Slovenská technická univerzita v Bratislave, Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

2. zadanie

Dátové štruktúry a algoritmy

Michal Slovík

INFO-3

V tomto zadaní sme mali za úlohu vytvoriť databázu sociálnych sietí. Ja som ju implementoval pomocou AVL stromov. Presnejšie AVL stromu v AVL strome. Keďže jeden strom slúžil na ukladanie dát o stránke samotnej a vnútorný strom mal údaje o užívateľoch, ktorí jej dali *like*.

Vyvažovanie AVL stromov bolo výhodne aj pri abecednom zoraďovaní a tiež poskytujú dostatočne rýchly prístup ku všetkým svojím vrcholom. Hľadanie, vkladanie a mazanie majú zložitosť O(log n) v priemernom ale aj v najhoršom prípade. Pridávanie a mazanie si môže vyžadovať vyváženie stromu jedno alebo viacerými rotáciami.

Takže okrem bežného zadefinovania AVL mohol každý vrchol obsahovať ďalší strom AVL so všetkými vlastnosťami.

```
Struct page *subname;  // AVL strom v strome
struct page *left;
struct page *right;
```

FUNKCIA LIKE()

Dostane ako parametre meno stránky, a užívateľa ktorý jej dal svoj like, tieto informácie pošlem do ďalšej funkcie `addName(web, helpPage , helpUser);` ktorá postupne ukladá do stromov.

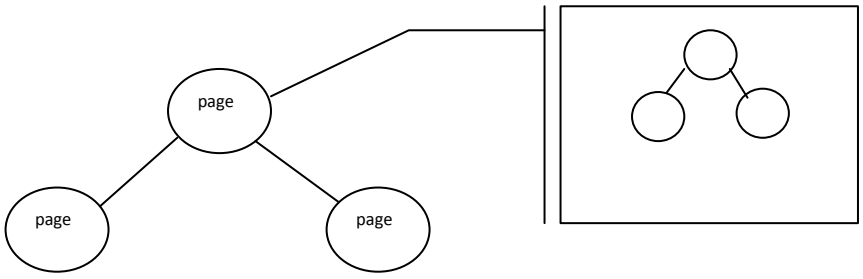
Najskôr vytvorím strom pre stránku ak neexistuje, skontrolujem či nie je potrebné vyvážiť strom po vložení nového prvku, ak nie tak vráti nový strom a do tohto stromu vkladám nový strom s menom človeka, ktorý jej dal like. Takže používam len jednu funkciu insert (`addNameToTree`) na obidve možnosti. Predošlá funkcia určí čo sa bude ukladať a kam ako vidno v kóde:

```
page *addName(page *root, char *nameOfPage, char *nameOfPeople) {

        page *lastNode = addNameToTree(&root,nameOfPage);

        addNameToTree(&lastNode->subname, nameOfPeople);

}
```



FUNKCIA UNLIKE()

Pracuje jednoduchšie ako like, pretože najskôr iba zisťuje či daná stránka existuje a či jej vôbec niekto dal svoj like. Ako náhle existuje, pracujeme len s jej vnútorným stromom, ostatné nie sú v tej chvíli potrebné. A v tomto strome vymazávame podľa pravidiel AVL stromu. To znamená pozeráme či strom nie je prázdny, a potom kde sa nachádza náš vrchol, v ľavom alebo pravom podstrome. Ako náhle ho nájdeme zisťujeme či má nasledovníkov a ak áno na ktorých stranách. Po samotnom odstránení znova kontrolujeme vyváženosť stromu. Už upravené vraciame späť.

FUNKCIA GETUSER()

Podobne ako pri funkcii unlike, najskôr hľadáme stránku, ktorú si vypýtal užívateľ. Potom zistíme koľko ľudí sa nachádza v tomto podstrome. Ak by si užívateľ popýtal číslo mimo rozsah. Tak by funkcia vrátila null.

Potom pomocou rekurzívneho prehľadávania hľadá k-tého človeka. Ako náhle ho nájde vráti jeho meno. Avšak pokračuje ďalej čo nie je úplne výhodne.

DOKÁZANIE ZLOŽITOSTI POMOCOU TESTOVAČA

Bežný AVL strom ma zložitosť O(log n). Ja však mám strom v strome, t.z. že celková zložitosť bude vychádzať $O(\log n)^2$ v priemernom aj v najhoršom prípade, na rozdiel od Binárneho vyváženého stromu kde je by tiež zložitosť bola riešená podobne ale iba v priemernom prípade v najhoršom prípade by to narástlo na n*m, (počet stránok * počet like)

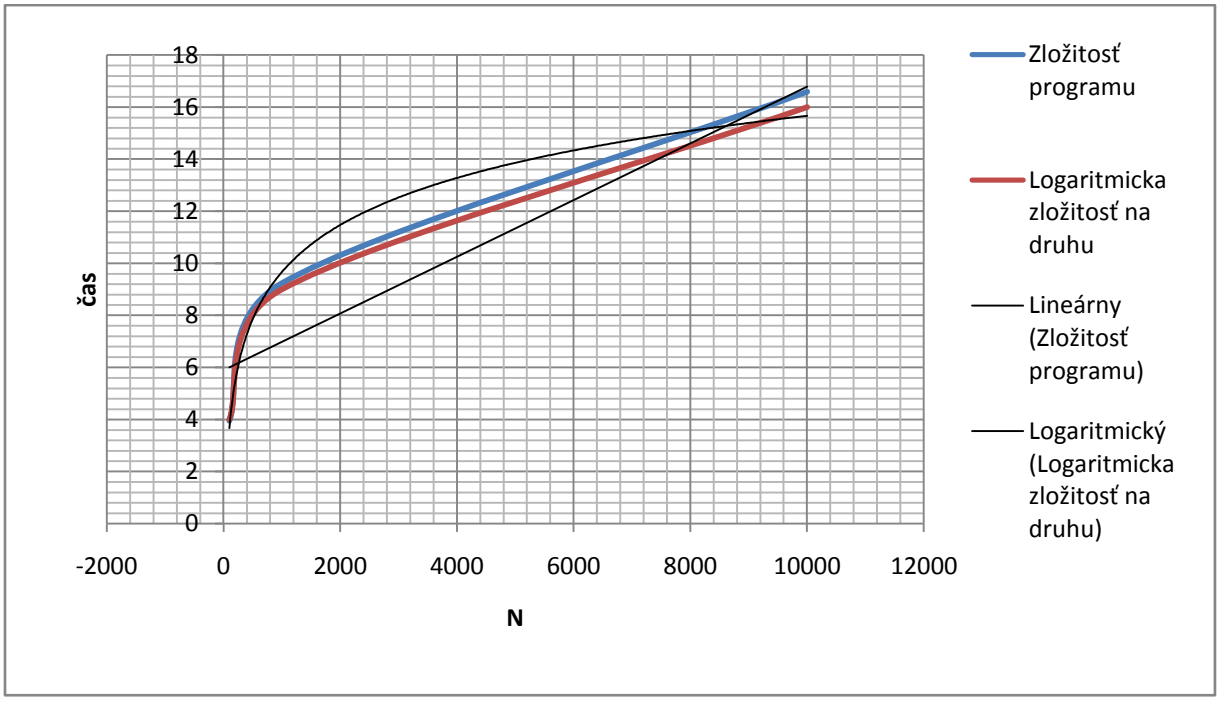
Pomocou testovača som si zisťoval ako dlho beží program pri jednotlivých počtoch stránok, užívateľov.

	100 stranok + 1 like + 1 unlike + 1 getuser	**1000 stránok + 100 like + 10 unlike + 10 getuser**
1.	4,84	8,5
2.	3,73	8,89
3.	3,42	8,88
4	3,2	8,89
5	3,27	9,04
6	3,27	9,2
7	4,57	9,82
8	4,63	8,89
9	4,07	9,02
10	3,49	8,89
	3,949	9,232
	$\text{Log}(100)^2 = 4$	$\text{Log}\ (1000)^2 = 9$

V tabuľke sú časy, za ktoré dokázal program vykonať správne všetky operácie. To znamená, že program, ktorý som testoval pre n == 100, sa zložitosťou približoval hodnote 4, čo je aj hodnota $\log(100)^2$. A rovnako aj pre vyššie n platí, ako vidno na tabuľke.

Priestorová zložitosť je pri AVL stromoch O(n). Kedže vkladáme n užívateľov.

PRE MALÉ N



PRE VEĽKÉ N