Struktura podataka TRIE

Andreja Ilić
e-mail: ilic_andrejko@yahoo.com
Prirodno Matematički Fakultet u Nišu

Struktura *Trie* predstavlja jednu od lakših struktura podataka namenjenih indeksiranju i pretrazi stringova. Većina struktura ovog tipa nisu podržane od strane standardnih biblioteka.

Neka *word* predstavlja konkretan strig a sa *dinctionary* označimo skup različitih reči. Želimo da konstruišemo takvu 'strukturu' kojoj ćemo moći brzo ispitati da li odredjena reč pripada skupu ili ne. Postoje dosta standardnih struktura podataka koje podržavaju dati upit (npr. *set*, *HashSet* ...). Medjutim strukturu *Trie* karakterišu dve osobine koje navedene strukture nemaju:

- Ubacivanje nove reči kao i ispitivanje pripadnosti je složenosti $O(|word|)$ (dakle ne zavisi od veličine rečnika)
- Pored same činjenice da li reč pripada skupu ili ne, iz strukture *Trie* možemo pronaći: reči čije je *edit-distance* rastojanje 1 (reči dobijene zamenom, brisanjem ili dodavanjem jednog karaktera); reči koje počinju na dati prefiks...

Naziv *Trie* potiče iz reči *retrival* što znači pretraživanje. *Edward Fredkin* je 1960. godine izdao rad pod nazivom *'Trie Memory'* u kome je opisao ovu strukturu. Ideja koja se krije iza samog naziva, kako je on rekao, se odnosi na povezivanje dve osnovne karakteristike strukture: oblik stabla i funkcija pretrage.

Kao što smo već napomenuli, kostur strukture će biti predstavljen korenskim stablo. Glavna ideja strkture jeste da sam čvor ne sadrži u potpunosti infromaciju, već se ona nalazi na putu od njega do korena stabla. Dakle, za razliku od klasičnih strukutra koje se reprezentuju stablom, svaki čvor će kao podatak čuvati samo jedan karakter, dok se sama reč koju on reprezentuje dobija konkateniranjem karaktera na putu od korena do posmatranog čvora.

Strukturu možemo definisati na sledeći način:

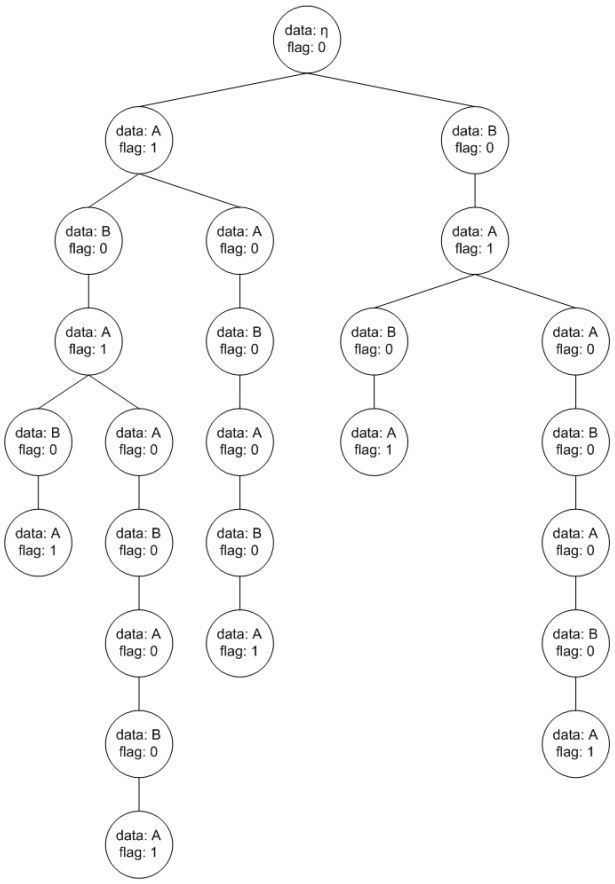
Data je skup W različitih reči definisanih nad konačnom azbukom $\Sigma \cup \eta$, gde η označava prazan karakter. Korensko stablo T koje zadovoljava sledeće uslove:

- svaki čvor stabla sadrži dva podatka: karakter *data* i binarnu vrednost $flag$
- podaci korena stabla su $data = \eta$ i $flag = 0$
- za svaki čvor stabla reč koja je dobijena konkateniranjem karaktera *data* sa čvorova na put od korena do njega, predstavlja prefiks neke reči iz skupa W. Prefiks će predstavljati celu reč akko je vrednost $flag$-a u čvoru jednak 1
- za svaku reč $w \in W$ i svaki prefiks p reči w postoji tačno jedan čvor čija je reč dobijena gore opisanim konkateniranjem upravo prefiks p
- čvor može imati najviše jednog direktnog potomka za dato $\varepsilon \in \Sigma$

naziva se $Trie$ stablo nad rečnikom W.

Iako formalno deluje komplikovano, konstrukcija stabla je vrlo jednostavna. Pre nego što počemo sa implementacijom, razmotrićemo $Trie$ stablo dobijeno nad sufiksima 6-te Fibonačijeve reči[1] *abaababa*. Dakle imamo da je

$$W = \{a, ba, aba, baba, ababa, aababa, baababa, abaababa\}$$



Slika 1: Trie nad sufiksima 6-te Fibonačijeve reči

Označimo sa $word(v)$, gde je v čvor stabla, reč dobijenu konkateniranjem karaktera sa puta od korena stabla do samog čvora. Posmatrajmo sada proizvoljan čvor v stabla, koji je različit od korena. Tada imamo da $word(v)$ predstavlja prefiks neke reči iz skupa W. Tačnije broj reči koje imaju $word(v)$ kao prefiks jednak je broj čvorova iz podstabla čvora v (podstabla čiji je koren v) kod kojih je $flag$ jednak 1. Takodje vidimo da su svi listovi imaju postavljen $flag$ bit, jer oni sigurno predstavljaju kraj nekih reči iz rečnika W.

Implementacija

U ovom delu ćemo izložiti implementaciju osnovne verzije strukture $Trie$, kojom ispitujemo egzaktno poklapanje. Algoritam će biti predstavljen pseudo kodom. Naime, strukturu opisujemo pomoću tri osnovne funkcije:

- *AddWord (string word)* - metoda za dodavanje nove reči *word* u kolekciju
- *Contains (string Word)* - metoda za ispitivanje da li reč *word* postoji u kolekciji

[1]Fibonačijeve reči se dobijaju rekurzivnim algoritmom sličnim samoj konstrkciji Fibonačijevih brojeva. Definišemo ih kao: $Fib_1 = a$, $Fib_2 = b$, dok je $Fib_n = Fib_{n-1}Fib_{n-2}$ za $n > 2$. Fibonačijeve reči su interesantne jer imaju veliki broj karakteristika periodničnosti i ponavljanja.

- *Count (string prefix)* - funkcija za računanje broja reči iz kolekcije koje sadrže $prefix$ kao prefiks

Na početku postavljamo da je koren stabla *root* jedini čvor u stablu, pri čemu naravno postavljamo njegove atribute na polazne vrednosti. Dodavanje nove reči se izvršava na sledeći način: krećemo od korena stabla i 'silazimo' u potomke koji za *data* sadrže odgovarajući karakter. Ukoliko završimo u čvoru koji nema takvog potomka, kreiramo novi čvor sa datim *data* i postavljamo ga kao potomka čvora u kome se nalazimo. Čvoru u kome smo završili postavljamo $flag$ na 1.

Funkcija za dodavanje nove reči u kolekciju

Input: Reč koju dodajemo: *word*

```
1  currentNode = root;
2  for k ← 1 to length(word) do
3      if currentNode nema potomka sa data = word[k] then
4          kreiramo novi čvor child;
5          child.data = word[k];
6          child.flag = 0;
7          postaviti currentNode za roditelja čvora child;
8      end
9      currentNode = potomak čvora currentNode sa data = word[k];
10 end
11 currentNode.flag = 1;
```

Ispitivanje pripadnosti reči u kolekciju je jako slično dodavanju nove. Krećemo se po stablu počev od korena. Ukoliko dati potomak ne postoji, reč ne pripada kolekciju. U slučaju da samo završili obilazak svih karaktera reči, jednostavno treba ispitati da li je $flag$ u čvoru u kome smo završili postavljen ili ne.

Funkcija za ispitivanje da li data reč pripada kolekciji

Input: Reč koju ispitujemo: *word*
Output: $true/falase$: da li reč pripada kolekciji ili ne

```
1  currentNode = root;
2  for k ← 1 to length(word) do
3      if currentNode nema potomka sa data = word[k] then
4          return false;
5      end
6      currentNode = potomak čvora currentNode sa data = word[k];
7  end
8  if currentNode.flag = 0 then
9      return false;
10 end
11 return true;
```

Na kraju razmotrimo kako se izvršava funkcija *Count (string prefix)* . Na početku ćemo se kretati po stablu preko karaktera iz prefiksa. Ukoliko je to nemoguće, vraćamo 0 kao rezultat. U suprotnom smo završili u nekom čvoru stabla v. Rezultat upita predstavlja broj čvorova u podstablu sa korenom v koji imaju postavljen bit $flag$. Ovo možemo prebrojati bilo kojom metodom pretrage grafa. Implementacija se ostavlja čitaocu za vežbu.