Anotace

- Rozděl a panuj (poznámky),
- předání funkce v parametru.

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.
- Rozděl a panuj rozdělí instanci na přesně definované menší instance,

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.
- Rozděl a panuj rozdělí instanci na přesně definované menší instance,
- vyřeší každou zvlášť a z jejich řešení zjistí řešení původní instance.

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.
- Rozděl a panuj rozdělí instanci na přesně definované menší instance,
- vyřeší každou zvlášť a z jejich řešení zjistí řešení původní instance.
- Příklady: Řízení podniku (ředitel leteckých závodů zpravidla neřeší, jak přinýtovat zadní část křídla),

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.
- Rozděl a panuj rozdělí instanci na přesně definované menší instance,
- vyřeší každou zvlášť a z jejich řešení zjistí řešení původní instance.
- Příklady: Řízení podniku (ředitel leteckých závodů zpravidla neřeší, jak přinýtovat zadní část křídla),
- Příklady (algoritmické): Quicksort, binární vyhledávání.

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.
- Příklad: $T(n) = T(\frac{n}{2}) + k$ (binární vyhledání)

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.
- Příklad: $T(n) = T(\frac{n}{2}) + k$ (binární vyhledání)
- Co když v Quicksortu jako pivot vybereme medián?

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.
- Příklad: $T(n) = T(\frac{n}{2}) + k$ (binární vyhledání)
- Co když v Quicksortu jako pivot vybereme medián?
- V tom případě máme rekurenci: $T(n) = 2T(\frac{n}{2}) + kn$.

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.
- Příklad: $T(n) = T(\frac{n}{2}) + k$ (binární vyhledání)
- Co když v Quicksortu jako pivot vybereme medián?
- V tom případě máme rekurenci: $T(n) = 2T(\frac{n}{2}) + kn$.
- Nelze použít jinou metodu?

Master theorem (symetrická verze)

Theorem

Je-li $T(1) = c$ *a* $T(n) = a \cdot T(\frac{n}{b}) + \Theta(n^d)$, *kde* $a \geq 1$, $b > 1$, $d \geq 0$ *a* a, b *přirozená čísla, potom platí:*

- $T(n) \in \Theta(n^d)$, *pokud* $a < b^d$,
- $T(n) \in \Theta(n^d \log n)$, *pokud* $a = b^d$
- *a* $T(n) \in \Theta(n^{\log_b a})$, *pokud* $a > b^d$.

Master theorem – myšlenka důkazu:

- Výpočet si představíme "po hladinách" podle hloubky rekurze.
- Zjistíme složitost každé hladiny zvlášť,
- zjistíme, která bude trvat nejdéle,
- posčítáme.
- Hladin bude $\log_b n$, protože čekáme, až z n zbyde v argumentu jen konstanta.
- Jaká je složitost hladiny k?: $a^k \cdot (\frac{n}{b^k})^d$.

- Která hladina bude trvat nejdéle?
- První (pokud $\left(\frac{a}{b^d}\right)^k < 1$), což je první případ.
- Nebo poslední (pokud $\left(\frac{a}{b^d}\right)^k > 1$), což je třetí případ.
- Nebo všechny stejně (pokud $\left(\frac{a}{b^d}\right)^k = 1$), což je druhý případ.
- Složitost je tedy součtem geometrické posloupnosti!
- Kvocient této řady je menší než jedna, větší než jedna, nebo právě jedna.
- Je-li kvocient různý od jedné, odhadneme součet největším z členů, zbytek je konstanta.
- Je-li kvocient jedna, součet je: hloubka krát složitost libovolné hladiny. A hloubka je $\log n$.

Analýza master-theorem:

- Binární vyhledání: $T(n) = T(\frac{n}{2}) + \Theta(n^0)$
- druhý případ $a = 1, b = 2, d = 0, a = b^d$, tedy složitost binárního vyhledání je $\Theta(\log n)$.
- Quicksort obecně: $T(n) = T(n_1) + T(n - n_1) + \Theta(n^1)$. Master theorem mlčí!
- Quicksort vybereme-li za pivot medián a umíme-li medián vybrat s lineární složitostí: $T(n) = 2T(\frac{n}{2}) + \Theta(n^1)$. Jde opět o druhý případ $a = 2, b = 2, d = 1, a = b^d$, získáváme opět složitost $\Theta(n \log n)$.

Další příklad – násobení matic:

- $A = \begin{pmatrix} A_{11}, A_{12} \\ A_{21}, A_{22} \end{pmatrix}$, $B = \begin{pmatrix} B_{11}, B_{12} \\ B_{21}, B_{22} \end{pmatrix}$, $C = \begin{pmatrix} C_{11}, C_{12} \\ C_{21}, C_{22} \end{pmatrix}$.

Další příklad – násobení matic:

- $A = \begin{pmatrix} A_{11}, A_{12} \\ A_{21}, A_{22} \end{pmatrix}$, $B = \begin{pmatrix} B_{11}, B_{12} \\ B_{21}, B_{22} \end{pmatrix}$, $C = \begin{pmatrix} C_{11}, C_{12} \\ C_{21}, C_{22} \end{pmatrix}$.
- cíl:

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}.$$

Další příklad – násobení matic:

- $A = \begin{pmatrix} A_{11}, A_{12} \\ A_{21}, A_{22} \end{pmatrix}$, $B = \begin{pmatrix} B_{11}, B_{12} \\ B_{21}, B_{22} \end{pmatrix}$, $C = \begin{pmatrix} C_{11}, C_{12} \\ C_{21}, C_{22} \end{pmatrix}$.
- cíl:
  $C_{11} = A_{11}B_{11} + A_{12}B_{21}$
  $C_{12} = A_{11}B_{12} + A_{12}B_{22}$
  $C_{21} = A_{21}B_{11} + A_{22}B_{21}$
  $C_{22} = A_{21}B_{12} + A_{22}B_{22}$.
- Trik: Definujeme matice $M_1, \ldots M_7$, aby $C_{11} \ldots C_{22}$ šlo vyjádřit jako jejich součet resp. rozdíl.

Další příklad – násobení matic:

Strassenuv algoritmus

- $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$,
  $M_2 = (A_{21} + A_{22})B_{11}$,
  $M_3 = A_{11}(B_{12} - B_{22})$,
  $M_4 = A_{22}(B_{21} - B_{11})$,
  $M_5 = (A_{11} + A_{12})B_{22}$,
  $M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$,
  $M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$.

Další příklad – násobení matic:

Strassenuv algoritmus

- $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$,
  $M_2 = (A_{21} + A_{22})B_{11}$,
  $M_3 = A_{11}(B_{12} - B_{22})$,
  $M_4 = A_{22}(B_{21} - B_{11})$,
  $M_5 = (A_{11} + A_{12})B_{22}$,
  $M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$,
  $M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$.
- $C_{11} = M_1 + M_4 - M_5 + M_7$,
  $C_{12} = M_3 + M_5$,
  $C_{21} = M_2 + M_4$,
  $C_{22} = M_1 - M_2 + M_3 + M_6$.

Další příklad – násobení matic:

- Naivní algoritmus: $T(n) = \Theta(n^3)$.
- Strassenův algoritmus použije jen 7 násobení matic o polovinu menších,
- režie každé iterace je kvadratická (vůči šířce matice),
- rekurence tedy vychází: $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$.
- Jde o 3. případ ($a = 7, b = 2, d = 2, a > b^d$) a tedy složitost Strassenova algoritmu je: $\Theta(n^{\log_2 7})$.
- Dokazujte toto indukcí...

Medián v lineárním čase

- Quicksort potřebuje rozdělit hodnoty na menší a větší

Medián v lineárním čase

- Quicksort potřebuje rozdělit hodnoty na menší a větší
- ale jak zvolit pivot?

Medián v lineárním čase

- Quicksort potřebuje rozdělit hodnoty na menší a větší
- ale jak zvolit pivot?
- Deterministická volba (první prvek posloupnosti) vede k ošklivé složitosti v nejhorším případě,

Medián v lineárním čase

- Quicksort potřebuje rozdělit hodnoty na menší a větší
- ale jak zvolit pivot?
- Deterministická volba (první prvek posloupnosti) vede k ošklivé složitosti v nejhorším případě,
- ideální by bylo volit medián,

Medián v lineárním čase

- Quicksort potřebuje rozdělit hodnoty na menší a větší
- ale jak zvolit pivot?
- Deterministická volba (první prvek posloupnosti) vede k ošklivé složitosti v nejhorším případě,
- ideální by bylo volit medián,
- ale je nutné získat ho v lineárním čase.

Medián v lineárním čase

- Rozděl vstup na pětice,
- v každé pětici najdi medián,
- najdi medián mediánů (tedy medián mezi mediány pětic),
- rozděl vstup na menší a větší,
- zjisti, zda se k-tý nejmenší nachází mezi většími nebo menšími
- pokračuj s hledáním příslušné hromádce.

Medián lineárně detaily

- Jak najít mediány pětic?

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?
- Rekurzívně (zavolej se na pole mediánů pětic).

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?
- Rekurzívně (zavolej se na pole mediánů pětic).
- Jak "pokračovat na příslušné hromádce?"

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?
- Rekurzívně (zavolej se na pole mediánů pětic).
- Jak "pokračovat na příslušné hromádce?"
- Rekurzívně (zavolej se buďto na menší hromádku a hledej k-tý nejmenší, nebo máme-li hledat v hromádce větších hodnot budiž l počet prvků na menší hromádce a hledej v hromádce větších $k - l$-tý nejmenší.

Proč je algoritmus lineární?

Protože:

- mediánů pětic je přibližně pětina délky vstupu,
- hromádka menších čísel stejně jako hromádka větších čísel bude mít velikost aspoň $3/10$,
- tudíž každá z hromádek bude mít též velikost nejvýš $7/10$.
- Zbytek je jen indukce.

Indukce:

Složitost algoritmu vede k rekurenci:

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + kn.$$

Ukážeme, že existuje l takové, že $T(n) \leq ln$:

$$T(n) \leq \frac{ln}{5} + \frac{7ln}{10} + kn = kn + \frac{9}{10}ln$$

a tedy stačí volit $l \geq 10k$.

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`
- `var funkcni_promenna:jmeno_typu;`

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`
- `var funkcni_promenna:jmeno_typu;`
- `function porovnej(argumenty:typy);...`

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`
- `var funkcni_promenna:jmeno_typu;`
- `function porovnej(argumenty:typy);...`
- `funkcni_promenna:=porovnej;`

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`
- `var funkcni_promenna:jmeno_typu;`
- `function porovnej(argumenty:typy);...`
- `funkcni_promenna:=porovnej;`
- `funkcni_promenna(parametry);`

Příklad

```
type porovfce=function (a,b:integer):boolean;
var p:porovfce;
function cmp(a,b:integer):boolean;
begin
      cmp:=(a<b);
end;
procedure por(c:porovfce;a,b:integer);
begin
      if(c(a,b)) then writeln('Prvni vetsi!');
end;
begin
      p:=cmp;
      if(p(10,20)) then writeln('Prvni vetsi');
      por(cmp,10,20);
end
```

Konec

...děkuji za pozornost...

Otázky?