Zadanie 2 - Sociálna sieť

Autor: Ondrej Harnúšek

Použité dátové štruktúry

Hashovacia tabuľka - uloženie záznamov o stránkach

- v tabuľke sú uložené pointery na prvý prvok spájaného zoznamu stránok, (na začiatku NULL)
- záznam o stránke obsahuje koreň AVL stromu užívateľov lajkujúcich stránku
- hashkod je generovaný polynomialnou akumuláciou - ktorá je vhodná pre reťazce, ma dobre rozptýlenie
- kolízie sú riešené zreťazením (pri alfa>0.5 menšia časová zložitosť ako otvorené adresovanie)

AVL strom - uloženie záznamov o užívateľoch lajkujucich stránku

- záznam o užívateľovi obsahuje pointer na priamych potomkov v strome, výsku v strome a váhu = počet všetkých potomkov užívateľa v strome
- strom užívateľov si udržuje rotáciami nízku výsku

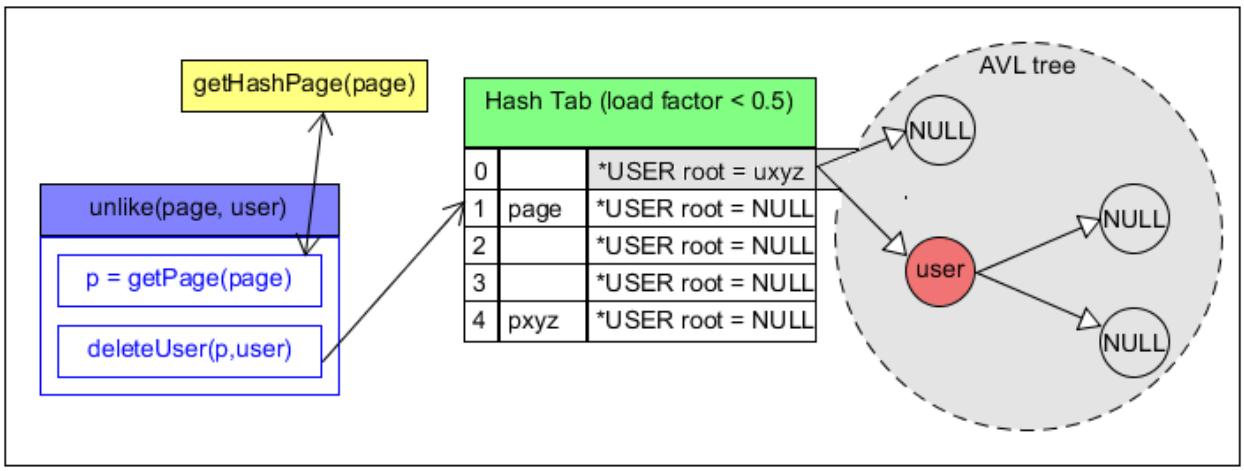

Diagram dátových štruktúr a implementácie funkcie unlike()

Opis použitého algoritmu

- init - inicializácia veľkosti tabuľky (prvočíslo) a alokácia hashovacej tabuľky stránok = pole pointerov na štruktúru stránky
- like-nájde/vytvori stránku v tabuľke a pridá užívateľa do stromu v stránke
- unlike-nájde stránku v tabuľke a vymaže užívateľa zo stromu v stránke
- getuser-pohybuje sa v strome užívatelov podla počtu všetkých potomkov v ľavom a pravom podstrome, počty sú v uložené v priamych potomkoch

-princíp:

0.ak je hľadané k > počet užívateľov v strome vracia NULL 1.ak veľkosť je veľkosť ľavého podstromu +1 > hľadané 'k' idem rekur. do ľavého podstromu 2.ak veľkosť je veľkosť ľavého podstromu +1 < hľadané 'k' idem rekur. do pravého podstromu 3.ak sa rovná 'k', našiel som užívateľa

Odhad priestorovej zložitosti

- veľkosť hashovacej tabuľky je konštantná, ale dĺžka spájaných zoznamov stránok závisí od počtu stránok 'S', takže O(S)
- veľkosť stromu = počet záznamov, zavisí od poctu užívateľov 'u' stránky 'S': O(u).
- ak 'U' = suma počtu užívateľov v stránke cez všetky stránky(t.j. zarátavaných aj opakovane ak sú v inej stránke) tak:
- Celková priestorová zložitosť O(S+U)

Odhad časovej zložitosti

- vyhľadanie v stromoch AVL: O(log n)
- vyhľadanie v tabuľke je pri dobrej hashovacej funkcii a nízkemu faktoru naplnenia : O(1)
- init-vykonáva sa konštantný počet operácii O(1)
- like-vyhľadanie v tabuľke + insert v strome O(log n)
- unlike-vyhľadanie v tabuľke + delete v strome O(log n)
- getuser - vďaka uloženým počtom potomkov nie je nutné prechádzať užívateľmi od 1. po k-teho !{O(k)}.
- Ale zložitosť je podobná ako pri vyhľadávaní v strome O(log n)

Zhodnotenie

Podľa môjho testovania je najväčším problémom implementácie zvolenie správnej veľkosti hashovacej tabuľky. Toto by sa dalo vyriešiť vytvorením premennej alfa=počet_stránok/velkosť_tabuľky a pri alfa>1 rehashovaním celej tabuľky do tabuľky dvojnásobnej veľkosti.

Testovač

1. test - základný

- 2 stranky, kazda lajkuje 20 pouzivatelov
- postupne vykona: 10 like + 5 unlike + 5 getuser + 10 like + 15 getuser
- lajky uklada aj do pomocnej, abecedne utriedenej, pamate
- a s prvkami v nej kontroluje spravnu navratovu hodnotu funkcie getuser

2. test - väčší

- vygeneroval som súbory pages.txt a users.txt s **tisíckami mien** stránok a užívateľov
- \1. Všetci užívatelia lajkujú všetky stránky
- \2. Všetci užívatelia odlajkujú všetky stránky
- \3. Funkcia getuser(1) - t.j prvý v poradím musí vrátit NULL

Protokol

```
File 'pages.txt' with 1000 names was generated!
File 'users.txt' with 1000 names was generated!
Implementacia OK!
execution time : 1.597 s
```

```
File 'pages.txt' with 10000 names was generated!
File 'users.txt' with 1000 names was generated!
Implementacia OK!
execution time : 15.444 s

File 'pages.txt' with 1000 names was generated!
File 'users.txt' with 10000 names was generated!
Implementacia OK!
execution time : 20.274 s

File 'pages.txt' with 5000 names was generated!
File 'users.txt' with 5000 names was generated!
Implementacia OK!
execution time : 45.516 s
```

Zdrojový kód testov: tl;dr

```
int basic_test()  // 2 stranky: 20 pouzivatelov- 10 like + 5 unlike + 5 get + 10 like + 15 get
{
    int i ,j, k;
    char user[5];                          //meno uzivatela
    char page1[] = "Ab", page2[] = "BA";  //mena stranok =rovnaky hash

    // zdrojove udaje - uzivatelia, ktory postupne budu lajkovat stranku
    int size1=20, size2=20;
    int source1[] = {32 ,87 ,66 ,28 ,39 ,86 ,44 ,54 ,24 ,96 ,47 ,92 ,41 ,94 ,17 ,51 ,99 ,34 ,81,
10};
    int source2[] = {24, 63, 31, 25, 85, 34, 99, 42, 36, 78, 32, 71, 35, 55, 15, 74, 23, 44, 65,
30};

    int *likePage1 = (int*) calloc(size1,sizeof(int));//obsahuje zoradeny zoznam uzivatelov
lajkujucich stranku1
    int *likePage2 = (int*) calloc(size2,sizeof(int));//obsahuje zoradeny zoznam uzivatelov
lajkujucich stranku2

    init();

// 10 X like
    for(i=0; i<10; i++)
    {
        //stranka 1
        itoa(source1[i],user,10);
        like(page1,user);     // like stranky
        likePage1[0] = source1[i]; // ulozi like do kontrolnej pamati
        qsort(likePage1,size1,sizeof(int),comp);    // utriedi
        //stranka 2
        itoa(source2[i],user,10);
        like(page2,user);       // like stranky
        likePage2[0] = source2[i]; // ulozi like do kontrolnej pamati
        qsort(likePage2,size2,sizeof(int),comp);    // utriedi
    }
```

```
// 5 X unlike
    for(i=0; i<10; i++)
    {
        j=0;
        if(i%2 == 0)        //parne = unlajkuje stranku 1
        {
            itoa(source1[i],user,10);
            unlike(page1,user);
            while(likePage1[j]!=source1[i])
                j++;
            likePage1[j]=0;
            qsort(likePage1,size1,sizeof(int),comp);    // utriedi
        }
        else                //NEparne = unlajkuje stranku 2
        {
            itoa(source2[i],user,10);
            unlike(page2,user);
            while(likePage2[j]!=source2[i])
                j++;
            likePage2[j]=0;
            qsort(likePage2,size2,sizeof(int),comp);    // utriedi
        }
    }

// 5 X getuser
    for(i=0, k=0; i<size1; i++, k++)    //stranka 1
    {
        if(likePage1[i] == 0)
            k--;
        else
            if(likePage1[i] != atoi(getuser(page1,k+1)))
                return 1;
    }
    for(i=0, k=0; i<size1; i++, k++)    //stranka 2
    {
        if(likePage2[i] == 0)
            k--;
        else
            if(likePage2[i] != atoi(getuser(page2,k+1)))
                return 1;
    }

// 10 X like
    for(i=10; i<20; i++)
    {
        //stranka 1
        itoa(source1[i],user,10);
        like(page1,user);           // like stranky
        likePage1[0] = source1[i];  // ulozi like do kontrolnej pamati
        qsort(likePage1,size1,sizeof(int),comp);    // utriedi
        //stranka 2
        itoa(source2[i],user,10);

        like(page2,user);            // like stranky
```

```
        likePage2[0] = source2[i];   // ulozi like do kontrolnej pamati
        qsort(likePage2,size2,sizeof(int),comp);
    }

// 15 X getuser
    for(i=0, k=0; i<size1; i++, k++)    //stranka 1
    {
        if(likePage1[i] == 0)
            k--;
        else
            if(likePage1[i] != atoi(getuser(page1,k+1)))
                return 1;
    }
    for(i=0, k=0; i<size1; i++, k++)    //stranka 2
    {

        if(likePage2[i] == 0)
            k--;
        else
            if(likePage2[i] != atoi(getuser(page2,k+1)))
                return 1;
    }
//uvolnenie kontrolnych pamati
    free(likePage1);
    free(likePage2);
    return 0;
}
int bigger_test()
{
    int i,j;
    FILE *file_pages = fopen("pages.txt","r");
    FILE *file_users = fopen("users.txt","r");
    char page[35],user[35];
    init();
//like
    for(i=0; fgets(page,35,file_pages)!=0; i++)//all pages
    {
        for(j=0; fgets(user,35,file_users)!=0; j++)//all users
        {
            like(page,user);
        }
        rewind(file_users);
    }
    rewind(file_pages);
//unlike
    for(i=0; fgets(page,35,file_pages)!=0; i++)//all pages
    {
        for(j=0; fgets(user,35,file_users)!=0; j++)//all users
        {
            unlike(page,user);
        }
        rewind(file_users);

    }
```

```
    rewind(file_pages);
//getuser
    for(i=0; fgets(page,35,file_pages)!=0; i++)//all pages
    {
        if(getuser(page,1) != NULL)
            return 1;
    }
//close all
    fclose(file_pages);
    fclose(file_users);
    return 0;
}
```