Anotace

- Základní třídící algoritmy
- Quicksort
- Jednotky (oddělený překlad)

Problém třídění – jednoduché třídící algoritmy

- Bublinkové třídění (BubbleSort),
- zatřiďování alias třídění přímým vkládáním (InsertSort),
- třídění vyběrem (SelectSort),
- QuickSort.

Bublinkové třídění

- Geometrická interpretace:
  Bublinky v kapalině jdou zpravidla vzhůru
- Myšlenka: Porovnáváme po sobě jdoucí čísla (ve smyslu sousední v zadaném poli) od prvního k poslednímu, jsou-li v nesprávném pořadí, prohodíme je.
- Prvky "probublávají""správným"směrem.
- Opakujeme bublání, dokud se prohazuje.

Bubblesort v pseudokódu

```
prohazovalose:=true;
while prohazovalose:=true do
begin
    for i:=1 to pocet - 1 do
    begin
        prohazovalose:=false;
        if pole[i]>pole[i+1] then
        begin prohod(pole[i],pole[i+1]);
              prohazovalose:=true;
        end;
    end;
end;
```

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V *i*-tém kroku dojede *i*-tý největší na své místo!

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.
- Složitost `BubbleSortu` je tedy $O(n^2)$.

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.
- Složitost `BubbleSortu` je tedy $O(n^2)$.
- Implementaci, kdy se střídavě bublá z jedné strany na druhou a z druhé na první se říká `ShakeSort` a funguje pro něj stejný odhad složitosti.

Třídění přímým výběrem a zatřiďováním

Přímý výběr:

- Opakuj, dokud není tříděné pole prázdné:
- Najdi v poli minimum a přesuň ho na konec setříděného pole.

Zatřiďování:

- Opakuj, dokud není tříděné pole prázdné:
- Vyjmi z něj první prvek a zatřiď do cílového pole, tedy: najdi pozici, kam prvek patří, přidej ho tam a zbytek setříděného pole posuň (o jedna dál).

Analýza složitosti: $n$krát opakujeme proces, který trvá nejvýše n kroků, tedy také $O(n^2)$.

Quicksort – třídění za pomoci rekurze – idea:

- Pokud třídíme jedno číslo, nic nedělej (posloupnost je setříděna),
  tedy vrať posloupnost tak, jak jsme ji dostali.
- V poli POLE vyber jeden prvek (dále pivot).
- Rozděl POLE na pole A obsahující prvky menší než pivot
- a na pole B obsahující prvky větší nebo rovné pivotu.
- Pomocí sebe sama setřiď pole A,
- pomocí sebe sama setřiď pole B,
- Vypiš: pole A, pivot, pole B.

Quicksort

Implementace bude na webu.

Quicksort – analýza složitosti

- V nejhorším případě: $\Theta(n^2)$.
- V nejlepším případě: $\Theta(n \log n)$.
- V průměrném případě: $\Theta(n \log n)$.

Quicksort – varianty a složitosti

- Randomizovaný quicksort: Pivotem volíme náhodný prvek,

Quicksort – varianty a složitosti

- Randomizovaný quicksort: Pivotem volíme náhodný prvek,
- složitost v průměrném případě $\Theta(n \log n)$.

Quicksort – varianty a složitosti

- Randomizovaný quicksort: Pivotem volíme náhodný prvek,
- složitost v průměrném případě $\Theta(n \log n)$.
- Analýza se opírá o netriviální (i když známá) tvrzení teorie pravděpodobnosti.

Quicksort – přirozené otázky?

- Na čem závisí složitost?

Quicksort – přirozené otázky?

- Na čem závisí složitost?
- Jak volbu pivota ovlivnit?

Quicksort – přirozené otázky?

- Na čem závisí složitost?
- Jak volbu pivota ovlivnit?
- Lze volit pivot tak, aby quicksort provedl vždy jen $O(\log n)$ "fází"?

Quicksort – přirozené otázky?

- Na čem závisí složitost?
- Jak volbu pivota ovlivnit?
- Lze volit pivot tak, aby quicksort provedl vždy jen $O(\log n)$ "fází"?
- Mediánem nazveme takový prvek, že alespoň polovina prvků je alespoň taková jako on a alespoň polovina nejvýše taková.

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-theorememem:

- 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:

- 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

- Byl algoritmus Quicksort.
- Problémem bylo, jak volit pivot.
- Volíme-li špatně, děláme mnoho iterací rekurze.
- Hledáme-li pivot dlouho, nepříjemně to trvá.
- Jak hledat medián v lineárním čase?
- Ve skutečnosti nebudeme hledat medián, ale k-tý nejmenší prvek.

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 $l \geq 10k$.

Odstrašující příklady `for`-cyklus

- Co se stane, když změníme obsah cyklící proměnné ve `for`-cyklu?
- ```
  for i:=1 to 10 do
  begin
        writeln(i);
        if (i= 3) then
              i:=1000;
  end;
  ```
- Co řekne FPC? A co BPC? A co GPC? A co jeho option `--borland-pascal`?
  ```

Parametry programu

- copy c:\io.sys c:\windows\kram.krm
- copy con program.exe
- dir *.pas
- jedná se o volání programů s parametry.
- Parametry jsou elegantní způsob načtení dat.

Jednotky – oddělený překlad

- Občas máme obecně využitelné funkce, které chceme používat v různých programech současně.
- Buďto je můžeme okopírovat mezi zdrojovými soubory (špatný nápad)
- nebo je dáme do zvláštního souboru, který budeme kompilovat zvlášť
- Tomu (druhému) řešení říkáme jednotky.

Jednotky – výhody a nevýhody

- Kód se rozlézá ve více souborech,
- jeden kód nemusíme psát vícekrát, chceme-li jej sdílet ve více programech.

Jednotky – syntax a sémantika

- Místo slovem `program` zahájíme soubor slovem `unit`,
- opět následuje jméno jednotky a tentokrát už se kontroluje!
- Jednotka má `interface` (popis, co je vidět zvenku)
- a část implementační (zahájenou slovem `implementation`).

Jednotky – interface

- Interface popisuje, co z jednotky je "vidět".
- V části interface jsou:
- definice proměnných (viditelných zvenku),
- prototypy funkcí a procedur (rovněž viditelných zvenku),
- prototypem rozumíme hlavičku funkce ("první řádek").

Jednotky – impelementace

- To, co nemá být vidět zvenku, tedy:
- Samotné definice funkcí,
- definice proměnných, které nemají být vidět zvenku,
- definice pomocných funkcí (které nemají být vidět zvenku).
- Jednotku ukončíme `end.` (!)

Jednotky – příklad

```
unit trideni;
interface
      type po=array[0..9] of integer;
      procedure bublej(var pole:array of integer);
      procedure vytrid(var a:po);
      procedure zatridovani(var a:po);
      procedure quicksort(var pole:array of
integer;kolik:integer);
      procedure vypis(a:array of integer);
```

Jednotky – příklad (pokr.)

```
...
implementation
      var zatrideno:integer;
      procedure bublej(var pole:array of integer);
      ...
      function najdi_min(var a:po):integer;
      {Pozor, funkci najdi_min neni zvenku videt!}
      ...
      procedure vytrid(var a:po):integer;
      ...
      ...
end.
```

Použití jednotky

- Chceme-li jednotku použít, oznámíme to pomocí klíčového slova `uses`, před názvem dotyčné jednotky:
- Příklad: `uses trideni;`

Použití jednotky – příklad

```
program trid;
uses trideni;
var p:array [0..9] of integer;
i:integer;
begin
for i:=0 to 9 do
read(p[i]);
quicksort(p,10);
vypis(p);
end.
```

Standardní jednotky

Turbo Pascal je vybaven několika standardními jednotkami:

- crt,
- dos,
- graph,
- printer,
- ...

Jednotky se mohou lišit pro různé překladače!

Jednotka `crt`

- Jednotka pro práci s obrazovkou a klávesnicí (barvy, zvuky)
- Proměnné: `LastMode` (údaj o posledním textovém módu před přechodem do grafiky),
- `TextAttr` (aktuální atribut zobrazení ovládaný pomocí `TextBackground` a `TextColor`),
- Procedura `TextBackground` nastaví barvu pozadí, proc. `TextColor` nastaví barvu popředí,
- funkce `keypressed` (vrací `boolean` a říká, zda byla stisknuta klávesa), `clrscr` (smaže obrazovku).

Jednotky dos, graph a printer

- Jednotka `dos` slouží především k práci se soubory, adresáři, disky...
- Jednotka `graph` slouží k práci s grafikou (`InitGraph`, `CloseGraph`, `GraphResult`, `SetColor`, `GetColor`...).
- Jednotka `Printer` slouží k tisku.
- Všechny tyto jednotky obsahují mnoho funkcí, které si nastudujete v případě potřeby.

Podivný příklad:

Ukazoval jsem několikrát program obsahující:

```
program nic;
uses crt;
...
begin
... repeat until keypressed;
end.
```

Co to bylo?

Podivný příklad:

Ukazoval jsem několikrát program obsahující:

```
program nic;
uses crt;
...
begin
... repeat until keypressed;
end.
```

Co to bylo?

Použití jednotky `crt`, resp. její funkce `keypressed`.

Direktiva `forward`

- Občas nám mezi dvěma funkcemi vznikne vzájemná závislost:
- Jedna funkce volá druhou a druhá funkce volá funkci první.
- Překladač Pascalu se začne bouřit při prvním volání druhé funkce, protože ta ještě není definována!
- Proto před definicí první funkce musíme oznámit, že bude existovat druhá funkce.
- Napíšeme její prototyp a místo definice těla dáme klíčové slovo `forward`:
- `procedure dva(a:integer);forward;`

Forward příklad:

```
program qq;
procedure dva(a:integer);forward;
procedure jedna(a:integer);
begin
      dva(a);
end;
procedure dva(a:integer);
begin
      jedna(a);
end;
begin
      jedna(1);
      {Ponechme stranou, že program nic nedela!}
end.
```