Napredni algoritmi i strukture podataka

Streaming podataka, Count-min sketch, HyperLogLog, Lookup tabele

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Streaming podatak

- Izraz *streaming* koristi se za opisivanje neprekidnih tokova podataka koji se kontinuirano generiše $[A, B, C, ..., \infty]$
- Ovaj pojam donosi konstantan niz podataka koji se mogu koristiti bez prethodnog skladištenja
- Ove podatke možemo da transformišemo, da ih skladištimo ili reagujemo kako dolaze
- Ovakve skupove podataka generišu razne vrste izvora, u različitim formatima i obimu

Count–min sketch - problem 1

Zaposlili ste se u Twitteru (jeee), vaš prvi zadatak je da napravite sistem za evidenciju hash tagova u objavama, da bi sledeći tim mogao da implementira bolji *trending* feature. Od vas se očekuje da napravite evidenciju frekfencije *hash tagova*, i pred vas su stavljena sledeća ograničenja i zahtevi:

- Sistem mora da radi sa *streaming* podacima
- Sistem mora da koristi malo resursa
- Sistem treba da omogući laku paralelizaciju
- 100 % preciznost nije obavezna

Predlozi :) ?

Count–min sketch - problem 2

Zaposlili ste se u Youtube-u (opaaa), vaš prvi zadatak je da napravite sistem za evidenciju pregleda video-a, da bi sledeći tim mogao da implementira bolji *recommender* sistem. Od vas se očekuje da napravite evidenciju frekfencije pregleda videa, i pred vas su stavljena sledeća ograničenja i zahtevi:

- Sistem mora da radi sa *streaming* podacima
- Sistem treba da koristi malo resursa
- Sistem treba da omogući laku paralelizaciju
- 100 % preciznost nije obavezna

Predlozi :) ?

Count-min sketch - Uvod

- *Count-min sketch* je probabilistčka strukture podataka koja služi kao **tabela učestalosti dogadjaja** u *stream-u* podataka
- Ova strutura koristi *hash funkcije* za preslikavanje dogadjaja na frekvencije
- Za razliku od hash tabele koristi manje prostora, na račun **precenjivanja nekih** dogadjaja nastalih zbog **kolizija** hash funkcija
- Jednom kreirana, struktura **ne raste**, ma šta radili sa njom – zgodna osoboina
- Zbog ovih osobina, često se koristi u sistemima koji rade za izuzetno velikom količinom podataka
- Druga vrlo zgodna primena su strimovi podataka – nema kraja podacima :)

- Ova struktura koristi **k** hash funkcija, slično kao i Bloom Filter
- Count-min sketch predstavlja tabelu gde registrujemo učestalost dogadjaja
- Svaka hash funkcija h_i se koristi **za korespodentni red** u tabeli
- Tabela ima **m** kolona, a vrednosti **nećemo birati nasumično**
- Preciznost ove strukture **zavisi od toga koliko redova dodajemo**, tj. koliko **hash funkcija koristimo**
- Više redova veća preciznost, više redova veća struktura – balans

Inicijalizacija tabele

- Inicijalno svaka ćelija unutar Count-min sketch (CMS) tabele se postavlja na vrednost **0**
- Zbog daljih operacija, ovo će biti neutralni element
- Ako imamo CMS sa **k** redova i **m** kolona onda je proces inicijalizacije sledeći:
  - $\forall i \in \{0, 1, ..., k\}$
  - $\forall j \in \{0, 1, ..., m\}$
  - $CMS[i, j] = 0$
- Dakle prodjemo kroz tabelu, i na svaki presek postavimo vrednost 0
- Može se izvesti relativno brzo

Count-min sketch - Dodavanje

Ako dobijemo element iz stream-a sa ključem **K**, postupak dodavanja je sledeći:

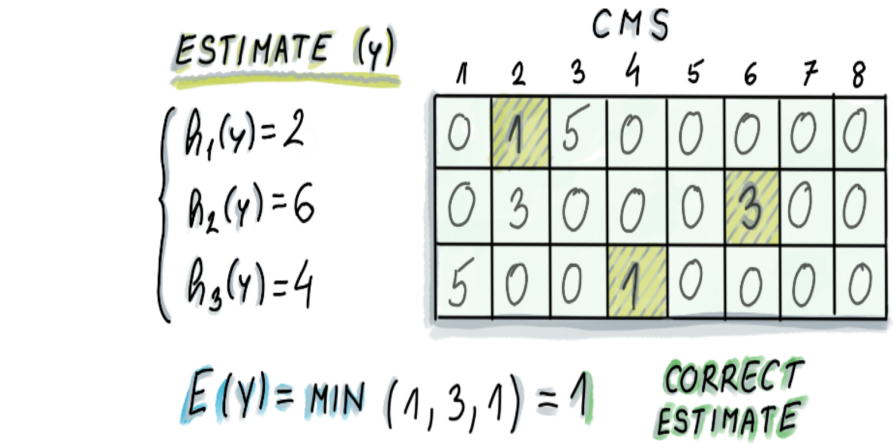
- Propustimo element **K** kroz **svaku hash funkciju**: $\forall\ h_i \in \{1, ..., k\}$
- Dakle svaka *hash* funkcija h_i je **red** u tabeli
- Dobijemo vrednost kolone: $j = h_i(K)\ \%\ m$ – slično kao i kod Bloom Filter-a
- Na preseku reda i kolone povećamo vrednost za **1**: $CMS[i, j] += 1$
- Moguće je obaviti operaciju relativno brzo

Count-min sketch - Dobijanje vrednosti

Ako želimo da vidimo učestalost elementa **K** u tabeli, postupak je sledeći:

- Propustimo element **K** kroz **svaku hash funkciju**: $\forall\ h_i \in \{1, ..., k\}$
- Dobijemo vrednost kolone: $j = h_i(K)\ \%\ m$ – slično kao i kod Bloom Filter-a
- Formiramo **niz** vrednosti sa odgovarajućih pozicija $R[i] = CMS[i, j], i \in \{0, ..., k\}$
- Uzmemo minimum iz niza i to je procena učestalosti dogadjaja **K**
  $E(K) = min(R[i]), i \in \{1, ..., k\}$
- Moguće je obaviti operaciju relativno brzo

Count-min sketch - Primer



(Algorithms and Data Structures for Massive Datasets, Medjedovic, D. and Tahirovic, E. and Dedovic, I. ISBN: 9781638356561, Manning)

Count-min sketch - Izbor parametara

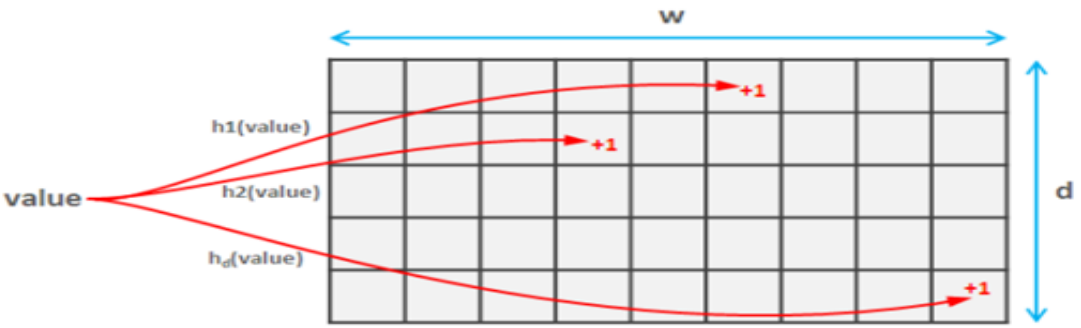
- Parametre **k** i **m** nećemo nasumično birati
- Kao i kod Bloom Filtera možemo da se oslonimo na malo matematike
- Ako hoćemo da definišemo tabelu veličine $k \times m$ treba da izaberemo preciznost (ε) koju želimo da postignemo, kao i sigurnost sa kojom dolazimo do tačnosti (δ)
- Dobijamo $k = [\ln \frac{1}{\delta}]$ i $w = [\frac{\epsilon}{\varepsilon}]$, gde je ϵ Ojlerov broj

ε	1 - δ	w	d	wd
0.1	0.9	28	3	84
0.1	0.99	28	5	140
0.1	0.999	28	7	196
0.01	0.9	272	3	816
0.01	0.99	272	5	1360
0.01	0.999	272	7	1940
0.001	0.999	2719	7	19033

(Introduction to Probabilistic Data Structures, DZone)

Napomena: $d = k, w = m$

Count-min sketch - Pitanja?



Pitanja :) ?

Count-min sketch - Dodatni materijali

- An improved data stream summary: the count-min sketch and its applications
- Algorithms and Data Structures for Massive Datasets
- Live example
- Probabilistic Data Structures and Algorithms for Big Data Applications

HyperLogLog - Problem 1

Zaposlili ste se u Facebook-u (you rocks), i od vas se traži da izračunate broj različitih korisnika koji su posetili Facebook u datoj nedelji, gde se svaka osoba prijavljuje više puta dnevno. Ovo rezultuje velikim skupom podataka sa mnogo duplikata. Od vas se zahteva da:

- Ne potrosite previse resursa
- 100 % tačan podatak nije obavezan
- Lako paralelizujemo proces
- Sistem treba da radi i sa *streaming* podacima

Predlozi :) ?

HyperLogLog - Problem 2

Zaposlili ste se u Google-u (bravo majstori), i od vas se traži da izračunate broj različitih stvari koje su korisnici pretraživali svaki dan. Ovo rezultuje velikim skupom podataka sa mnogo duplikata. Od vas se zahteva da:

- Ne potrosite previse resursa
- 100 % tačan podatak nije obavezan
- Lako paralelizujemo proces
- Sistem treba da radi i sa *streaming* podacima

Predlozi :) ?

HyperLogLog - Uvod

- HyperLogLog (HLL) je probabilistička struktura podataka koja se koristi za izračunavanje kardinalnosti velikih skupova podataka (broj različitih elemenata u skupu) – *Count-distinct problem*
- Kao i Bloom Filter i Count-min sketch, i on se oslanja na *hash funkcije*
- Za razliku od prethodne dve strukture, **on nema potrebu da skladišti** hash-eve
- HLL u se memoriji reprezentuje kao fiksna strutura koja neće rasti sa dodavanjem elemenata
- HLL rešava problem pronalaženja kardinalnosti masovnog skupa podataka koji koristi manje od **1,5 KB** memorije i sa procenom greške manjom od **2 %**

- Sam algoritam je relativno jednostavan, ali matematika i dokazi u pozadini nisu baš :/
- Kao i prethodna dva algoritma, danas se prilično intenzivno koristi u raznim aplikacijama sa velikim skupovima podataka
- Dosta se koristi kod *streaming* aplikcija
- Pogotovo je koristan u Big Data i Cloud aplikacijama gde su skopovi podataka jako veliki
- Zbog svojih osobina (kao i prethodne struture) mogu se čak koristiti i na sistemima sa ograničenim resursima, sa (skoro) identičnim performansama

Intuicija Flajolet i Martin

- Metrika Flajolet i Martin broji *nula* bitove na početku heširanih vrednosti
- Kod nasumičnih skupova podataka, sekvenca od k **uzastopnih** *nula* bitova će se pojaviti **jednom u svakih** 2^k **elemenata**, u proseku
- Potražimo sekvence, i zabeležimo najdužu sekvencu *nula* bitova da bismo procenili ukupan broj jedinstvenih elemenata
- Medjutim, ovo još uvek nije sjajna procena
  - U najboljem slučaju može nam dati procenu broja elemenata stepena dvojke uz ogromnu varijansu
  - Sa pozitivne strane, da bismo zabeležili sekvencu vodećih nula bitova u 32 bita, potreban nam je broj od 5 bita

HyperLogLog - Ideja

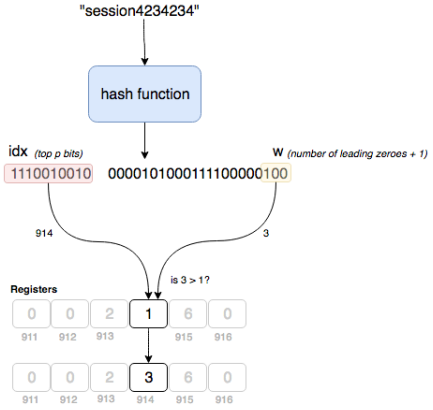
- Ako imamo dovoljno veliku kolekciju brojeva fiksne veličine (npr. 32/64/128 bita) i pronadjemo broj koji ima maksimlanih k **uzastopnih nula bitova**, možemo biti **gotovo** sigurni da postoji **najmanje** 2^k **brojeva** u toj kolekciji
- HLL prvo primenjuje hash funkciju na sve vrednosti i predstavlja 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 **+1**

- 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 strukture
- Vrednost **p** je obično u intervalu $[4, 16]$
- Veća vrednost **p** smanjuje grešku u brojanju, koristeći više memorije
- Nakon toga treba da izračunamo koliki nam set **m** treba, koristeći formulu $\mathrm{m} = 2^{\mathrm{p}}$

HyperLogLog - dodavanje

- Pretpostavimo da imamo HyperLogLog definisan sa preciznošću 10 ($p = 10$)
- Kao rezultat toga, znamo da je veličina seta $m = 2^{10}$ (po formuli $m = 2^p$)
- Ako korisnik hoće da dodam vrednost **X** u HyperLogLog, vrednost treba da heširmao i pretvorimo u binarni oblik
- Recimo da dobijamo vrednost: 11100100100000101000111100000100
- Iz dobijene binarne vrednosti zaključujemo da je vrednost bucket-a gde ćemo upisati vrednost *1110010010* tj. **914** – preciznost p
- Vrednost koju upisujemo dobijamo tako što prebrojimo broj **uzastopnih** nula sa kraja niza i na to dodamo **+1**
- Sada znamo gde upisujemo **bucket: 914**, i koja je **vrednost: 3** koja se upisuje

HyperLogLog - dodavanje slikovito



(https://djharper.dev/demos/hyperloglog/adding/)

HyperLogLog – kardinalnost

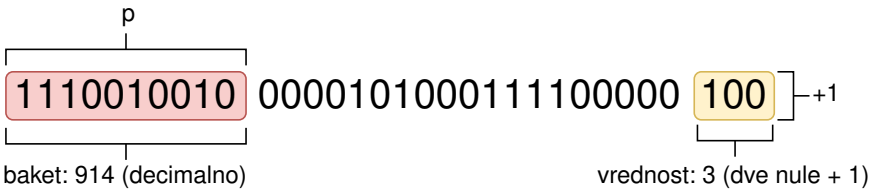
- Durand-Flajolet je izveo konstantu da ispravi pristrasnost ka većim procenama (algoritam se zove LogLog).
- $\text{CARDINALITY}_{\text{HLL}} = \text{constant} * m * \frac{m}{\sum_{n=1}^{m} 2^{-R_j}}$
- R_j označavaju registar ili pojedinačnu promenljiva koja sadrži najduži niz uzastopnih nula
- Izraz $\sum_{j=1}^{m} 2^{-R_j}$ se naziva *harmonijska sredina* čime se postiže smanjenje greške bez povećanja potrebne memorije (Za dokaz konsultovati originalan rad)
- Vrednost promenljive constant se obično računa i stvar je procene

Tabela 1 prikaze vrednosti konstante za najčešće vrednosti m in const.

m	const
2^4	0.673
2^5	0.697
2^6	0.709
$\geqslant 2^7$	$\frac{0.7213*\mathrm{m}}{\mathrm{m}+1.079}$

Table: Kombinacije različitih parametara

HyperLogLog - Pitanja



Pitanja :) ?

HyperLogLog - Dodatni materijali

- HyperLogLog Paper
- HyperLogLog playground
- Facebook engineering HyperLogLog
- Algorithms and Data Structures for Massive Datasets
- Probabilistic Data Structures and Algorithms for Big Data Applications

Opažanje

- Ako pogledamo prethodna dva mehanizma, vidimo da postoje **neke tabele**, gde unapred imam definisane vrednosti...
- Da li ih je potrebno uvek računati, ako već znamo vrenosti za dosta situacija?
- Kako ovo možemo izbeći, i kako možemo poboljšati nase mehanizme?

Ideje :)?

Lookup tabele – ideja

- Ideja iza Lookup tabela (LUTs) je relativno jednostavna
- To je je niz ili tabela koja zamenjuje računanje odredjenih parametara jednostavnijom operacijom indeksiranja
- Ideja je da, unapred **zakucamo** u nekakav niz ili tabelu **unapred poznate vrednost**i koje se često koriste ili su nekakve **podrazumevan vrednosti**
- Na taj način ne moramo stalno da ih računamo, čime možemo dodatno *ubrzati* program ili vreme potrebno za instanciranje komponente a na uštrb malo prostora

Prednosti LUT-a

- **Direktno adresiranje** vrednosti koristeći indeks ili nekakav ključ – brzo
- Iako jednostavna, ova ideja se pokazuje vrlo korisna u raznim problemima
- Izbor boja iz nekakve matrice, predefinisani parametri za Count-min sketch, HyperLogLog itd.
- Dodatno nam omogućva da odredjene strukture, algoritme ili proračune možemo da vršimo i na uredjajima sa ograničenom količinom resursa
- Zbog svoje jednostavnosti, mogu se često spustiti i u sam *hardware* čine dodatno možemo ubrzati izvršavanje

Važna napomena

Formule za Bloom filter, Count-min sketch HyperLogLog ne trebate da učite napamet!!!
Nemojte to sebi raditi!

Zanimljivo

Bioinformamtika, medicina i probabilističke strukture :)

To Petabytes and beyond: recent advances in probabilistic and signal processing algorithms and their application to metagenomics