*Tirane, me 10.06.2019*

Emri: Sofia Demetja Grupi II \_\_B2 \_

## 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

***//1.NEPERMJET LISTES SE LIDHUR NJE DREJTIMORE***

#include<conio.h>

#include<iostream>

#include<fstream>

#include<time.h>

using namespace std;

struct Node{

string word; //deklarimi i struktures liste

Node\* next;

};

bool exist (Node\* head,string w){ //funksioni boolean per te kontrolluar nqs eksizton

Node\* tmp=head; // \* shenjon ne koken e listes

while(tmp!=NULL){ //per sa kohe qe lista eshte joBoshe

if(tmp->word==w){ // nqs e kemi gjetur fjalen kthejme true

return true;

}

tmp=tmp->next; } //nqs jo kalojme te pasardhesi

return false; //nqs ende nuk eshte gjetur kthejme false

}

Node\* add(Node\* head,string w){ //vendosja e fjaleve ne liste nga perpara

if(exist(head,w)){ //perdorim funskionin existe te deklaruar me lart qe merr si parameter koken e listes

return head; //dhe w per fjalen dhe kthejme listen nqs ekziston fjala

}

else { //ne te kundertet krijojme nje liste te re

Node\* tmp=new Node; //vendosja e fjaleve ne fillim te listes pasi ka dhe kompleksitet me te ulet dhe

tmp->word=w; // procesim me te shpejte

tmp->next=head;

head=tmp;

return head;

}

}

void print (Node\* head){ //funksioni per te printuar (funksion qe nuk kkthen vlere)

Node\* tmp=head; //shenojme ne koken e listet

int i=1; //indeksim per printimin e fjaleve

while(tmp!=NULL){ //bejme bredhjen ne liste persakohe qe e kemi te ndryshme nga NULL

cout<<i<<" "<<tmp->word<<endl; //printojme fjalet te cdo nyje te listes

tmp=tmp->next; //per te kaluar tek nyja tjeter

i++; //dhe rrisim indeksin sa here e shtojme fjalen ne nyje te listes

}

}

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

clock\_t start, end; //per te llogaritur kompleksitetin e funksioneve te perdorura

start = clock();

ifstream in;

Node\* head=new Node;

int count=0; //integer qe numeron sa fjale kemi te vendosura ne liste

in.open("FJALORI.txt"); //hapim skedarin tekst

while(!in.eof()){ //bejme bredhjen ne skedar endoffile

string str;

in>>str; // vendosja e fjaleve ne skedar

//cout<<str<<endl;

if(count==0){ //nese lista eshte boshe ath fusim fjalen e pare si koke te listes

head->word=str;

head->next=NULL;

count++; //rrisin count sa here fusim nje fjale te re ne listen nje drejtimore

}

else{ //nqs lista nuk eshte boshe bejme therritje e funskionit add i cili i vendos fjalet ne koken e listes

head=add(head,str); //vendosje ne fillim te listes

}

}

print(head); //printimin e listes

end = clock();

double time\_taken = double(end - start) / double(CLOCKS\_PER\_SEC);

cout<<"FJALORI yne ka keto fjale te vendosura si mesiper: "<<endl;

cout<<"KOha e ekzekutimit eshte "<<time\_taken<<"s"<<endl;

getch();

return 0;

}

***//2.NEPERMJET PEMES BINARE***

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

int i = 1;

struct Node { //deklarimi i struktures se pemes binare

string word; //mbajtesi fjales

Node \*left; //shenjues ne te majte

Node \*right; //shenjues ne te djathte

};

bool exist(Node \*root, string w) { //therritja e funksionit exist i tipit boolean

if(root==NULL) { //qe kontrollon nqs rrenja eshte boshe do ktheje false ne te kundert kryhen veprimet me poshte:

return false;

}

if(root->word==w) { //rasti i pare fjala mund te ndodhet ne rrenjen e pemes, nese po kthejme true

return true;

}

if(w<root->word) { //rasti i dyte nese fjala eshte me e vogel ne rend alfabetik, se ajo qe ndodhet ne rrenje

return exist(root->left,w); //therrasim metoden rekursive qe ben kerkimin ne nenpemen e majte

} else {

return exist(root->right,w); //rasti i trete ku behet kerkimi ne nenpemen e djathte

}

}

Node \*newNode(string w) { //funksion per krijimin e nje kulmi te ri

Node \*temp = new Node; //rezervimin ne memorie per kulmin e ri

temp->word = w; //vendosja e fjales

temp->left = NULL; //biri i majte NULL

temp->right = NULL; //biri i djathte NULL

return temp; //kthejme kulmin

}

Node \*add(Node \*root, string w) { //therritja e funskionit add per te vendosur fjalet ne skedar nepermjet pemes binare

if(root==NULL) //nqs pema eshte boshe

return newNode(w); //kthejme nje kulm qe korrespondon me rrenjen , pra funksionin e krijuar me lart per te krijuar nje kulm

if(exist(root,w)) //nqs fjala ndodhet ne rrenje

return root; //kthejme direkt pemen

if(w<root->word) { //nese fjala eshte ne rend alfabetik me te vogel se rrenja

root->left = add(root->left,w); //bejme therritjen rekursive per vendosjen e fjales

} else {

root->right = add(root->right,w); //e kunderta nese eshte ne rend alfabetik me te madh

}

return root;

}

void print(Node \*root) // therritja e funksionit per printimin

{

if (root != NULL) //derisa nuk kemi arritur ne fund te pemes

{

print(root->left); //printojme birin e majte

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

i++; //rrisim i sa here behet nje veprim i tille

print(root->right); //printojme birin e djathte

}

}

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

ifstream in;

Node\* head = new Node;

int count = 0;

in.open("FJALORI.txt"); //hapim skedarin tekst

while(!in.eof()){

string str;

in>>str;

//cout<<str<<endl;

if(count == 0){

head->word = str;

head->left = NULL;

head->right = NULL;

count++;

}

else {

head = add(head,str);

}

}

cout<<"FJALORI yne ka keto fjale te vendosura si meposhte: "<<endl;

print(head);

getchar();

return 0;

}

***//3.NEPERMJET PEMES SE BALANCUAR***

#include <iostream>

#include <string.h>

#include <fstream>

#include <conio.h>

using namespace std;

int i = 1;

struct Node { //deklarimi i struktures se pemes se balancuar

string word;

Node \*left;

Node \*right;

int height; //merr dhe integer lartesine ne pemen e balancuar

};

int max(int x, int y) { // funksioni max kthen gjatesine maximale ndermjet dy gjatesive

if(x>y)

return x; //veprimet perkatese

return y;

}

int height(Node \*tmp) { //kthen gjatesine e nje kulmi

if(tmp==NULL)

return 0;

return tmp->height;

}

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

if(root==NULL) {

return false;

}

if(root->word==w) { //rasti i pare nqs fjala ndodhet ne rrenjen e pemes kthejme true

return true;

}

if(w>root->word) { // rasti i dyte nqs fjalan eshte me a madhe ne rend alfabetik se fjala ne rrenje

return exist(root->left,w); //therritet funskioni exist i deklaruar me lart

} else {

return exist(root->right,w);

}

}

Node \*rightRotate(Node \*y)

{

Node \*x = y->left;

Node \*T2 = x->right;

// Perform rotation

x->right = y;

y->left = T2;

// Update heights

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

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

// Return new root

return x;

}

Node \*leftRotate(Node \*x)

{

Node \*y = x->right;

Node \*T2 = y->left;

// Perform rotation

y->left = x;

x->right = T2;

// Update heights

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

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

// Return new root

return y;

}

int getBalance(Node \*tmp)

{

if (tmp == NULL)

return 0;

return height(tmp->left) - height(tmp->right); //zbritja e dy gjatesive te bijeve te nje kulmi per te gjetur gjatesine e ketij kulmi

}

Node \*newNode(string w) { //krijimi i nje nyje te re me krijim e kulmi dhe bijeve te majte e te djathte

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) //nqs eshte boshe e vendosim newNode

return newNode(w);

if(exist(root,w)) //nese ekziston ne rrenje bejme kontrollin ne te

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)); // kulmi lplus maksimumin midis ketyre te dyjave

int balance = getBalance(root);

// Left left Case

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

return rightRotate(root);

// Right Right Case

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

return leftRotate(root);

// Left Right Case

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

{

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

return rightRotate(root);

}

// Right Left Case

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

{

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

return leftRotate(root);

}

return root;

}

void print(Node \*root)

{

if (root != NULL)

{

print(root->left);

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

i++;

print(root->right);

}

}

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

ifstream in;

Node\* head = new Node;

int count = 0;

in.open("FJALORI.txt"); //hapim skedarin tekst while(!in.eof()){

while(!in.eof()){

string str;

in>>str;

//cout<<str<<endl;

if(count == 0){

head = add(NULL,str);

count++;

}

else {

head = add(head,str);

} }

in.close();

cout<<"Fjalori yne ka keto fjale te vendosura: "<<endl;

print(head);

getchar();

return 0;}

***//4.NEPERMJET TABELES HASH \*nepermjete metodes channing***

#include <iostream>

#include <string>

#include <fstream>

#define MAX\_SIZE 9

using namespace std;

struct element

{

string fjale; //deklarimi i struktures se tabeles HASH

struct element \*next;

};

struct element \*tabele[MAX\_SIZE]= {NULL}; //mbushja e tabeles me elemente NULL

int Hash(string fjala) //funskioni Hash qe merr si parameter nje string fjalen

{ int gjatesi=fjala.length();

int shuma=0;

for(int i=0; i<gjatesi; i++)

shuma=shuma+(int)fjala.at(i);

return shuma%MAX\_SIZE; // e njevlefshme me celes%N

}

void shtoNeTabHash(string fjala) //funskioni qe shton ne tabelen hash fjalet e marra nga skedari

{ int poz=Hash(fjala);

element \*tmp=new element;

tmp->fjale=fjala;

tmp->next=tabele[poz];

tabele[poz]=tmp;

}

bool gjendetNeListe(string fjala,element \*l)

{

while(l!=NULL)

{

if(l->fjale==fjala)

return true;

l=l->next;

}

return false;

}

bool gjendet(string fjala)

{ int poz=Hash(fjala);

return gjendetNeListe(fjala,tabele[poz]);

} void afishoListe(element \*l)

{

while(l!=NULL)

{ cout<<l->fjale<<endl;

l=l->next;

}

}

void afishoTabeleHash()

{

for(int i=0; i<MAX\_SIZE; i++)

{

cout<<"liste "<<i+1<<" //////"<<endl;

afishoListe(tabele[i]);

}

}

int main()

{

ifstream S;

string str;

S.open("FJALORI.txt");

while(S>>str)

{

if(str.at(0)=='.'||str.at(0)==',')

str=str.substr(1,str.length());

else

if(str.at(str.length()-1)=='.'||str.at(str.length()-1)==',')

str=str.substr(0,str.length()-1);

if(!gjendet(str))

{

shtoNeTabHash(str);

} } S.close();

afishoTabeleHash();

return 0;

}

Kompleksiteti i algoritmit per:

1. **rastin e perdorimit te listave te lidhura eshte:**

Ne rastin me te keq funksionet exist, add dhe print kane kompleksitet ***O(n).*** Me pas ne main bejme leximin e skedarit deri ne fund per n- fjale.

**2. rastin e perdorimit te pemeve binare eshte:**

Ne rastin me te keq te mundshem funksionet exist,add dhe print kane kompleksitet ***0(n).*** Me pas na main bejme leximin e skedarit deri ne fund per n-fjale.

**3. rastin e perdorimit te pemeve te balancuara eshte:**

Ne rastin me te keq te mundshem funksionet exist,add dhe print kane kompleksitet ***0(2\*log2n).*** Kjo sepse pas shtimit te kulmit bejme dhe kontrollin per balancen.

**4. rastin e perdorimit te tabelave hash eshte:**

Funksioni gjendet ne rastin me te keq ka kompleksitet O(n***).*** Shtimi eshte si te lista O(1). Perfundimisht kompleksiteti eshte ***O(n).***

**Struktura per 100 fjale per:**

Listen nje drejtimore -> 0,015s  
Pemen e kerkimit binar -> 0,012s  
Pemen e balancuar -> 0,024s  
Tabelen Hash -> 0,018s

**Struktura per 1000 fjale per:**

Listen nje drejtimore -> 0,012s  
Pemen e kerkimit binar -> 0,022s  
Pemen e balancuar -> 0,06s  
Tabelen Hash -> 0,062s

**Struktura per 100000 fjale per:**

Listen nje drejtimore -> 1,06s  
Pemen e kerkimit binar -> 0,188s  
Pemen e balancuar -> 0,281s  
Tabelen Hash -> 0,203s

**Struktura per 1000000 fjale per:**

Listen nje drejtimore -> 4,875s  
Pemen e kerkimit binar -> 1,324s  
Pemen e balancuar -> 1837,s  
Tabelen Hash -> 1,7543s

## 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

package detyre;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.util.HashSet;

import org.jsoup.\*;

import org.jsoup.nodes.\*;

import org.jsoup.select.\*;

public class WebCrawler {

private String baseURL;

private static HashSet<String> containerURL;

private static int depth\_max;

public static void getURLs(String url, int depth) throws IOException {

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

return;}

if(!containerURL.contains(url) && depth<=depth\_max) {

try {

String str = ">> Depth " + depth + " [" + url + "]";

writeToFile(str);

System.out.println(str);

containerURL.add(url);

Document document = Jsoup.connect(url).get();

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

depth++;

for (Element page : linket) {

getURLs(page.attr("abs:href"), depth); } }

catch(IOException ex) {

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

public static void writeToFile(String str) throws IOException {

try {

BufferedWriter bw = new BufferedWriter(new FileWriter("C:/Users/User/Desktop/WebCrawler.txt",true));

bw.write(str);

bw.newLine();

bw.close(); }

catch(IOException ex) {

System.out.println("Ndodhi nje gabim!"); } }

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

containerURL = new HashSet<>();

depth\_max = 2;

BufferedWriter bw = new BufferedWriter(new FileWriter("C:/Users/User/Desktop/WebCrawler.txt"));

getURLs("https://google.com",0);}

}

**Kompleksiteti i algoritmit eshte :**

Kompleksiteti i loop-es i cile eshte cikel qe kerkon te gjitha url eshte O(n).Por brenda kesa loop-e kemi therritjen rekursive e cila varet nga parametri depth qe do zgjedhim i cili ka kompleksitet O(n^depth).  
=> Algoritmi jone ka kompleksitet ***O(n^(depth\*n)).***