UNICORN COLLEGE

UCL Koordinátor běhů předmětů (Křížová Martina)
Základy návrhu a optimalizace algoritmů (verze 2014) Léto 2016 ▸ 04 Domácí úkoly / práce

UCL Exercise Assignment
UCL-BT:ALG14A16S.CZ/XA02_PT

ALG Cvičení - Hrabičkové třídění

Zadat úkol skupině č. 1 | Zadat úkol skupině č. 2

Kontext

Hovořili jsme o několika třídicích algoritmech využívajících nejrůznějších principů pro seřazení dat. Cílem tohoto cvičení je seznámit se s dalším třídícím algoritmem a v praxi ověřit některé teoretické závěry, ke kterým jsme došli.

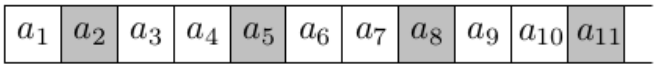
Pro lepší zobrazení matematických vzorců můžete zvolit Zadání v PDF (v příloze) .

Hrabičkové třídění

Následující třídící algoritmus, nazvěme ho *hrabičkové třídění*, si bere jako inspiraci jednotlivé průchody insertion sortu -- tedy zpětné průchody, ve kterých se vezme další nový prvek z pole, například a_i, a zatřídí se do již setříděné úvodní části posloupnosti $a_1 \leq a_2 \leq \ldots \leq a_{i-1}$ (čímž se setříděná posloupnost rozšíří na i prvků). Říkejme tomuto průchodu *včlenění prvku* a_i. Insertion sort tedy postupně *včleňuje* prvek za prvkem, tedy $a_1, a_2, \ldots, a_n$.

Hrabičkové třídění pracuje ve fázích. Každá fáze začíná volbou přirozeného čísla $k \in N$ (jeho přesnou volbu vysvětlíme později). V každé takové fázi pak hrabičkové třídění prochází od začátku pole a na každý prvek provádí postupně operaci *včlenění* do již prošlé úvodní části posloupnosti. Důležitý rozdíl je ale v tom, že při těchto *včleněních* se uvažuje pouze jedna -tina prvků, a to takové prvky, které jsou od zatřiďovaného prvku **vzdálené násobky čísla** k -- můžeme si to představit tak, že se uvažují jenom prvky, na které ukazují pomyslné *hrábě*. Tedy pro prvek a_i to jsou prvky $a_{i-k}, a_{i-2k}, a_{i-3k}, \ldots, a_{i-\lfloor \frac{i-1}{k} \rfloor k}$, ostatní prvky pole se při těchto jednotlivých operacích *včlenění* neuvažují a úplně se přeskakují. Po jedné takové fázi říkáme, že je pole k-*shrabané*.

Důležité je si uvědomit, že prvky se při jedné této jedné fázi postupně *včleňují* všechny, jen každé *včlenění* je k-násobně rychlejší než jedno *včelenění* u klasického insertion sortu. Pomyslné hrábě se nám po poli postupně posouvají zleva doprava a po každých k krocích hroty hrábí znovu ukazují na prvky, na které ukazovaly před k kroky, jen je hrotů nad polem o jeden víc.



$$a_2 \leq a_5 \leq a_8 \leq a_{11}$$

Obrázek: Včlenění prvku a_{11} se pro $k = 3$ týká pouze prvků a_2, a_5, a_8, a_{11}

Můžeme si všimnout, že úplně všechny prvky včleňovat není nutné: k prvních prvků můžeme přeskočit, protože každá z posloupností o délce 1, které jsou tvořeny prvky $a_1, \ldots, a_k$, je již setříděná.

Kouzlo hrabičkového třídění je v tom, že pokud pole budeme ze začátku *shrabávat* s vysokým číslem k, algoritmus bude rychle přesouvat malé prvky na velké vzdálenosti z konce pole k jeho začátku a opačně, čímž zbyde méně práce pro hrabání s menším k. Navíc, pokud k_1-*shrabané* pole následně k_2-*shrabeme* pro $k_2 < k_1$, zůstane i k_1-*shrabané*. Tedy pokud budeme pole hrabat postupně pro vybraná zmenšující se k a skončíme nakonec s $k = 1$, dostaneme setříděnou posloupnost za použití menšího počtu kroků, než by provedl původní insertion sort. Uvědomte si, že originální insertion sort je vlastně hrabičkové třídění s tím, že vezmem pouze jedno jediné $k = 1$.

Celý algoritmus hrabičkového třídění pro pole $a = (a_1, a_2, ..., a_n)$ je zobrazen v následujícím pseudokódu.

for správná čísla $k = \{k_l > k_{l-1} > \ldots > k_1\}$ **do**
  // prvních k prvků je už k-shrabaných, začni tedy od $k+1$
  for $i = \{(k+1), (k+2), \ldots, n\}$ **do**
    // zatřiď a_i *do k-shrabané úvodní podposloupnosti* $a_1, \ldots, a_{i-1}$*, tedy:*
    // skákej zpět po k prvcích a prohazuj k-sousední prvky, dokud je to potřeba:
    for $l = \{i, (i-k), (i-2k), \ldots\}$ **do**
      pokud jsou a_{l-k} *a* a_l *ve špatném pořadí (tj.* $a_{l-k} > a_l$*),*
      prohoď jejich hodnoty, jinak vyskoč z cyklu
    end for
  end for
end for

Výběr vhodné posloupnosti čísel $k_l, k_{l-1}, \ldots, k_1$ značně ovlivňuje výslednou rychlost celého hrabičkového třídění. Vaším úkolem bude implementovat algoritmus s následujícími možnostmi (K1), (K2) a (K3)

1. $\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{4} \rfloor, \ldots, 1$, tj. pro $n = 30$ např. 15, 7, 3, 1
2. $\frac{3^a - 1}{2}$ (pro $a = 1, 2, 3, \ldots$), výsledné číslo ne větší než $\lceil n/3 \rceil$, tj. např. 121, 40, 13, 4, 1
3. sestupná čísla tvaru $2^r 3^s$ pro $r, s \in Z_0^+ : 2^r 3^q < n$, tj. např. 16, 12, 9, 8, 6, 4, 3, 2, 1

a porovnat je s rychlostmi *insertion sortu* a *quicksortu*.

Zadání

Implementace

K dispozici (v příloze) máte implementace insertion sortu a quicksortu jako metody třídy **Array**. Vaším úkolem je naimplementovat algoritmus hrabičkového třídění jako další metodu hrabisort! třídy **Array** a uložit ji do souboru `hrabisort.rb` s následující kostrou

```ruby
class Array
  def hrabisort!(ktype = 1)
    #
    # ...
    #
    return self
  end
end
```

`hrabisort!` bude mít jeden nepovinný parametr $ktype \in \{1, 2, 3\}$, který bude určovat, která z možností (K1), (K2) nebo (K3) se použije pro generování hodnot k.

Výstup: soubor `hrabisort.rb`

Test

Unit testy jsou, vzhledem k synchronizaci s předmětem PES, ze zadání vynechány. Pokud máte však o programování vážnější zájem, doporučujeme si je udělat, neboť tvoří elementární znalost programátorské praxe a ušetří vám při řešení mnoha úloh spoustu práce v hledání zbytečných chyb nebo manuálního testování.

V případě zájmu testy uložte do souboru `hrabisort_test.rb`. Testy byly v plány dvojího druhu.

Otestování všech možností na malých polích

Pro malá pole není z časového hlediska problém otestovat všechny permutace jejích prvků. K tomuto účelu použijte přiloženou třídu **PermutationGenerator**. Ta ve svém konstruktoru očekává pole nějakých prvků. Následně lze na instanci této třídy volat

metodu `next`, která bude postupně vracet všechny permutace konstruktoru předaného pole, až na závěr vrátí `nil`. Například můžete tedy postupovat přibližně následujícím způsobem:

```ruby
require_relative "permutation_generator"
require_relative "quicksort"

generator = PermutationGenerator.new([0, 1, 2, 3, 4, 5, 6])
while array = generator.next()
  referential = array.dup
  quick = array.quicksort!
  referential.sort!
  assert referential == quick
end
```

Tímto způsobem otestujte všechny permutace polí velikosti nula až sedm.

Otestování náhodně velkých polí

Nelze se samozřejmě spokojit s testy pouze na takto malých polích. Každý algoritmus proto rovněž otestujte na náhodně velkém množství náhodně velkých polí. Počet takovýchto testů a maximální velikost pole volte na základě vlastní úvahy.

Nepovinný výstup: soubor `hrabisort_test.rb`

Benchmark

Změřte dobu běhu hrabičkového třídění varianty K1, K2 a K3, a dále insertion sortu a quicksortu na různě velkých polích. Měření proveďte postupně pro následující dvojice čísel (m, n), kde m je velikost jednotlivých polí a n vyjadřuje počet opakování: (10, 50000), (40, 10000), (160, 2000), (640, 200), (2560, 20), (10240, 5). Pro tyto dvojice proveďte měření následovně:

1. Vytvořte n polí velikosti m. Každé z polí bude obsahovat m náhodných hodnot z rozmezí 0 až $n - 1$ (výběr s opakováním).
2. Pro každý z výše zmíněných algoritmů změřte dohromady celkový čas, jak dlouho potrvá setřídění všech n polí (v cyklu za sebou).

Výsledkem tedy bude celková doba setřídění n polí o m prvcích, postupně pro pět různých třídicích algoritmů. Počítejte s tím, že výpočet bude několik desítek vteřin trvat, především kvůli poslední dvojici (m, n).

K měření doby trvání nějakého výpočtu použijeme modulu **Benchmark**. Přímo voláním metody `Benchmark.realtime`, které předáme blok, získáme čas (v sekundách), jaký trvalo blok vykonat. Použití tedy může vypadat například přibližně takto:

```ruby
require "benchmark"
# cyklus přes všechny dvojice (m, n)
  # vytvoření n polí délky m
  # pro každý algoritmus pak:
  # - zduplikování polí
  # - změření času setřídění
  t_quick_mn = Benchmark.realtime do
    arrays.each do |array|
      array.quicksort!
    end
  end
```

Nakonec nás zajímá

1. **doba** t_{alg} potřebná pro setřídění jednoho pole: $t_{alg} = \frac{\text{namereny cas}}{n}, \quad alg \in \{K1, K2, K3, insert, quick\}$
2. **poměr** r_{alg} doby t_{alg} vůči asymptoticky nejlepšímu průměrnému času třídění ve vnitřní paměti, tj. $r_{alg} = \frac{t_{alg}}{m \log m}, \quad alg \in \{K1, K2, K3, insert, quick\}$

Několik důležitých upozornění:

- Je nezbytné, aby všechny algoritmy prováděly třídění vždy stejné sady n polí. Jinak není možné výsledky porovnávat.
- Uvědomte si, že všechny algoritmy jsou implementovány tak, že modifikují přímo vstupní pole. Máte-li tedy nějaké pole třídit opakovaně různými algoritmy, musíte si bokem udržovat jeho původní, nemodifikovanou kopii. Dobu kopírování pole pak ovšem

nezahrnujte do bloku s měřením času.

Výstup benchmarku

Zpracujte kód měření do skriptu `hrabibench.rb` tak, že výstupem skriptu budou dva soubory `times.txt` s hodnotami t_{alg} a `ratios.txt` s hodnotami r_{alg} v následujícím formátu

```
# m       K1        K2        K3        insert    quick
10        1.23e-5   1.07e-5   ...       ...
40        2.94e-5   ...       ...
160       7.24e-4   ...
640       3.92e-2
2560      4.37e-1
10240     2.12
```

Formát těchto dvou souborů tedy bude prostý text s úvodním řádkem s popisky sloupců uvozeným znakem `#`. Následovat budou řádky s hodnotami ve sloupcích širokých 10 znaků. První sloupec budou hodnoty m (tedy délky polí), další sloupce budou jednotlivé časy nebo poměry zaokrouhlené na 3 platné cifry v uvedeném pořadí.

Výstup: soubor `hrabibench.rb`

Grafy a shrnutí výsledků

Nakonec obě sady hodnot zpracujte do následujících tří grafů, vždy s hodnotami m na vodorovné ose s lineární škálou, a na svislé ose

- s časy t_{alg} v lineární škále,
- s časy t_{alg} v logaritmické škále,
- s poměry r_{alg} v lineární škále.

Podle grafů pak posuďte, jakou průměrnou složitost mohou mít jednotlivá hrabičková třídění (K1), (K2) a (K3). Dále se zkuste zamyslet nad výhodami či nevýhodami hrabičkového třídění oproti quicksortu. Tuto úvahu v délce cca 1000 až 2000 znaků pak spolu s grafy uložte ve formátu PDF.

Výstup: soubor `vysledky.pdf`

Poznámky

Co se tedy očekává, že odevzdáte:

- Implementaci hrabičkového třídění: `hrabisort.rb`
- Testovací skript ověřující korektnost hrabičkového třídění: `hrabisort_test.rb`
- Skript provádějící měření rychlosti algoritmů: `hrabibench.rb`
- Shrnutí naměřených výsledků (ve formátu PDF): `vysledky.pdf`

Způsob odevzdání

- všechny (příp. čtyři) požadované soubory uložte do adresáře `prijmeni_jmeno_du2` , kde `prijmeni` a `jmeno` jsou vaše příjmení a jméno malými písmeny bez diakritiky
- adresář zabalte do souboru `prijmeni_jmeno_du2.zip` nebo `prijmeni_jmeno_du2.7z`
- v +4U artefaktu s odevzdáním vyplňte na začátek políčka **Název** řetězec `DU2`
- Nedodržení tohoto způsobu uložení bude mít za následek snížení počtu bodů, případně nemožnost úkol opravit.

Nezapomeňte mít počítač během měření co nejvíce „v klidu“, ideálně tak, aby úlohu nenarušovaly žádné další aplikace. I přehrávač hudby, který působí dojmem, že při hraní nebere takřka žádný procesorový čas (případně by se člověk mohl domnívat, že pokud bere,

tak bere „všem stejně“), musí čas od času načíst z disku do paměti nějaká další data k přehrávání, což sice netrvá čas postřehnutelný okem, nicméně měření v řádu desítek či stovek milisekund může snadno ovlivnit.

Řešení

Řešení posílejte tlačítkem níže.

Odeslat řešení domácího úkolu

Zdroje

- ALG Datové struktury
- ALG Třídění I
- ALG Třídění II