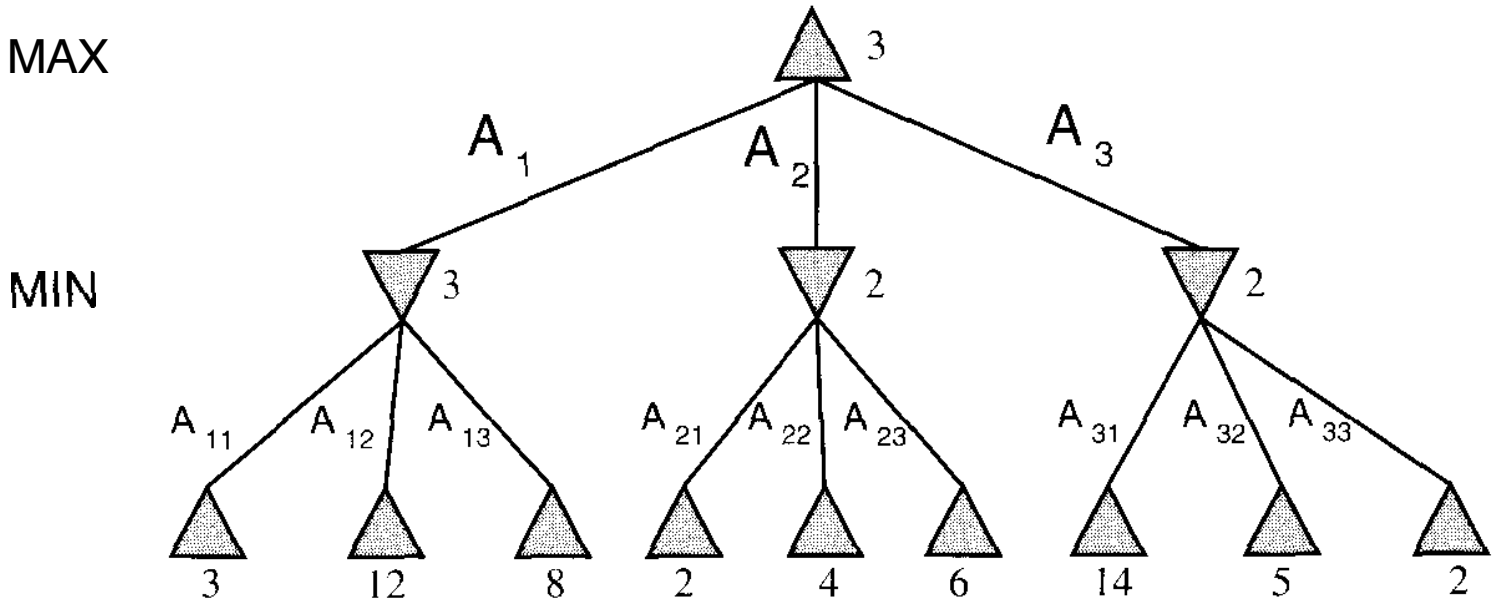
Speciální oborový seminář

Umělá inteligence

26. 2. 2019



Prohledávání stavového prostoru

Množina stavů S

Počáteční stav s_0

Množina cílových stavů G

Množina operátorů F=$\{f_1, f_2, \ldots, f_k\}$

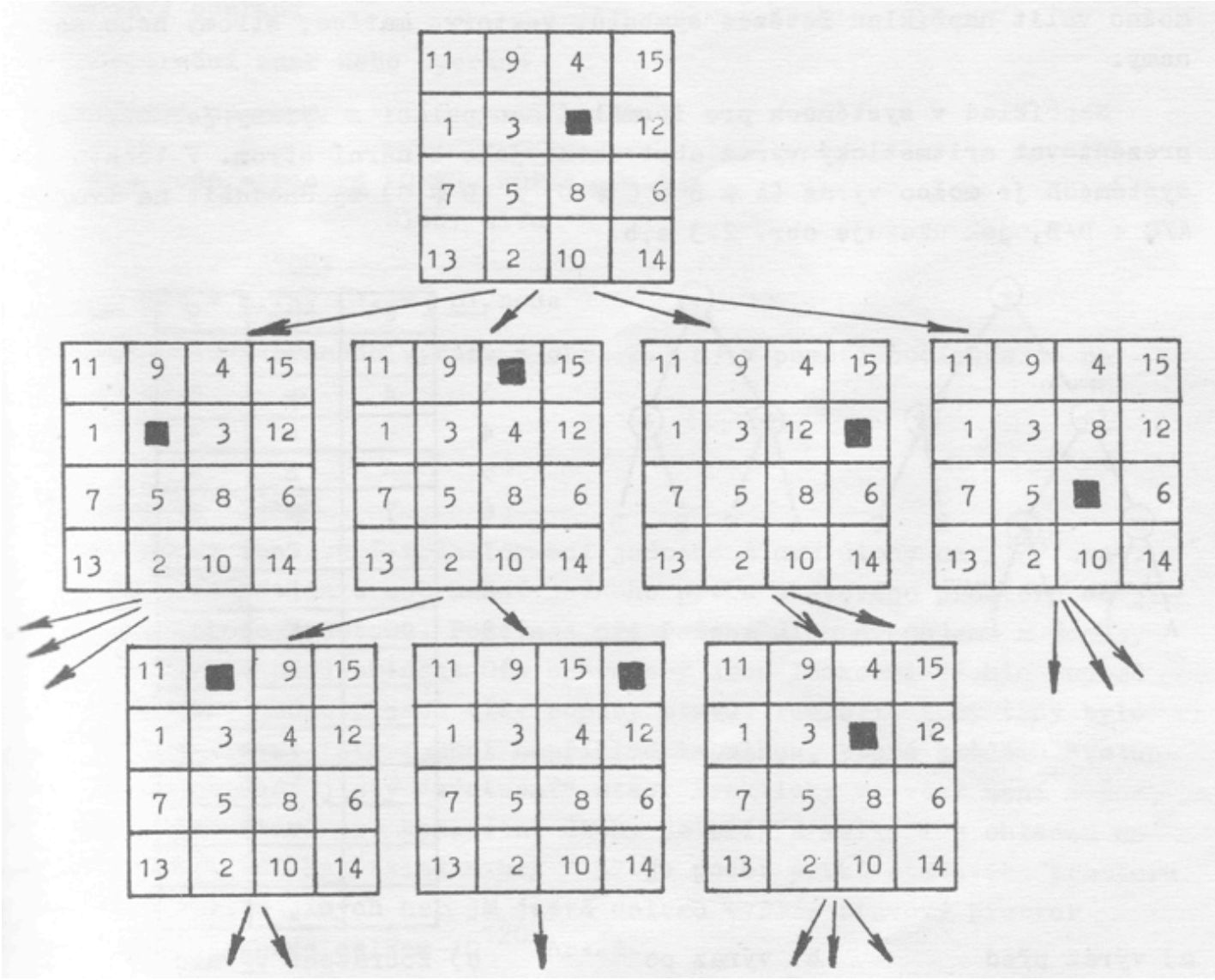
$\Gamma(x)$ = množina stavů, které vzniknou aplikací operátorů z F na stav x

Reprezentace stavového prostoru orientovaným grafem (stromem)

Expanze vrcholu x

- aplikace $\Gamma(x)$

Příklad: Lloydova “15”



Prohledávání stavového prostoru

Průchod do hloubky (DFS)

Průchod do šířky (BFS)

Best-first search (uspořádaný výběr)

Heuristické prohledávání

- ohodnocovaci funkce f(x)=g(x)+h(x)

Algoritmus A*

Prohledávání stavového prostoru: uspořádaný výběr

Obecnější situace

- přechod ze stavu s_1 do stavu s_2 má přiřazenu cenu $c(s_1,s_2)$
- cena řešení $s_1,\ldots,s_n = \sum c(s_i,s_{i+1})$
- hledáme optimální řešení = řešení minimální ceny
- příklad: problém obchodního cestujícího

Uspořádaný výběr

- "expanze" stavu minimální ceny
- variace na téma Dijkstra

Heuristické prohledávání

Rafinovanější strategie

- expanze stavu, který má "největší šanci" na to, že povede k cíli
- jak takový stav najít?

Zavedeme ohodnocující funkci f

- $f(s) = g(s) + h(s)$
  - » $g(s)$ = cena optimální cesty ze startu do s
  - » $h(s)$ = cena optimální cesty z s do cíle
- g ani h neznáme $\Rightarrow$ použijeme odhad
- $\hat{f}(s) = \hat{g}(s) + \hat{h}(s)$

Heuristické prohledávání: A*

Algoritmus A*

- používá ohodnocující funkci $\hat{\mathrm{f}}(s) = \hat{\mathrm{g}}(s) + \hat{\mathrm{h}}(s)$
- kde $\hat{\mathrm{g}}(s)$= cena nalezené cesty se startu do s
- jak odhadnout $\hat{\mathrm{h}}(s)$?

Věta. Pokud existuje δ>0 tak, že cena žádné hrany neklesne pod δ a $\hat{\mathrm{h}}(s) \leq h(s)$ pro každý stav s, pak první řešení nalezené algoritmem A* je řešení optimální.

Heuristické prohledávání: A*

Buďte A_1 a A_2 algoritmy A* s heuristickými funkcemi $\hat{h}_1$ a $\hat{h}_2$. Říkáme, že A_1 je *lépe informován* než A_2, pokud pro každý stav s platí $\hat{h}_1(s) \geq \hat{h}_2(s)$.

- Je-li A_1 lépe informován než A_2, pak A_2 expanduje všechny stavy, které expanduje A_1. Naopak to obecně neplatí.

Přednáška Umělá inteligence I NAIL069

Metoda větví a mezí.

Příklad 1 (motivační): Piškvorky

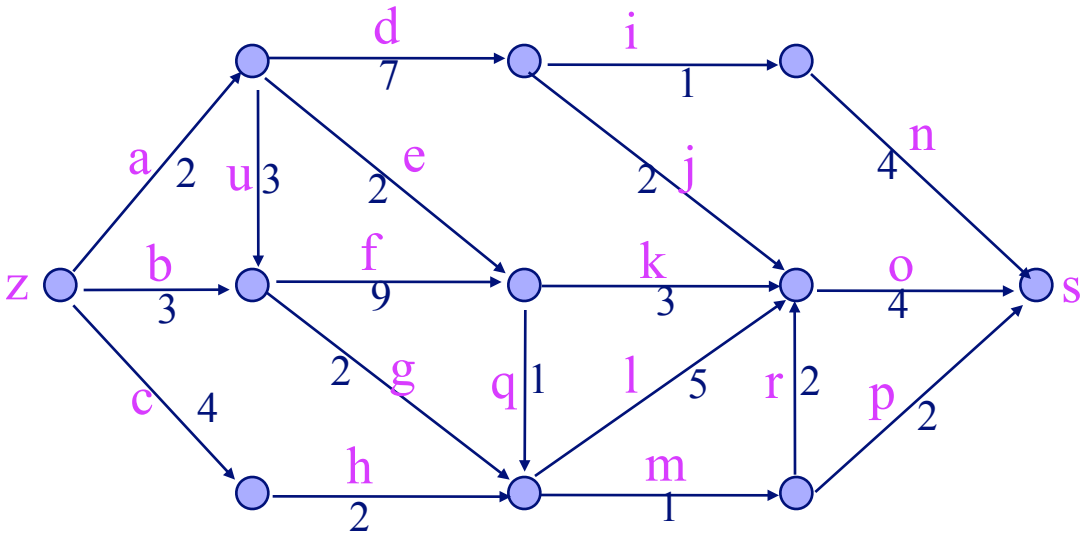
Minimaxový algoritmus

- generování stromu hry do hloubky *h*
- každý list *l* ohodnocen funkcí f(*l*)
- je-li určeno ohodnocení všech synů, lze ohodnotit otce

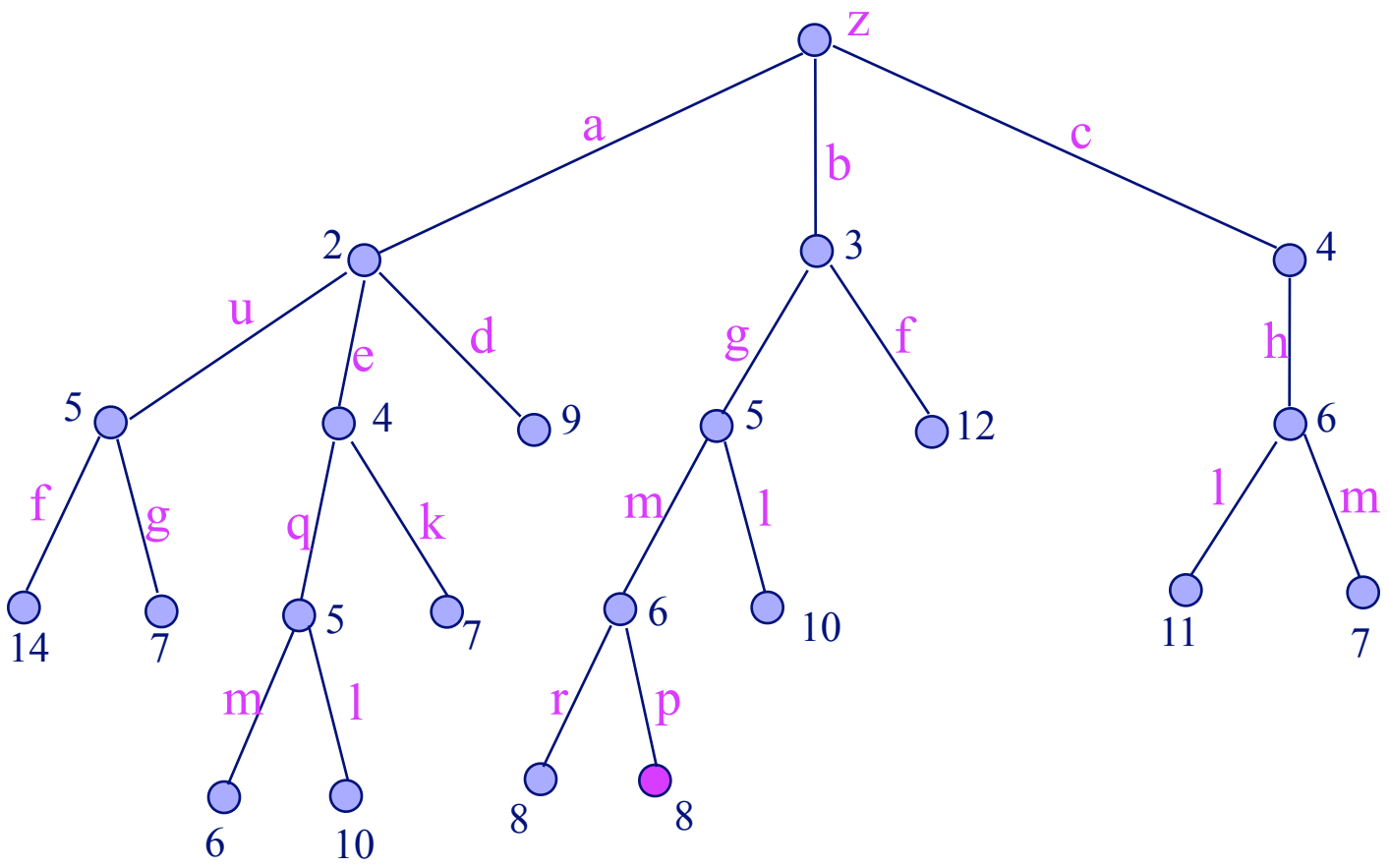
Zrychlení: α-β procedura

- „odřezávání“ neperspektivních větví stromu hry

Příklad 2 (nerealistický): Metoda větví a mezí pro problém nejkratší cesty



Metoda větví a mezí pro problém nejkratší cesty - pokračování



Popis metody větví a mezí

Optimalizační problém: hledá se max/min jisté veličiny

Omezíme se na minimalizační problémy

Potřebujeme

- větvení: množinu (částečných) řešení, reprezentovanou jedním vrcholem stromu řešení, lze rozložit na podmnožiny. Každá podmnožina je pak reprezentována synem původního vrcholu
- dolní mez: máme k dispozici efektivní algoritmus pro výpočet dolní meze ceny každého řešení v dané podmnožině

Algoritmus

AktivníMnožina := {0} //0 je nějaké počáteční řešení
DolníMez := ∞; ZatímNejŘešení := NIL;

while AktivníMnožina ≠ ∅ **do**

vyber (a odstraň) v z AktivníMnožiny;

urči syny $s_1,\ldots,s_k$ vrcholu v

pro každé i spočti $z(s_i)$ =dolní mez na hodnotu řešení s_i

for i:=1 **to** k **do if** $z(s_i) \geq$ DolníMez **then** odstraň s_i

else if s_i je úplné řešení **then** DolníMez:=$z(s_i)$;
ZatímNejŘešení:= s_i

else AktivníMnožina :=AktivníMnožina$\cup\{s_i\}$

Jak vybrat vrchol *v* z AktivníMnožiny?

LIFO (DFS)

FIFO (BFS)

vrchol s minimální hodnotou heuristické funkce

populární volba:

- vrchol *v* s minimální hodnotou z(*v*)

Příklad 3 (realistický): Problém obchodního cestujícího

Definice: *1-kostrou* grafu G s vrcholy V(G)={1,..,n} rozumíme graf, tvořený kostrou grafu G -{1}, k němuž přidáme vrchol 1 jakož i 2 hrany s ním incidentní.

Buď dáno ohodnocení hran c:E(G) → **N**. Definujme ohodnocení1-kostry K jako $c(K):=\Sigma_{e\in E(K)}c(e)$. *Minimální 1-kostra* je 1-kostra s minimálním ohodnocením.

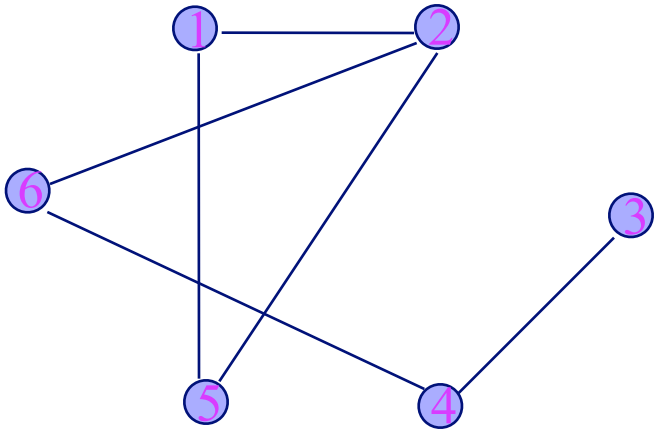
Existuje efektivní algoritmus, který určí minimální 1-kostru.

Každá cesta obchodního cestujícího je 1-kostra.
Je-li c* cena optimání cesty obchodního cestujícího, platí

$$\min_{K \text{ je 1-kostra}} c(K) \leq c^*$$

Příklad - zadání & první krok

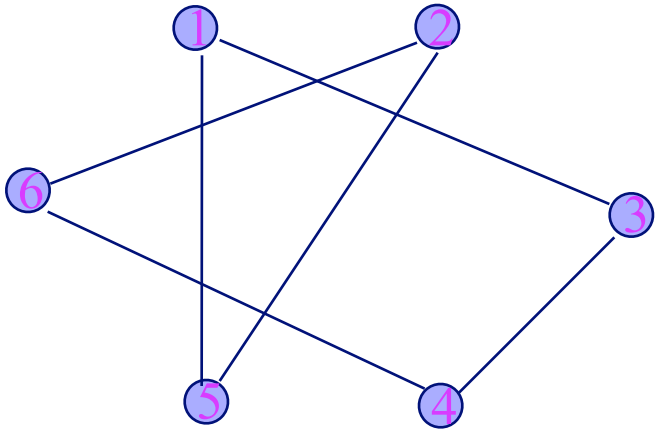
	1	2	3	4	5	6
1	0	4	10	18	5	10
2	4	0	12	8	2	6
3	10	12	0	4	18	16
4	18	8	4	0	14	6
5	5	2	18	14	0	16
6	10	6	16	6	16	0



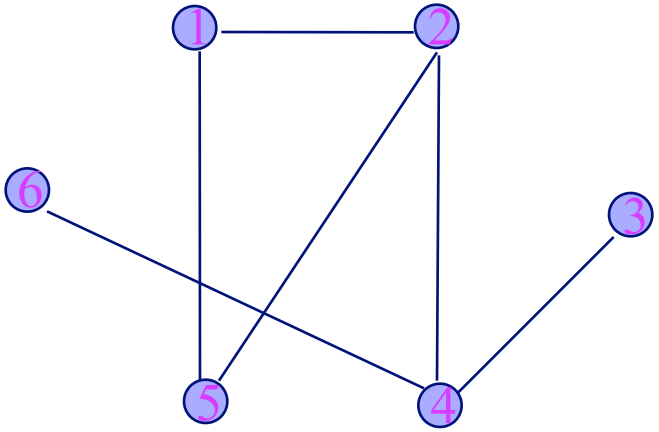
0

0 27

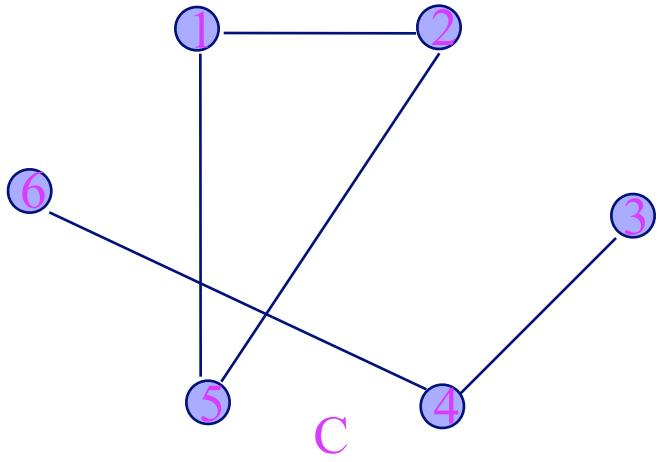
Příklad - expanze kořene stromu řešení



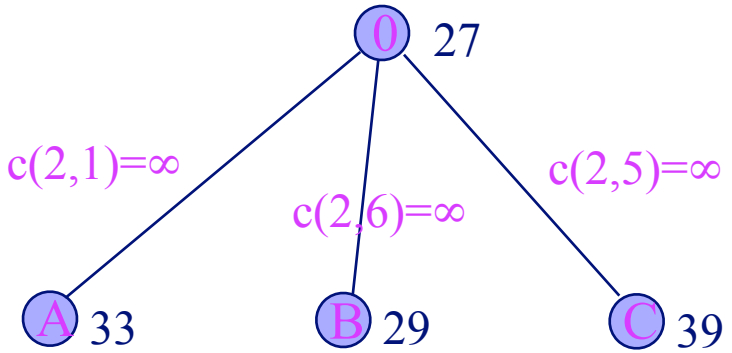
A



B



C



Zlepšení strategie pro POC (Held,Karp,1971)

Lepší dolní odhad z

Idea: Pro každý vrchol *i* urči jeho penalizaci π_i a transformuj ohodnocení hran

$c(i,j) \rightarrow c'(i,j) := c(i,j) + \pi_i + \pi_j$

Cena každé cesty obchodního cestujícího se zvýší o $2\Sigma\pi_i$

$c'^* \geq \min_{K \text{ je 1-kostra}} c'(K)$

$c^* + 2\Sigma\pi_i \geq \min_{K \text{ je 1-kostra}} [c(K) + \Sigma d_K(i)\pi_i]$

$c^* \geq \min_{K \text{ je 1-kostra}} [\underbrace{c(K)}_{z} + \Sigma(d_K(i)-2)\pi_i]$

Jak určit π, aby dolní odhad byl co nejtěsnější?

Určení π(t,p) //t - zvolená konstanta,p - # iterací

for i:=1 **to** n **do** $\pi_i := 0$

for j:=1 **to** p **do** najdi minimální 1-kostru K vzhledem k cenám c’

for i:=1 **to** n

do $\pi_i := \pi_i + t\,(d_K(i)-2)$

Zlepšení strategie pro POC - pokračování

Rozklad na navzájem disjunktní podproblémy

Množiny X=hrany, které musím použít

Y=hrany, které nesmím použít

$z_{X,Y}(\pi) = \min \{c(K)+\Sigma_{i=1}^{n}(d_k(i)-2)\pi_i \mid$

K je 1-kostra, $E(K) \supseteq X$, $Y \cap E(K) \neq \emptyset\}$

Každý vrchol bude ohodnocen čtveřicí

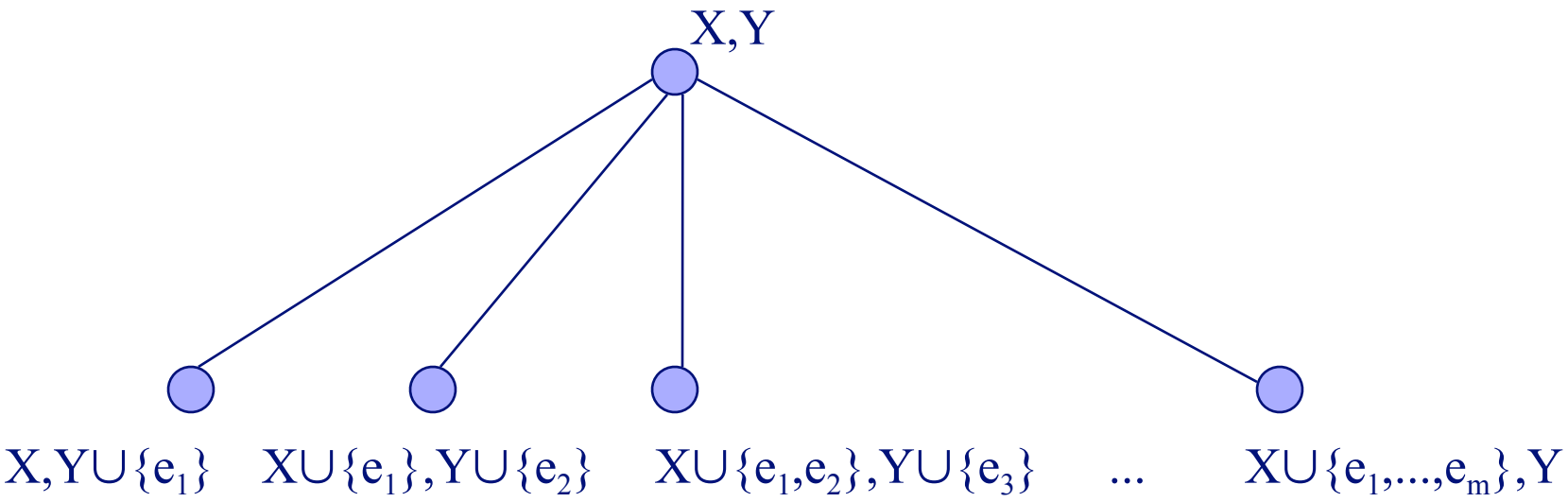
$(X,Y,\pi,z_{X,Y}(\pi))$

Inicializace $(\emptyset, \emptyset,0,z(0))$

Všechny hrany, které neleží v $X \cup Y$, uspořádáme do posloupnosti $e_1, e_2, \ldots, e_m$ tak, aby

$$z_{X,Y \cup \{e_1\}}(\pi) \geq z_{X,Y \cup \{e_2\}}(\pi) \geq \ldots \geq z_{X,Y \cup \{e_m\}}(\pi)$$

Rozklad na navzájem disjunktní podproblémy



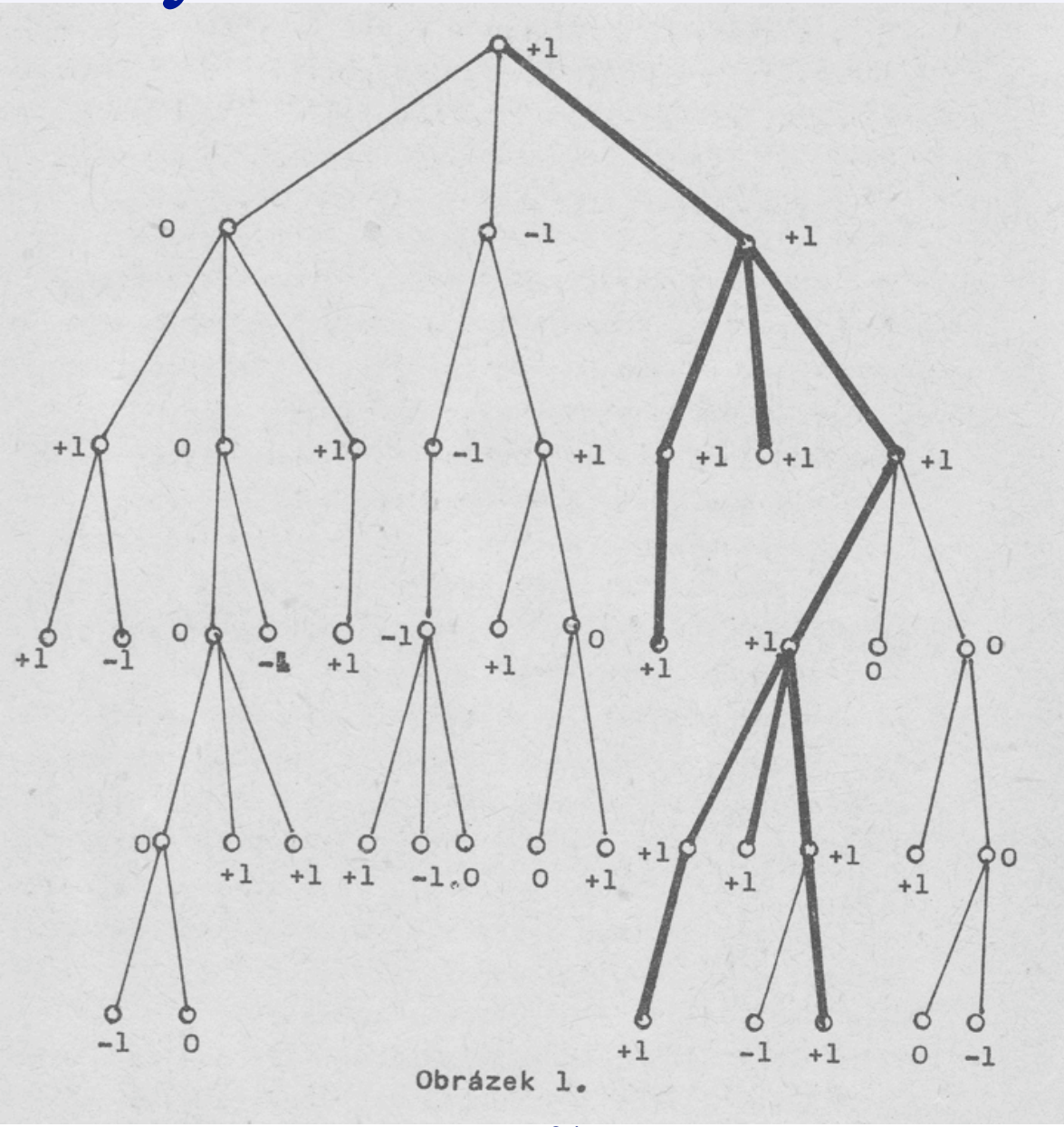
Prohledávání stromu hry

Hra

- 2 hráči: bílý, černý
- s úplnou informací
  - » *pozice*
- konečnost
  - » konečně mnoho tahů z každé přípustné pozice
  - » každá *partie* je konečná
- 3 (2) možný výsledky hry
  - » vyhraje bílý, vyhraje černý, remiza

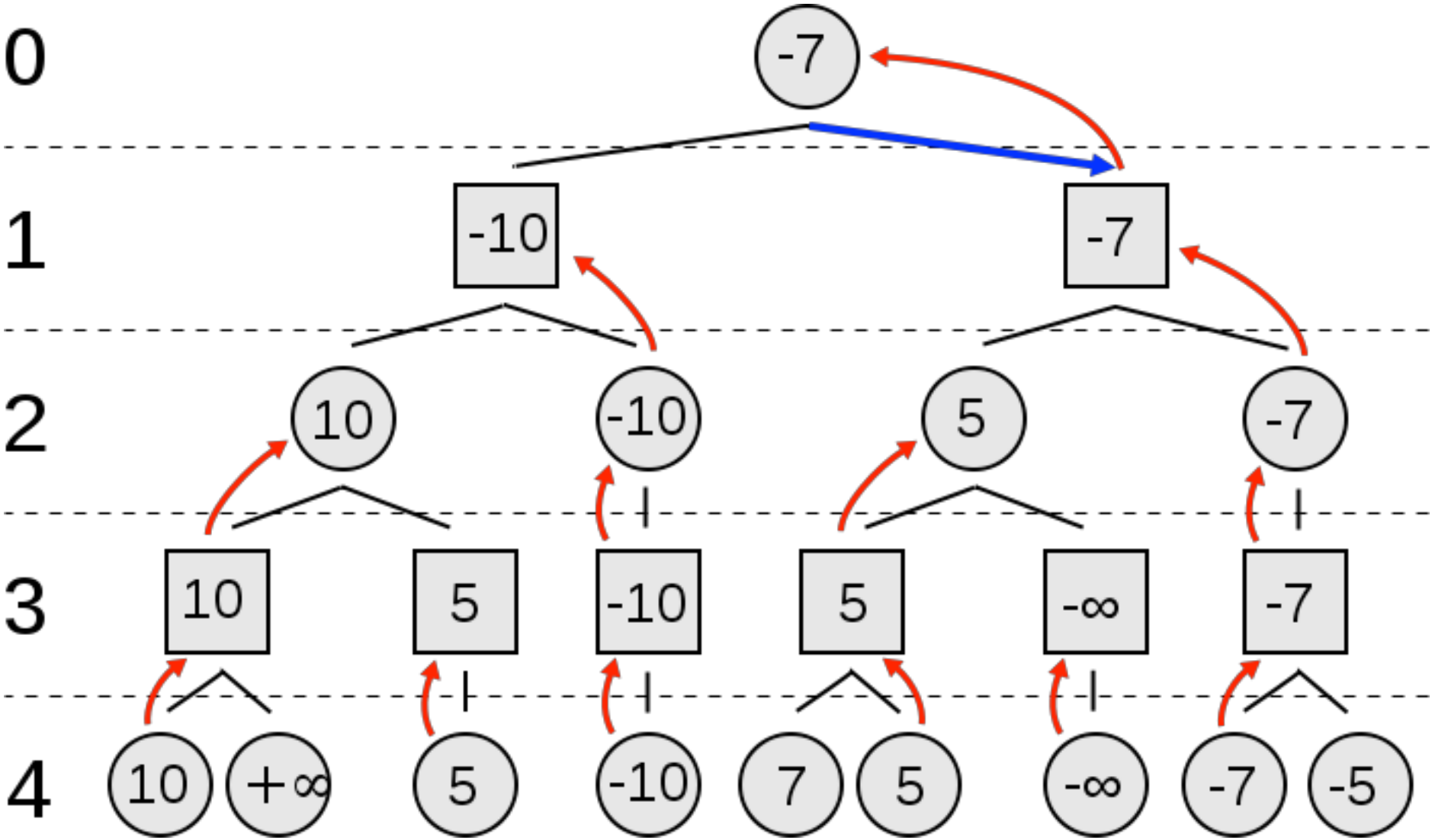
Pro každou hru tohoto typu existuje neprohrávací (vítězná) strategie jednoho z hráčů.

Strom hry



Obrázek 1.

Minimax s omezenou hloubkou

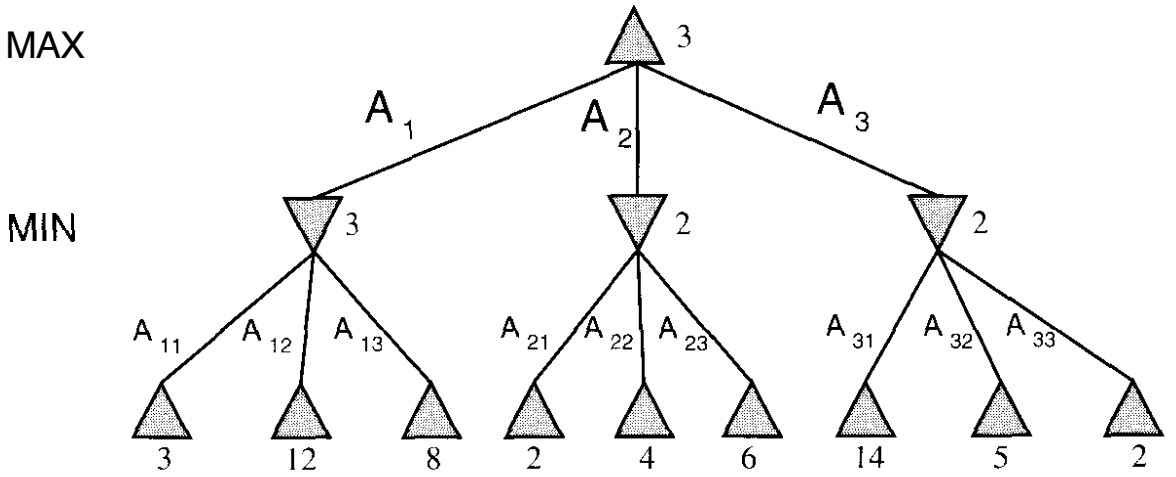


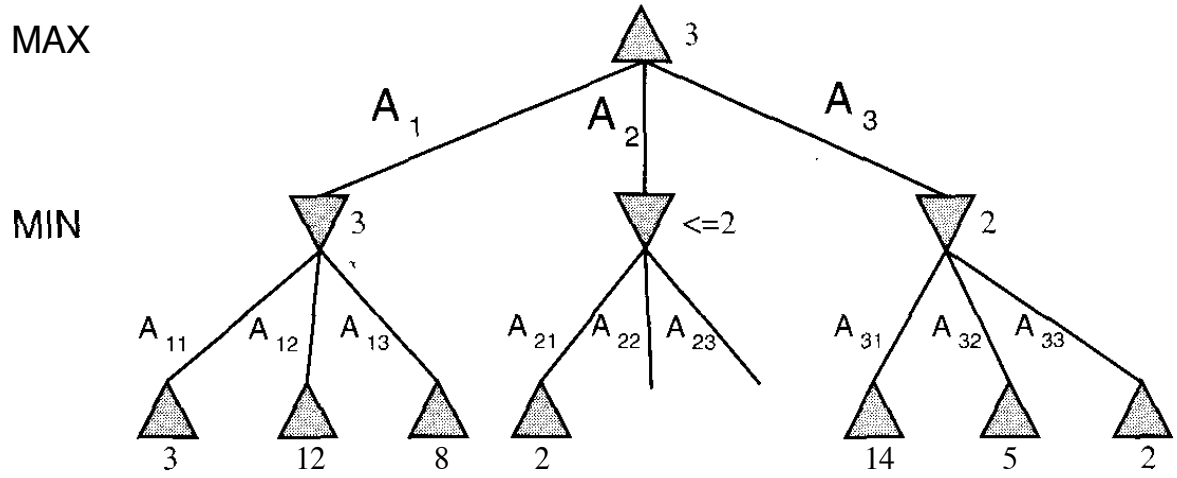
Minimaxový algoritmus

```
function minimax(vrchol, hloubka)
 if  hloubka = 0 or vrchol je koncový
 then return ohodnocení vrcholu

 if hraje_max(vrchol) then
  nej := -∞
  foreach dítě vrcholu do
   nej := max(nej, minimax(dítě, hloubka-1))
 else /* hraje_min(vrchol) */
  nej := ∞
  foreach dítě vrcholu do
   nej := min(nej, minimax(dítě, hloubka-1))
 return nej
```

Prořezávání minimaxového stromu

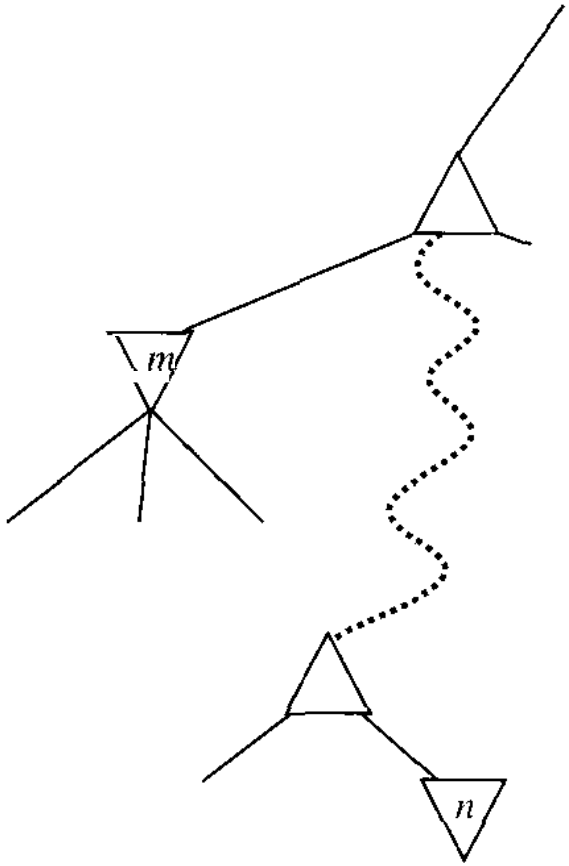




Prořezávání minimaxového stromu

Hráč

Oponent



Hráč

Oponent

m je pro Hráče lepší než *n*

⇒ Hráč si nevybere *n*

⇒ větev do *n* je zbytečná!

Alfa-beta procedura

Minimax ⇒ DFS ⇒ stačí uvažovat cestu z kořene do aktuálního vrcholu

α := nejlepší volba pro hráče MAX na této cestě
β := nejlepší volba pro hráče MIN na této cestě

Odříznout podstrom (= ukončit rekurzivní volání)

- lze v případě
- že vygenerovaná hodnota bude zcela jistě
- ležet mimo interval $\langle \alpha, \beta \rangle$

```
/* počáteční volání */
alphabeta(start, hloubka, -∞, +∞)
```

Alfa-beta procedura

```
function alphabeta(vrchol, hloubka, α, β)
 if  hloubka = 0 or vrchol je koncový
 then return ohodnocení vrcholu

 if hraje_max(vrchol) then
  nej := -∞
  foreach dítě vrcholu do
   nej := max{nej,alphabeta(dítě, hloubka-1, α, β)}
   α := max{α,nej}
   if β ≤ α then break  /* β-řez */
  return nej

 else /* hraje_min(vrchol) */
  nej := ∞
  foreach dítě vrcholu do
   nej := min{nej,alphabeta(dítě, hloubka-1, α, β)}
   β := min{β,nej}
   if β ≤ α then break /* α-řez */
  return nej
```

Alfa-beta procedura: alternativa

Vrací jen hodnoty z $\langle \alpha, \beta \rangle$

```
function alphabeta(vrchol, hloubka, α, β)
 if  hloubka = 0 or vrchol je koncový
 then return ohodnocení vrcholu

 if hraje_max(vrchol) then
  foreach dítě vrcholu do
   α := max{α,alphabeta(dítě, hloubka-1,α, β)}
   if β ≤ α then return β  /* β-řez */
  return α

 else /* hraje_min(vrchol) */
  foreach dítě vrcholu do
   β := min{β,alphabeta(dítě, hloubka-1, α, β)}
   if β ≤ α then return α /* α-řez */
  return β
```

Negamax

Sloučení obou větví (max / min) výpočtu

- max(a,b) = -min(-a,-b)

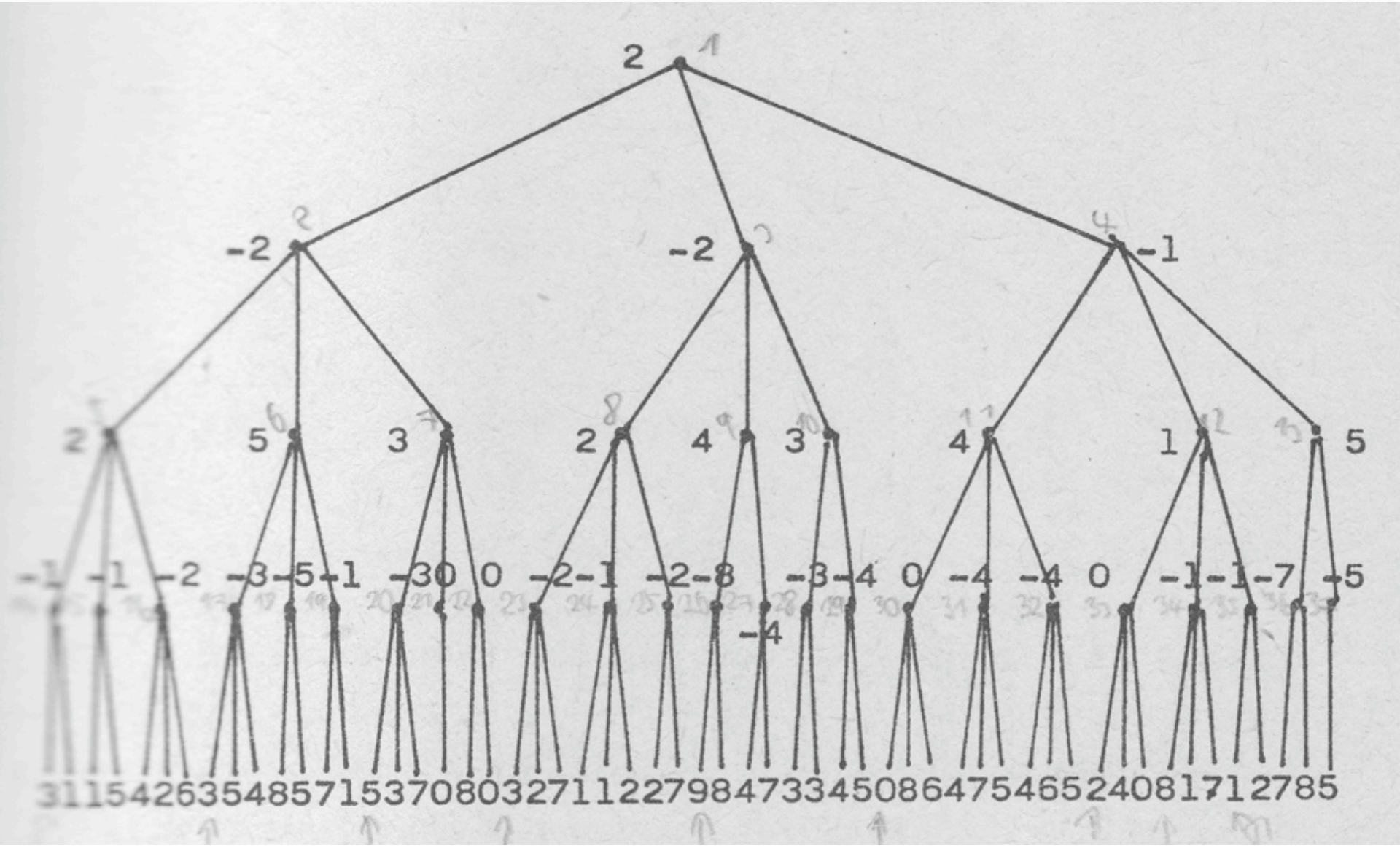
```
function minimax(vrchol, hloubka)

 if  hloubka = 0 or vrchol je koncový
 then return ohodnocení vrcholu

 α := -∞
 foreach dítě vrcholu do
  α := max{α, -minimax(dítě, hloubka-1)}

 return α
```

Negamax


2
-2
-2
-1
2
5
3
2
4
3
4
1
5
-1 -1 -2 -3 -5 -1 -3 0 0 -2 -1 -2 -8 -3 -4 0 -4 -4 0 -1 -1 -7 -5
-4
3 1 1 5 4 2 6 3 5 4 8 5 7 1 5 3 7 0 8 0 3 2 7 1 1 2 2 7 9 8 4 7 3 3 4 5 0 8 6 4 7 5 4 6 5 2 4 0 8 1 7 1 2 7 8 5

Alfa-beta: negamaxová verze

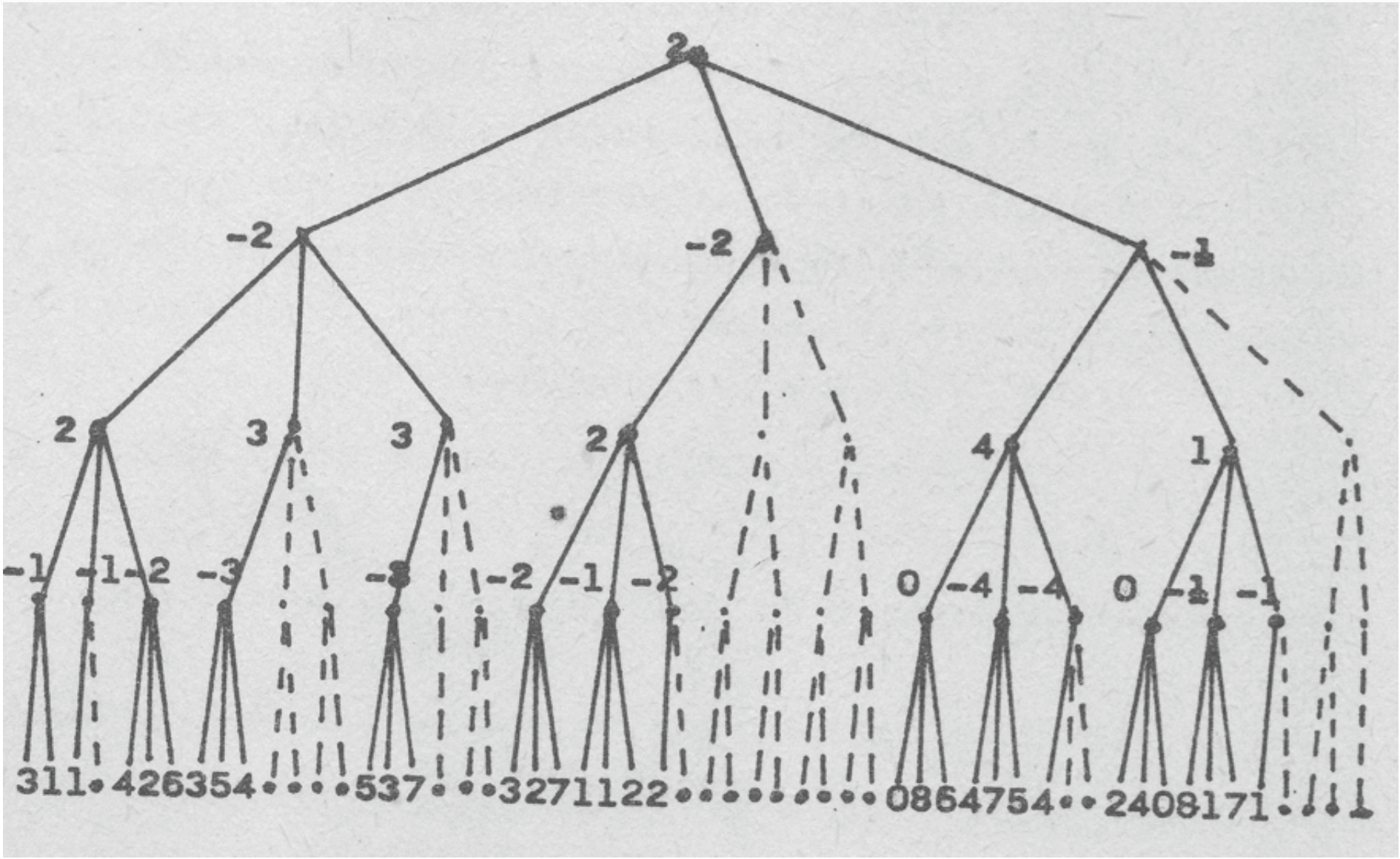
```
function alphabeta(vrchol, hloubka, α, β)
 if  hloubka = 0 or vrchol je koncový
 then return ohodnocení vrcholu

 foreach dítě vrcholu
  α := max{α, -alphabeta(dítě,hloubka-1,-β,-α)}
  if β ≤ α then return β

 return α

/* počáteční volání */
alphabeta(start, hloubka, -∞, +∞)
```

Alfa-beta prořezávání



Účinnost alfa-beta prořezávání

větvící faktor b

- konstantní nebo průměrný

hloubka d

- počet půltahů

počet navštívených koncových vrcholů

- nejhorší případ $O(b^d)$
  - » minimax
- nejlepší případ $O(b^{d/2})$