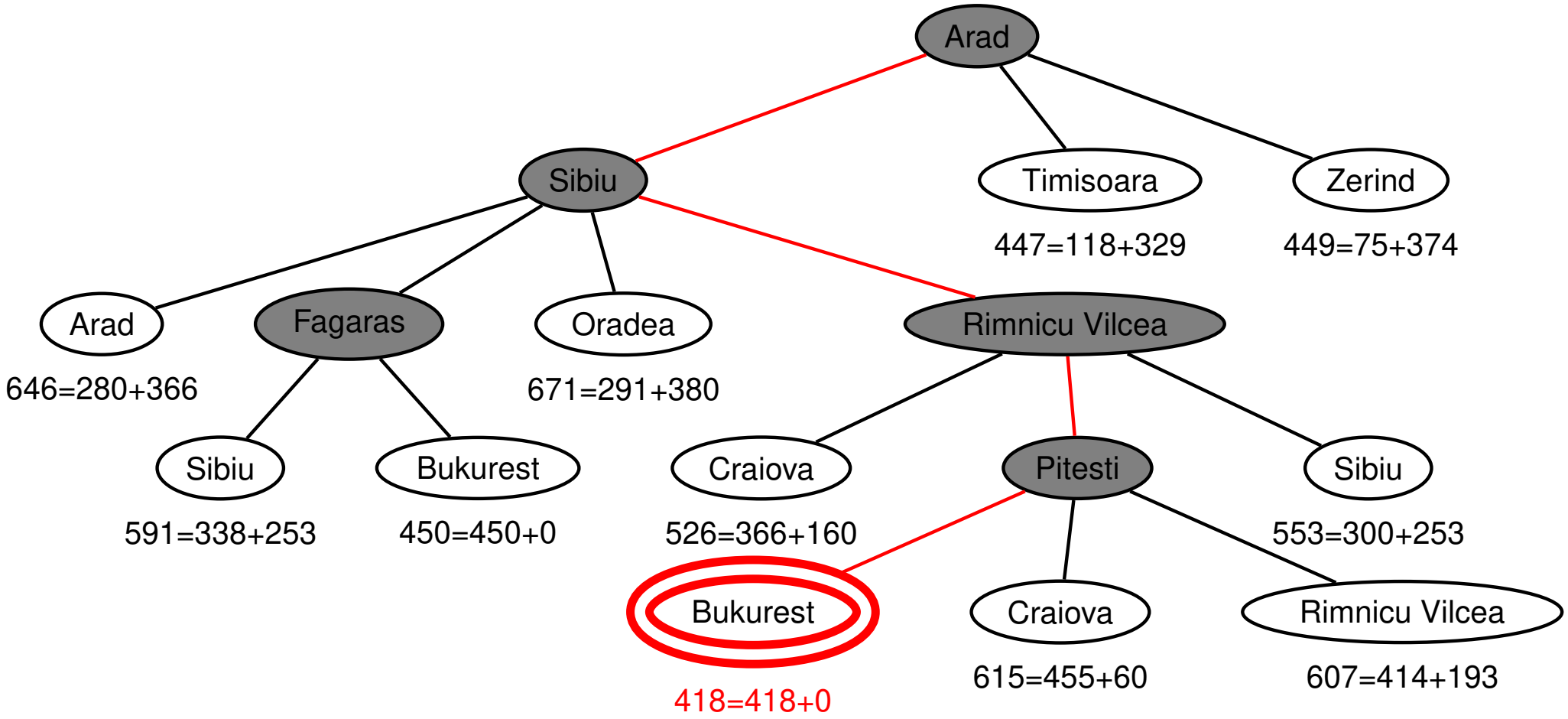
$$0 \leq h(n) \leq h^*(n), \text{ kde } h^*(n) \text{ je skutečná cena cesty z } n \text{ do cíle}$$

tj. odhad se volí vždycky **kratší** nebo roven ceně libovolné **možné** cesty do cíle
Např. přímá vzdálenost $h_{\mathrm{vzd_Buk}}$ nikdy není delší než (jakákoliv) cesta

# Heuristické hledání A* – příklad

Hledání cesty z města *Arad* do města *Bukurest*

ohodnocovací funkce $f(n) = g(n) + h(n) = g(n) + h_{\mathrm{vzd_Buk}}(n)$, přímá vzdálenost z $n$ do Bukuresti



# Hledání nejlepší cesty – algoritmus A*

reprezentace uzlů:

- **l(N,F/G)** ... listový uzel **N**, **F** $= f(N) =$ **G** $+ h(N)$, **G** $= g(N)$
- **t(N,F/G,Subs)** ... podstrom s kořenovým uzlem **N**, **Subs** seznam podstromů seřazených podle $f$, **G** $= g(N)$ a **F** $=$ $f$-hodnota nejnadějnějšího následníka uzlu $N$

**biggest(-Big)** *horní závora pro cenu nejlepší cesty např.* **biggest(9999).**

```
bestsearch(Start,Solution) :- biggest(Big), expand([],l(Start ,0/0), Big,_,yes,Solution).

expand(P,l(N,_),_,_,yes,[N|P]) :- goal(N). % cíl
% list – generuj následníky a expanduj je v rámci Bound
expand(P,l(N,F/G),Bound,Tree1,Solved,Sol) :- F=<Bound,
    (bagof(M/C,(move(N,M,C),not(member(M,P))),Succ),!,succlist(G,Succ,Ts),
    bestf(Ts,F1), expand(P,t(N,F1/G,Ts),Bound,Tree1,Solved,Sol);Solved=never).
% nelist , f<Bound – expanduj nejslibnější podstrom, pokračuj dle výsledku
expand(P,t(N,F/G,[T|Ts]),Bound,Tree1,Solved,Sol) :- F=<Bound, bestf(Ts,BF),
    min(Bound,BF,Bound1),expand([N|P],T,Bound1,T1,Solved1,Sol),
    continue(P,t(N,F/G,[T1|Ts]),Bound,Tree1,Solved1,Solved,Sol).
expand(_,t(_,_ ,[]), _,_,never,_) :- !. % nejsou další následovníci
expand(_,Tree,Bound,Tree,no,_) :- f(Tree,F), F>Bound. % limit
% pokrač. →
```

**expand(+Path,+Tr,+Bnd,-Tr1,?Solved,-Sol)**
**Path** – *cesta mezi kořenem a* **Tr**
**Tr** – *prohledávaný podstrom*
**Bnd** – $f$*-limita pro expandování* **Tr**
**Tr1** – **Tr** *expandovaný až po* **Bnd**
**Solved** – **yes**, **no**, **never**
**Sol** – *cesta z kořene do cílového uzlu*

## Hledání nejlepší cesty – algoritmus A* pokrač.

```
continue(_,_,_,_,yes,yes,Sol).
continue(P,t(N,F/G,[T1|Ts]),Bound,Tree1,Solved1,Solved,Sol) :-
    (Solved=no,insert(T1,Ts,NTs);Solved=never,NTs=Ts),
    bestf(NTs,F1),expand(P,t(N,F1/G,NTs),Bound,Tree1,Solved,Sol).

succlist(_ ,[],[]).
succlist(G0,[N/C|NCs],Ts) :- G is G0+C,h(N,H),F is G+H,
    succlist(G0,NCs,Ts1), insert(l(N,F/G),Ts1,Ts).

insert(T,Ts,[T|Ts]) :- f(T,F),bestf(Ts,F1),F=<F1,!.
insert(T,[T1|Ts],[T1|Ts1]) :- insert(T,Ts,Ts1).

f(l(_,F/_),F).
f(t(_,F/_,_),F).

bestf([T|_],F) :- f(T,F).
bestf ([], Big) :- biggest(Big).

min(X,Y,X) :- X=<Y,!.
min(X,Y,Y).
```

**continue( +Path, +Tree, +Bound, -NewTree, +SubtrSolved, ?TreeSolved, -Solution)**
*volba způsobu pokračování podle výsledků* **expand**

**succlist( +G0, [+Node1/+Cost1, ...], [l(-BestNode,-BestF/-G), ...])**
*setřídění seznamu listů podle $f$-hodnot*

*vloží* **T** *do seznamu stromů* **Ts** *podle $f$*

*"vytáhne"* **F** *ze struktury*

*nejlepší $f$-hodnota ze seznamu stromů*

# Hledání nejlepší cesty A* – vlastnosti

➜ expanduje uzly podle $f(n) = g(n) + h(n)$

A* expanduje **všechny** uzly s $f(n) < C^*$
A* expanduje **některé** uzly s $f(n) = C^*$
A* **ne**expanduje **žádné** uzly s $f(n) > C^*$

➜ *úplnost* **je** úplný (pokud $\left[\text{počet uzlů s } f < C^*\right] \neq \infty$)

*optimálnost* **je** optimální

*časová složitost* $O\big((b^*)^d\big)$, exponenciální v délce řešení $d$

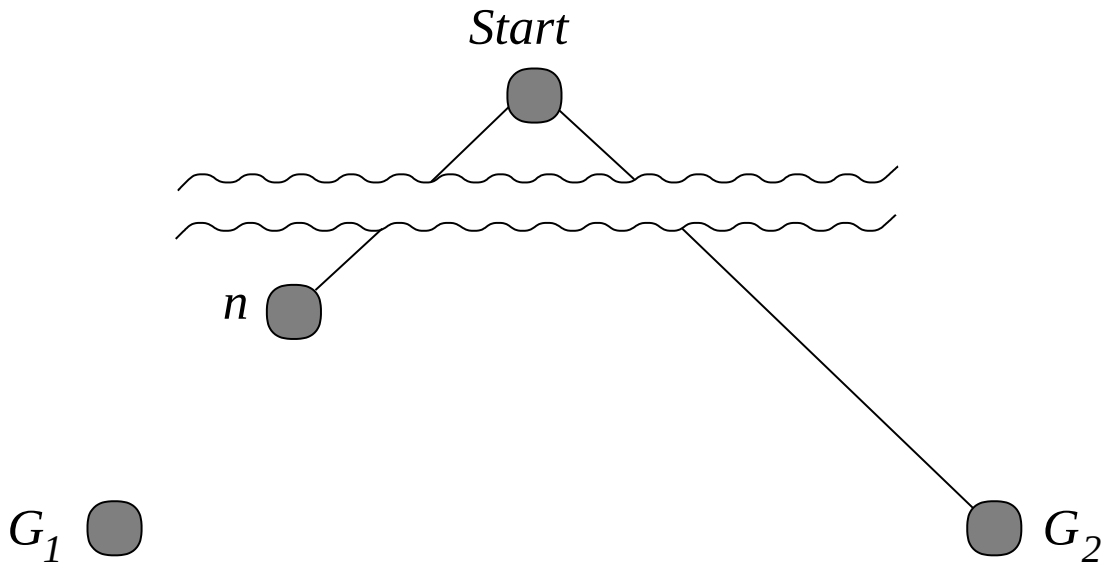
$b^*$ … tzv. *efektivní faktor větvení*, viz dále

*prostorová složitost* $O\big((b^*)^d\big)$, každý uzel v paměti

Problém s prostorovou složitostí řeší některé nedávné algoritmy (např. *Memory-bounded heuristic search*)

# Důkaz optimálnosti algoritmu A*

- předpokládejme, že byl vygenerován nějaký **suboptimální cíl** $G_2$ a je uložen ve frontě.
- dále nechť $n$ je **neexpandovaný** uzel na nejkratší cestě k **optimálnímu cíli** $G_1$ (tj. *chybně neexpandovaný* uzel ve správném řešení)




Pak

$$\begin{aligned} f(G_2) &= g(G_2) && \text{protože } h(G_2) = 0 \\ &> g(G_1) && \text{protože } G_2 \text{ je suboptimální} \\ &\geq f(n) && \text{protože } h \text{ je přípustná} \end{aligned}$$

tedy $f(G_2) > f(n)$ a $\Rightarrow$ A* nikdy nevybere $G_2$ pro expanzi dřív než expanduje $n$

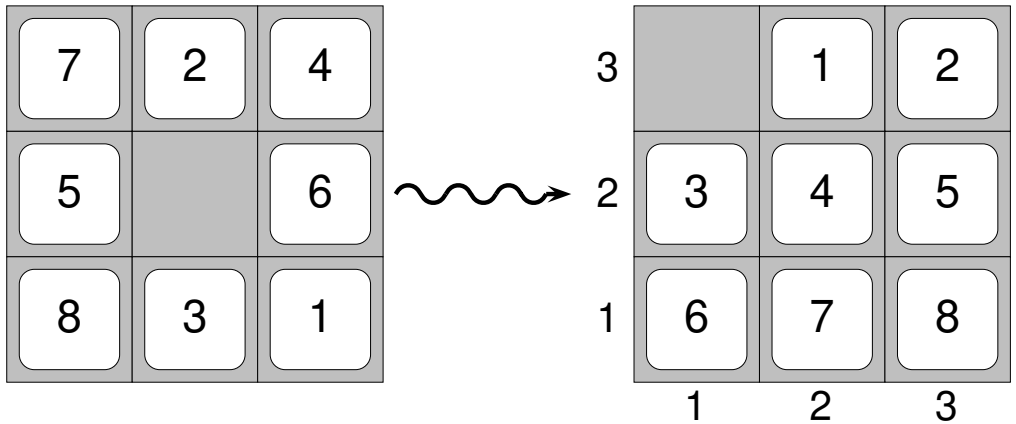
$\rightarrow$ **spor** s předpokladem, že $n$ je *neexpandovaný uzel* $\square$

# PŘÍKLAD – ŘEŠENÍ POSUNOVAČKY

konfigurace = seznam dvojic **X**/**Y** (sloupec/řádek) = [pozice$_{\text{díry}}$, pozice$_{\text{kámen č.1}}$, ...]

**goal**([1/3, 2/3, 3/3, 1/2, 2/2, 3/2, 1/1, 2/1, 3/1]).

**S**=




**Volba přípustné heuristické funkce $h$:**

➜ $h_1(n)$ = počet dlaždiček, které nejsou na svém místě $h_1(\mathbf{S}) = 8$

➜ $h_2(n)$ = součet **manhattanských vzdáleností** dlaždic od svých správných pozic

$h_2(\mathbf{S}) = 3_7 + 1_2 + 2_4 + 2_5 + 3_6 + 2_8 + 2_3 + 3_1 = 18$

$h_1$ i $h_2$ jsou přípustné ... $h^*(S) = 26$

# Jak najít dobrou heuristiku?

## Jak najít přípustnou heuristickou funkci?

- je možné najít obecné pravidlo, jak objevit heuristiku $h_1$ nebo $h_2$?
- $h_1$ i $h_2$ jsou délky cest pro **zjednodušené verze** problému Posunovačka:
  - při **přenášení** dlaždice kamkoliv – $h_1$=počet kroků nejkratšího řešení
  - při **posouvání** dlaždice kamkoliv **o 1 pole** (i na plné) – $h_2$=počet kroků nejkratšího řešení
- **relaxovaný problém** – méně omezení na akce než původní problém

  ***Cena** optimálního řešení **relaxovaného problému je přípustná heuristika** pro původní problém.*

  optimální řešení původního problému = řešení relaxovaného problému

**Posunovačka a relaxovaná posunovačka:**

- dlaždice se může přesunout z A na B $\Leftrightarrow$ A sousedí s B $\wedge$ B je prázdná.
- (a) dlaždice se může přesunout z A na B $\Leftrightarrow$ A sousedí s B. ..... $h_2$

  (b) dlaždice se může přesunout z A na B $\Leftrightarrow$ B je prázdná. ..... Gaschnigova heuristika

  (c) dlaždice se může přesunout z A na B. ..... $h_1$

## Určení kvality heuristiky

**efektivní faktor větvení** $b^*$ – $N$... počet vygenerovaných uzlů, $d$... hloubka řešení:

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

např.: když A$^*$ najde řešení po 52 uzlech v hloubce 5 ... $b^* = 1.92$

heuristika je tím **lepší**, čím **blíže** je $b^*$ **hodnotě 1**.

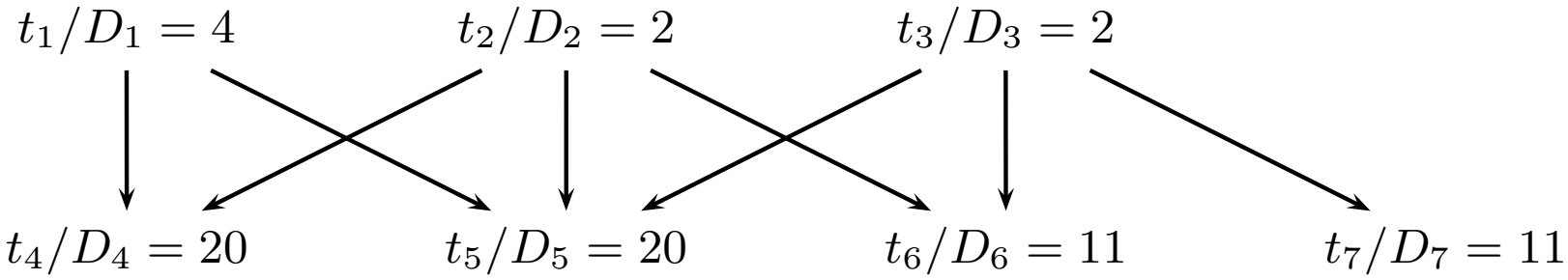
☞ **měření** $b^*$ na malé množině testovacích sad – dobrá představa o **přínosu heuristiky**

| | Průměrný počet uzlů | | | Efektivní faktor větvení $b^*$ | | |
|---|---|---|---|---|---|---|
| $d$ | IDS | A$^*(h_1)$ | A$^*(h_2)$ | IDS | A$^*(h_1)$ | A$^*(h_2)$ |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 3644035 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 18 | – | 3056 | 363 | – | 1.46 | 1.26 |
| 24 | – | 39135 | 1641 | – | 1.48 | 1.26 |

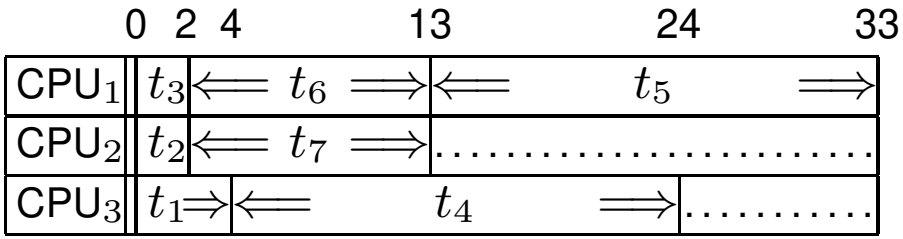
$h_2$ **dominuje** $h_1$ $\big(\forall n : h_2(n) \geq h_1(n)\big)$ ... $h_2$ je lepší (nebo stejná) než $h_1$ ve všech případech

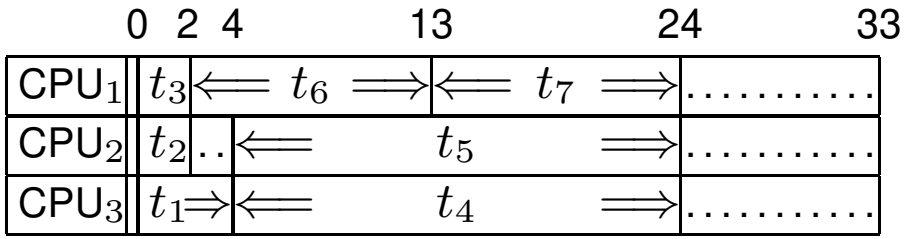
# Příklad – rozvrh práce procesorů

- **úlohy** $t_i$ s potřebným **časem** na zpracování $D_i$ (např.: $i = 1, \ldots, 7$)
- $m$ **procesorů** (např.: $m = 3$)
- relace **precedence** mezi úlohami – které úlohy mohou začít až po skončení dané úlohy




- problém: najít **rozvrh práce** pro každý procesor s minimalizací celkového času



## PŘÍKLAD – ROZVRH PRÁCE PROCESORŮ pokrač.

- stavy: **nezařazené_úlohy*zařazené_úlohy*čas_ukončení**

  např.: **[WaitingTask1/D1,WaitingTask2/D2,...]*[Task1/F1,Task2/F2,...]*FinTime**

  udržujeme **F1** $\leq$ **F2** $\leq$ **F3** . . .

- přechodová funkce **move(+Uzel, -NaslUzel, -Cena)**:

**move( +Uzel, -NaslUzel, -Cena)**
**Uzel** – *aktuální stav*
**NaslUzel** – *nový stav*
**Cena** – *cena přechodu*

```
move(Tasks1*[_/F|Active1]*Fin1, Tasks2*Active2*Fin2, Cost) :-
    del1(Task/D,Tasks1,Tasks2), not (member(T/_,Tasks2),before(T,Task)),
    not (member(T1/F1,Active1),F<F1,before(T1,Task)),
    Time is F+D, insert(Task/Time,Active1,Active2,Fin1,Fin2), Cost is Fin2-Fin1.
move(Tasks*[_/F|Active1]*Fin,Tasks*Active2*Fin,0) :- insertidle(F,Active1,Active2).

before(T1,T2) :- precedence(T1,T2).
before(T1,T2) :- precedence(T,T2),before(T1,T).

insert(S/A,[T/B|L],[S/A,T/B|L],F,F) :- A=<B,!.
insert(S/A,[T/B|L],[T/B|L1],F1,F2) :- insert(S/A,L,L1,F1,F2).
insert(S/A ,[],[S/A],_,A).

insertidle(A,[T/B|L],[idle/B,T/B|L]) :- A<B,!.
insertidle(A,[T/B|L],[T/B|L1]) :- insertidle(A,L,L1).

goal([]*_*_).
```

**before( +Task1, +Task2)**
*tranzitivní obal relace* **precedence**

# PŘÍKLAD – ROZVRH PRÁCE PROCESORŮ pokrač.

- počáteční uzel: **start** ([ t1 /4, t2 /2, t3 /2, t4 /20, t5 /20, t6 /11, t7 /11]∗[ idle /0, idle /0, idle /0]∗0).
- heuristika

  optimální (nedosažitelný) čas:

$$\textbf{Finall} = \frac{\sum_i D_i + \sum_j F_j}{m}$$

skutečný čas výpočtu: **Fin** $= \max(F_j)$

**heuristická funkce** $h$:

$$\textbf{H} = \begin{cases} \textbf{Finall} - \textbf{Fin}, & \text{když } \textbf{Finall} > \textbf{Fin} \\ 0, & \text{jinak} \end{cases}$$

```
h(Tasks ∗ Processors ∗ Fin, H) :−
  totaltime(Tasks, Tottime),
  sumnum(Processors, Ftime, N),
  Finall is (Tottime + Ftime)/N,
  ( Finall > Fin, !, H is Finall − Fin
    ;
    H = 0).

totaltime ([], 0).
totaltime([_/D | Tasks], T) :−
  totaltime(Tasks, T1), T is T1 + D.

sumnum([], 0, 0).
sumnum([_/T | Procs], FT, N) :−
  sumnum(Procs, FT1, N1),
  N is N1 + 1, FT is FT1 + T.

precedence(t1, t4). precedence(t1, t5).
...
```