Algoritmi in podatkovne strukture – 2

Rang in izbira

Rang in izbira

Operacije nad vrsto s prednostjo in slovarjem (izbor):

dodajanje: `Insert(`S`, x)` – v S dodamo nov element `x`.

izločanje: `Delete(`S`, x) --> y` – iz S izločimo element `x`. Rezultat `y` je lahko Boolova vrednost `true` ali `false`, ki sporoči ali je bil element uspešno izločen ali ne, ali pa operacija ničesar ne vrne.

najdi: `Find(`S`, x) --> y` – v S poiščemo element `y`, ki je najboljši približek elementu `x`.

Novi operaciji (štejemo od 1 do n):

rang: `Rank(`S`, x) --> i` – kateri element po vrsti je v S je x.

izbira: `Select(`S`, i)` – poiščemo v S `i`-ti element po velikosti;

rang': `Rank(`S`, &x) --> i` – v S smo našli element `x` in imamo referenco nanj ter se sprašujemo, kateri element po vrsti je v S.

Izpeljanke

Z uporabo na novo definiranih operacij lahko implementiramo nekatere prejšnje operacije:

- `Min(`S`)` $\equiv$ `Select(`S`, 1)`
- `DelMin(`S`)` $\equiv$ `Delete(`S`, Select(`S`, 1))`
- `Left(`S`, x)` $\equiv$ `Select(`S`, Rank(`S`, x)+1)`
- `Right(`S`, x)` $\equiv$ `Select(`S`, Rank(`S`, x)-1)`

Opazimo:

- Če nove operacije uspemo realizirati v času $o(\log n)$, potem bomo tudi stare operacije lahko realizirali v tem času.
- Torej?
- Kako implementirati novi operaciji?

Rešitev

Očitna (trivialna): *Imamo polje urejenih elementov iz S.*

- `Rank` in `Select` sta preprosti – odgovori hitri: $O(1)$;
- `Find` opravimo z razpolavljanjem – odgovor v $O(\log n)$;
- `Insert` in `Delete` (lahko) zahtevata premik vseh elementov v polju – **popravljanji počasni** $O(n)$.

Kaj pa drevo?

Lastnosti – *invariance*

Za vsak poddrevo (tudi za celotno drevo) uravnoteženega dvojiškega drevesa velja:

- v korenu poddrevesa r je shranjen ključ k in nekaj podatkov za uravnotežanje;
- elementi v levem poddrevesu L so manjši od korena;
- elementi v desnem poddrevesu R so večji od korena.

Dodajmo še eno informacijo v koren poddrevesa:

- število elementov v poddrevesu c

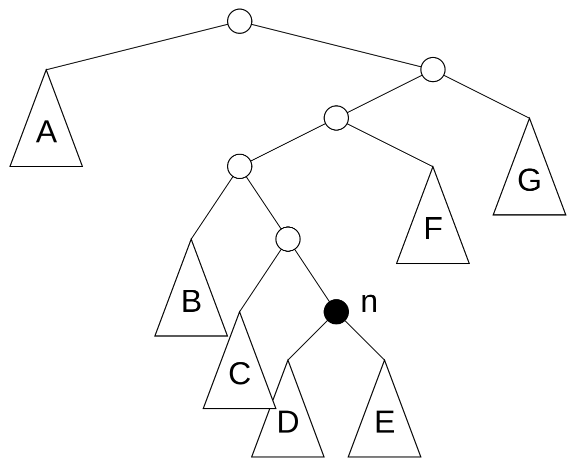
Kako pa je pri B drevesih?

NAMIG: Predvsem, kje je informacija o številu elementov c.

Lokalna opažanja

- `r.c`= `L.c` + `R.c` + 1
- koren je `L.c`-ti element v poddrevesu
- najmanjši element v desnem poddrevesu je (`L.c`+1)-vi element v poddrevesu

Globalna opažanja



- `Rank(n)` = `A.c` + 1 + `B.c` + 1 + `C.c` + 1 + `D.c` + 1
- v splošnem:

  na poti med korenom in n, prištejemo za vsa vozlišča, kjer gremo v desno poddrevo in za končno vozlišče n: velikost *levega* poddrevesa + 1

  Kako to dokazati?

  NAMIG: Z indukcijo.
- Kaj pa B drevesa?

Rang

```
int Size(Tree t) {
  if t == null return 0
  else         return t.c
}
int Rank(int x) {
  int poVrsti= Size(L) + 1;
  if x < k       return L.Rank(x)
  else if x == k return poVrsti
  else           return poVrsti + R.Rank(x)
}
```

- čas: $O(h) = O(\log n)$ – **super**
- prostor: $O(n)$ – **super**

Kaj pa če bi v vozlišču hranili preprosto podatek `poVrsti`?

Izbira

```
Node Select(int i) {
  int poVrsti= Size(L) + 1;
  if i  < poVrsti return L.Select(i)
  if i == poVrsti return this
  else            return R.Select(i-poVrsti)
}
```

- čas: $O(h) = O(\log n)$ – **super**
- prostor: $O(n)$ – **super**

Vstavljanje in izločanje

- dovolj, da ohranjamo lastnosti s prosojnice 5
- če vstavimo nov element (novo vozlišče) v, je število elementov v njegovem poddrevesu `v.c= 1`
- enojno in dvojno vrtenje (tako AVL kot RB drevesa)
- razcep in zlivanje pri B drevesih

Zahtevnost

mera/operacija	zahtevnost
dodajanje	$O(\log n)$
izločanje	$O(\log n)$
rank	$O(\log n)$
izbira	$O(\log n)$
iskanje	$O(\log n)$
najmanjši	$O(\log n)$
odreži	$O(\log n)$
spreminjanje	$O(\log n)$
levi sosed	$O(\log n)$
desni sosed	$O(\log n)$
velikost	$O(n)$

Komentarji:

- Vse operacije slovarja in vrste s prednostjo lahko izvedemo v času $O \log n$.
- Lahko naredimo hitreje?
- Kaj pa če bi v vozlišču hranili podatek `poVrsti`?