Objašnjenja algoritama

Algoritam za rangiranje stranica

U skupu podataka nad kojim se radi rangiranje nalaze se putanje svake stranice kao i broj ponavljanja tražene reči unutar stranice. U grafu se nalaze dva rečnika u vidu polja *incoming* i *outgoing*. Ključ oba rečnika predstavlja putanja stranice dok vrednost čine liste stranica koje upućuju na traženu stranicu, odnosno na koje ona upućuje.
U osnovi algoritam za računanje ranga se oslanja na PageRank algoritam koji rang stranice računa na osnovu linkova koji na nju upućuju pomoću formule:

$PR(A) = 1 - d + d \cdot \sum_{B = incoming(A)} PR(B) \div L(B)$, gde je *PR(X)* trenutna vrednost ranga stranice, *L(X)* broj stranica na koje stranica *X* upućuje, a *d* faktor prigušenja, koji predstavlja verovatnoću da korisnik koji prati linkove u bilo kom trenutku prestane sa pretragom.
Kako bismo algoritam prilagodili potrebama konkretnog problema i specifikaciji zadatka uvedene su određene modifikacije:

- *Uslov da linkovi koji i sami sadrže traženu reč doprinose više nego sama pojava te reči ili link koji ne sadrži traženu reč* je postignut tako što su sabirci pomnoženi određenim koeficijentima u slučajevima da sadrže, odnosno ne sadrže datu reč, u ovom slučaju su te vrednosti 1,2 i 0,8 respektivno. Do ovih brojki se došlo isključivo empirijskim putem.
- Samim tim i *linkovi koji ne sadrže zadatu reč imaju uticaj na jačinu ranga*.
- *Broj ponavljanja reči koja se pretražuje* direktno utiče na “jačinu” linka dodavanjem sabirka koji predstavlja odnos broja ponavljanja date reči i prosečan broj ponavljanja date reči nad skupom rezultata, tj. linkovi čija bi vrednost bila 0 se ne uzimaju u obzir.
- Ukoliko se radi o *pretrazi na osnovu više reči* od kojih svaki rezultat pretrage sadrži bar jednu od zadatih reči, rang se dodatno “pojačava” množenjem sa brojem reči koje se u njemu pojavljuju, npr. ukoliko stranica sadrži 2 reči od ukupno 3 koje se pretražuju njen rang se dodatno množi brojem 2 čime joj se daje dodatna dodatna prednost u odnosu na stranice koje sadrže samo jednu od zadatih reči. Ovaj deo algoritma se primenjuje van funkcije i ne odnosi se i na naprednu pretragu kod koje bi se, usled pojavljivanja drugih operacija i davanja prednosti, algoritam znatno zakomplikovao.
- Sami rangovi su na kraju prikazani procentualno.

Napomena: Prilikom osnovnih operacija nad skupovima, konkretno unije i preseka, vrednosti koje predstavljaju broj ponavljanja reči u HTML stranici se sabiraju.

Algoritam za sortiranje: HeapSort O(nlogn)

Prilikom izbora algoritma za sortiranje uzeli smo u obzir njegovu vremensku kompleksnost ali i performanse u odnosu na veličinu podataka. Kako se konkretan problem odnosi na relativno male sekvence (u test primeru je bilo uključeno sve ukupno 508 HTML stranica) nije bilo potrebe uvoditi algoritam poput BucketSort-a. Takođe smo izveli nekoliko testova sa različitim

algoritmima (BubbleSort, MergeSort, QuickSort, HeapSort…) prilikom kojih se HeapSort pokazao čak do 10 puta brži od ostalih, iako su razlike bile gotovo neprimetne za samog čoveka.

Strukture podataka

Trie:

Struktura: Svaki čvor sadrži Dictionary koji sadrži decu (key = char, value = node), takođe sadrži indikator (True/False) da li je reč završena kao i skup fajlova koji sadrže traženu reč.
Vremenska kompleksnost metoda:
add(word, file) -> O(duzina reči)
search(word) -> O(duzina reči)

Graph:

Struktura: Čvorove grafa predstavljaju putanje stranica, dok su ivice predstavljene pomoću rečnika *incoming* i *outgoing* koji su već pomenuti kod algoritma za rangiranje.
Vremenska kompleksnost metoda:
insert_vertex(filename) -> O(1)
insert_edge(origin, destination) -> O(max(len(outgoing(origin)), len(incoming(destination)))

Set

Struktura: Elementi skupa su ujedno i ključevi rečnika koji omogućuje direktan pristup. Elementima se dodeljuje i dodatna vrednost koja predstavlja broj ponavljanja tražene reči u stranici.
Vremenska kompleksnost metoda:
add(element, value) -> O(1)
Unija *__or__(self, other)* -> O(n1 + n2) gde je n1 broj elemenata prvog skupa a n2 drugog
Presek *__and__(self, other)* -> O(min(n1, n2))
Razlika *__sub__(self, other)* -> O(n1)