Anotace

- Středník II (alias Checkpoint Bravo)!! 6. 5. 2011
- Poznámka k vnějšímu třídění,
- odstranění rekurze,
- hashování,
- programování her.

Vnější třídění

- týkalo se tříděnína vnější paměti (pomalé a velké),

Vnější třídění

- týkalo se tříděnína vnější paměti (pomalé a velké),
- zejména mergesort, ale také heapsort,

Vnější třídění

- týkalo se tříděnína vnější paměti (pomalé a velké),
- zejména mergesort, ale také heapsort,
- samotné naprogramování použije:

Vnější třídění

- týkalo se tříděnína vnější paměti (pomalé a velké),
- zejména mergesort, ale také heapsort,
- samotné naprogramování použije:
  - strukturu,

Vnější třídění

- týkalo se tříděnína vnější paměti (pomalé a velké),
- zejména mergesort, ale také heapsort,
- samotné naprogramování použije:
  - strukturu,
  - `soubor: file of neco` – binární soubor,

Vnější třídění

- týkalo se tříděnína vnější paměti (pomalé a velké),
- zejména mergesort, ale také heapsort,
- samotné naprogramování použije:
  - strukturu,
  - `soubor: file of neco` – binární soubor,
  - `pod_polstarem: neco;` – prostor na nechtěně načtený údaj

Vnější třídění

- týkalo se tříděnína vnější paměti (pomalé a velké),
- zejména mergesort, ale také heapsort,
- samotné naprogramování použije:
  - strukturu,
  - `soubor: file of neco` – binární soubor,
  - `pod_polstarem: neco;` – prostor na nechtěně načtený údaj
  - obslužné funkce jako `get_next_neco`, `konec_iterace`

Vnější třídění

- týkalo se tříděnína vnější paměti (pomalé a velké),
- zejména mergesort, ale také heapsort,
- samotné naprogramování použije:
  - strukturu,
  - `soubor: file of neco` – binární soubor,
  - `pod_polstarem: neco;` – prostor na nechtěně načtený údaj
  - obslužné funkce jako `get_next_neco`, `konec_iterace`
- Nebo to vše lze udělat objektově a obslužné funkce "přivřít" do té třídy.

Odstranění rekurze obecně

- Rekurze je pěkná věc, ale podle teorie lze každý program napsat v podobě jednoho jediného cyklu (while) bez volání dalších funkcí.

Odstranění rekurze obecně

- Rekurze je pěkná věc, ale podle teorie lze každý program napsat v podobě jednoho jediného cyklu (while) bez volání dalších funkcí.
- Sice takový program není k přečtení, ale jedná se o zajímavý teoretický závěr.

Odstranění rekurze obecně

- Rekurze je pěkná věc, ale podle teorie lze každý program napsat v podobě jednoho jediného cyklu (while) bez volání dalších funkcí.
- Sice takový program není k přečtení, ale jedná se o zajímavý teoretický závěr.
- Jak to udělat?

Odstranění rekurze obecně

- Rekurze je pěkná věc, ale podle teorie lze každý program napsat v podobě jednoho jediného cyklu (while) bez volání dalších funkcí.
- Sice takový program není k přečtení, ale jedná se o zajímavý teoretický závěr.
- Jak to udělat?
- Uděláme totéž, co za nás udělá překladač, když spustíme rekurzi, tedy...

Odstranění rekurze

- Vyrobíme nové lokální proměnné, skočíme na správné místo programu a zapamatujeme si, kam se máme vrátit.

Odstranění rekurze

- Vyrobíme nové lokální proměnné, skočíme na správné místo programu a zapamatujeme si, kam se máme vrátit.
- Ve skutečnosti si jen (na zásobník) uložíme údaje o dosavadních datech, nahradíme daty "zarekurzenými" a údaj odkud jsme se "zavolali" a skočíme na "začátek" funkce.

Odstranění rekurze

- Vyrobíme nové lokální proměnné, skočíme na správné místo programu a zapamatujeme si, kam se máme vrátit.
- Ve skutečnosti si jen (na zásobník) uložíme údaje o dosavadních datech, nahradíme daty "zarekurzenými" a údaj odkud jsme se "zavolali" a skočíme na "začátek" funkce.
- Při "odrekurzení" poznamenáme návratové údaje, vytáhneme ze zásobníku původní obsahy lokálních proměnných a údaj odkud jsme se zavolali (a skočíme tam).

Quicksort s logaritmickou pamětí

- Quicksort může vytvořit velkou a malou část.

Quicksort s logaritmickou pamětí

- Quicksort může vytvořit velkou a malou část.
- Pak potřebujeme lineárně paměti při klasickém rekurzívním řešení.

Quicksort s logaritmickou pamětí

- Quicksort může vytvořit velkou a malou část.
- Pak potřebujeme lineárně paměti při klasickém rekurzívním řešení.
- Při třídění druhé části nepotřebujeme návratové údaje.

Quicksort s logaritmickou pamětí

- Quicksort může vytvořit velkou a malou část.
- Pak potřebujeme lineárně paměti při klasickém rekurzívním řešení.
- Při třídění druhé části nepotřebujeme návratové údaje.
- Proto napřed setřídíme část, která je menší (co do množství dat).

Quicksort s logaritmickou pamětí

- Quicksort může vytvořit velkou a malou část.
- Pak potřebujeme lineárně paměti při klasickém rekurzívním řešení.
- Při třídění druhé části nepotřebujeme návratové údaje.
- Proto napřed setřídíme část, která je menší (co do množství dat).
- A problém můžeme řešit hybridně, tedy na první část se rekurzívně zavolat (i když i v tomto případě umíme rekurzi odbourat, neumíme odbourat její paměťové nároky).

Quicksort s logaritmickou pamětí

- Quicksort může vytvořit velkou a malou část.
- Pak potřebujeme lineárně paměti při klasickém rekurzívním řešení.
- Při třídění druhé části nepotřebujeme návratové údaje.
- Proto napřed setřídíme část, která je menší (co do množství dat).
- A problém můžeme řešit hybridně, tedy na první část se rekurzívně zavolat (i když i v tomto případě umíme rekurzi odbourat, neumíme odbourat její paměťové nároky).
- Tím, že rekurzi (nebo její náhražku) spustíme jen na menší část, pracujeme v i-té úrovni rekurze nejvýše s $\frac{n}{2^i}$ hodnotami.

Quicksort s omezenou rekurzí

Hybridní implementace v pseudokódu

```
procedure quicksort(levy,pravy);
begin
      while (levy<>pravy) do
      begin index_pivota:=rozdel;
            if(index_pivota>(pravy-levy)/2) then
            begin quicksort(index_pivota+1,pravy);
                  pravy:=levy+index_pivota
            end else begin
                  quicksort(levy,index_pivota);
                  levy:=levy+index_pivota+1;
end;  end;  end;
```

Hashování

- Máme-li data, kterými lze indexovat, ale hodnoty by byly příliš velké (například řetězce), má smysl zkusit spočítat nějakou funkci (například posčítat ordinální hodnoty jednotlivých znaků mod 256). Tuto funkci označíme h.

Hashování

- Máme-li data, kterými lze indexovat, ale hodnoty by byly příliš velké (například řetězce), má smysl zkusit spočítat nějakou funkci (například posčítat ordinální hodnoty jednotlivých znaků mod 256). Tuto funkci označíme h.
- Uděláme tak tabulku mnohem menší velikosti, než je universum.

Hashování

- Máme-li data, kterými lze indexovat, ale hodnoty by byly příliš velké (například řetězce), má smysl zkusit spočítat nějakou funkci (například posčítat ordinální hodnoty jednotlivých znaků mod 256). Tuto funkci označíme h.
- Uděláme tak tabulku mnohem menší velikosti, než je universum.
- Tomu říkáme hashování.

Hashování

- Máme-li data, kterými lze indexovat, ale hodnoty by byly příliš velké (například řetězce), má smysl zkusit spočítat nějakou funkci (například posčítat ordinální hodnoty jednotlivých znaků mod 256). Tuto funkci označíme h.
- Uděláme tak tabulku mnohem menší velikosti, než je universum.
- Tomu říkáme hashování.
- Jenže více prvků může kandidovat na totéž místo tabulky, tomu říkáme kolize.

Hashování

řešení kolizí

- Různé metody řešení kolizí:

Hashování

řešení kolizí

- Různé metody řešení kolizí:
- Uděláme spojový seznam,

Hashování

řešení kolizí

- Různé metody řešení kolizí:
- Uděláme spojový seznam,
- srůstající hashování, tedy umísťujeme prvky do přihrádek, kam nepatří,

Hashování

řešení kolizí

- Různé metody řešení kolizí:
- Uděláme spojový seznam,
- srůstající hashování, tedy umísťujeme prvky do přihrádek, kam nepatří,
- z toho plyne problém s implementací `delete` (mazat prakticky nelze, protože bychom mohli roztrhnout řetízek).

Hashování

řešení kolizí

- Různé metody řešení kolizí:
- Uděláme spojový seznam,
- srůstající hashování, tedy umísťujeme prvky do přihrádek, kam nepatří,
- z toho plyne problém s implementací `delete` (mazat prakticky nelze, protože bychom mohli roztrhnout řetízek).
- Na to kam umístit prvek v kolizi existuje mnoho metod. Buďto vybereme další volné místo, nebo obstaráme funkci, která určí další kandidátské místo.

Hashování

řešení kolizí

- Různé metody řešení kolizí:
- Uděláme spojový seznam,
- srůstající hashování, tedy umísťujeme prvky do přihrádek, kam nepatří,
- z toho plyne problém s implementací `delete` (mazat prakticky nelze, protože bychom mohli roztrhnout řetízek).
- Na to kam umístit prvek v kolizi existuje mnoho metod. Buďto vybereme další volné místo, nebo obstaráme funkci, která určí další kandidátské místo.
- Pokud známe množství dat, můžeme se pokusit o perfektní hashování, a to do pole kvadraticky velkého (se středním počtem kolizí menším než 1), nebo dokonce do pole velikosti $4n$, ale pak předem musíme znát data.

Teorie her

- Kombinatorická hra je hrou dvou hráčů. Stav hry je určen pozicí nějakých předmětů. Všechny zúčastněné předměty jsou viditelné. Jde o tzv. hru s úplnou informací.

Teorie her

- Kombinatorická hra je hrou dvou hráčů. Stav hry je určen pozicí nějakých předmětů. Všechny zúčastněné předměty jsou viditelné. Jde o tzv. hru s úplnou informací.
- Příklad: Nimm, Podivná hra, Dáma, Šachy, Halma, Mlýn, Otrávená čokoláda...

Teorie her

- Kombinatorická hra je hrou dvou hráčů. Stav hry je určen pozicí nějakých předmětů. Všechny zúčastněné předměty jsou viditelné. Jde o tzv. hru s úplnou informací.
- Příklad: Nimm, Podivná hra, Dáma, Šachy, Halma, Mlýn, Otrávená čokoláda...
- Kombinatorickými hrami nejsou: Poker, Prší, Mariáš, Black Jack, závody formulí...

Teorie her

- Kombinatorická hra je hrou dvou hráčů. Stav hry je určen pozicí nějakých předmětů. Všechny zúčastněné předměty jsou viditelné. Jde o tzv. hru s úplnou informací.
- Příklad: Nimm, Podivná hra, Dáma, Šachy, Halma, Mlýn, Otrávená čokoláda...
- Kombinatorickými hrami nejsou: Poker, Prší, Mariáš, Black Jack, závody formulí...
- Zaměříme se na hrací část, ne na vstup a výstup.

Teorie her

- Kombinatorická hra je hrou dvou hráčů. Stav hry je určen pozicí nějakých předmětů. Všechny zúčastněné předměty jsou viditelné. Jde o tzv. hru s úplnou informací.
- Příklad: Nimm, Podivná hra, Dáma, Šachy, Halma, Mlýn, Otrávená čokoláda...
- Kombinatorickými hrami nejsou: Poker, Prší, Mariáš, Black Jack, závody formulí...
- Zaměříme se na hrací část, ne na vstup a výstup.
- U her předpokládáme, že hrají rozumně se chovající jedinci (s motivací vyhrát).

Al-Capone a Babinský na sebe práskají

A,B	Babinský neudá	Babinský udá
Al-Capone neudá	0,0	25,0
Al-Capone udá	0,25	10,10

Al-Capone a Babinský na sebe práskají

A,B	Babinský neudá	Babinský udá
Al-Capone neudá	0,0	25,0
Al-Capone udá	0,25	10,10

Optimum je udat!

Shannonova věta

Theorem (Shannon)

Každá kombinatorická hra má pro některého z hráčů neprohrávající strategii.

Důkaz.

Náznak: Buďto platí, že si jeden z hráčů může vynutit zacyklení hry (a tak neprohrát), nebo budeme zkoumat predikáty:
Existuje náš tah, že pro každý tah protihráče existuje náš tah, že pro každý tah protihráče... protihráč prohraje.
Pro každý náš tah existuje tah protihráče, že pro každý náš tah... my prohrajeme.
Formule jsou konečné, počty tahů jsou také konečné, jsou to vzájemně negace a lze je algoritmicky rozhodnout. □

Shannonova věta

Theorem (Shannon)

Každá kombinatorická hra má pro některého z hráčů neprohrávající strategii.

Corollary

Pokud je ve hře remíza vyloučena, jeden z hráčů má vyhrávající strategii.

Graf hry

- Ke hře (případně její instanci) definujeme orientovaný graf:
- Vrcholy: Stavy hry,
- Hrany: Možnosti přechodů mezi jednotlivými stavy.
- Příklad pro Nimm, kdy odebíráme 1 nebo 2 sirky (na tabuli).
- Každému stavu můžeme přiřadit barvu říkající, zda se odtud vyhrává nebo prohrává.

Příklad grafů her

- Na hracím plánu tvaru orientovaného grafu vyrážíme z určeného vrcholu. Taháme jedním padesátníkem.

Příklad grafů her

- Na hracím plánu tvaru orientovaného grafu vyrážíme z určeného vrcholu. Taháme jedním padesátníkem.
- Máme dojet do jednoho z cílových vrcholů. Kdo dojede, vyhraje.

Příklad grafů her

- Na hracím plánu tvaru orientovaného grafu vyrážíme z určeného vrcholu. Taháme jedním padesátníkem.
- Máme dojet do jednoho z cílových vrcholů. Kdo dojede, vyhraje.
- Graf hry máme přímo zadaný a jde jen o to, který vrchol vyhrává.

Příklad grafů her

- Na hracím plánu tvaru orientovaného grafu vyrážíme z určeného vrcholu. Taháme jedním padesátníkem.
- Máme dojet do jednoho z cílových vrcholů. Kdo dojede, vyhraje.
- Graf hry máme přímo zadaný a jde jen o to, který vrchol vyhrává.
- Podivná hra: Graf hry si zakreslíme na šachovnici. Vrcholy jsou políčka, hrany vedou tudy, kudy může figurka.

Příklad grafů her

- Na hracím plánu tvaru orientovaného grafu vyrážíme z určeného vrcholu. Taháme jedním padesátníkem.
- Máme dojet do jednoho z cílových vrcholů. Kdo dojede, vyhraje.
- Graf hry máme přímo zadaný a jde jen o to, který vrchol vyhrává.
- Podivná hra: Graf hry si zakreslíme na šachovnici. Vrcholy jsou políčka, hrany vedou tudy, kudy může figurka.
- Stačí říct, ze kterého vrcholu se vyhrává, prohrává, nebo zda existuje cyklus, po kterém mají oba hráči zájem bloudit (resp. zda si někdo z hráčů může bloudění po této kružnici vynutit).

AND-OR stromy

- Máme-li graf konečné hry, můžeme z něj postavit strom odpovídající hře.

AND-OR stromy

- Máme-li graf konečné hry, můžeme z něj postavit strom odpovídající hře.
- V tomto stromě nás zajímá, zda existuje větev, po které pokud pojedeme, tak vyhrajeme.

AND-OR stromy

- Máme-li graf konečné hry, můžeme z něj postavit strom odpovídající hře.
- V tomto stromě nás zajímá, zda existuje větev, po které pokud pojedeme, tak vyhrajeme.
- Tato větev se pozná tak, že ve všech synech jejího koncového vrcholu existuje vyhrávající cesta, tedy ...

AND-OR stromy

- Máme-li graf konečné hry, můžeme z něj postavit strom odpovídající hře.
- V tomto stromě nás zajímá, zda existuje větev, po které pokud pojedeme, tak vyhrajeme.
- Tato větev se pozná tak, že ve všech synech jejího koncového vrcholu existuje vyhrávající cesta, tedy ...
- buďto vyhrajeme v prvním synu, nebo ve druhém synu, nebo ve třetím...

AND-OR stromy

- Máme-li graf konečné hry, můžeme z něj postavit strom odpovídající hře.
- V tomto stromě nás zajímá, zda existuje větev, po které pokud pojedeme, tak vyhrajeme.
- Tato větev se pozná tak, že ve všech synech jejího koncového vrcholu existuje vyhrávající cesta, tedy ...
- buďto vyhrajeme v prvním synu, nebo ve druhém synu, nebo ve třetím...
- V k-tém synu vyhrajeme, jestliže protihráč prohraje v prvním synu a současně ve druhém synu a současně ve třetím synu... tohoto stavu.

AND-OR stromy

- Máme-li graf konečné hry, můžeme z něj postavit strom odpovídající hře.
- V tomto stromě nás zajímá, zda existuje větev, po které pokud pojedeme, tak vyhrajeme.
- Tato větev se pozná tak, že ve všech synech jejího koncového vrcholu existuje vyhrávající cesta, tedy ...
- buďto vyhrajeme v prvním synu, nebo ve druhém synu, nebo ve třetím...
- V k-tém synu vyhrajeme, jestliže protihráč prohraje v prvním synu a současně ve druhém synu a současně ve třetím synu... tohoto stavu.
- Prohraje tam, jestliže my (pro dotyčný stav) umíme vyhrát buďto v prvním synu, nebo ve druhém, anebo ve třetím...

AND-OR stromy

- Máme-li graf konečné hry, můžeme z něj postavit strom odpovídající hře.
- V tomto stromě nás zajímá, zda existuje větev, po které pokud pojedeme, tak vyhrajeme.
- Tato větev se pozná tak, že ve všech synech jejího koncového vrcholu existuje vyhrávající cesta, tedy ...
- buďto vyhrajeme v prvním synu, nebo ve druhém synu, nebo ve třetím...
- V k-tém synu vyhrajeme, jestliže protihráč prohraje v prvním synu a současně ve druhém synu a současně ve třetím synu... tohoto stavu.
- Prohraje tam, jestliže my (pro dotyčný stav) umíme vyhrát buďto v prvním synu, nebo ve druhém, anebo ve třetím...
- Podmínky AND a OR se stále střídají, proto AND-OR strom.

Hry s ohodnocením

Definition

Hra s ohodnocením je taková hra, kdy cílové stavy jsou ohodnoceny číslem. Jeden hráč se pokouší výsledek maximalizovat, druhý minimalizovat.

Definition

Hra s nulovým součtem je taková hra, ve které zisk jednoho hráče je roven ztrátě druhého hráče.

Algoritmus MINIMAX

- Algoritmus lze použít pro hry s ohodnocením.

Algoritmus MINIMAX

- Algoritmus lze použít pro hry s ohodnocením.
- Postavíme strom hry.

Algoritmus MINIMAX

- Algoritmus lze použít pro hry s ohodnocením.
- Postavíme strom hry.
- Začneme od koncových vrcholů.

Algoritmus MINIMAX

- Algoritmus lze použít pro hry s ohodnocením.
- Postavíme strom hry.
- Začneme od koncových vrcholů.
- Hodnota podstromu je minimum resp. maximum z hodnot synů (podle toho, zda hraje minimalizující nebo maximalizující hráč).

Algoritmus NEGAMAX

- Varianta algoritmu MINIMAX pro hry s nulovým součtem:

$$\max_{i \in S} -f(i) = \min_{i \in S} f(i).$$

Algoritmus NEGAMAX

- Varianta algoritmu MINIMAX pro hry s nulovým součtem:

$$\max_{i \in S} -f(i) = \min_{i \in S} f(i).$$

- Jde vlastně o totéž, je ovšem jednodušší na naprogramování.

Heuristiky

- Obvykle se pokoušíme neprohledávat zbytečně všechno, pokud najdeme jednu možnost výhry, nemusíme hledat i všechny ostatní.

Heuristiky

- Obvykle se pokoušíme neprohledávat zbytečně všechno, pokud najdeme jednu možnost výhry, nemusíme hledat i všechny ostatní.
- α-β-prořezávání: Umíme-li v nějakém synu S vyhrát aspoň α a najdeme v některém následujícím synu T, že protihráč nás umí dotlačit na méně, nemá smysl vrchol T dále zkoumat.

Heuristiky

- Obvykle se pokoušíme neprohledávat zbytečně všechno, pokud najdeme jednu možnost výhry, nemusíme hledat i všechny ostatní.
- α-β-prořezávání: Umíme-li v nějakém synu S vyhrát aspoň α a najdeme v některém následujícím synu T, že protihráč nás umí dotlačit na méně, nemá smysl vrchol T dále zkoumat.
- Pro opačný případ se používá β: Pokud nás nepřítel umí zatlačit na nejvýš β a v jiném synu mu utečeme přes, nemá smysl ten druhý syn zkoumat dále.

Reálné hry

Šachy, dáma, halma, mlýn...

- Můžeme postavit strom hry, ten je ale příliš velký.

Reálné hry

Šachy, dáma, halma, mlýn...

- Můžeme postavit strom hry, ten je ale příliš velký.
- Nasadíme proto všelijaké heuristiky. Ty dosavadní ale stejně daleko nevedou.

Reálné hry

Šachy, dáma, halma, mlýn...

- Můžeme postavit strom hry, ten je ale příliš velký.
- Nasadíme proto všelijaké heuristiky. Ty dosavadní ale stejně daleko nevedou.
- Statická ohodnocovací funkce: Funkce, která se pokouší odhadnout, zda je pozice perspektivní (dobrá) nebo ne.

Reálné hry

Šachy, dáma, halma, mlýn...

- Můžeme postavit strom hry, ten je ale příliš velký.
- Nasadíme proto všelijaké heuristiky. Ty dosavadní ale stejně daleko nevedou.
- Statická ohodnocovací funkce: Funkce, která se pokouší odhadnout, zda je pozice perspektivní (dobrá) nebo ne.
- Prohledáváme strom hry jen po nějakou dobu (do nějaké hloubky). Na nalezené (neterminální) pozice nasadíme statickou ohodnocovací funkci.

Reálné hry

Šachy, dáma, halma, mlýn...

- Můžeme postavit strom hry, ten je ale příliš velký.
- Nasadíme proto všelijaké heuristiky. Ty dosavadní ale stejně daleko nevedou.
- Statická ohodnocovací funkce: Funkce, která se pokouší odhadnout, zda je pozice perspektivní (dobrá) nebo ne.
- Prohledáváme strom hry jen po nějakou dobu (do nějaké hloubky). Na nalezené (neterminální) pozice nasadíme statickou ohodnocovací funkci.
- U šachů například můžeme počítat materiální převahu a body za ohrožené figurky (Colossus na Atari kolem roku 1985).

Konec

...děkuji za pozornost...

Otázky?