2-INF-237 Vybrané partie z dátových štruktúr

2-INF-237 Selected Topics in Data Structures

- Instructor: Broňa Brejová
- E-mail: brejova@fmph.uniba.sk
- Office: M163

- Course webpage: http://compbio.fmph.uniba.sk/vyuka/vpds/

Vybrané partie z dátových štruktúr

Broňa Brejová

- Prednášky utorok 9:50-11:20 M-VIII, streda 15:40-17:10 M-II
- Moodle na odovzdávanie úloh a evidenciu bodov,
  prihláste sa na predmet
- Konzultácie: po dohode
- Neváhajte klásť otázky na hodine, na konzultáciách, príp. emailom
- Prednášky môžu byť v angličtine

Ciele predmetu

- Oboznámenie sa s dátovými štruktúrami nepreberanými na základných bakalárskych predmetoch a s metódami ich analýzy. Zhrnieme tiež základné algoritmy na vyhľadávanie vzorky (slova) v texte.
- Použitie a prehĺbenie znalostí z predchádzajúcich predmetov týkajúcich sa tvorby a analýzy efektívnych algoritmov a dátových štruktúr.
- Získavanie skúseností v práci s odbornou literatúrou, navrhovaní a vyhodnocovaní výpočtových experimentov.

Pravidlá

Všetci:

Aktivita na hodine 10% (+ najviac 10% bonus)

Prezentácia článku 15%

Alternatíva A:

Domáca úloha 15%

Skúška 60%

Alternatíva B:

Domáca úloha 15%

Projekt 60%

Alternatíva C:

Projekt 75%

Alternatíva A je základ

V prípade záujmu o B alebo C sa príďte dohodnúť do **31.3.**

A: >=90, B:80...89, C: 70...79, D: 60...69, E: 50...59, FX: <50

Prácu cez semester berte vážne.

Pravidlá

Domáca úloha

V prvej polovici semestra, implementácia algoritmov a ich experimentálne porovnanie

Prezentácie

Vyberte si vedecký článok súvisiaci s témou predmetu (do 24.4.)

Prezentácie cez prednášky na konci semestra

Skúška

Písomná skúška (teória, riešenie problémov)

Povolený ťahák 2 listy A4

Pravidlá

Projekt (nepovinný)

Mini-diplomovka (5-15 strán)

Rôzne možnosti: Prehľad/implementácia a experimenty/teória

Výber témy do 31.3., dohodnite sa s vyučujúcou

Aktivita

Odpovedanie otázky cez prednášku, prezentácia nepovinnej DÚ

Písomné riešenie nepovinných DÚ

Písanie/zlepšovanie poznámok z prednášok na predmetovej wiki

Hľadanie chýb na prednáške

Neopisovať

Môžete sa o domácej úlohe rozprávať so spolužiakmi

Ale každý vlastnú implementáciu, experimenty a text

Nekopírovať z internetu, citujte prípadné zdroje

Literature

- P. Brass. Advanced Data Structures. Cambridge University Press 2008. In library I-INF-B-67
- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to Algorithms. MIT Press 2001. In library D-INF-C-1.
- D. Gusfield (1997) Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press. In library I-INF-G-8.
- MIT course Advanced Data Structures by Erik Demaine
- Gnarley trees: website by Kuko Kováč
- Research papers
- Class notes, slides

Opakovanie

Aké dátové štruktúry ste doteraz videli?

Aké základné operácie podporujú a v akom čase? (v najhoršom/amortizovanom/priemernom prípade)

Preberané témy

- Úsporné dátové štruktúry
- Prioritné rady (Fibonacciho haldy) a amortizovaná zložitosť
- Splay stromy, dynamické (link-cut) stromy
- RMQ a LCA
- Práca s textom: vyhľadávanie kľúčových slov, sufixové stromy a polia, klasické vyhľadávanie vzorky v texte, Burrowsova–Wheelerova transformácia
- Hešovanie
- Celočíselné kľúče
- Externá pamäť
- Perzistentné štruktúry
- Geometrické dátové štruktúry

Amortized analysis (amortizovaná analýza)

Usual worst-case analysis: analyze the running time of a single operation on the worst input.

Here: analyze the worst case for a whole sequence of operations

Definition:

If the real cost of ith operation c_i, we can say that amortized cost is $\hat{c}_i$ if $\sum_i c_i \leq \sum_i \hat{c}_i$

Example: vector (dynamic array)

Stores a sequence of elements

Supports operations

– `add(x)` : adds element x to the end of the stored sequence

– `get(i)` : get i-th element of the sequence

– `set(i,x)` : set i-th element of the sequence to value x

Implementation

Automatically growing arrray

– stores its size s and number of elements n

– when $n = s$, allocate a new array of size $2s$ and copy all elements over

Operation add for vector

```
1  void add(vector &a, dataType x) {
2    if (a.n == a.s) {
3      dataType *temp = new dataType[a.s * 2];
4      for (int i = 0; i < a.n; i++) {   // copy elements
5        temp[i] = a.a[i];
6      }
7      delete [] a.a;
8      a.a = temp;
9      a.s *= 2;
10   }
11   a.a[a.n] = x; a.n++;
12 }
```

Analysis of running time

Worst-case

`get` $O(1)$

`add` $O(n)$

Amortized

`get` $O(1)$

`add` $O(1)$

Plan:

– We will analyze amortized complexity by two different methods.

– We will consider only operation `add`.

– We will count how many times we run `temp[i] = a.a[i];`

– If this line is executed t times, running time is $O(1 + t)$.

Accounting method

- Charge $t_j(n)$ for operation o_j (its amortized cost)
- If $t_j(n) >$ cost of operation, distribute the rest to various accounts
- If $t_j(n) <$ cost of operation, charge some accounts
- All accounts have non-negative amounts at all times

For vector:

- Charge 2 for each `add` operation
- Account for each element of the array
- If no copying, store 1 in account $n - 1$, 1 in account $s - n$
- Otherwise use 1 from account i to copy i, then distribute 2 as above

Accounting method for vector

- Charge 2 for each `add` operation
- Account for each element of the array
- If no copying, store 1 in account $n-1$, 1 in account $s-n$
- Otherwise use 1 from account i to copy i, then distribute 2 as above

Prove non-negative account balances

– what is the status of accounts just before and after copying?

Conclusion

- Amortized cost 2 per `add`
- Overall for n `add` operations, line executed at most $2n$ times
- Amortized complexity $O(t_i + 1) = O(2 + 1) = O(1)$

Potential method

Let D_i be data structure after i-th operation

Potential function $\Phi(D_i)$ (analog of total amount in accounts)

Real cost of i-th operation c_i

Amortized cost of i-th operation $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Total amortized cost

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \left(\sum_{i=1}^{n} c_i\right) + \Phi(D_n) - \Phi(D_0)\end{aligned}$$

If $\Phi(D_n) \geq \Phi(D_0)$, amortized cost is upper bound on the true cost

Usually $\Phi(D_0) = 0$, so we need $\Phi(D_n)$ non-negative

Potential method for vector

- $\Phi = 2n - s$
- Just before resizing: $\Phi =$
- Just after resizing: $\Phi =$
- Actual cost of resizing: $c =$
- Amortized cost of resizing: $\hat{c} = c + \Phi_{after} - \Phi_{before} =$
- Amortized cost of increasing n: $\hat{c} = c + \Delta\Phi =$
- At the beginning assume $s = n = 0$, $\Phi(D_0) = 0$
- First `add`: increase s and n to 1, $c_i = 0$, $\Delta\Phi =$
- $\Phi(D_i) \geq \Phi(D_0) = 0$, because

Potential method for vector

- $\Phi = 2n - s$
- Just before resizing: $n = s$, $\Phi = n$
- Just after resizing: $2n = s$, $\Phi = 0$
- Actual cost of resizing: $c = n$
- Amortized cost of resizing:
  $\hat{c} = c + \Phi_{\text{after}} - \Phi_{\text{before}} = n + 0 - n = 0$
- Amortized cost of increasing n: $\hat{c} = c + \Delta\Phi = 0 + 2$
- At the beginning assume $s = n = 0$, $\Phi(D_0) = 0$
- First `add`: increase s and n to 1, $c_i = 0$, $\Delta\Phi = 1$
- $\Phi(D_i) \geq \Phi(D_0) = 0$, because $n \geq s/2$

Amortized analysis (amortizovaná analýza)

Analyze the worst case for a whole sequence of operations.

Real cost of ith operation c_i, amortized $\hat{c}_i$

We need $\sum c_i \leq \sum \hat{c}_i$

Accounting method:

Create accounts associated with values, nodes etc.

Operations with $c_i < \hat{c}_i$ store $\hat{c}_i - c_i$ in some accounts

Operations with $c_i > \hat{c}_i$ use money from the accounts

Potential method:

Keep only total in all acounts, potential function $\Phi(D)$

$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$