Konkurentni autocomplete

C program za Linux sistem koji korisniku obezbeđuje autocomplete funkcionalnost. Korisnik unosi proizvoljan prefiks, i na ekranu dobija reči koje počinju sa tim prefiksom. Potencijalne reči se čitaju iz tekstualnih datoteka. Reči u memoriji se skladište u trie strukturi, koja podržava konkurentno ubacivanje reči, kao i konkurentnu pretragu.

Priloženi su podaci koji mogu da se koriste za testiranje, u okviru arhive data.zip.

Interakcija sa korisnikom

Korisnik sistemu može da zadaje komande preko komandne linije. Podržane su sledeće komande:

1. `_add_ <dir>`
2. `_stop_`

Komanda `_add_` dodaje novi direktorijum sa tekstualnim datotekama u sistem. Pri izvršavanju ove komande kreira se nova scanner nit koja će da skenira taj direktorijum. Naziv direktorijuma se daje bez poslednje kose crte.

Komanda `_stop_` gasi aplikaciju.

Ako korisnik na konzoli otkuca bilo šta drugo i pritisne enter, podrazumeva se da je poslednja reč uneta na toj liniji prefiks za koji korisnik želi da dobije autocomplete rezultate. Bilo koje reči pre poslednje se ignorišu. Na ekranu se odmah ispisuju sve reči koje su do sada pronađene koje počinju sa datim prefiksom. Ako pretraga još uvek traje, i pronađe se neka nova reč koja počinje sa tim prefiksom, treba je ispisati na ekranu. Korisnik ne može da unosi komande dok se ispisuju ove reči. Korisnik završava trenutnu pretragu kombinacijom tastera CTRL+D. Nakon unosa CTRL+D se zaustavlja ispis i korisnik može da unese novu komandu ili prefiks za pretragu.

Scanner nit

Svaka scanner nit obilazi svoj direktorijum i čita reči iz njega. Nit ignoriše bilo kakve poddirektorijume i čita sve datoteke, pretpostavljajući da one sadrže običan ASCII kodiran tekst. Ova nit prati koje datoteke je već pročitala, kao i kada su te datoteke poslednji put menjane pomoću deljenog niza koji sadrži spisak svih do sada skeniranih datoteka, kao i njihovih poslednjih vremena izmena.

Kada nit naiđe na neku datoteku koju nije već pročitala, ili datoteku koja je menjana od prethodnog čitanja, ona započinje čitanje. Nit čita reč po reč iz datoteke. Reči su razdvojene

znakovima SPACE (‘ ‘), TAB (‘\t’) ili ENTER (‘\n’). Ako reč sadrži bilo kakve karaktere osim malih ili velikih slova engleske abecede, cela se ignoriše. Reči imaju maksimalnu dužinu od 63 slova. Za reči duže od toga su zadržana samo prva 63 karaktera.
Reči se ubacuju u trie strukturu koja je deljena između svih scanner niti.

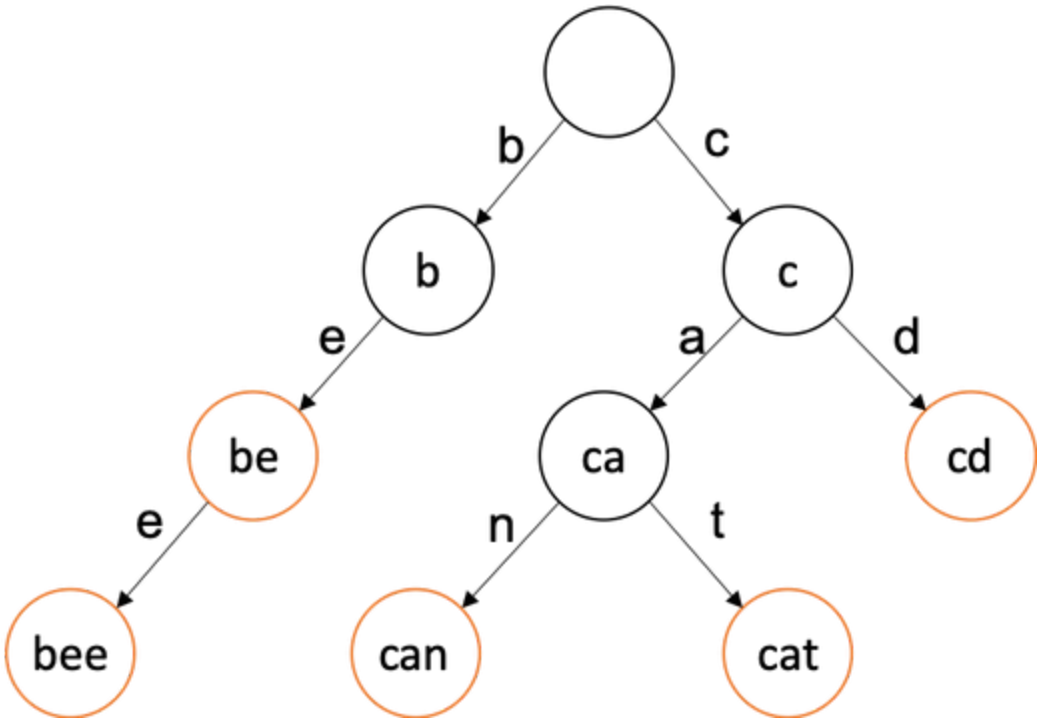
Kada nit obiđe svoj zadati direktorijum, ona pravi pauzu od pet sekundi, nakon čega ponovo počinje obilazak kompletnog direktorijuma. U novom obilasku čita samo nove i izmenjene datoteke. Nit treba da obavlja ovaj posao u beskonačnoj petlji. Ovo znači da sistem treba da detektuje i reaguje na dodavanje nove datoteke, kao i izmenu postojećih datoteka.

Trie struktura

Trie (ili prefiksno stablo) je struktura koja u svakom svom čvoru sadrži:

1. Slovo
2. Potencijalnih 26 pokazivača na decu čvorove
3. Pokazivač na roditelja
4. Flag koji naznačava kraj reči
5. Broj podreči, ne računajući samu sebe

Koren strukture je tipično prazan. Reč u trie strukturi se predstavlja kao niz čvorova od korena do lista, gde svaki naredni čvor u putanji predstavlja jedno slovo reči. Na narednoj slici je prikazan trie u koji su upisane reči: be, bee, can, cat, cd.



Narandžasti čvorovi imaju postavljen flag koji naznačava kraj reči. Primera radi, čvor **b** bi kod naše strukture imao 2 kao broj podreči, dok bi čvor **c** imao 3 kao broj podreči. Prikupljanje "autocomplete" rezultata za neki prefiks se svodi na obilazak čitavog podstabla, počevši od čvora koji predstavlja traženi prefiks. Broj podreči održavamo da bismo na početku tog obilaska mogli da inicijalizujemo strukturu koja će predstavljati rezultat pretrage. Ovaj brojač se osvežava praćenjem roditeljskih veza nakon ubacivanja nove reči kao lista u trie. Bitno je da u rezultatu bude tačno onoliko reči koliko piše u brojaču na čvoru prefiksa. Nije dozvoljeno da trie sam vrši ispis rezultata, već mora da može da ga vrati kao strukturu nekom pozivaocu. Rezultati koji su pronađeni naknadno treba da se prijave pozivaocu pomoću callback funkcije.

Konkurentnost

U postoji N + 1 nit, gde N predstavlja broj izvršenih `_add_` komandi, a jedna dodatna nit je main nit, iz čijeg konteksta se izvršava pretraga deljene trie strukture. Sve ove niti rade sa deljenom trie strukturom uz sledeća ograničenja:

- Nikada ne smeju da se ispisuju reči koje nisu tražene nekim prefiksom.
- U nizu skeniranih datoteka treba da se nalaze sve datoteke koje su do sada skenirane, sa njihovim tačnim poslednjim vremenima modifikacije.
- Nije dozvoljeno zaključavati čitavu trie strukturu.
- Ubacivanje čvorova, kao i pretraga trie strukture treba da mogu da se obavljaju konkurentno. Pritom je dozvoljeno da operacije dodavanja ili pretrage koje se rade nad istim prefiksom bilo koje dužine to rade uz međusobno isključivanje. Npr:
    - Dodavanje "vader" i "khan" treba da se izvršava konkurentno.
    - Dodavanje "vader" i "vargoth" treba da se međusobno isključuje.
    - Pretraživanje rezultata za prefiks "luk" i dodavanje reči "jaina" treba da se izvršava konkurentno.
    - Pretraživanje rezultata za prefiks "luk" i dodavanje reči "lucas" treba da se međusobno isključuje.

Poenta ove vrste međusobnog isključivanja je da se obezbedi što je više moguće konkurentnog rada, dok i dalje imamo garanciju da pretraga neće da bude prekinuta ubacivanjem, te da proizvede neočekivane rezultate, kao i da dva ubacivanja u istom delu stabla neće da se dešavaju istovremeno, i da se na taj način napravi loša struktura stabla.

Funkcije i strukture

```
#define MAX_WORD_LEN 64 //najveca dozvoljena duzina reci, uz \0
#define LETTERS 26 //broj slova u abecedi i broj dece u trie

extern void scanner_init(); //poziva se jednom na pocetku rada sistema
extern void *scanner_work(void *_args); //funkcija scanner niti
```

```
extern void trie_init(); //poziva se jednom na pocetku rada sistema
extern void trie_add_word(char *word); //operacija za dodavanje reci
extern search_result *trie_get_words(char *prefix); //operacija za pretragu
extern void trie_free_result(search_result *result); //rezultat se dinamicki alocira pri pretrazi, tako da treba da postoji funkcija za oslobadjanje tog rezultata
extern void trie_set_current_prefix(char *prefix, void (*callback)(char *word)); //podesavanje trenutno aktuelnog prefiksa i callback funkcije, kako bismo mogli da reagujemo na pronalazak novih reci sa datim prefiksom

typedef struct trie_node //cvor unutar trie strukture
{
     char c; //slovo ovog cvora
     int term; //flag za kraj reci
     int subwords; //broj reci u podstablu, ne racunajuci sebe
     struct trie_node *parent; //pokazivac ka roditelju
     struct trie_node *children[LETTERS]; //deca
} trie_node;

typedef struct search_result //rezultat pretrage
{
     int result_count; //duzina niza
     char **words; //niz stringova, svaki string duzine MAX_WORD_LEN
} search_result;

typedef struct scanned_file //datoteka koju je scanner vec skenirao
{
     char file_name[256]; //naziv datoteke
     time_t mod_time; //vreme poslednje modifikacije datoteke
} scanned_file;
```