

Napredni algoritmi i strukture podataka

Probabilističke strukture podataka, Hash funkcije, Bloom filter



Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Probabilističke strukture

- ▶ Probabilističke strukture podataka su klasa struktura podataka koja je izuzetno korisna za aplikacije velikih podataka (Big Data) i za striming
- ▶ Ove strukture podataka koriste *hash* funkcije za randomizaciju, i kompaktno predstavljanje skupa podataka
- ▶ Kolizije se zanemaruju, ali se greške mogu kontrolisati
- ▶ Nasuprot klasičnim (determinističkim), ovi strukture koriste **znatno** manje memorije, i imaju konstantno vreme upita za proizvoljan skup podataka
- ▶ Obično podržavaju operacije spajanja i ukrštanja, pa se stoga mogu lako paralelizovati

- ▶ Uobičajen pristup je upotreba neke vrste determinističke strukture podataka poput HashSet—a ili Hashtable—a za aplikacije velikih podataka i za striming
- ▶ Ali kada skup podataka sa kojim se bavimo postane veoma veliki, takve strukture podataka jednostavno nisu upotrebljive jer su podaci preveliki da bi stali u memoriju
- ▶ Probabilističke strukture koristimo za proveru da li je element prisutan u skupu, izračunavanje kardinaliteta, izračunavanje frekfencije pojave nekog elementa, itd.
- ▶ Primeri ovih struktura su **BloomFilter**, Locality-sensitive hashing, **Count-min sketch**, **Skip list**, Cuckoo filter, **HyperLogLog**, Quotient filter, itd

Hash funkcije

- ▶ Jednosmerne funkcije (one-way) su zamišljene sa idejom da bude teško da na osnovu izlaza (outputa) možemo dobiti originalni ulaz (input)
- ▶ Obično za ulaze **različite dužine daju izlaze uvek jednake dužine** – uniformno predstavljanje podatka
- ▶ Uvek za isti ulaz daje isti rezultat (izlaz)
- ▶ Dobar način da se proverí da li je neko kompromitovao ulazne (originalne) podatke
- ▶ *Mogu proizvesti kolizije - dva različita podataka imaju istu hash vrednost*
- ▶ Primeri: MD5, SHA-256, SHA-384, SHA-512, murmur itd

Problem 1

Potrebno je da proverimo da li je uneta e-mail adresa zauzeta, i pred nama su sledeća ograničenja:

- ▶ Treba da potrošimo izuzetno **malo** resursa
- ▶ Odgovor treba da dobijemo **brzo** (instant)
- ▶ Dopusšteno je tolerisati **false-positive** odgovore, zarad brzine i male potrošnje resursa

Predlozi :) ?

Problem 2

Potrebno je da pretražimo n čvorova koji sadrže korisničke podatke. Svaki čvor sadrži 100TB podataka (Facebook, Twitter, Instagram, ...). Ako znamo da se podaci sigurno nalaze na bar **3** čvora, i dalje treba da pretražimo veliku količinu podataka. Pretragu treba da ubrzamo tako što nećemo vršiti pretragu ako znamo da **ključ** nije na tom čvoru. I pred nama su sledeća ograničenja:

- ▶ Treba da smanjimo broj pretraga
- ▶ Odgovor treba da dobijemo **brzo**
- ▶ Treba da potrošimo izuzetno **malo** resursa

Predlozi :) ?

Bloom filter - uvod

- ▶ Bloom filter je probabilistička struktura podataka dizajnirana da brzo i efikasno odredi da li je neki element prisutan u skupu
- ▶ Bloom filter je fikasna struktura u pogledu korišćenja memorijskog prostora
- ▶ Bloom filter sa 1% false positive rate zahteva **samo 9,6 bita po elementu bez obzira na veličinu elemenata**

- ▶ Cena koja se plaća za ovu efikasnost je ta da Bloom filter može da nam kaže da li element **sigurno nije** u skupu, ili je on **možda** u skupu
- ▶ Bloom filter podržava: dodavanje, i pretragu elemenata
- ▶ (osnovni) Bloom filter ne podržava **brisanje** elemenata
- ▶ (osnovni) Bloom filter ne podržava vraćanje **broja** elemenata koji je zapisan
- ▶ Neke naprednije izvedbe ove strukture omogućavaju i ove funkcije
- ▶ Za većinu realnih primena, ove funkcije nam nisu preko potrebne

Bloom filter - parametri

Bloom filter zahteva nekoliko parametara za ispravan rad:

- ▶ Niz bitova veličine **m**, gde su svi bitovi inicijalno postavljeni na vrednost **0**
- ▶ **k** *hash* funkcija za izračunavanje heševa za dati ulaz
- ▶ Koristeći prethodne parametre, možemo odrediti poziciju bit-a koji treba da prebacimo sa **0** na **1** prilikom dodavanja elementa u filter

Bloom filter - dodavanje

Kada želimo da dodamo element u filter, koristimo par jednostavnih pravila:

- ▶ koristeći **k** hash funkcija ($h_1(x), h_2(x), \dots, h_k(x)$) izračunamo indekse u setu koje ćemo prebaciti sa **0** na **1**.
- ▶ ako se desi kolizija, tj. da je bit već postavljen na vrednost **1**, sve ok, nastavljamo dalje
- ▶ Za set veličine **m**, imamo **k** hash funkcija, onda je proces dobijanja indeksa sledeći:

$$h_1(\text{"key"}) \% m = i_1$$

$$h_2(\text{"key"}) \% m = i_2$$

$$\vdots$$

$$h_k(\text{"key"}) \% m = i_k$$

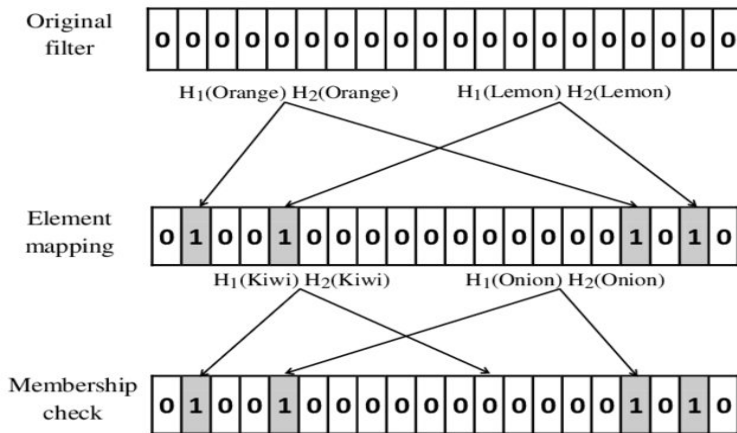
$$\text{Gde } i_{ith} \in \{0, 1, \dots, m - 1\}$$

Bloom filter - pretraga

Kada želimo da proverimo da li je element prisutan u filteru, koristimo par sledećih pravila:

- ▶ koristeći **k** hash funkcija ($h_1(x), h_2(x), \dots, h_k(x)$) potrebno je da izračunamo indekse u setu gde treba da proverimo da li je vrednost **0**
- ▶ Da bi smatrali da je element u setu, svih **k** indeksa treba da vrate vrednost **1**
- ▶ Ova operacija može da dovede do **false-positive** rezultata zbog kolizije *hesh* funkcija
- ▶ Zbog prethodne tvrdnje Bloom filter ne može da garantuje da je element **sigurno** prisutan u skupu
- ▶ Ali **sigurno** može da nam kaže ako on nije prisutan u skupu – vrlo korisna stvar

Bloom filter - primer



(Shubbar, Ro'aa and Ahmadi, Mahmood. (2019). A Filter-Based Design of Pending Interest Table in Named Data Networking. Journal of Network and Systems Management. 27. 10.1007/s10922-019-09495-y.)

Bloom filter - formule

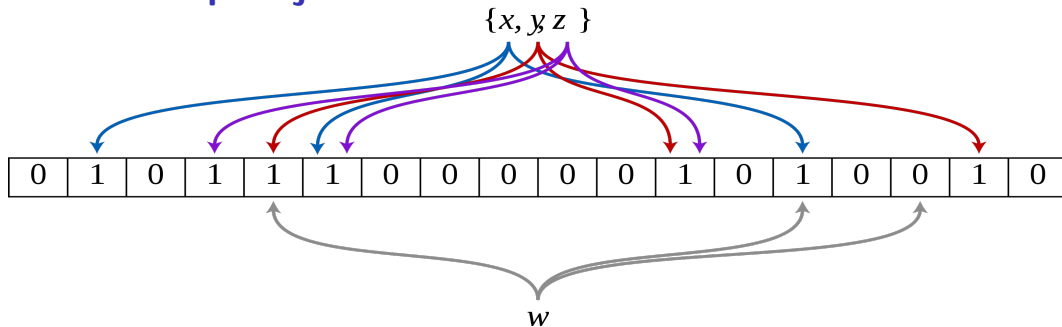
- ▶ Parametre **m** i **k** nećemo nasumično birati
- ▶ Njih biramo shodno tome koju verovatnoću **false-positive** dopuštamo u sistemu
- ▶ Ako pretpostavimo da će set sadržati **n** elemenata, onda verovatnoću **p** možemo izračunati sa: $p = (1 - [1 - \frac{1}{m}]^{kn})^k$
- ▶ Veličinu bit seta **m** možemo izračunati na sledeći način: $m = -\frac{n \ln p}{(\ln 2)^2}$
- ▶ Optimalan broj hash funkcija **k**, možemo izračunati na sledeći način: $k = \frac{m}{n} \ln 2$

Bloom filter - nedostaci

Bloom filter ima nekoliko nedostataka:

- ▶ Veličina Bloom filtera mora biti poznata unapred, što nije uvek moguće lako odrediti (Scalable Bloom Filter rešava taj problem)
- ▶ Bloom filter ne može da nam vrati listu elemenata koji su uneti
- ▶ Bloom filter ne može sa sigurnošću da nam vrati da li je element u setu
- ▶ Brisanje elemenata nije moguće (Counting Bloom Filter omogućava brisanje)

Bloom filter - pitanja



(Image by David Eppstein)

Pitanja :) ?

Bloom filter - dodatni materijali

- ▶ Theory and Practice of Bloom Filters for Distributed Systems
- ▶ Network Applications of Bloom Filters: A Survey
- ▶ Scalable Bloom Filters
- ▶ The Deletable Bloom filter - A new member of the Bloom family
- ▶ Bloom Filters A Tutorial, Analysis, and Survey
- ▶ Applications of Bloom Filter
- ▶ Probabilistic Data Structures and Algorithms for Big Data Applications
- ▶ Algorithms and Data Structures for Massive Datasets

Pročitati za narednu sedmicu

- ▶ Count-Min Sketch
- ▶ HyperLogLog