Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

Analiza velikih skupova podataka

Autori: Marin Šilić, Klemo Vladimir
Ak. god. 2017/2018

1. Laboratorijska vježba

U prvoj laboratorijskoj vježbi zadatak je ostvariti sažimanje tekstova korištenjem algoritma **Simhash**. Za razliku od kriptografskih algoritama sažimanja (koji su izuzetno osjetljivi na minimalne promjene ulaznog teksta), algoritam Simhash čuva sličnost ulaznih tekstova; ako su ulazni tekstovi vrlo slični (npr. nekoliko različitih riječi) onda se i sažeci generirani algoritmom Simhash razlikuju u malom broju bitova (po Hammingovoj udaljenosti). Generirani Simhash sažeci će se koristiti za identifikaciju sličnih tekstova.

U sklopu vježbe postoje dva zadatka za identifikaciju sličnih tekstova. U prvom zadatku (zadatak A) identifikaciju sličnih tekstova treba provesti **slijednim pretraživanjem sažetaka** svih tekstova. U drugom zadatku (zadatak B) potrebno je koristiti tehniku sažimanja **osjetlljvog na bliskost** (eng. *Locality Sensitive Hashing, LSH*). Ostatak dokumenta organiziran je na sljedeći način: ulomak 1.1 opisuje algoritam Simhash, ulomak 1.2 opisuje format ulazne datoteke za zadatke A i B, dok ulomak 1.3 opisuje algoritam sažimanja LSH.

1.1 Algoritam SimHash

Ulaz u algoritam simhash je niz znakova (tekst, dokument), a izlaz je niz znakova koji predstavlja **heksadecimalni zapis 128-bitnog** Simhash sažetka. Algoritam Simhash je detaljno opisan na predavanju "Detection of near-duplicate documents".

Algoritam Simhash interno koristi jedan od tradicionalnih algoritama sažimanja. U ovoj laboratorijskoj vježbi će se koristiti 128-bitni algoritam kriptografskog sažimanja **md5**. Dodatno, ulazni tekst u algoritam simhash će se razlučiti na niz jedinki odvojenih znakom razmaka. Npr. ako je ulaz u simhash tekst `"fakultet elektrotehnike i racunarstva"`, onda će simhash razdvojiti tekst na 4 jedinke `["fakultet", "elektrotehnike", "i", "racunarstva"]` te za svaku jedinku računati md5 sažetak. Izgrađeni md5 sažeci se analiziraju te se generira konačni sažetak kao izlaz algoritma simhash prema pseudokodu:

```
funkcija simhash(tekst):
  var sh = []
  jedinke = generiraj_jedinke(tekst)
  za svaku jedinku iz jedinke:
      hash = md5(jedinka)
      ako je i-ti bit u hash jednak 1:
        sh[i] += 1
      inače:
        sh[i] -= 1
  ako je i-ti element u sh >= 0:
    sh[i] = 1
  inače:
    sh[i] = 0
  vrati heksadecimalno(sh)
```

Napomena: prethodni pseudokod je samo jedna od više mogućih ilustracija rada algoritma i ne predstavlja obavezan predložak rješenja.

Primjer očekivanog sažetka generiranog algoritmom simhash za ulaz "**`fakultet elektrotehnike i racunarstva`**" jest "**f27c6b49c8fcec47ebeef2de783eaf57**".

Preporuča se ostvarenje vježbi u programskim jezicima Java i Python, asistenti su ostvarili vježbe u tim jezicima i mogu vam pružiti eventualno potrebnu pomoć. Pitanja možete poslati na službeni mail predmeta avsp@zemris.fer.hr.

Ako koristite programski jezik Python, onda je algoritam md5 dostupan u knjižnici **hashlib**. U programskom jeziku Java koristite isključivo paket

- **`org.apache.commons.codec.digest.DigestUtils`** (http://commons.apache.org/proper/commons-codec/apidocs/org/apache/commons/codec/digest/DigestUtils.html)

1.2 Zadatak A - slijedno pretraživanje sažetaka

Napomena: *Zadatak A nosi 50% bodova prve laboratorijske vježbe*.

Osim izračunavanja simhash sažetka, u ovoj laboratorijskoj vježbi je potrebno ostvariti i identificiranje sličnih tekstova. Format ulazne datoteke u program koji predajete u ovoj laboratorijskoj vježbi je:

```
N
tekst_0
...
tekst_N-1
Q
upit_0
...
upit_Q-1
```

U ulaznom zapisu, svaki redak završava znakom za kraj retka (**\n**). Linija 1. sadrži cijeli broj (**N <= 1000**) koji označava broj tekstova za koje se računaju sažeci. Nakon linije 1. slijedi N linija s N tekstova. Svaki tekst je zapisan u zasebnoj liniji te sadrži male engleske znakove. Nakon N ulaznih tekstova slijedi linija koja sadrži jedan cijeli broj koji označava broj upita (**Q <= 1000**). Posljednjih Q linija predstavlja upite za izračun sličnosti.

Jedan upit sadrži dva cijela broja odvojena prazninom (**I, K; 0 <= I <= N-1 i 0 <= K <= 31**). Za **svaki upit** program mora generirati cijeli broj koji označava ukupan broj tekstova čiji se sažeci razlikuju do na K bitova (uključujući K) od sažetka I-tog teksta (po Hammingovoj udaljenosti). Npr. ako je I=16 i K=3 onda izlaz za navedeni upit sadrži broj tekstova čiji se sažeci razlikuju od sažetka 17-tog (pretpostavlja se *0-based indexing* dokumenata) teksta do uključivo 3 bita.

Važne napomene:

- Vremensko ograničenje na izvođenje programa za bilo koju ulaznu definiciju automata jest 20 sekundi
- Ulazna točka za Java rješenja treba biti u **razredu SimHash**, a ulazna točka u Python rješenja treba biti u datoteci **SimHash.py**

1.2.1 Primjer za provjeru valjanosti

Na stranicama predmeta postavljen je primjer ulazne datoteke s pripadajućim očekivanim izlazom (*lab1A_primjer.zip*). Preporučamo provjeru ispravnosti na temelju zadanog primjera prije predaje vježbe na sustav *sprut*. Evaluacija ovog zadatka provodit će se na 5 ulaznih datoteka, uključujući i ovu iz primjera.

1.3 Zadatak B - LSH

Napomena: *Zadatak B nosi 50% bodova prve laboratorijske vježbe*.

U B zadatku format ulazne datoteke je potpuno jednak ulazu datoteke za A zadatak. Vrijede sljedeća ograničenja na ulazne parametre:

N <= 10^5

Q <= 10^5
0 <= I <= N-1 i 0 <= K <= 31

Iz gore navedenih ograničenja ulaznih parametara, očito je da se slični tekstovi za neki zadani tekst neće moći efikasno pronalaziti slijednim pretraživanjem. Potrebno je koristiti algoritam sažimanja LSH koji je opisan u predavanju dostupnom na AVSP_03a (*slide* 30-31).

U prvom koraku potrebno je, kao i u A zadatku, za sve ulazne tekstove stvoriti **128-bitne sažetke** korištenjem **SimHash** algoritma kako je to opisano u odjeljku 1.1. Na predavanjima smo za stvaranje sažetaka koristili algoritam MinHash dok ovdje koristimo brži algoritam SimHash, pritom vrijedi da je veličina sažetka **k = 128**.

U drugom koraku potrebno je na dobivene sažetke primijeniti algoritam LSH. Dakle, osnovna zamisao je sažetke podijeliti u **b pojasa**. U ovoj vježbi ćemo koristiti **b = 8**, što znači da će svaki pojas sadržavati **r = k/b = 128/8 = 16 bita**. Algoritam za svaki pojas sažima cjelobrojnu interpretaciju djela sažetka koji pripada pojasu. Svi parovi sažetaka koji se u barem jednom pojasu sažmu u jednaki pretinac postaju kandidati čiju sličnost treba ispitati. Pritom "*sažimanje u jednaki pretinac u nekom pojasu*" znači da su sažetci u tom pojasu identični. Primjerice, ako imamo dva sažetka veličine k = 4, S_1 = "01**01**" i S_2 = "10**01**" i broj pojasa b = 2 (r = k/b = 2), tada se sažetci S_1 i S_2 u drugom pojasu sažimaju u isti pretinac jer su sažetci u tom pojasu identični. Nakon što se postupak sažimanja sažetaka po svim pojasima završi, za svaki tekst dobivamo pripadajući skup tekstova koji su potencijalni kandidati za sličnost.

Pseudokod LSH algoritma je sljedeći.

```
list simHash = stvoriSimHash(tekstovi);
dictionary kandidati = {} ;
za pojas = 1 do b {
      pretinci = {};
      za trenutni_id = 0 do N - 1 {
            hash = simHash[trenutni_id];
            // hash sadrži 128 bita
            // Uzmi r = 16 bita u trenutnom pojasu
            // počevši od manje bitnih bitova
            // npr. za pojas = 1, uzimaju se bitovi od 0:15
            // za pojas = 2, uzimaju se bitovi od 16:31, itd.
            // korištenjem fje hash2int pretvori tih 16 bita u integer
            val = hash2int(pojas, hash);
            tekstovi_u_pretincu = {};
            ako (pretinci[val] != {}) {
                  tekstovi_u_pretincu = pretinci[val];
                  za svaki tekst_id in tekstovi_u_pretincu {
                        kandidati[trenutni_id].add(tekst_id);
                        kandidati[tekst_id].add(trenutni_id);
                  }
            } inače {
                  tekstovi_u_pretincu = {};
            }
            tekstovi_u_pretincu.add(trenutni_id);
            pretinci[val] = tekstovi_u_pretincu;
      }
}
```

Nakon što se završi algoritam LSH, u mapi `kandidati` za svaki ulazni tekst nalazi se skup tekstova kandidata za sličnost (preciznije, radi se o rednim brojevima tih tekstova s obzirom na poredak u ulaznoj datoteci).

U trećem koraku, program mora, kao i u A zadatku, generirati cijeli broj koji označava ukupan broj dokumenata čiji se sažeci razlikuju do na K bitova (uključujući K) od sažetka I-tog dokumenta (po Hammingovoj udaljenosti). Prilikom generiranja broja program treba koristiti mapu `kandidati`, tj. slične tekstove za I-ti tekst program treba tražiti isključivo u mapi `kandidati` izgrađenoj pomoću LSH algoritma.

Npr. ako je l=16 i K=3 onda program za navedeni upit traži u mapi `kandidati` skup tekstova kandidata pridruženih tekstu koji ima ključ 16, te ispisuje broj tekstova čiji se sažeci razlikuju od sažetka 17-tog (pretpostavlja se *0-based indexing* tekstova) teksta do uključivo 3 bita.

Važne napomene:

- Vremensko ograničenje na izvođenje programa za zadatak B za bilo koju ulaznu definiciju jest 100 sekundi
- Ulazna točka za Java rješenja treba biti u **razredu SimHashBuckets**, a ulazna točka u Python rješenja treba biti u datoteci **SimHashBuckets.py**

1.3.1 Primjer za provjeru valjanosti

Na stranicama predmeta postavljen je primjer ulazne datoteke s pripadajućim očekivanim izlazom (*lab1B_primjer.zip*). Preporučamo provjeru ispravnosti na temelju zadanog primjera prije predaje vježbe na sustav *sprut*. Evaluacija ovog zadatka provodit će se na 5 ulaznih datoteka, uključujući i ovu iz primjera.