Napredni algoritmi i strukture podataka

Batch i mikro batch obrada podataka, Skip list, SimHash

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Batch obrada podataka

- Batch obrada podataka je metod obrade (uglavnom) veceg obima podataka (mogu se ponavljati), gde su podaci prvobitno skladišteni
- Ovaj metod omogućava korisnicima da procesuiraju podatke kada su računarski resursi dostupni sa malo ili bez interakcije korisnika
- Batch obrada uglavom traje duže zbog masivnosti podatka koji treba da se obrade, ili repeticije posla
- Primer za ovaj tip obrade bi bili, indeksiranje interneta i *search engines* (Google, Bing, DuckDuckGo, ...), obrada DNK, genoma, treniranje velikih modela sistema mašinskog učenja, obrada podataka sudara čestica, snimanje galaksija, crnih rupa itd.

Mikro Batch

- *Stream* i *Batch* procesuiranje su dva ekstrema
- U nekim situacijam se kombinuju zarad boljih rezultata (npr. Big Data - Lambda arhitekture)
- Mikro batch obrada je mogućnost kombinacije ova dva ekstrema u jedan unificiran način obrade
- Ovaj način obrade podataka je koristan za prikupljanja podataka u malim grupama (batch) u svrhu obrade tih podataka kako dolaze (stream)
- Često se koristi zarad uštede resursa
- Podaci mogu biti grupisani na razne načine: vremenski (1s, 1min, itd.), memorijski (1MB, 10MB, itd).

Problem

Zaposlili ste se u uzbuljivom startapu koji pravi novi sistem za skladištenje podataka. Jedan deo ovog sistema čuva podatke u memoriji i od vas se očekuje da implementirate sistem sa nekim ograničenjima i zahtevima:

- Dodavanje i brisanje treba da bude relativno brzo, i algoritam za to treba da bude jednostavan
- Pretraga bi trebala da bude relativno brza
- Prostorno treba da budemo efikasni što više možemo
- Podaci su u memoriji ali ih može biti dosta

Predlozi :) ?

Skip list - uvod

- Skip list je probabilistička struktura podataka koja je napravljena na opštoj ideji spregnutih (linked) lista
- Ova lista koristi verovatnoću za izgradnju novih (viših) slojeva spregnutih lista, na originalnoj listi.
- Skip list je struktura podataka koja se može koristiti umesto balansiranih stabala
- Skip list koristi verovatnoću, a ne striktno prinudno balansiranje
- Dodavanje i brisanje su znatno jednostavniji i brži od ekvivalentnih algoritama balansiranih stabala
- Svaki dodatni sloj veza sadrži manje elemenata, ali nema novih elemenata

Doubly-linked list vs skip list

- Dobre osobine doubly-linked list:
  - Lako za dodavanje i brisanje za $\mathcal{O}(1)$ vreme
  - Nema potrebe za procenom ukupne potrebne memorije
- Loš osobine doubly-linked list:
  - Teško je pretragu izvesti ispod $\mathcal{O}(n)$ vremena — binary search ne radi
  - Teško je skočiti na sredinu
- Skip list rešava ove probleme
- Očekivano vreme pretrage je $\mathcal{O}(\log n)$

Skip list - ideja

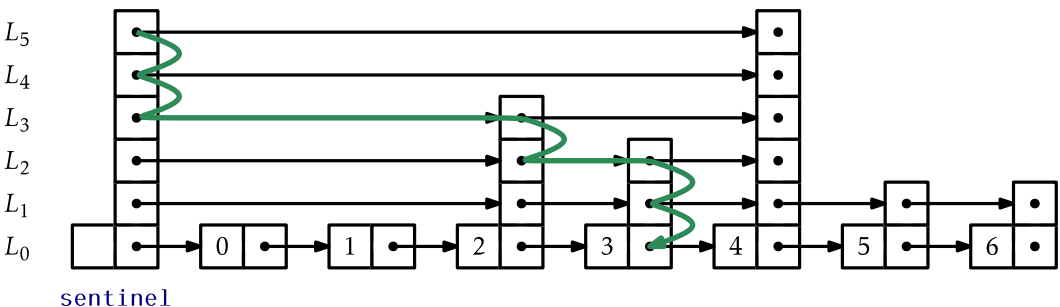
- Možete zamisliti ovu strukturu kao sistem metroa
- Postoje vozovi koji staju na **svakoj** stanici
- Ali, postoji i ekspresni voz koji staje na **manje** stanica
- Ovo čini ekspresni voz atraktivnom opcijom ako znate **gde** staje
- Početak (glava) i kraj (rep) imaju pokazivač ka **svakom** nivou
- Zove se Skip lista, jer omogućava **preskakanje** čvorova
- Čvorovi su **promenljive** visine od 1 do n pokazivača

Skip list - pretraga

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

- Počinjemo od *glave*
- Ako je k $=$ key, kraj našli smo element
- Ako je k $<$ next key, prelazimo na nivo ispod (drop down)
- Ako je k $>$ next key, idemo desno (scan forward)

Pretraga element **3**

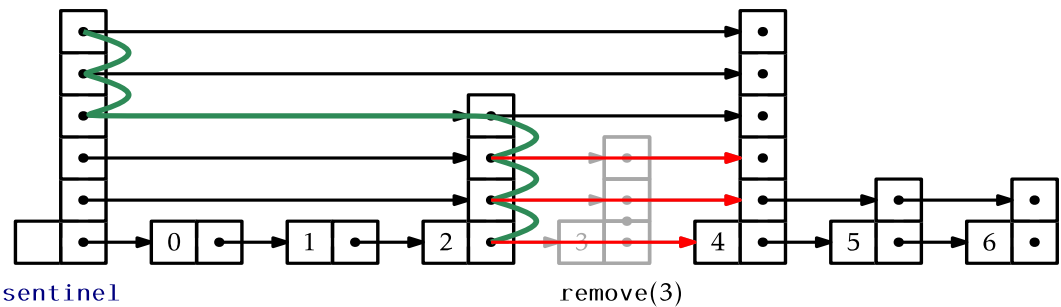


Skip list - brisanje

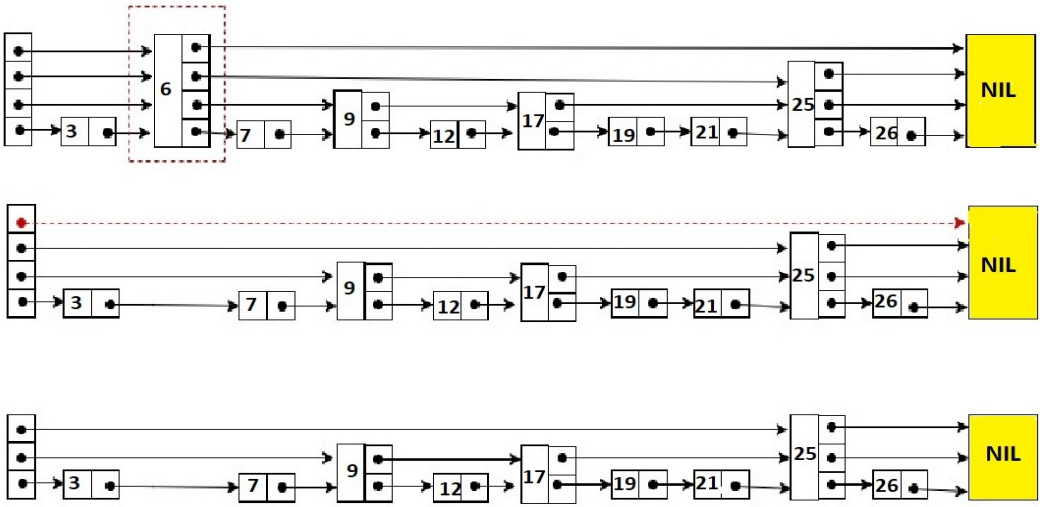
Brisanje elementa **k** se vrši po sledećim koracima:

- 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.

Brisanje elementa **3**



Brisanje elementa **6**, i smanjenje nivao liste

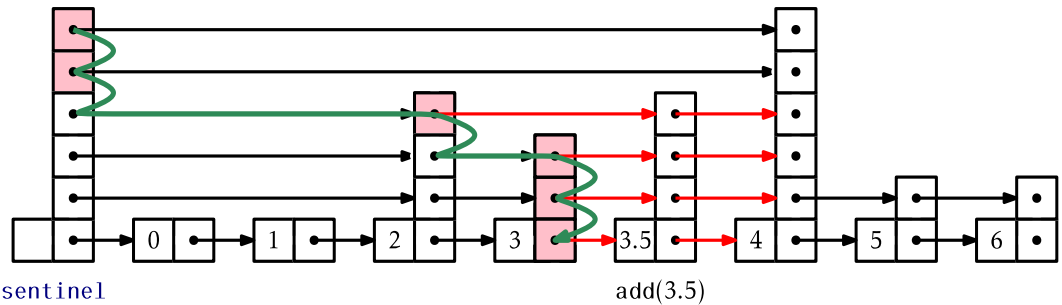


Skip list - dodavanje

- 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 – identične kao i kod linked liste
- **ALI**, treba i da odredimo koliko nivao naš element ima – koristimo verovatnoću
- Inicijalna visina svakog elementa je **0**
- Koristimo ideju bacanje novčića (Koje su šanse za bacanje jedne glave? 50 %. Dva zaredom? 25 %. Tri zaredom? 12,5 %.)
- Bacamo novčić, i sve dok dobijamo **1** (glava) povećavamo visinu elementa

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

Dodavanje element **3.5**



Problem 1

Zaposlili ste se u Google-u (bravo), i dobili ste zadatak da poboljšate njihov *crawler* mehanizam. Vaš zadatak je da utvrdite koje stranice su slične. Pred vama su sledeći zahtevi i ograničenja:

- Relativno laka paralelizacija posla
- Zauzimamo što manje resursa moguće
- Obrada podataka se radi nad velikim skupovima i u *batch-u*
- Jasan i jednostavan algoritam za razumevanje

Predlozi :) ?

Problem 2

Zaposlili ste se u istraživačkom centru. Vaš prvi posao kao inženjera jeste da nad dva skupa gena, ustanovite sličnosti, tako da treba da uporedimo dve sekvence gena. Pred vama su sldeći zahtevi i ograničenja:

- Relativno laka paralelizacija posla
- Zauzimamo što manje resursa moguće
- Jasan i jednostavan algoritam za razumevanje

Predlozi :) ?

Okruženje

- U mnogim aplikacijama, često imamo posla sa visokodimenzionalnim podacima
- Tekstualni dokumenti su predstavljeni kao vektori frekvencije reči, slike kao vektori intenziteta piksela ili profili korisnika kao vektori preferencija
- Tradicionalne metode kao što je prosta pretraga su nepraktične za velike skupove podataka
- Pronalaženje sličnih elemenata u visokodimenzionalnim prostorima je skupa operacija - "prokletstva dimenzionalnosti"
- Kako ovo rešiti?

LSH, ideja

- Ovde na scenu stupa *Locality Sensitive Hashing (LSH)*, koji nudi **skalabilno** rešenje
- *LSH* je **aproksimativna metoda** za probleme pretrage najbližeg suseda ili grupisanje vrednosti u visokodimenzionalnim prostorima
- Osnovna ideja je da se koriste *hash* funkcije koje čuvaju **lokalnost tačaka podataka**
- Tačke koje su **blizu** jedna drugoj u **originalnom prostoru** imaju veće šanse da budu **bliže u novom prostoru**
- Razne variajnte odredjivanja **blizine** tačaka:
  - Jaccard sličnost, koristi se za **skupove**
  - Kosinusna sličnost, koristi se za **vektore**
  - Euklidsko rastojanje, koristi se za tačke u Euklidskom prostoru

SimHash

- SimHash je tehnika za **brzu procenu** koliko su dva skupa podataka **slična**
- Dizajniran da **sačuva kosinusnu sličnost** izmedju vektora
- Radi tako što projektuje vektor na binarne *hash* vrednosti
- Fingerprint podataka je *hash* vrednost njegovih karakteristika – izbor je u zavisnosti koje karakteristike su nam bitne
- Snaga SimHash-a je u konverziji podataka u *hash* vrednost, i izračunavanje *Hemingovog rastojanja*
- Originalan skup podataka transformišemo u **vektor njegovih osobina** i radimo poredjenje

Hemingova udaljenost

- Hemingova udaljenost je metrika za poredjenje dva niza binarnih podataka **jednake dužine**
- Hemingova udaljenost je **broj pozicija** u kojima su dva bita različita
- Hemingovo rastojanje dobijamo koristeći **XOR** $(a \oplus b)$, a zatim računamo ukupan broj **jedinica** u rezultujućem nizu
- Koristi se za otkrivanje ili ispravljanje grešaka kada se podaci prenose preko mreže
- Koristi se u teoriji kodiranja za poredjenje podataka jednake dužine

- Slični skupovi podataka imaju slične *hash* vrednosti, samim tim i **vektor osobina**
- Što su skupovi podataka **sličniji**, to je Hemingova udaljenost **manja**
- Izuzetno **brz i efikasan** algoritam u pogledu skladišnog prostora
- Uglavom se koristi nad već **uskladištenim** podacima
- Pogodan za *batch* mehanizam obrade podataka
- Snaga je donekle u tome, što podatke već dobijemo spremne, nema potrebe za pripremom
- To znači da druge **pametne** mehanizme možemo iskoristiti za obradu, a ovaj mehanizam samo za sličnost – podela posla

SimHash - algoritam, primer tekst

- Set podataka (npr. tekst) podelimo na delove i uklonite zaustavne reči (ako ih ima)
- Dodelimo težine dobijenim vrednostima (npr. broj ponavljanja reči)
- Izračunamo **b**-bitni *hash* za svaki element iz dobijenog skupa, propuštajući element kroz *hash* funkciju, i pretvorimo u **binarni** oblik
- Za svaku dobijenu vrednost uradimo konverziju $\mathbf{0} \rightarrow \mathbf{-1}$
- Formiramo **tabelu**, tako što vrednosti stavimo **jedne ispod drugih**
- Sumiramo kolone, množeći težine sa vrednošću
- Ponovo izvršimo konverziju, ali sada za svaku vrednost u dobijenom rezultatu:
  - if $el \geqslant 0$, $el \leftarrow \mathbf{1}$
  - if $el < 0$, $el \leftarrow \mathbf{0}$
- Dobijamo **b**-bit fingerprint, vektor osobina, za ceo ulazni set – dužina zavisi od izbora *hash* funkcije
- Uradimo XOR operaciju sa drugim setom podataka i dobijamo Hemingovu udaljenost

SimHash - primer

- Pretpostavimo da imamo dokument sa narednim tekstom u sebu:
  - **Probabilistic data structures are fun, and fun is to learn them**
- Podelimo tekst na reči, obrišemo zaustavne reči, a za težine izaberemo (npr.) broj ponavljanja reči, dobijamo:
  - **probabilistic:1, data:1, structures:1, fun:2, learn:1, them:1**
- Propustimo svaku reč kroz neku heš funkciju h
- Dobijenu vrednost pretvorimo u binarni oblik
- Zbog jednostavnosti ispisa i proračuna, pretpostavimo da je izlaz nakon prethodne dve operacije 8-bitni
- Dobijamo, na primer, sledeće vrednosti za prethodni (test) skup podataka

Reč	Težina	Heš
probabilistic	1	11001010
data	1	10001101
structures	1	11010001
fun	2	11101110
learn	1	11001110
them	1	11101000

- Saberemo koline, i gde god je 0 postaje u -1, gde god je 1 ne menjamo
- Pomnožimo sa odgovarajućom težinom
- Primer za prvu kolonu bi glasio:
  - $1*1+1*1+1*1+2*1+1*1+1*1= 7 \rightarrow [7]$
- Da bi bilo zanimljivije, hajde da vidimo i primer za drugu kolonu:
  - $1*1+1*(-1) +1*1+2*1+1*1+1*1= 5 \rightarrow [7, 5]$

- Postupak je identičan za sve ostale kolone i dobijamo niz:
  - $[7\ 5\ 1 - 5\ 5\ 1\ 1 - 3]$
- Primenimo pravilo:
  - vrednost < 0, postaje 0
  - vrednost $\geqslant 0$, postaje 1
- Nakon primene ovog pravila imamo niz:
  - $[1\ 1\ 1\ 0\ 1\ 1\ 1\ 0]$
- Dobijeni niz predstavlja **jedinstveni identifikator** ili vektor osobina početnog dokumenta
- Ponovimo postupak sa drugi dokument (dobijemo npr. $[1\ 0\ 1\ 0\ 1\ 0\ 1\ 1]$) i uradimo **XOR** operaciju
  - $[1\ 1\ 1\ 0\ 1\ 1\ 1\ 0] \oplus [1\ 0\ 1\ 0\ 1\ 0\ 1\ 1] \rightarrow [1\ 0\ 0\ 0\ 1\ 0\ 1]$
- Broj jedinica u rezultatu je **Hemingovo rastojanje**, procena razlika dva podatka
  - $\mathrm{HammingDistance}([1\ 0\ 0\ 0\ 1\ 0\ 1]) \rightarrow 3$

Skip list - Dodatni materijali

- Skip list paper
- MIT Skip list
- Open data tructures Skip list
- Building a Skip list
- Probabilistic Data Structures and Algorithms for Big Data Applications

SimHash - dodatni materijali

- Pulse Code Modulation Techniques
- Similarity Estimation Techniques from Rounding Algorithms
- Detecting Near-Duplicates for Web Crawling
- Computing Text Similarity by Simhash+Hamming Distance
- Probabilistic Data Structures and Algorithms for Big Data Applications
- BLEND: A Fast, Memory-Efficient, and Accurate Mechanism to Find Fuzzy Seed Matches in Genome Analysis

Pročitati za narednu sedmicu

- Write-Ahead Log for Dummies (nije uvreda :))
- ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions