Napredni algoritmi i strukture podataka

HyperLogLog, SkipList

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

HyperLogLog

- HyperLogLog (HLL) je probabilistička struktura podataka koja se koristi za izračunavanje kardinaliteta (broj elemenata u skupu) velikih skupova podataka
- HLL nema potrebu da skladišti hash-eve
- HLL prvo primenjuje hash funkciju na sve vrednosti i predstavlja ih kao **cele brojeve iste veličine**
- Zatim ih pretvara u **binarne vrednosti** i procenjuje kardinalnost iz **heširane vrednosti**, umesto iz samih zapisa
- Izlaz hash funkcije je podeljen na dva dela
  - *Bakete* na osnovu vodećih *(leading)* bitova
  - *Vredosti* najveći mogući broj krajnjih uzastopnih *(consecutive)* nula
- Ako dobijemo više uzastopnih nula iz krajnjeg desnog bita za isti baket, ažuriraćemo taj baket.

- Oslanjamo se na nekoliko parametara:
  - **p** koliko vodećih bitova koristimo za baket
  - **m** veličina seta
- Prvo moramo da odredimo koliko vodećih bitova korisitmo za baket **p** (kolika je preciznost) obinočno u intervalu $[4, 16]$
- Veća vrednost **p** smanjuje grešku u brojanju, koristeći više memorije
- Nakon toga treba da izračunamo koliki nam set **m** trebam koristeći formulu $\mathrm{m} = 2^{\mathrm{p}}$

HyperLogLog - dodavanje

- Pretpostavimo da nakon hash funkcije i pretvaranja u binarni oblik, naš ključ **K** ima vrednost *1011011101101100000*
- Pretpostavimo da za preciznost odaberemo vrednost **4** ($p = 4$)
- Kao rezultat toga, znamo da je veličina seta $m = 2^4$ tj. **16** (po formuli $m = 2^p$)
- Iz dobijene binarne vrednosti *1011011101101100000* zaključujemo da je vrednost bucket-a gde ćemo upisati vrednost *1011* tj. **11**
- Vrednost koju upisujemo u baket *11* je *5*, zato što je broj nula sa kraja **5**, od ostalog dela binarnog zapisa *011101101100000*

HyperLogLog - kardinalitet

- Durand-Flajolet je izveo konstantu da ispravi pristrasnost ka većim procenama (algoritam se zove LogLog).
- $constant = 0.79402$
- $\text{CARDINALITY}_{\text{HLL}} = constant * m * \frac{m}{\sum_{n=1}^{m} 2^{-R_j}}$
- R_j označavaju broj nula od krajnjeg levog bita
- Izraz $\sum_{j=1}^{m} 2^{-R_j}$ se naziva *harmonijska sredina* čime se postiže smanjenje greške bez poveć anja potrebne memorije (Za dokaz konsultovati originalan rad)

Skip list - ideja

- Možete zamisliti ovu strukturu kao sistem metroa
- Postoje vozovi koji staje na svakoj stanici
- Ali, postoji i ekspresni voz koji staje na manje stanica
- Ovo čini ekspresni voz atraktivnom opcijom ako znate gde staje

Skip list - pretraga

Pretraga elementa **k** se vrši po sledećem algoritmu

- Ako je k $=$ key, kraj
- Ako je k $<$ next key, prelazimo na nivo ispod
- Ako je k $>=$ next key, idemo desno

Skip list - brisanje

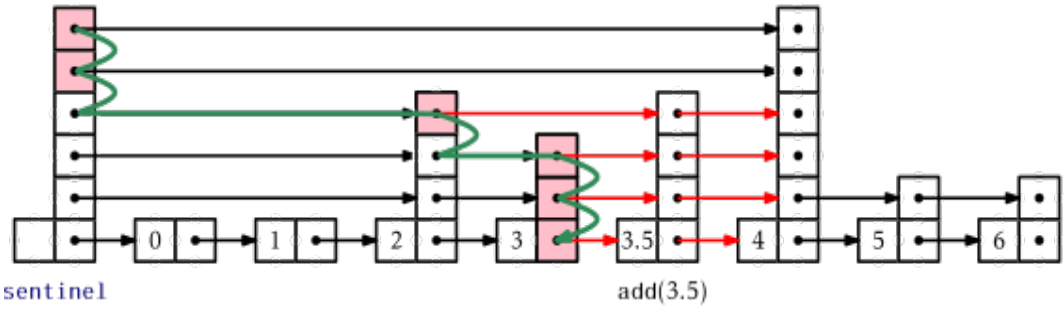
Brisanje elementa **k** se vrši po sledećim koracima:

- Lociramo koji element trebalo da se obriše, na osnovu prethodnog algoritma — **pretraga**
- Kada je element lociran, prevezujemo pokazivače da bi se element uklonio iz liste, baš kao što radimo u linked listi.
- Brisanje počinjemo od najnižeg nivoa i vršimo prevezivanje pokazivača sve dok ne stignemo do elementa
- Nakon brisanja elementa može postojati nivo bez elemenata, tako da ćemo i ove nivoe ukloniti, smanjivši nivo Skip liste.

Skip list - dodavanje

- Lociramo gde bi element trebalo da se doda, na osnovu prethodnog algoritma — **pretraga**
- Povežemo pokavivač prethodnog elementa na novokreiranim elementom
- Pokazivač novokreiranog elementa pokazuje na naredni element
- Ove operacije su identične kao i kod linked liste
- **ALI**, treba i da odredimo koliko nivao naš element ima

- Pronalazimo **k**
- Dodajemo nod u nivo **0**
- **while** $\mathrm{FLIP}() == \mathit{'GLAVA'}$
  - Dodajemo novi nivo
  - Povećavamo nivo elementa


0
1
2
3
3.5
4
5
6
sentinel
add(3.5)

Zadaci

- Implementirati HyperLogLog strukturu. Koristiti date pomoćne funkcije
- Implementirati SkipList strukturu