*Tirane, me 10.06.2019*

Emri:\_\_\_\_\_Gentjan Ferati\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Grupi II \_A\_ \_2\_

**Ushtrim 1 (5 pike)**

Fjaloret mund te implementohen nepermjet listave te lidhura nje drejtimore, pemeve binare, pemeve te balancuara dhe tabelave hash. Megjithate, jo te gjitha keto struktura kane te njejten performance.

Te shkruhet nje program qe lexon nje skedar text me madhesi te konsiderueshme dhe shfaq vetem njehere te gjitha fjalet e vecante qe shfaqen ne te. Gjate leximit te skedarit, programi duhet te krijoje nje fjalor me te gjitha fjalet qe jane shfaqur ne skedar. Sa here qe shfaqet nje fjale e re, dmth qe nuk ndodhet ne fjalor, programi duhet ta shtoje ate ne fjalor. Fjalori te realizohet:

1. nepermjet listes se lidhur nje drejtimore

2. nepermjet pemes binare

3. nepermjet pemes se balancuar

4. nepermjet tabeles hash.

Per secilen nga strukturat e mesiperme, te matet koha e ezkekutimit per:

1. nje skedar me 100 fjale

2. nje skedar me 1000 fjale

3. nje skedar me 100000 fjale

4. nje skedar me 1000000 fjale

Cila eshte varesia e kohes se ekzekutimit te programit (per secilen nga strukturat) nga madhesia e inputit? Cila nga strukturat rezulton me eficente? Po me pak?

**Pergjigje 1**

[Pershkrimi i zgjidhjes se ndertuar, shpjegime dhe komente argumentuese per zgjidhjen]

[**Kodi i programit**]

Kodet e secilit prej programeve jane bashkengjitur ne GitHub.Com se bashku me skedaret e fjaleve.

Kompleksiteti i algoritmit per:

* rastin e perdorimit te listave te lidhura eshte:

#include <iostream>

#include <fstream>

using namespace std;

struct Node{

string word;

Node\* next;

};

bool existsWord(Node\* head, string w){

Node\* tmp = head;

while(tmp != NULL){

if(tmp->word == w){

return true;

}

tmp = tmp->next;

}

return false;

}

Node\* add(Node\* head, string w) {

if(existsWord(head,w)){

return head;

}

else{

Node\* tmp = new Node;

tmp->word = w;

tmp->next = head;

head = tmp;

return head;

}

}

void printUniqueWords(Node\* head){

Node\* tmp = head;

int i = 1;

while(tmp != NULL){

cout<<i<<" "<<tmp->word<<endl;

tmp = tmp->next;

i++;

}

}

int main(int argc, char \* argv[]) {

ifstream in;

Node\* head = new Node;

int count = 0;

in.open("fjalori.txt");

while(!in.eof()){

string str;

in>>str;

if(count == 0){

head->word = str;

head->next = NULL;

count++;

}

else {

head = add(head,str);

}

}

printUniqueWords(head);

return 0;

}

Kemi perdorur tre funksione qe aksesojne listen. Nje qe shton element ne liste, nje qe kontrollon nese fjala ekzistion ne liste dhe nje funksion qe printon listen duke perdorur nje numer i qe fillon nga 1 per te pare numrin e fjaleve unike. Kompleksiteti i cdo funksioni nuk e kalon kompleksitetin O(n). Pra kompleksiteti eshte O(n).

2. rastin e perdorimit te pemeve binare eshte:

#include <iostream>

#include <fstream>

using namespace std;

int i = 1;

struct Node {

string word;

Node \*left;

Node \*right;

};

bool existsWord(Node \*root, string w) {

if(root==NULL) {

return false;

}

if(root->word==w) {

return true;

}

if(w<root->word) {

return existsWord(root->left,w);

} else {

return existsWord(root->right,w);

}

}

Node \*newNode(string w) {

Node \*temp = new Node;

temp->word = w;

temp->left = NULL;

temp->right = NULL;

return temp;

}

Node \*add(Node \*root, string w) {

if(root==NULL)

return newNode(w);

if(existsWord(root,w))

return root;

if(w<root->word) {

root->left = add(root->left,w);

} else {

root->right = add(root->right,w);

}

return root;

}

void printUniqueWords(Node \*root)

{

if (root != NULL)

{

printUniqueWords(root->left);

cout<<i<<" "<<root->word<<endl;

i++;

printUniqueWords(root->right);

}

}

int main(int argc, char \* argv[]) {

ifstream in;

Node\* head = new Node;

int count = 0;

in.open("fjalori.txt");

while(!in.eof()){

string str;

in>>str;

if(count == 0){

head->word = str;

head->left = NULL;

head->right = NULL;

count++;

}

else {

head = add(head,str);

}

}

printUniqueWords(head);

return 0;

}

Edhe ne pemen binare kemi perdorur tre funksione si ne listat nje drejtimore. Kemi perdorur bredhjen pararendore per kontrollin nese fjala ekzistion si dhe per prinitimin. Kompleksiteti eshte gjithashtu O(n).

3. rastin e perdorimit te pemeve te balancuara eshte:

#include <iostream>

#include <fstream>

using namespace std;

int i = 1;

struct Node {

string word;

Node \*left;

Node \*right;

int height;

};

int max(int x, int y) {

if(x>y)

return x;

return y;

}

int height(Node \*tmp) {

if(tmp==NULL)

return 0;

return tmp->height;

}

bool existsWord(Node \*root, string w) {

if(root==NULL) {

return false;

}

if(root->word==w) {

return true;

}

if(w>root->word) {

return existsWord(root->left,w);

} else {

return existsWord(root->right,w);

}

}

Node \*rightRotate(Node \*y)

{

Node \*x = y->left;

Node \*T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left), height(y->right))+1;

x->height = max(height(x->left), height(x->right))+1;

return x;

}

Node \*leftRotate(Node \*x)

{

Node \*y = x->right;

Node \*T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left), height(x->right))+1;

y->height = max(height(y->left), height(y->right))+1;

return y;

}

int getBalance(Node \*tmp)

{

if (tmp == NULL)

return 0;

return height(tmp->left) - height(tmp->right);

}

Node \*newNode(string w) {

Node \*temp = new Node;

temp->word = w;

temp->left = NULL;

temp->right = NULL;

temp->height = 1;

return temp;

}

Node \*add(Node \*root, string w) {

if(root==NULL)

return newNode(w);

if(existsWord(root,w))

return root;

if(w<root->word) {

root->left = add(root->left,w);

} else {

root->right = add(root->right,w);

}

root->height = 1 + max(height(root->left),height(root->right));

int balance = getBalance(root);

if (balance > 1 && w < root->left->word)

return rightRotate(root);

if (balance < -1 && w > root->right->word)

return leftRotate(root);

if (balance > 1 && w > root->left->word)

{

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && w < root->right->word)

{

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

void printUniqueWords(Node \*root)

{

if (root != NULL)

{

printUniqueWords(root->left);

cout<<i<<" "<<root->word<<endl;

i++;

printUniqueWords(root->right);

}

}

int main(int argc, char \* argv[]) {

ifstream in;

Node\* head = new Node;

int count = 0;

in.open("fjalori.txt");

while(!in.eof()){

string str;

in>>str;

if(count == 0){

head = add(NULL,str);

count++;

}

else {

head = add(head,str);

}

}

in.close();

printUniqueWords(head);

return 0;

}

Ndryshe nga pema e kerkimit binar, pema e balancuar ka dhe funksionet e rrotullimin kur pema behet e pabalancuar kur shtojm elementet e rinj. Ne kete rast kemi kompleksitet O(log2(n)). Por duke marre parasysh edhe hapjen e skedarit edhe rrotullimin kemi kompleksitet O(2log2(n)).

4. rastin e perdorimit te tabelave hash eshte:

#include <iostream>

#include <fstream>

using namespace std;

int c = 0; // Celsi

int j = 1; // Printimi

struct Node {

int key;

string word;

Node \*next;

};

Node \*addElement(Node \*head, int k, string w)

{

Node \* tmp = new Node;

tmp->key = k;

tmp->word = w;

tmp->next = head;

head = tmp;

return head;

}

bool existsWord(Node \*T[], int n, string w) {

for(int i=0;i<n;i++) {

if(T[i]!=NULL && T[i]->word == w) {

return true;

}

}

return false;

}

void printUniqueWords(Node \*T[], int n) {

for(int i=0;i<n;i++) {

if(T[i]!=NULL) {

cout<<j<<" "<<T[i]->word<<endl;

j++;

}

}

}

int main(int argc, char \* argv[]) {

int N = 10000;

Node \*hashTable[N];

for(int i=0;i<N;i++) {

hashTable[i] = NULL;

}

ifstream in;

int count = 0;

int i = 0;

in.open("fjalori.txt");

while(!in.eof() && i<N){

string str;

in>>str;

if(!existsWord(hashTable,N,str)) {

hashTable[i] = addElement(hashTable[i],c,str);

i++;

c++;

}

}

in.close();

printUniqueWords(hashTable,N);

return 0;

}

Ne rastin e tabeles hash kemi perdorur menyren me adresim direkt. Ne kete rast kompleksiteti eshte O(n).

Koha e ekzekutimit per cdo strukutere dhe numrin e fjaleve:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipi | 100 Fjale | 1.000 Fjale | 100.000 Fjale | 1.000.000 Fjale |
| Liste Nje Drejtimore | 0.143 s | 0.342 s | 2.254 s | 16.446 s |
| Peme BST | 0.174 s | 0.328 s | 1.965 s | 4.502 s |
| Peme AVL | 0.129 s | 0.322 s | 3.356 s | 19.464 s |
| Tabele Hash | 0.157 s | 0.313 s | 2.015 s | 9.512 s |

Si perfundim mund te themi se per nje numer te madh fjalesh pema kerkimit binar rezulton si nje nga strukturat qe ka kohen me te vogel te ekzekutimit. Por per nje numer me te vogel fjalesh pema e balancuar jep shpejtesi me te shpejt te ekzekutimit dhe arsyja qe ajo fillon te ngadalsohet me rritjen e numrit te fjaleve eshte sepse ka kompleksitet me te madh se strukturat e tjera.

**Ushtrim 2 (5 pike)**

*Web crawler* eshte lloj software i cili sherben per te zbuluar faqe qe jane online te publikuara. Nje *web crawler* kontrollon faqet e internetit per adresa URL. Nese gjenden te tilla ne permbajtjen e faqeve, atehere faqet e identifikuara nga keto adresa jane te arritshme prej faqes aktuale. *Web crawler* fillon nga me nje liste adresash faqesh, te cilat kontrollohen te parat. Pas tyre, vizitohen faqet adresat e se cilave gjendeshin ne faqet paraardhese. Ky proces mund te vazhdoje deri per nje nivel te caktuar thellesie.

Te ndertohet nje program i cili realizon nje *web crawler* te thjeshte. Programi duhet te perdore *command- line* per te marre 3 argumenta:

* Adresen e faqes prej nga do te filloje kontrolli (psh. [www.fshn.edu.al](http://www.fshn.edu.al/))
* Nivelin e thellesise se kontrollit (psh. nese thellesia eshte 4 atehere duke nisur nga faqja e pare, kontrollohen faqet e gjetura ne faqen e pare (niveli pare), pastaj faqet e gjetura ne faqet e nivelit te pare (niveli dyte), pastaj faqet e gjetura ne faqet e nivelit te dyte (niveli trete) dhe se fundmi faqet e gjetura ne faqet e nivelit te trete.
* Direktorine ku do te ruhen rezultatet e kontrollit (dmth te gjitha adresat e faqeve te arritshme nga faqja me adresen e dhene si argumenti i pare).

Programi duhet te lexoje permbajtjen e faqes me adresen e dhene si argument i pare dhe te kerkoje aty per adresa URL. Keto adresa duhet te shkruhen ne direktorine e dhene si parameter i trete dhe duhet te sherbejne per ciklin tjeter te kontrollit. Ky proces duhet te vazhdoje derisa te arrihet niveli i dhene si argument i dyte.

Programi duhet te shmange ciklet e pafundme, dmth viziten e nje faqeje qe eshte vizituar dhe kontrolluar me pare. Psh. nese faqja [www.fshn.edu.al](http://www.fshn.edu.al/) permban link per tek faqja [www.unitir.edu.al](http://www.unitir.edu.al/) dhe kjo e fundit permban link per tek faqja [www.fshn.edu.al](http://www.fshn.edu.al/) atehere jemi ne kushtet e nje cikli te pafundem.

**Pergjigje 2**

[Pershkrimi i zgjidhjes se ndertuar, shpjegime dhe komente argumentuese per zgjidhjen]

**import** java.io.\*;

**import** java.util.\*;

**import** org.jsoup.\*;

**import** org.jsoup.nodes.\*;

**import** org.jsoup.select.\*;

**public** **class** WebCrawler {

**private** **static** String *path*;

**private** **static** **int** *thellesia\_max*;

**private** **static** HashSet<String> *hs*;

**public** **static** **void** merrFaqet(String url, **int** thellesia) **throws** IOException {

**if**(url==**null** || url=="") {

**return**;

}

**if**( !*hs*.contains(url) && thellesia <= *thellesia\_max*) {

**try** {

*hs*.add(url);

PrintWriter pw = **new** PrintWriter(**new** FileWriter(*path*,**true**));

pw.println("Thellesia: "+thellesia+" | Url: "+url);

pw.close();

Document doc = Jsoup.*connect*(url).get();

Elements linket = doc.select("a[href]");

thellesia += 1;

**for**(Element faqe: linket) {

*merrFaqet*(faqe.attr("abs:href"),thellesia);

}

} **catch** (IOException ex) {

System.***out***.println("Error: " + ex.getMessage());

}

}

}

**public** **static** **void** main(String[] args) **throws** IOException {

**if**(args.length<3) {

System.***out***.println("Mungojn argumentat!");

**return**;

}

String url = args[0];

*thellesia\_max* = Integer.*parseInt*(args[1]);

*path* = args[2];

*hs* = **new** HashSet<String>();

PrintWriter pw = **new** PrintWriter(**new** File(*path*));

pw.close();

*merrFaqet*(url,0);

}

}

Per kete ushtrim kemi perdorur gjuhen e programimit JAVA. Si strukture per mbajtjen e linkeve kemi perdorur HashSet e cila eshte nje strukture e cila nuk lejon te dhenat duplicate. Kemi perdorur Jsoup per te bere parsimin e linkeve ne nje funksion rekursiv i cili vazhdon deri ne momentin kur arrihet thellesia e vendosur nga perdoruesi. Kompleksiteti i ketij programi varet nga thellesia e vendosur nga perdoruesi si dhe nga numri i linkeve qe mund t permbaje secila faqe qe gjendet gjate rruges.