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
- Jak najít dobrou heuristiku?

Příklad – cesta na mapě

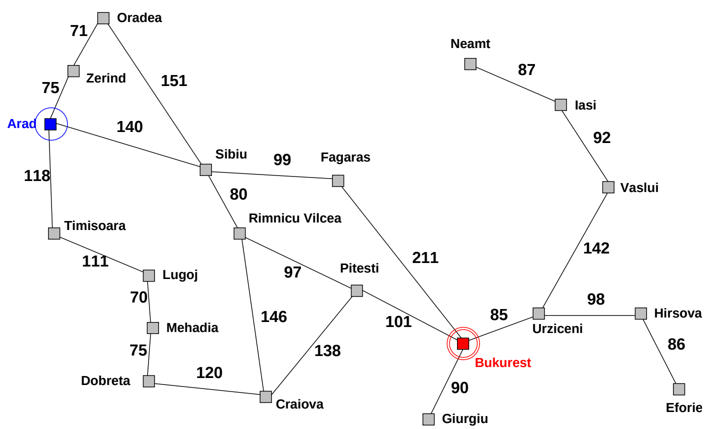
Najdi cestu z města **Arad** do města **Bukurest**

Města:
- **Arad**
- **Bukurest**
- Craiova
- Dobreta
- Eforie
- Fagaras
- Giurgiu
- Hirsova
- Iasi
- Lugoj
- Mehadia
- Neamt
- ...

Cesty:

Arad	↔	Timisoara	118
Arad	↔	Sibiu	140
Arad	↔	Zerind	75
Timisoara	↔	Lugoj	111
Sibiu	↔	Fagaras	99
Sibiu	↔	Rimnicu Vilcea	80
Zerind	↔	Oradea	71
...	↔	...	
Giurgiu	↔	**Bukurest**	90
Pitesti	↔	**Bukurest**	101
Fagaras	↔	**Bukurest**	211
Urziceni	↔	**Bukurest**	85

Příklad – 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
Vaslui	199
Zerind	374

Příklad – cesta na mapě

Neinformované prohledávání:

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

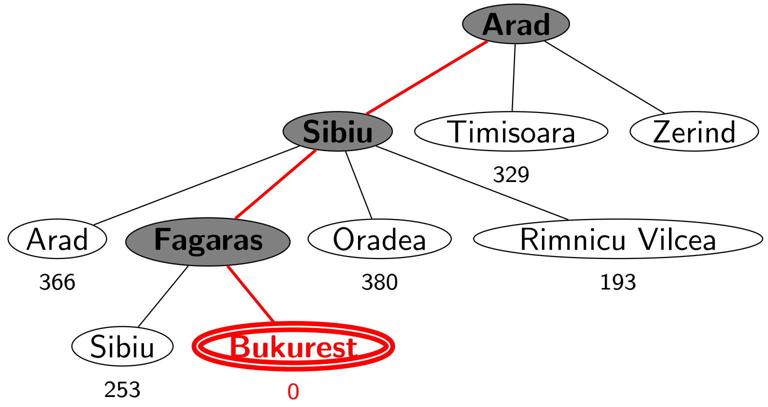
Informované prohledávání:
má navíc informaci o (odhadu) blízkosti stavu k cílovému stavu – **heuristická funkce** (heuristika)

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 (stavu) od cíle
- čím *menší* $h(n)$, tím blíže k cíli, $h(\mathrm{Goal}) = 0$.
- nejjednodušší varianta – **hladové heuristické hledání**, *Greedy best-first search*
  $f(n) = h(n)$

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 ($g(\mathrm{Arad} \to \mathrm{Sibiu} \to \mathrm{Fagaras} \to \mathrm{Bukurest}) = 450$) je sice úspěšná, ale **není optimální** ($g(\mathrm{Arad} \to \mathrm{Sibiu} \to \mathrm{RimnicuVilcea} \to \mathrm{Pitesti} \to \mathrm{Bukurest}) = 418$)
-

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 $h(n)$:

$$\mathbf{f(n) = g(n) + h(n)}$$

$g(n)$	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

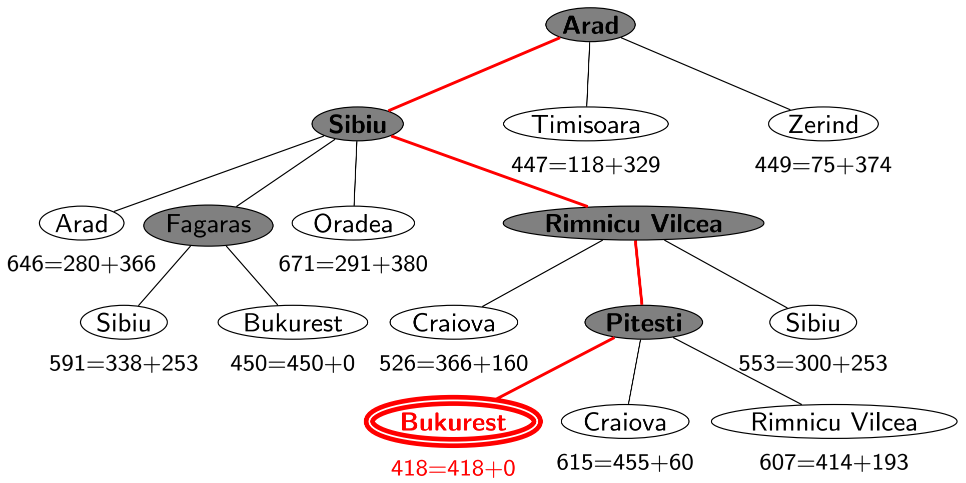
- A* algoritmus vyžaduje tzv. **přípustnou** (*admissible*) heuristiku:

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

  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 $f(n) = g(n) + h(n) = g(n) + h_{\text{vzd\_Buk}}(n)$



Hledání nejlepší cesty A* – vlastnosti

- expanduje uzly podle $\mathbf{f(n)} = g(n) + h(n)$

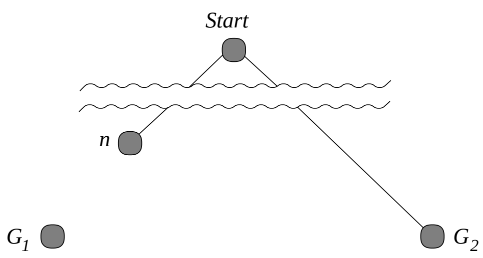
  A* expanduje **všechny** uzly s $f(n) < C^*$
  A* expanduje **některé** uzly s $f(n) = C^*$
  A* **ne**expanduje **žádné** uzly s $f(n) > C^*$

- *úplnost* **je** úplný (pokud [počet uzlů s $f < C^*$] $\neq \infty$)
  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 algoritmy jako *IDA**, *RBFS*

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

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

tedy $f(G_2) > f(n)$ $\Rightarrow$ A* nikdy nevybere G_2 pro expanzi dřív než expanduje n $\rightarrow$ **spor** s předpokladem, že n je *neexpandovaný uzel* □

Hledání nejlepší cesty – algoritmus A*

reprezentace uzlů:

- **l(N,F/G)** … listový uzel **N**, $\mathbf{F} = f(N) = \mathbf{G} + h(N)$, $\mathbf{G} = g(N)$
- **t(N,F/G,Subs)** … podstrom s kořenem **N**, **Subs** podstromy seřazené podle f, $\mathbf{G} = g(N)$ a $\mathbf{F} =$ f-hodnota nejnadějnějšího následníka 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),\+ 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(+Path, +Tree, +Bound, -NewTree, +SubtrSolved, ?TreeSolved, ?Solution)
volba způsobu pokračování podle výsledků **expand** (**SubtrSolved**)

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

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

```
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).
```

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

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

"vytáhne" **F** ze struktury

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

nejlepší f-hodnota ze seznamu stromů

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


Příklad – řešení posunovačky

konfigurace = seznam souřadnic **X/Y**: [pozice$_{\text{díry}}$, pozice$_{\text{kámen č.1}}$, ...]

```
start([2/2, 3/1, 2/3, 2/1, 3/3,
    1/2, 3/2, 1/3, 1/1]).
goal([1/3, 2/3, 3/3, 1/2, 2/2,
    3/2, 1/1, 2/1, 3/1]).
```

S=

7	2	4
5		6
8	3	1

⟿

3		1	2
2	3	4	5
1	6	7	8
	1	2	3

move(+Uzel, -NaslUzel,-Cena) pomocí pohybů mezery (cena vždy 1)

```
move([XB/YB | Numbers], [XL/YB | NewNumbers], 1) :- % doleva
    XB>1, XL is XB − 1, replace(XL/YB, XB/YB, Numbers, NewNumbers).
move([XB/YB | Numbers], [XR/YB | NewNumbers], 1) :- % doprava
    XB<3, XR is XB + 1, replace(XR/YB, XB/YB, Numbers, NewNumbers).
move([XB/YB | Numbers], [XB/YD | NewNumbers], 1) :- % dolu
    YB>1, YD is YB − 1, replace(XB/YD, XB/YB, Numbers, NewNumbers).
move([XB/YB | Numbers], [XB/YU | NewNumbers], 1) :- % nahoru
    YB<3, YU is YB + 1, replace(XB/YU, XB/YB, Numbers, NewNumbers).

% replace(+Co, +Cim, +Seznam, −NovySeznam)
replace(Co,Cim,[Co|T],[Cim|T]):- !.
replace(Co,Cim,[H|T1],[H|T2]) :- replace(Co,Cim,T1,T2).
```


Příklad – řešení posunovačky pokrač.

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 ... $h^*(S) = 26$

```
:- start(Start), bestsearch(Start, Solution),
   reverse(Solution, RSolution), writelist(RSolution).
 1: [2/2, 3/1, 2/3, 2/1, 3/3, 1/2, 3/2, 1/3, 1/1]
 2: [1/2, 3/1, 2/3, 2/1, 3/3, 2/2, 3/2, 1/3, 1/1]
...
26: [1/2, 2/3, 3/3, 1/3, 2/2, 3/2, 1/1, 2/1, 3/1]
27: [1/3, 2/3, 3/3, 1/2, 2/2, 3/2, 1/1, 2/1, 3/1]
```


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 ⇔ A sousedí s B ∧ B je prázdná
- (a) dlaždice se může přesunout z A na B ⇔ A sousedí s B .. h_2
  (b) dlaždice se může přesunout z A na B ⇔ B je prázdná ... Gaschnigova h.
  (c) dlaždice se může přesunout z A na B .... h_1

Určení kvality heuristiky

efektivní faktor větvení b^* – $N\ldots$ počet vygenerovaných uzlů, $d\ldots$ hloubka řešení, idealizovaný strom s $N+1$ uzly má faktor větvení b^* (reálné číslo):

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

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 množině testovacích sad – dobrá představa o **přínosu heuristiky**

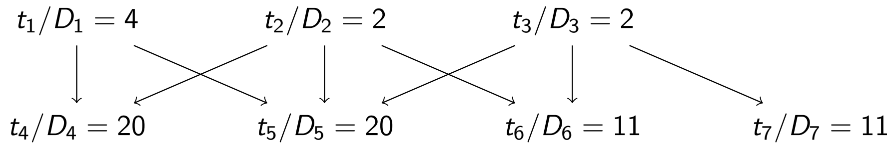
8-posunovačka

	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

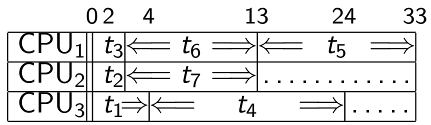
h_2 **dominuje** h_1 $\big(\forall n : h_2(n) \geq h_1(n)\big)$ ... h_2 je **lepší** (nebo stejná) než h_1

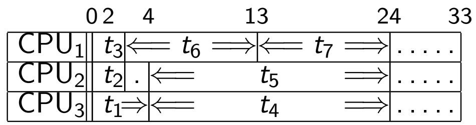
Příklad – rozvrh práce procesorů

- **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*běžící_úlohy*čas_ukončení**

  např.: **[WaitingT1/D1,WaitingT2/D2,...]*[Task1/F1,Task2/F2,Task3/F3]*FinTime**

  běžící_úlohy udržujeme setříděné $\mathbf{F1} \leq \mathbf{F2} \leq \mathbf{F3}$
- přechodová funkce **move(+Uzel, -NaslUzel, -Cena)**:

```
move(Tasks1*[_/F|Active1]*Fin1, Tasks2*Active2*Fin2, Cost) :-
    del1(Task/D,Tasks1,Tasks2),
    \+ (member(T/_,Tasks2),before(T,Task)), % kontrola predence v čekajících
    \+ (member(T1/F1,Active1),F<F1,before(T1,Task)), % a v zařazených úlohách
    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).
```

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

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


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:

$$\textbf{Fin} = \max(F_j)$$

heuristická funkce h:

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

```
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).
...
```

Příklad – rozvrh práce procesorů – pokrač.

```
:- start(Start), write('Pocatecni stav:'), write(Start), nl,
     bestsearch(Start, Solution),
     write('Nalezene reseni:'), nl,
     reverse(Solution,RSolution), writelist(RSolution).
```

```
Pocatecni stav: [t1/4,t2/2,t3/2,t4/20,t5/20,t6/11,t7/11]*[idle/0,idle/0,idle/0]*0
Nalezene reseni:
1: [t1/4,t2/2,t3/2,t4/20,t5/20,t6/11,t7/11]*[idle/0,idle/0,idle/0]*0
2: [t1/4,t2/2,t4/20,t5/20,t6/11,t7/11]*[idle/0,idle/0,t3/2]*2
3: [t1/4,t4/20,t5/20,t6/11,t7/11]*[idle/0,t2/2,t3/2]*2
4: [t4/20,t5/20,t6/11,t7/11]*[t2/2,t3/2,t1/4]*4
5: [t4/20,t5/20,t6/11]*[t3/2,t1/4,t7/13]*13
6: [t4/20,t5/20,t6/11]*[idle/4,t1/4,t7/13]*13
7: [t5/20,t6/11]*[t1/4,t7/13,t4/24]*24
8: [t6/11]*[t7/13,t5/24,t4/24]*24
9: []*[t6/24,t5/24,t4/24]*24
```