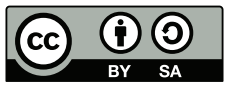
brmiversity: Umělá inteligence a teoretická informatika

Přednáška č. 10

Petr Baudiš ⟨pasky@ucw.cz⟩

brmlab 2011



brmlab
hackerspace prague

Outline

Strojové učení

Učící se agent: Datový vstup a výstup, rozhodovací problém, užitková funkce

Máme trénovací množinu

- Učení s učitelem vs. bez učitele
- Rozpoznávání vs. samoorganizace

Nemáme trénovací množinu (i když. . .)

- Exploration—exploitation dilemma
- Zpětnovazebné učení

brmlab hackerspace prague

Zpětnovazebné učení

Dnes: Nemáme klasickou trénovací množinu,
ale **feedback** — učitelem je prostředí.
Data si často musíme shánět "sami" průběžně.
Jinak řečeno: Agentův **stav** se mění a provádí **akce**.

Reinforcement learning je ztělesněním jádra UI.
Agenta umístíme do prostředí, sám se musí naučit
svoji "přechodovou funkci".

Zpětnovazebné učení

- Strategie agenta (policy) $\pi : S \to A$
  $\pi(s) = a$ zvolí ve stavu s akci a
- Na základě akcí dostáváme **zpětnou vazbu**
  Někdy hned, ale často až za dlouho!
  Jak to modelovat?
- **Pasivní učení:** Během učení používáme fixní π
- **Aktivní učení:** Strategii π průběžně měníme

Bellmannova rovnice

- Každý stav má **ocenění** R (reward)
- K zajímavému stavu se můžeme dostat různými cestami
- **Užitek** je celkové "hodnocení výkonu" agenta, distribuované přes posloupnost stavů
- $U[s_0, s_1, s_2, \ldots] = R(s_0) + \gamma(R(s_1) + \gamma(R(s_2) + \cdots)) \quad \gamma \in [0, 1]$

Bellmannova rovnice

- Každý stav má **ocenění** R (reward)
- K zajímavému stavu se můžeme dostat různými cestami
- **Užitek** je celkové "hodnocení výkonu" agenta, distribuované přes posloupnost stavů
- $U[s_0, s_1, s_2, \ldots] = R(s_0) + \gamma(R(s_1) + \gamma(R(s_2) + \cdots)) \quad \gamma \in [0, 1]$
- Akce nemusí vždy dopadnout stejně — pravděpodobnostní rozdělení; Markovský rozhodovací proces (MDP)!
- $\pi(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s')$
  (vyber akci s nejvyšším *očekávaným* užitkem)

brmlab hackerspace prague

Bellmannova rovnice

- Každý stav má **ocenění** R (reward)
- K zajímavému stavu se můžeme dostat různými cestami
- **Užitek** je celkové "hodnocení výkonu" agenta, distribuované přes posloupnost stavů
- $U[s_0, s_1, s_2, \ldots] = R(s_0) + \gamma(R(s_1) + \gamma(R(s_2) + \cdots)) \quad \gamma \in [0, 1]$
- Akce nemusí vždy dopadnout stejně — pravděpodobnostní rozdělení; Markovský rozhodovací proces (MDP)!
- $\pi(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s')$ (vyber akci s nejvyšším *očekávaným* užitkem)
- Volí-li agent optimální akci, užitek stavu závisí na užitku okolních stavů
- **Bellmanova rovnice:** $U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s')$

brmlab hackerspace prague

Pasivní učení

- Agent neprozkoumává prostředí, vzorkuje s pevnou π
- Chceme *ohodnotit* π, tzn. zjistit očekávaný U^π
- Neznáme $P(s'|s,a)$ ani $R(s)$, potřebujeme zjistit U

Přímý model

- Naivní: Užitek stavu buď průměrné ocenění cesty do cíle
- Ignorujeme interakce mezi stavy a ostří π (Bellmanova rovnice)
- Funguje, ale konverguje pomalu!

Adaptivní dynamické programování

- ADP: Ze vzorků získáme R, postupně dopočítáváme P a U
- P na základě empirických četností
- U **iterativním vzorkováním** pomocí Bellmanovy rovnice (Monte Carlo metoda; TODO)
- Funguje, ale časově náročné kroky a hodně paměti

Temporální diference

- TD-learning: Počítáme přímo U podle "skoku užitku"
- U inicializuji podle R; ve stavu s dostanu reinforcement R, vykonám akci a a ocitnu se ve stavu s'
- TD(0) pravidlo: $U(s) \leftarrow U(s) + \alpha(R(s) + \gamma U(s') - U(s))$
- $\alpha \in (0, 1)$ je parametr učení; díváme se o stav *dopředu* a provádíme *backup*
- Méně paměti, mnoho krátkých kroků, neudržuje konzistenci

Jak to funguje?

- $U[s_0, s_1, s_2, \ldots] = R(s_0) + \gamma(R(s_1) + \gamma(R(s_2) + \cdots))$
  $U[s_0, s_1, s_2, \ldots] = R(s_0) + \gamma U[s_1, s_2, \ldots]$
- Posouváme se k $U(s_0) = \mathbb{E}[R(s_0) + \gamma U(s_1)]$

Aktivní učení

Aktivní učení: Adaptivní strategie.
Měli bychom při učení již co nejlépe fungovat.
Učit se chceme co nejefektivněji.
Problém výběru akcí — zase!

Aktivní učení

Aktivní učení: Adaptivní strategie.
Měli bychom při učení již co nejlépe fungovat.
Učit se chceme co nejefektivněji.
Problém výběru akcí — zase!

Hladový agent se i průběžně drží
$\pi(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$
Je něco lepšího?

Aktivní učení

Aktivní učení: Adaptivní strategie.
Měli bychom při učení již co nejlépe fungovat.
Učit se chceme co nejefektivněji.
Problém výběru akcí — zase!

Hladový agent se i průběžně drží
$\pi(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$
Je něco lepšího?

Exploration: Data o alternativách.
Exploitation: Výběr nejlepší alternativy.
Exploration a exploitation nemohou být v opozici natrvalo.

brmlab hackerspace prague

Exploration-exploitation dilemma

- Problém mnohorukého loupežníka (multi-armed bandit)
  (TODO obrázek)
- Robot v bludišti
- Hraní šachů nebo Go
- Genetický algoritmus

Strategie řešení

- **Semi-uniformní:** ε-greedy
- **Oceňovací:** $U^+ \leftarrow R(s) + \gamma \max_a f(U^+(a), N(s,a))$
  $f(u,n)$ klesá v u, roste v n — třeba prahová
  $u = U^+$ zajišťuje preferenci neprozkoumaných oblastí
  Alternativa: upper confidence bounds
- Optimální strategie vybere nejlepší akci exponenciálně častěji než jakoukoliv jinou

Aktivní TD agent

Druhy aktivních agentů

- **Utility based agent** se učí užitek pro stavy (nutný model)
- **Q-learning based agent** se učí očekávaný užitek pro akce
- **Reflex based agent** se učí přímo $\pi(s)$

- Naivně: Přechodový model ADP agenta (učíme se $P(s'|s,a)$, apriorní znalost), update funkce užitku jako TD(0)
- Q-učení: Bezmodelový — učíme se hodnotu akce $Q(s,a)$
  Místo $U(s)$ se TD(0) aplikuje na $Q(s,a)$
  $U(s) \leftarrow U(s) + \alpha(R(s) + \gamma U(s') - U(s))$

Aktivní TD agent

Druhy aktivních agentů

- **Utility based agent** se učí užitek pro stavy (nutný model)
- **Q-learning based agent** se učí očekávaný užitek pro akce
- **Reflex based agent** se učí přímo $\pi(s)$

- Naivně: Přechodový model ADP agenta (učíme se $P(s'|s,a)$, apriorní znalost), update funkce užitku jako TD(0)
- Q-učení: Bezmodelový — učíme se hodnotu akce $Q(s,a)$
  Místo $U(s)$ se TD(0) aplikuje na $Q(s,a)$
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$

Aktivní TD agent

Druhy aktivních agentů

- **Utility based agent** se učí užitek pro stavy (nutný model)
- **Q-learning based agent** se učí očekávaný užitek pro akce
- **Reflex based agent** se učí přímo $\pi(s)$

- Naivně: Přechodový model ADP agenta (učíme se $P(s'|s,a)$, apriorní znalost), update funkce užitku jako TD(0)
- Q-učení: Bezmodelový — učíme se hodnotu akce $Q(s,a)$
  Místo $U(s)$ se TD(0) aplikuje na $Q(s,a)$
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$
- SARSA: State-Action-Reward-State-Action
  Při update čekáme na další akci
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$
- Jaký je rozdíl pro hladového agenta?

brmlab hackerspace prague

Aktivní TD agent

Druhy aktivních agentů

- **Utility based agent** se učí užitek pro stavy (nutný model)
- **Q-learning based agent** se učí očekávaný užitek pro akce
- **Reflex based agent** se učí přímo $\pi(s)$

- Naivně: Přechodový model ADP agenta (učíme se $P(s'|s,a)$, apriorní znalost), update funkce užitku jako TD(0)
- Q-učení: Bezmodelový — učíme se hodnotu akce $Q(s,a)$
  Místo $U(s)$ se TD(0) aplikuje na $Q(s,a)$
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$
- SARSA: State-Action-Reward-State-Action
  Při update čekáme na další akci
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$
- Jaký je rozdíl pro hladového agenta? Žádný!
- A při průzkumu?

brmlab hackerspace prague

Aktivní TD agent

Druhy aktivních agentů

- **Utility based agent** se učí užitek pro stavy (nutný model)
- **Q-learning based agent** se učí očekávaný užitek pro akce
- **Reflex based agent** se učí přímo $\pi(s)$

- Naivně: Přechodový model ADP agenta (učíme se $P(s'|s,a)$, apriorní znalost), update funkce užitku jako TD(0)
- Q-učení: Bezmodelový — učíme se hodnotu akce $Q(s,a)$
  Místo $U(s)$ se TD(0) aplikuje na $Q(s,a)$
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$
- SARSA: State-Action-Reward-State-Action
  Při update čekáme na další akci
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$
- Jaký je rozdíl pro hladového agenta? Žádný!
- A při průzkumu? SARSA zesiluje efekt strategie

brmlab hackerspace prague

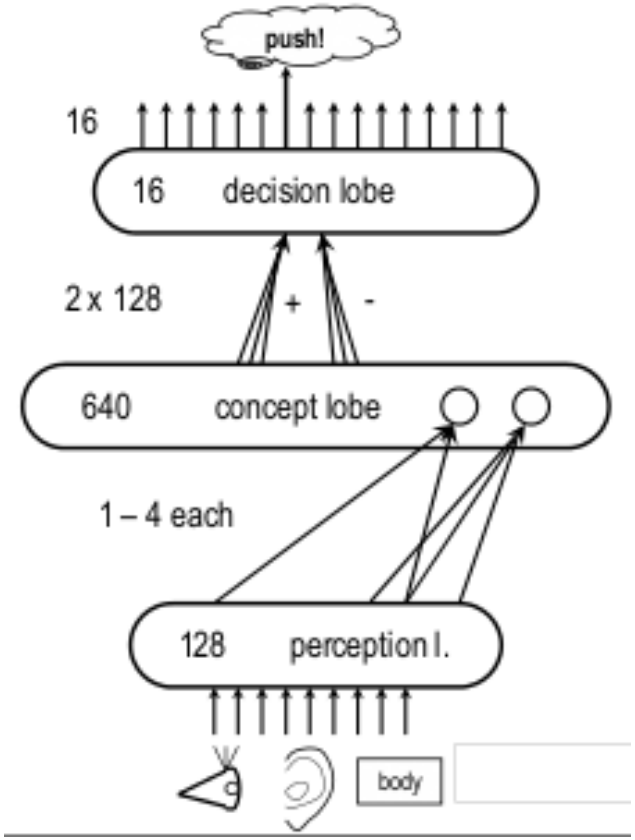
Aktivní TD agent

Druhy aktivních agentů

- **Utility based agent** se učí užitek pro stavy (nutný model)
- **Q-learning based agent** se učí očekávaný užitek pro akce
- **Reflex based agent** se učí přímo $\pi(s)$

- Naivně: Přechodový model ADP agenta (učíme se $P(s'|s,a)$, apriorní znalost), update funkce užitku jako TD(0)
- Q-učení: Bezmodelový — učíme se hodnotu akce $Q(s,a)$
  Místo $U(s)$ se TD(0) aplikuje na $Q(s,a)$
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$
- SARSA: State-Action-Reward-State-Action
  Při update čekáme na další akci
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$
- Jaký je rozdíl pro hladového agenta? Žádný!
- A při průzkumu? SARSA zesiluje efekt strategie (dvojsečné)

Zpětnovazebné učení v neuronových sítích

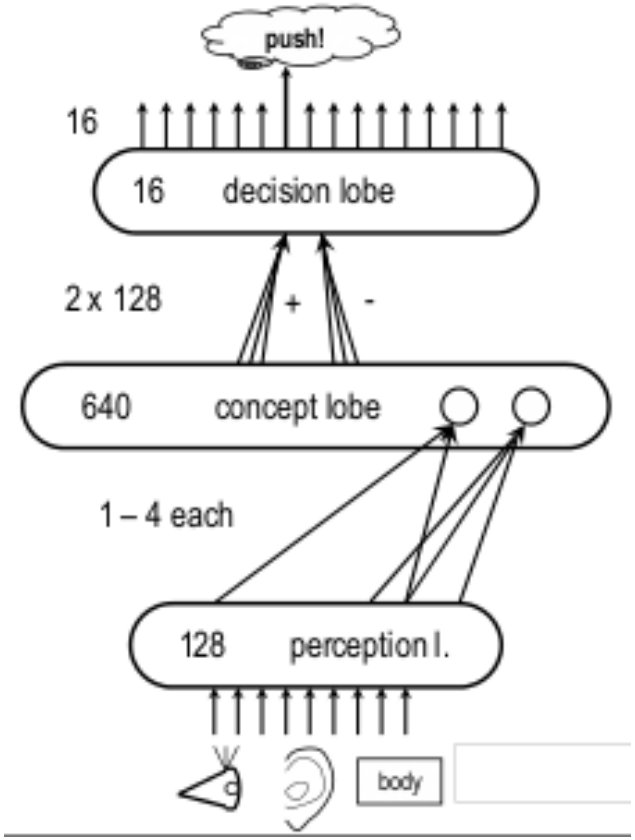


(Brom, 2006)

- Mozek umělé bytosti "Norn"
- Perception: egg, toy, Norn, hunger, sexdrive
- Concept: egg + hunger, Norn + sexdrive, toy + egg + sexdrive
- Decision: eat, mate, run, play
- Pomocný okruh pro Attention selection
- Chceme podporovat užitečné koncepty a vázat je na dobré akce
- Náhodně inicializované váhy a zpětná vazba!
- Reinforcement záleží na změně *pudů* (drives: hlad, sex, . . .)

brmlab hackerspace prague

Zpětnovazebné učení v neuronových sítích



(Brom, 2006)

- Mozek umělé bytosti "Norn"
- Perception: egg, toy, Norn, hunger, sexdrive
- Concept: egg + hunger, Norn + sexdrive, toy + egg + sexdrive
- Decision: eat, mate, run, play
- Suspectibility α a průběžný reinforcement r
- Vykonání akce zvýší α spojům po cestě, to se s časem opět snižuje
- Po snížení váhy k nule synapse migruje
- Robustnost: Short-term weight, long-term weight

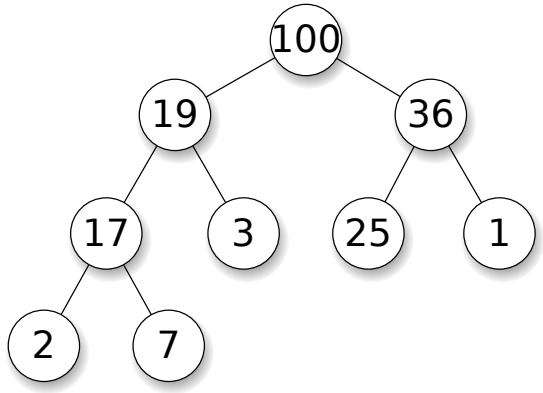
brmlab hackerspace prague

Otázky?

Příště UI: Reprezentace znalostí.
Příště Adaptivní agenti: Etologie a populační dynamika.

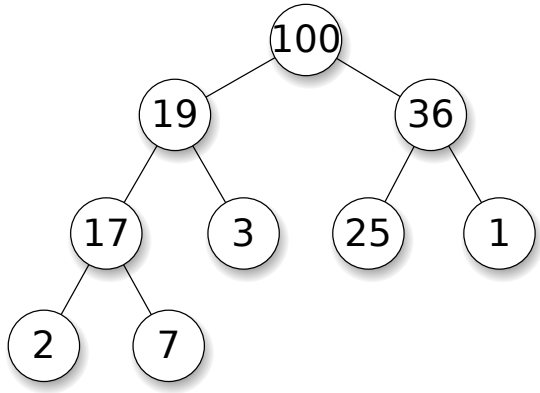
Outline

Halda



- Chceme ukládat data tak, abychom vždy mohli snadno vytáhnout minimum
- Speciálně uspořádaný strom: synové jsou vždy větší než otec
- Tedy v kořeni je nejmenší prvek
- Chceme podporovat zejména operace INSERT, DELETEMIN
- Může se hodit i INCREASE a DECREASE

Regulární halda



- Pěkně "zarovnaný" strom, každý otec kromě těch u dna má dva syny (má hloubku $\log n$)
- Mezi syny není žádné pořadí
- INSERT: Přidáme na "konec" haldy (na dně), posouváme nahoru, dokud je porušena podmínka
- DELETEMIN: Prohodíme kořen a poslední prvek, umažeme poslední prvek, prvek z kořene posouváme dolů
- Složitosti $O(\log n)$
- Heap sort!

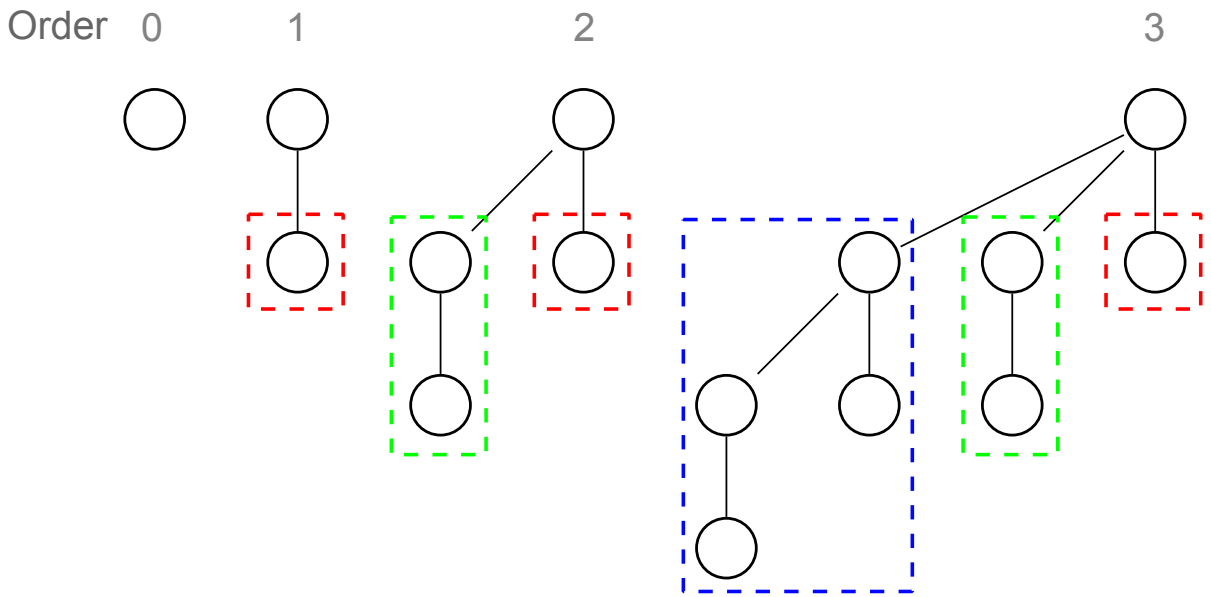
Levicové (leftist) haldy

- Místo probublávání bude základní operace MERGE dvou hald
- INSERT: MERGE s jednoprvkovou haldou
- DELETEMIN: MERGE synů kořene

- Línější strukturální podmínka: *levý syn má vždy delší cestu k nejbližšímu listu*
- Překvapení: délka pravé cesty je nejvýše logaritmická!
- Záruka efektivity: MERGE půjde po pravé cestě

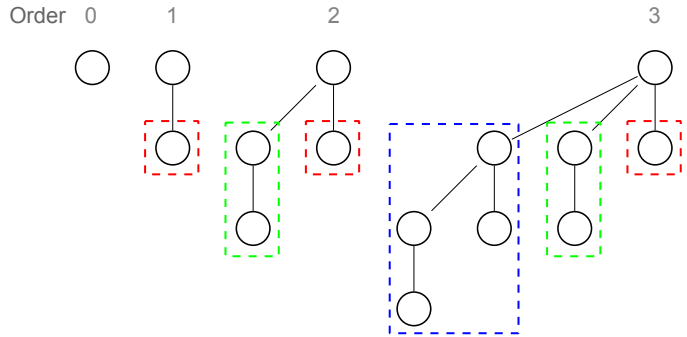
- $MERGE(A, B)$ nastaví $RIGHT(A) = B$ a opraví haldu (haldová a cestová podmínka)
- Pokud již $RIGHT(A)$ předtím existovalo, pustíme pak $MERGE(RIGHT(A), OLDRIGHT(A))$
- DECREASE a INCREASE: Utrhneme podstrom, rekurzivně opravíme rodiče a MERGE

Binomiální stromy



- Binomiální strom: H_0 je jeden uzel, $H_{i+1} = \oplus(H_i, H'_i)$ kde H'_i přidáme jako dalšího syna kořene H_i

Binomiální haldy



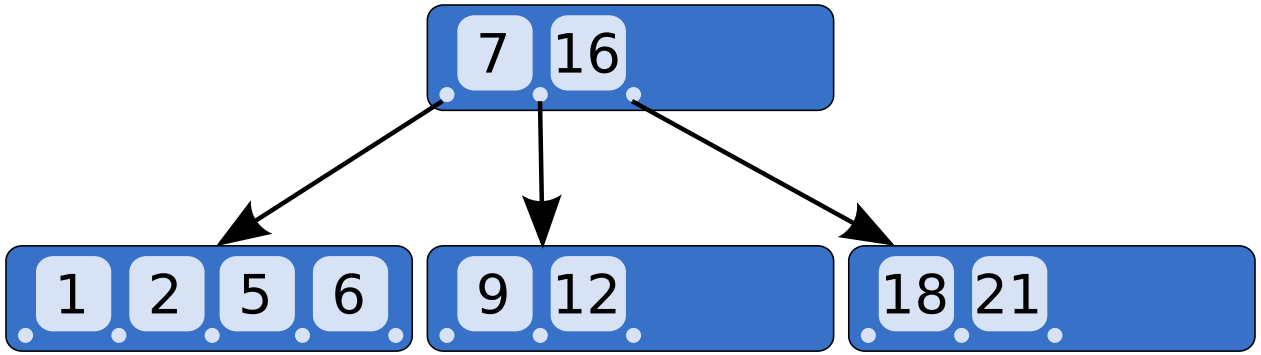
- Binomiální halda: Binomiální les s haldovou podmínkou, maximálně jeden strom každé velikosti
- $MERGE(A, B)$ funguje jako sčítání dvou binárních čísel! Existuje-li H_i jen v jedné, zkopíruje se; jinak vyrobím přenos $H_{i+1}^T = \oplus(H_i^A, H_i^B)$
- MIN je relativně pomalé $O(\log N)$ — musí projít celý les
- Liná binomiální halda: MERGE neslučuje, zato MIN ano; amortizovaně MIN stále $O(\log N)$

Fibonacciho haldy

- Zobecnění líné binomiální haldy — chceme zrychlit INCREASE a DECREASE
- Zrušíme strukturální podmínku na H_i
  i bude popisovat pouze *rank*, totiž počet synů kořenu
  (uvnitř synů to může vypadat, jakkoliv nám dovolí algoritmus)
- Vrchol je *obarvený*, byl-li mu někdy utrhnut syn

- MERGE a MIN jako u líné binomální haldy
- DECREASE a INCREASE: Utrhneme podstrom, rekurzivně opravíme rodiče a MERGE (jako v leftist haldě!)
- Oprava rodičů: Označené také utrhneme
- MIN je stále amortizovaně $O(\log N)$ (dokážeme, že maximální počet synů každého uzlu je nejvýše $\log N$, dá se ukázat přes Fibonacciho posloupnost)

B-stromy

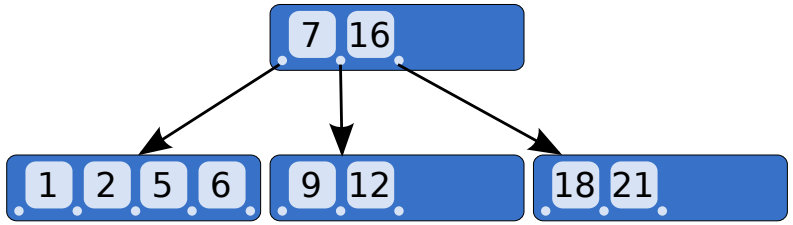


- Vyhledávací strom s jednou hodnotou v uzlu: nevhodný pro pomalý přímý přístup (I/O, cache)
- B-strom: Mnoho definic, nejobecnější je (a, b) strom

- (a, b) strom $(a \leq b)$: každý vnitřní vrchol kromě kořene má alespoň a a nejvíce b synů a všechny cesty od kořene k listu mají stejnou délku
- V praxi navíc: $a \geq 2$, $b \geq 2b - 1$; kořen je buď list, nebo má alespoň 2 a nejvíce b synů

brmlab hackerspace prague

B-stromy



- Populární: 2-3 strom, v praxi ovšem často spíše 128-256 strom apod. (v závislosti na velikosti bloku)
- Díky $b \geq 2b - 1$ mohu vždy sloučit dva a-prázdné uzly nebo rozštěpit jeden plný
- Vyvažování v INSERT a DELETE: Propaguji chybu vzhůru — snažím se vyměňovat prvky se sourozenci, v extrémním případě rozštěpím kořen

- A-sort: Třídění pomocí přidávání do B-stromu, INSERT začíná v listu (výhodné pro částečně předtříděné posloupnosti)

brmlab hackerspace prague

Otázky?

Příště: Datové struktury v externí paměti.

Děkuji vám

pasky@ucw.cz

Příště: Neuronové sítě. Adaptivní agenti. Evoluční algoritmy (koevoluce, otevřená evoluce, teoretická analýza). Složitost (vlastnosti tříd složitosti).

brmlab hackerspace prague