Student1 – Nikola Kolovic RA133/2017

Trie stablo

Trie stablo je struktura podataka koja je pogodna za cuvanje stringova. (primer: cuvanje svih reci iz teksta). Za potrebe programa Search Engine koristili smo standardni trie. Standardni trie ima osobinu da svaki cvor osim korena sadrzi jedan karakter i putanja od korena do lista daje jedan string.

Standardni trie trosi O(n) prostora gde n predstavlja ukupnu duzinu stringova.  
Za dodavanje i pretragu elemenata trosi O(dm) vremena, m- duzina stringa, d- velicina alfabeta

t

y

a

e

TrieNode se sastoji od elemenata (char(karakter), children(niz dece za svaki cvor), endOfWord(Boolean koji cuva da li se u datom cvoru zavrsava rec) i polje stranice koje predstavlja set html linkova koje sadrze datu rec (polje stranice je neprazno samo ukoliko je endOfWord=True)

Dodavanje elemenata u trie:

add\_word(word,file)  
**Input:** word: rec koja se unosi  
**Input:** file: fajl u kojem je pronadjena

1. Za trenutni cvor postavljamo root cvor
2. Prolazimo kroz sve karaktere koji se nalaze u reci
3. Ako se karakter ne nalazi u polju children tekuceg cvora
   1. U children trenutnog cvora dodajemo cvor sa tekucim karakterom
4. Za trenutni cvor postavljamo cvor koji sadrzi dati karakter u children
5. Ukoliko nismo prosli kroz sve karaktere u reci vracamo se na korak 2
6. U suprotnom, trenutnom cvoru postavljamo vrednost endOfFile na true
7. U polju stranice dodajemo u skup dati html file

Pretraga elemenata u trie:

search(word)  
**Input:** word – rec koja se pretrazuje

1. Za trenutni cvor postavljamo root cvor
2. Prolazimo kroz sve karaktere koji se nalaze u reci
3. Ako se karakter ne nalazi u polju children tekuceg cvora
4. Rec nije pronadjena, vracamo string “False” i None
5. Za trenutni cvor postavljamo cvor koji sadrzi dati karakter u children
6. Ukoliko nismo prosli kroz sve karaktere u reci, vracamo se na korak 2
7. U suprotnom, rec je pronadjena, vracamo “True” i polje stranice

Parsiranje i validacija unosa obicne pretrage

Parsiranje obicne pretrage:

Parsiranje obicne pretrage parsira uneti kriterijum tako sto pomocu funckije .strip() otklanja   
 sve praznine ispred prvog karaktera i iza poslednjeg. Zatim zamenjuje sve tabove sa praznim   
 stringom(brise unete tabove sa tastature) pomocu funkcije .replace(‘\t’,’’). Zatim kreira novi   
 niz kriterijuma pomocu komande re.split(‘ ‘, kriteterijum) tako sto splituje string po razmaku   
 i stavlja u niz kriterijuma. U slucaju da je korisnik uneo vise razmaka jedan pored drugog u   
 nizu kriterijuma ce se pojaviti razmak, koji moramo ukloniti da bi niz kriterijuma sadrzao   
 samo operatore i reci koje pretrazujemo.

Primer: **input**: “ python and java “  
 **output**: [“python”,”and”,”java”]

Validacija unosa obicne pretrage:

Validacija unosa pretrage sluzi da proveri da li je kriterijum u pravilnom formatu. Za potrebe   
ovog programa uzeli smo da je skup operatora (**OR,AND,NOT, “ “** prazan string ce gledati kao OR operator).   
**Pravilan format unosa:** Uslov1 OPERATOR Uslov2 | NOT Uslov1 | Uslov1 Uslov2 … UslovN  
U prvom slucaju izmedju 2 uslova mora se nalaziti operator, u drugom slucaju se radi negacija uslova tako sto nakon operatora “NOT” ide uslov, treci slucaj je kada unosimo N uslova izmedju kojih se nalazi razmak koji se ponasa kao operator “OR”  
U slucaju kada je ispostovan format unosa funkcija vraca True, u suprotnom False.

Obicna pretraga

slozenijaPretraga(kriterijum, operacija)  
**input**: kriterijum – niz kriterijuma pretrage  
**input**: operacija – string koji pokazuje koju operaciju izvrsiti nad nizom kriterijuma(“OR”,”AND”,”NOT”,”KOMPLEMENT”)

**Algoritam:**

1.Pomocu if-elif proveravamo koji je operator pretrage i ulazimo u odgovorajuci blok  
 2.Ukoliko je operacija “OR”  
 a. Iteriramo kroz sve uslove u kriterijumu  
 b. Pozivamo pretrazivanje iz globalnog trie stable  
 c. Ukoliko je rec iz uslova pronadjena u RESULT\_SKUP dodajemo skup html stranica  
 u kojima se ta rec nalazi  
 d. Ukoliko nismo dosli do kraja niza kriterijuma vracamo se na korak a), u suprotnom  
 e. Prolazimo kroz sve skupova HTML dokumenata koji ispunjavaju uslov i pomocu   
 operacije | ugradjene u Set odredjujemo RESULT\_SET  
 3.Ukoliko je operacija “AND”   
 a. Iteriramo kroz sve uslove u kriterijumu  
 b. Pozivamo pretrazivanje iz globalnog trie stable  
 c. Ukoliko je rec iz uslova pronadjena  
 c1. Ukoliko je kriterijum prvi, u RESULT\_SET stavljamo pretrazeni skup   
 stranica da ne bi radili operaciju & sa praznim skupom  
 c2. Ukoliko je kriterijum drugi, njega smestamo u drugu promenljivu  
 d. Kada prodjemo kroz sve uslove, pomocu operacije & ugradjene u Set odredjujemo   
 RESULT\_SET  
 3.Ukoliko je operacija “NOT”  
 a. Ponavljamo korake a), b),c) iz koraka 3.  
 b. Pomocu operacije – ugradjene u Set odredjujemo RESULT\_SET  
 4.Ukoliko je operacija “KOMPLEMENT”  
 a. Pretrazujemo trie za zadati kriterijum  
 b. Ako je za dati kriterijum pronadjen skup html fajlova taj skup stavljamo kao  
 rezultujuci  
 c. U odnosu na taj rezultujuci skup radimo komplement pomocu funkcije u Set-u   
 i dobijamo trazeni RESULT\_SET

Napredna pretraga(#parser)

Parsiranje za naprednu pretragu je napravljeno tako sto se kriterijum pretrage prebacuje iz infiksne u postfiksnu notaciju, zatim iz postfiksne notacije u stablo parsiranja. Stablo parsiranja je binarno stablo.

Prebacivanje kriterijuma iz infiksne u postfiksnu notaciju(Algoritam)  
  
Uvodimo mapu prioriteta(kljuc je operator a vrednost je prioritet), “!” ima najveci prioritet 4, “&&” ima prioritet 3, “||” ima prioritet 2 i “(“ ima prioritet 1.  
Takodje uvodimo brojac koji ce brojati uzastopno pojavljivanje obicne reci, to znaci da ukoliko imamo vrednost brojaca vecu od 0 znaci da se kriterijum pojavio uzastupno 2 ili vise puta a da izmedju nije bilo operatora (u specifikaciji se podrazumeva “||”).

1.Iteriramo od prvog do poslednjeg stringa u kriterijumu  
 2.Ukoliko naidjemo na levu zagradu stavimo je na stek i resetujemo brojac na 0  
 3.Ukoliko naidjemo na desnu zagradu sve dok se na vrhu steka ne naidje na levu zagradu   
 skidamo element sa vrha steka i stavljamo u rezultujuci niz i resetujemo brojac na 0  
 4.U slucaju da naidjemo na operatore “!”,”||” ili “&&” sve dok stek nije prazan i prioritet   
 vrha steka je veci ili jednak prioritetu trenutnog operatora u rezultujuci niz stavljamo   
 vrednost sa vrha steka, nakon cega stavljamo operator u rezultujuci niz i resetujemo brojac   
 5.Ukoliko naidjemo na obicnu rec(kriterijum), kriterijum pretrage ubacujemo u rezultujuci  
 niz, takodje proveravamo ako je brojac veci od 0 dodajemo u rezultujuci niz i “||”, nakon   
 cega uvecavamo brojac  
 6.Kada prodjemo kroz sve kriterijume u rezultujuci niz ubacujemo sve preostale elemente   
 sa steka

Kreiranje binarnog stabla (stablo parsiranja) – Algoritam

1.Iteriramo kroz elemente niza kriterijuma koji je u postfiksnoj notaciji  
 2.Ukoliko element nije operator kreiramo cvor stable i stavljamo na stek  
 3.U suprotnom, ukoliko je element operator  
 a. Ako je u pitanju “!” kreiramo cvor sa vrednoscu “!” a za levo dete stavljamo  
 element sa vrha steka, desno dete ostaje prazno(None)  
 b. Ako je u pitanju neki drugi operator kreiramo cvor sa vrednoscu tog operatora  
 i kao levo i desno dete postavljamo prve dve vrednosti sa vrha steka  
 c. Na stek stavljamo novokreirani cvor  
 4.Kada se zavrsi iteriranje kroz elemente na vrhu steka se nalazi root cvor

**Dusan Petrovic**

**RA122-2017**

**Student 2**

# Graph

* Implementirana kao 2 recnika, outgoing - izlazne stranice, incoming - ulazne stranice
* Vrednosti recnika su liste stranica (ulaznih/izlaznih)
* Metoda koja se koristi nakon parsiranja add\_from\_html sluzi za dodavanje stranice i njenih linkova
* Na pocetku se dodaje cvor (to je moguce zato sto su putanje do HTML dokumenata jedinstvene)
* Nakon toga svaki njegov link dodajemo u listu sa odgovarajucim kljucem u mapi izlaznih grana #add\_edge
* Zatim inicijalizujemo listu u recniku ulaznih grana za zadati link ako je to potrebno i na kraju dodajemo posmatranu stranicu kao ulaznu za posmatrani link #add\_edge

# Skup(#set)

* Implementirano kao recnik ciji je kljuc putanja HTML stranice a vrednost broj pojavljivanja date reci
* **OR** - implementirano kao unija, odnosno sve stranice koje se ne nalaze u posmatranom skupu a nalaze se u prosledjenom dodaju se u povratnu vrednost
* **AND** - vracaju se samo one stranice koje se pojavljuju i u posmatranom skupu i u prosledjenom skupu
* **KOMPLEMENT** (NOT) - na osnovu pocetno odredjenog NADSKUPA (pri parsiranju direktorijuma), u kom se nalaze sve stranice, vracaju se one stranice iz NADSKUPA koje se ne nalaze u posmatranom skupu

# Rangiranje

-Podeljeno u 3 dela

1. rangiranje na osnovu broja trazenih reci u stranicama
2. rangiranje na osnovu broja trazenih reci u linkovanim stranicama
3. rangiranje na osnovu broja linkova ka posmatranoj stranici

1. rangiranje na osnovu broja trazenih reci u stranicama

* Kreiramo recnik sa kljucem - Putanja HTML stranice cija je vrednost - Rang
* Inicijalizujemo rang svake stranice iz RESULT\_SET-a na 0
* Zatim prolazimo kroz parsirani kriterijum pretrage (kroz sve reci) i trazimo u TRIE stablu Skup svih stranica koje sadrze rec
* Rang stranice predstavlja vrednost iz Skupa ciji je kljuc putanja posmatrane stranice. Vrednost je broj pojavljivanja reci.
* Na kraju svaki rang mnozimo sa 0.5 (svaka reci daje +0.5 rangu) i vracamo prethodno izmenjeni recnik

2. rangiranje na osnovu broja trazenih reci u linkovanim stranicama

* Rangiranje na osnovu broja reci u linkovanim stranicama.
* Korisnik unosi koliko linkovanih stranica zeli da posmatra i na osnovu toga se rekurzivno poziva algoritam za rangiranje
* Posto smo u prethodnom koraku odredili rang na osnovu broja reci, to sada koristimo za drugo rangiranje
* Prolazi se kroz stranice iz dobijenog Recnika, i algoritmu se prosledjuje
  + Recnik,
  + Lista ulaznih cvorova (inocming) za posmatranu stranicu
  + Mnozilac (0.3 u mom slucaju) - koji utice na rangiranje, svaka pojava reci u linkovanim stranicama daje mnozilac\*0.5\*broj reci Rangu, i sluzi za odredjivanje baznog slucaja, i
  + Graf
* Prolazimo kroz ulazne cvorove, i vrsi se provera da li je mnozilac dovoljno mali, na osnovu globalVar.n
  + Ako je manji, dobijamo najdalji cvor koji posmatramo i sabiramo sve rangove na tom nivou
  + Ako mnozilac nije manji opet pozivamo funkciju za drugo rangiranje s tim da mnozilac delimo sa 3 (dalji cvor manje utice na rang)
* Vracanjem iz funkcije proverava se da li smo dosli do kraja liste ulaznih cvorova.
* Ako jesmo dodajemo trenutni rang stranice na globalVar.zbir\_rangiranja - koja cuva rang, i mnozimo globalVar.n sa 3 da bismo mogli da realizujemo bazni slucaj i da saberemo rang i za cvorove koje su blize od najdaljeg koji se posmatra
* Na kraju se vraca zbir\*mnozilac
* Vracanjem u pozivajucu petlju, dodajemo povratnu vrednost funkcije (bice 0) i dodajemo globalVar.zbir\_rangiranja (trazeni rang)

3. rangiranje na osnovu broja linkova ka posmatranoj stranici

* Trece rangiranje na osnovu broja linkova
* Za svaku stranicu prolazimo kroz njenu listu ulaznih grana
* Svaka stranica koja se nalazi u RESULT\_SETU daje +1 njenom rangu (svaki link koji sadrzi trazenu rec rang += 1 ili koji ne sadrzi (u slucaju sa NOT)
* Svaka stranica koja se NE nalzi u RESULT\_SETU daje +0.3 njenom rangu (svaki link koji ne sadrzi trazenu rec rang += 0.3 ili je sadrzi (u slucaju sa NOT)

# Paginacija

-Inicijalno se prikazuje 10 stranica, po defaultu, i svaki pogresan unos pri promeni broja stranica vraca broj prikazanih stranica na 10

-Postoje opcije za:

1. prikaz sledecih N stranica
2. prethodnih N stranica
3. za promenu broja prikazanih stranica

U slucaju kada se dolazi do kraja, ispisuju se preostale stranice

# Sortiranje

Izabrani algoritam **Merge sort**.

vremenska kompleksnost algoritma - **O(n\*logn)**

Algoritam:

* -Trazimo polovinu prosledjene liste i delimo je na pola sve dok ne dodjemo do lista, odnosno kada vise ne mozemo da je podelimo
* -Poredimo leve i desne delove i u svakom koraku od leve i desne strane dobijamo jedan uredjen niz
* **PRIMER**: 4 3 2 5 9 6 1 (delimo na 2 dela)
* - levi: 4 3 2 <- dalje delimo desni: 5 9 6 1
* L: 4 <- ne moze dalje D: 3 2 <-on se deli
  + - L: 3 D: 2
  + vracamo uredjen niz: 3,2
* L: 4 D: 3,2
  + vracamo uredjen niz 4,3,2
* L: 4,3,2 D:5 9 6 1 <- sada se on deli
  + L:5 9 <-dalje delimo D: 6 1
  + L: 5 D: 9
* vracamo uredjen niz: 9,5
* L:9,5 D: 6 1 <- sada se on deli
  + - L: 6 D: 1
    - vracamo uredjen niz: 6,1
* L: 9,5 D: 6,1
* vracamo uredjen niz: 9,6,5,1
* L: 4,3,2 D: 9,6,5,1
* konacno, vraca se niz: 9,6,5,4,3,2,1

# Evaluacija stabla parsiranja

Vremenska kompleksnost algoritma : **O(n)**

**Algoritam**:

If t nije None then

If t = list then

Return t.vrednost # Skup stranica dobijen pretragom Trie

Else

A = evaluacija(t.left)

B = evaluacija(t.right)

return A operator B

gde je operator vrednost iz t