

Napredni algoritmi i strukture podataka

Streaming podataka, Count-min sketch, HyperLogLog



Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Streaming podatak

- ▶ Izraz *streaming* koristi se za opisivanje neprekidnih tokova podataka koji se kontinuirano generiše $[A, B, C, \dots, \infty]$
- ▶ 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 frekvencije *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 sustavljena sledeća ograničenja i zahevi:

- ▶ 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 probabilistička struktura podataka koja služi kao **tabela učestalosti događaja** u *stream-u* podataka
- ▶ Ova struktura koristi *hash funkcije* za preslikavanje događaja na frekvencije
- ▶ Za razliku od hash tabele koristi manje prostora, na račun precenjivanja nekih događaja nastalih zbog kolizija hash funkcija
- ▶ Jednom kreirana, struktura ne raste, ma šta radili sa njom

- ▶ Ova struktura koristi **k** hash funkcija, slično kao i Bloom Filter
- ▶ Count-min sketch predstavlja tabelu gde registrujemo učestalost događaja
- ▶ Svaka hash funkcija (k_i) se koristi **za korespondentni red** u tabeli
- ▶ Tabela ima **m** kolona, vrednost **nećemo birati nasumično**
- ▶ Preciznost ove strukture zavisi od toga koliko redova dodajemo, tj. koliko **hash funkcija koristimo**

- ▶ Inicijalno svaka ćelija unutar Count-min sketch (CMS) tabele se postavlja na vrednost **0**
- ▶ Ako imamo CMS sa **k** redova i **m** kolona onda je proces inicijaliacije sledeći:
 - ▶ $\forall i \in \{0, 1, \dots, k\}$
 - ▶ $\forall j \in \{0, 1, \dots, m\}$
 - ▶ $\text{CMS}[i, j] = 0$

Count-min sketch - Dodavanje

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

- ▶ Propustimo element **K** kroz **svaku hash funkciju**: $\forall h_i \in \{1, \dots, k\}$
- ▶ Dobijemo vrednost kolone: $j = h_i(K) \% m$
- ▶ Na preseku reda i kolone povećamo vrednost za **1**: $CMS[i, j] += 1$

Count-min sketch - Dobijanje vrednosti

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

- ▶ Propustimo element **K** kroz **svaku hash funkciju**: $\forall h_i \in \{1, \dots, k\}$
- ▶ Dobijemo vrednost kolone: $j = h_i(K) \% m$
- ▶ Formiramo niz vrednosti sa odgovarajućih pozicija $R[i] = CMS[i, j], i \in \{0, \dots, k\}$
- ▶ Uzmemo minimum iz niza i to je procena učestalosti događaja **K**
 $E(K) = \min(R[i]), i \in \{1, \dots, k\}$

Count-min sketch - Primer

ESTIMATE (y)

$\begin{cases} h_1(y) = 2 \\ h_2(y) = 6 \\ h_3(y) = 4 \end{cases}$

CMS

1	2	3	4	5	6	7	8
0	1	5	0	0	0	0	0
0	3	0	0	0	3	0	0
5	0	0	1	0	0	0	0

$E(y) = \min(1, 3, 1) = 1$ CORRECT ESTIMATE

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

Count-min sketch - Izbor parametara

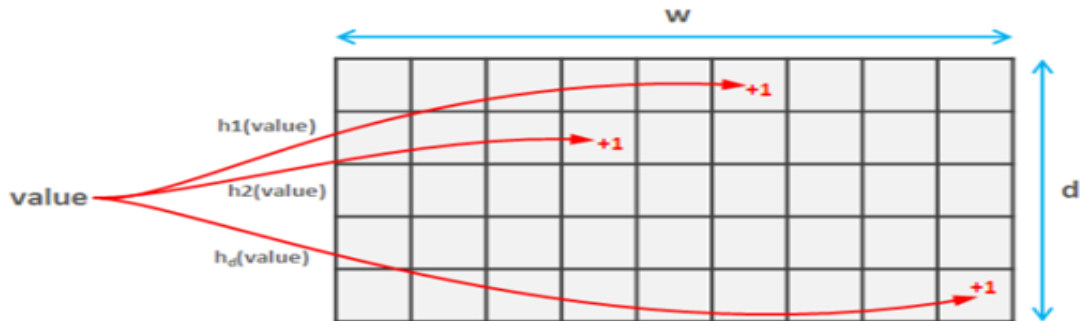
- ▶ Parametre **k** i **m** nećemo nasumično birati
- ▶ Kao i kod Bloom Filtera možemo da se oslonimo na malo matematike
- ▶ Ako hoćemo da definišemo tabelu veličine $k \times m$ treba da izaberemo preciznost (ϵ) koju želimo da postignemo, kao i sigurnost sa kojom dolazimo do tačnosti (δ)
- ▶ Dobijamo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

ϵ	$1 - \delta$	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

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 potrošite 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 kardinaliteta velikih skupova podataka
- ▶ 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 struktura 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
- ▶ Pogotovo je koristan u Big Data, Streamin i Cloud aplikacijama gde su skupovi podataka veliki
- ▶ Zbog svoje osobine (kao i prethodne struture) mogu se čak koristiti i na sistemima sa ograničenim resursima

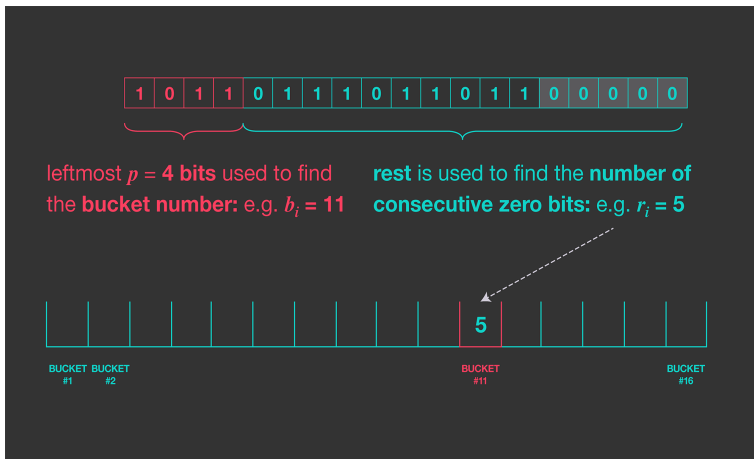
HyperLogLog - Ideja

- ▶ 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, ažuriracćemo taj baket.

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

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 in Presto: A significantly faster way to handle cardinality estimation, Facebook engineering)

HyperLogLog - kardinalitet

- ▶ Durand-Flajolet je izveo konstantu da ispravi pristrasnost ka većim procenama (algoritam se zove LogLog).
- ▶ $\text{constant} = 0.79402$
- ▶ $\text{CARDINALITY}_{\text{HLL}} = \text{constant} * m * \frac{m}{\sum_{n=1}^m 2^{-R_j}}$
- ▶ 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 povećanja potrebne memorije (Za dokaz konsultovati originalan rad)

HyperLogLog - Pitanja

Pitanja :) ?

HyperLogLog - Dodatni materijali

- ▶ HyperLogLog Paper
- ▶ HyperLogLog playground
- ▶ Facebook engineering HyperLogLog
- ▶ Algorithms and Data Structures for Massive Datasets

Važna napomena

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