NTIN060 | Homework 07/9

Milan Wikarski (milan@wikarski.sk)

Zadanie

Korupce za časů korony. Někteří zákazníci nechtějí čekat a tak se rozhodnou uplácet, čímž se dostanou ze své současné pozice na pozici první (kde je opět někdo může předběhnout). Sestrojte datovou strukturu pro uložení seznamu tak, abychom uměli rychle najítk-tý prvek a přesunout ho na začátek. (AVL strom?)

Riešenie a dôkaz správnosti

Frontu zákazníkov budeme reprezentovať AVL stromom, ktorý si mierne upravíme a definujeme operáciu `FindNth()`.

Strom

Strom `T` bude mať v sebe uložený údaj o veľkosti (tj. počet vrcholov), budeme ho označovať `|T|`. Ďalej bude mať odkaz na prvý vrchol `v0` (môže platiť `v0 = ⌀`, ak `|T| = 0`).

Vrcholy

Každý vrchol `v` bude mať v sebe odkaz na pravé dieťa (budeme označovať `r(v)`) a ľavé dieťa (budeme označovať `l(v)`).

Podstrom, ktorý má koreň vo vrchole `v` budeme označovať `T(v)`.

Každý vrchol bude mať uloženú veľkosť jeho podstromu, tj. `|T(v)|`, pričom platí `|T(v) = 0|` ak `v = ⌀`.

Vrcholy budú v AVL strome zoradené podľa kľúčov, ktoré predstavujú poradie zákazníka vo fronte. Každý zákazník `Z` má pridelený kľúč `k(Z) ∈ { ..., -2, -1, 0, 1, 2, ...}`. Ak uvažujeme `n` zákazníkov, kľúče **nemusia** byť iba čísla z množiny `{1, 2, ..., n}`. Jedinou podmienkou je, že ak zákazník `A` je vo fronte pred zákazníkom `B`, potom platí `k(A) < k(B)`. Takisto pripúšťame záporné kľúče. Strom v sebe bude mať uloženú informáciu o hodnote najnižšieho kľúča, ktorý obsahuje. Túto hodnotu označíme `min_key`.

Operácie Insert(), Delete()

Náš AVL strom bude implementovať operácie:

- `Insert(key, value)`: pridá nový vrchol do stromu s kľúčom `key` a hodnoutou `value` (hodnota bude v našom prípade zákazník) a strom vyváži.
- `Delete(key)`: vymaže vrchol s kľúčom `key` zo stromu a strom vyváži.

Operácia FindNth()

Okrem bežných operácií implementujeme aj operáciu `FindNth(order)`. Táto operácia bude fungovať následovne: ak by sme zoradili vrcholy stromu podľa hodnoty kľúčov, tak by táto operácia vrátila vrchol na pozícii `order`, pričom uvažujeme jednotkový index. (toto je len pre vysvetlenie - operácia bude implementovaná inak).

Pri implementácii `FindNth()` využijeme fakt, že pre každý vrchol `v` v AVL strome platí, že všetky vrcholy, ktoré sú v jeho pravom podstrome `T(r(v))` majú väčší kľúč, ako vrchol `v` a všetky vrcholy, ktoré sú v jeho ľavom podstrome `T(l(v))` majú menší kľúč, ako vrchol `v`.

Začneme úvahou, že ak máme `|T|` vrcholov a my máme nájsť vrchol `v`, ktorého poradie má byť `order`, potom musí existovať práve `|T| - order` vrcholov s kľúčom väčším, ako `v`.

Budeme prehľadávať strom. Označíme `v` vrchol, v ktorom sa práve nachádzame a budeme si držať informáciu o tom, koľko vrcholov je väčších, ako vrchol `v`. Túto hodnotu si označíme `larger`. Hľadať začneme vo vrchole `v0` a položíme `larger <- |T(r(v0))|` (počet vrcholov napravo).

V každom kroku sa pozrieme na to, koľko vrcholov je väčších, ako vrchol `v`.

- Ak je ich presne `|T| - order`, tak sme hotoví a vrátime vrchol `v`.
- Ak ich je menej, potrebujeme ísť naľavo na vrchol `l(v)`. Položíme `v <- l(v)`. Teraz vieme, že vrchol, z ktorého sme prišli je väčší, ako vrchol `v` (lebo sme prišli zprava) a každý vrchol z pravého podstromu vrcholu `v` bude väčší, ako vrchol `v`. Zvýšime teda hodnotu `larger` o `1 + |T(r(v))|` (poznámka 1: tento výraz platí aj v prípade, ak `v` nemá pravého potomka, pretože sme definovali `|T(∅)| = 0`).
- Ak ich je viac, potrebujeme ísť napravo na vrchol `r(v)`. Položíme `v <- r(v)`. Teraz už vieme, že vrchol `v` nie je väčší ako vrchol `v` (je rovný) a celý ľavý podstrom vrcholu `v` bude menší, ako vrchol `v`. Znížime teda hodnotu `larger` o `1 + |T(l(v))|` (viz. poznámka 1).

Všetko dokopy

Najprv použijeme operáciu `FindNth()`, aby sme našli n-tého zákazníka vo fronte. Tohoto zákazníka potom z fronty vyhodíme pomocou operácie `Delete()`. Následne ho vrátime naspäť do fronty pomocou operácie `Insert()` a priradíme mu nový kľúč `k`, ktorý bude mať hodnotu `k <- min_key - 1`. Nakoniec položíme `min_key <- k`.

Správnosť

Úvaha, že vrchol, ktorý má poradie `order` je práve ten, pre ktorý existuje `|T| - order` väčších vrcholov je zrejme správna.

Náš algoritmus sa v každom kroku presunie buď naľavo alebo napravo, a teda sa v každom kroku nachádza v nižšej vrstve, ako v predošlom kroku. To znamená, že po konečnom množstve krokov sa zastaví.

Algoritmus

```
ALGORITMUS: FindNth
---------------------------------------------------------------
 INPUT: AVL Strom T, poradie order
OUTPUT: Vrchol, s daným poradím
 NOTES: Zavolaj FindNth2() s rovnakým inputom
        Začni vo vrchole v0
        Uvažuj, že všetky vrcholy napravo sú väčšie

1. return FindNth2(T, order, |T(r(v0))|, v0)
```

```
ALGORITMUS: FindNth2
---------------------------------------------------------------
 INPUT: AVL Strom `T`, poradie `order`, počet väčších vrcholov `larger`, vrchol `v`
OUTPUT: Vrchol, s daným poradím

1.  if (larger > |T| - order):
2.    v <- r(v)                              ◁ Choď napravo
3.    larger <- larger - (1 + |T(l(v))|)     ◁ Odrátaj vrchol, na ktorom sa nachádzaš a celý ľavý podstrom
4.    return FindNth2(order, larger, v)      ◁ Pokračuj v hľadaní
5.  else if (larger < |T| - order)
6.    v <- l(v)                              ◁ Choď naľavo
7.    larger <- larger + (1 + |T(r(v))|)     ◁ Prirátaj vrchol, z ktorého si prišiel a celý pravý podstrom
8.    return FindNth2(order, larger, v)      ◁ Pokračuj v hľadaní
```

```
9.  else:
10.   return v                              ◁ Vráť nájdený vrchol
```

```
ALGORITMUS: Predbehnutie
---------------------------------------------------------------
 INPUT: AVL Strom `T`, poradie `order`
OUTPUT: Strom T s novým poradím

1. zakaznik <- FindNth(T, order)
2. Delete(T, k(zakaznik))
3. Insert(T, min_key - 1, zakaznik)
4. min_key <- min_key - 1
5. return T
```

Časová a priestorová zložitosť

Označme si `n = |T|` počet vrcholov AVL stromu `T`.

Časová zložitosť

AVL strom má v každom okamihu najviac `O(log(n))` vrstiev. Operácie `Insert()` aj `Delete()` pracujú (aj s vyvažovaním) v čase `O(log(n))`. Operácia `FindNth()`, ktorú sme definovali sa v každom kroku presunie do ďalšej vrstvy a v každej vrstve vykoná iba konštantné množstvo výpočtov. Jej zložitosť teda je `O(log(n))`. Náš algoritmus využíva každú operáciu práve raz, a teda jeho výsledná časová zložitosť je `O(3 * log(n)) = O(log(n))`.

Priestorová zložitosť

Do AVL stromu sme museli každému vrcholu pridať informáciu o veľkosti jeho podstromu, čo zaberie `O(n)` pamäte.