*Tirane, me 10.06.2019*

Emri:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Kevin Dule\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Grupi II \_A2\_\_ \_\_\_

## 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 me listë.

Listat jane tipe abstrakte te dhenash të cilat përfaqësojnë një bashkësi nyjesh, ku secila prej tyre ruan një paraardhës dhe një pasardhës të vetëm.

#include <conio.h>

#include <iostream>

#include <fstream>

#include <time.h>

using namespace std;

struct Node{//fillimisht ndertojme strukturen e nyjes

string word;//ajo do te mbaje nje fjale

Node\* next;//dhe nje shenjues per elementin pasardhes

};

bool exist(Node\* head, string w){//funksioni ne fjale ben kerkimin nese nje fjale gjendet ne liste apo jo

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

while(tmp != NULL){//iteracioni ben bredhjen ne liste gjersa ajo eshte e ndryshme nga null

if(tmp->word == w){//nese gjendet fjala e marre si parameter kthen true

return true;

}

tmp = tmp->next;

}

return false;

}

Node\* add(Node\* head, string w) {//funksion per te shtuar ne liste

if(exist(head,w)){//nese elementi ekziston

return head;//kthejme listen

}

else{//ne te kundert bejme shtimin e saj perpara

Node\* tmp = new Node;//krijojme nyjen e re

tmp->word = w;//fjala qe do mbaje eshte fjala qe merret si parameter

tmp->next = head;//pasardhesi eshte koka

head = tmp;//koke bejme nyjen e saposhtuar

return head;

}

}

void print(Node\* head){//funksioni i printimit i cili ka nje logjike te thjeshte

Node\* tmp = head;//shenojme ne krye

int i = 1;//variabel per rankimin e fjaleve

while(tmp != NULL){//derisa ky iteracion te perfundoje do te printojme te gjithe elementet e listes

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

tmp = tmp->next;

i++;

}

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

ifstream in;

clock\_t start,end;

Node\* head = new Node;

int count = 0;// counti na sherben per te specifikuar nese lista eshte bosh

start = clock();

in.open("Fjalori1000000.txt");//behet hapja e skedarit

while(!in.eof()){

string str;

in>>str;

//cout<<str<<endl;

if(count == 0){

head->word = str;

head->next = NULL;

count++;//tashme rrisim countin kemi futur nje fjale dhe lista seshte bosh

}

else {

head = add(head,str);

}

}

print(head);

end = clock();

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

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

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

getchar();

return 0;

}

//Rasti me BST

Pemët e kërkimit binar quhen pemët binare në të cilat çelësat e elementeve të nënpemës së majtë janë më të vegjël ose baraz me çelësin e rrënjës ndërsa ata të nënpemës së djathtë përfaqësohen nga një çelës më i madh se ai i rrënjës.

#include <iostream>

#include <string>

#include <fstream>

#include <time.h>

using namespace std;

int index = 1;

struct Node {//krijome strukturen per nyjen

string word;//do te mbaj nje fjale

Node \*left;//shenjues per majtas

Node \*right;//shenjues per birin e djathte

};

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

if(root==NULL) {//kushti nese pema eshte bosh

return false;

}

if(root->word==newW) {//kerkojme fillimisht nese fjala ndodhet ne rrenjen e pemes

return true;

}

if(newW<root->word) {//kusht i cili kontrollon nese fjala eshte ne rend me te ulet alfabetik

return exist(root->left,newW);//therritje e metodes rekrusive

} else {

return exist(root->right,newW);//ne rast se fjala eshte ne rend alfabetik me te madh

}

}

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

Node \*temp = new Node;

temp->word = w;

temp->left = NULL;

temp->right = NULL;

return temp;

}

Node \*add(Node \*root, string w) {//funksion per shtimin e kulmit te ri

if(root==NULL)//rasti nese pema eshte bosh

return newNode(w);

if(exist(root,w))//therrasim funskion exist, nese fjala ekziston

return root;//kthejme pemen

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

root->left = add(root->left,w);//e shtojme ne te majte nepermjet therritjes se metodes rekursive

} else {

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

}

return root;

}

void print(Node \*root)//printimi i pemes i cili behet sipas menyres nenpeme e majte, rrenje, nenpeme e djathte

{

if (root != NULL)

{

print(root->left);

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

index++;

print(root->right);

}

}

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

ifstream in;

clock\_t start, end;

Node\* head = new Node;

int c = 0;

in.open("Fjalori1000.txt");

start = clock();

while(!in.eof()){

string str;

in>>str;

if(c == 0){

head->word = str;

head->left = NULL;

head->right = NULL;

c++;

}

else {

head = add(head,str);

}

}

print(head);

end = clock();

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

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

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

getchar();

return 0;

}

//me AVL

Pemet e balancuara jane peme te kerkimit binar ne te cilat lartesia e cdo kulmi duhet te jete e balancuar. Ajo eshte e balancuar nese diferenca ndermjet lartesive te bijve eshte pakten 1.

#include <iostream>

#include <string.h>

#include <fstream>

#include <time.h>

using namespace std;

int i = 1;

struct Node {//krijimi i struktures per pemen e balancuar, e njejte si BST

string word;

Node \*left;

Node \*right;

int height;//na duhet dhe nje variabel per te matur lartesine e kulmit, pasi duhet te kontrollojme vazhdimisht nese pema qendron e balancuar apo jo pas shtimit te ndonje kulmi

};

int returnmax(int x, int y) {//funksion per gjetjen e maksimumit ndermjet dy gjatesive

if(x>y)

return x;

return y;

}

int height(Node \*tmp) {//funksion qe na kthen gjatesine e kulmit

if(tmp==NULL)

return 0;

return tmp->height;

}

bool exist(Node \*root, string w) {//funksion per kerkimin e ndonje fjale, e njejte si te BST

if(root==NULL) {

return false;

}

if(root->word==w) {

return true;

}

if(w>root->word) {

return exist(root->left,w);

} else {

return exist(root->right,w);

}

}

Node \*rightRotate(Node \*y)//kryerja e rrotullimit per te bere pemen e balancuar nese ajo nuk eshte nga e djathta

{

Node \*x = y->left;

Node \*T2 = x->right;

//realizimi i rrotullimit

x->right = y;

y->left = T2;

// riperditesim i gjatesise

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)//kryerja e rrotullimi per nga e majta

{

Node \*y = x->right;

Node \*T2 = y->left;

// realizimi i rrotullimit

y->left = x;

x->right = T2;

//perditesimi i gjatesise

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)//kthene balancen pra ndryshesen e gjatesive te nenpemes se majte me ate te djathe e cila duhet te jete ta pakten 1

{

if (tmp == NULL)

return 0;

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

}

Node \*newNode(string w) {//funksion qe ben krijimin e kulmit te ri

Node \*temp = new Node;

temp->word = w;

temp->left = NULL;

temp->right = NULL;

temp->height = 1;

return temp;

}

Node \*add(Node \*root, string w) {//funksioni qe shton nje kulm te ri

//pjesa e pare njejte si te BST

if(root==NULL)

return newNode(w);

if(exist(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));

//marrim balancen

int balance = getBalance(root);

//Majtas Majtas

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

return rightRotate(root);

// Djathtas Djathtas

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

return leftRotate(root);

// Majtas Djathtas

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

{

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

return rightRotate(root);

}

// Djathtas Majtas

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

{

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

return leftRotate(root);

}

return root;

}

void print(Node \*root)//funksioni i printimit

{

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;

clock\_t start, end;

start = clock();

in.open("Fjalori1000000.txt");

while(!in.eof()){

string str;

in>>str;

if(count == 0){

head = add(NULL,str);

count++;

}

else {

head = add(head,str);

}

}

in.close();

print(head);

end = clock();

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

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

cout<<"Koha e ekzekutimit te programit tone eshte: "<<time\_taken<<" s"<<endl;

getchar();

return 0;

}

//me tabela hash

#include <iostream>

#include <string>

#include <fstream>

#define MAX\_SIZE 9

using namespace std;

struct element

{

string fjale;

struct element \*next;

};

struct element \*tabele[MAX\_SIZE]= {NULL};

void shtoNeTabHash(string);

int HASH(string);

bool gjendet(string);

bool gjendetNeListe(string,element \*);

void afishoListe(element \*l);

void afishoTabeleHash();

int main()

{

ifstream inData;

string str;

inData.open("Fjalori1000.txt");

while(inData>>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);

}

}

inData.close();

afishoTabeleHash();

return 0;

}

int HASH(string fjala)

{

int gjatesi=fjala.length();

int shuma=0;

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

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

return shuma%MAX\_SIZE;

}

void shtoNeTabHash(string fjala)

{

int poz=HASH(fjala);

element \*tmp=new element;

tmp->fjale=fjala;

tmp->next=tabele[poz];

tabele[poz]=tmp;

}

bool gjendet(string fjala)

{

int poz=HASH(fjala);

return gjendetNeListe(fjala,tabele[poz]);

}

bool gjendetNeListe(string fjala,element \*l)

{

while(l)

{

if(l->fjale==fjala)

return true;

l=l->next;

}

return false;

}

void afishoListe(element \*l)

{

while(l)

{

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]);

}

}

Kompleksiteti i algoritmit per:

Ne rastin e tabelave Hash kemi perdorur tekniken Chaining. Kemi daklaruar permasen e tabeles hash me MAX\_SIZE 9. Funksioni HASH ben te mundur konvertimin e fjales ne sistemin ASCII. Marrim germat nje nga nje dhe me pas kthejme modulin shumes me 9. Supozojme se fjala eshte “BCA” shuma e ketyre karaktereve ne ASCII eshte 189%9 = 0 pra kjo fjale do te vendoset ne koken e listes qe shenon ne tabelen hash me pozicion 0.

1. rastin e perdorimit te listave te lidhura eshte:

Ne rastin e listave te lidhura kemi 3 funksione exist(), add() dhe print(). Ne rastin me te keq exist() duhet te bej bredhjen e te gjithe listes dhe kompleksiteti behet o(n). Funksioni add() ben shtim ne fillim te listes me kompleksitet o(1). Ky funksion mban brenda dhe funksioni exist(). Perfundimisht, kompleksiteti eshte o(n).

2. rastin e perdorimit te pemeve binare eshte:

Ne rastin e peme binare funksioni exist ne rastin me te keq do te jete o(n) nese sekuenca e elementeve te dhene ka qene e renditur, add behet pakashume si te listat me kompleksitet o(n). Printimi i pemes behet ne menyre nderrendore.

3. rastin e perdorimit te pemeve te balancuara eshte:

Nje rast i ngjashme me BST eshte dhe AVL, pa harruar operacionet e rrotullimit nga e djathta dhe e majta. Funksionet add, exist dhe print ne rastin me te keq kane kompleksitet o(log2n). Perfundimisht, kompleksiteti i kesaj strukture se bashku me hapjen e skedarit eshte o(2\*log2n) pasi behet dhe kontrolli per balancimin e pemes.

4. rastin e perdorimit te tabelave hash eshte:

Ne kete rast shtimi ne nje liste te caktuar behet me kompleksitet o(1). Kerkimi per ndonje fjale behet si tek lista dhe ne rastin me te keq ky kompleksitet eshte o(n). Perfundimisht, kompleksiteti final eshte o(n).

Perfundimisht:

|  |  |
| --- | --- |
| Struktura | Kompleksiteti |
| Lista lineare | O(n) |
| Pema e kerkimit binar | O(n) |
| Pema e balancuar | O(2\*log2n) |
| Tabela Hash | O(n) |

**Perfundimi**

**Shohim se struktura me efikase eshte pema e balancuar. Por per ndonje sasi te caktuar te dhenash Pema e kerkimit binar mund te kete kompleksitet o(log2n). Pra rreth dy here me e shpejte se pema e balancuar.**

**Tabela e koheve te ekzekutimi(e cila ndikohet nga parametrat e procesorit):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Struktura | 100 fjale | 1000 fjale | 100000 fjale | 1000000 fjale |
| Liste linear | **0.015 s** | **0.034 s** | **1.06 s** | **4.875 s** |
| Peme e kerkimit binar | **0.012 s** | **0.022 s** | **0.188 s** | **1.324s** |
| Peme e balancuar | **0.024 s** | **0.06s** | **0.281 s** | **1.837 s** |
| Tabela Hash | **0.018 s** | **0.062 s** | **0.203 s** | **1.7543 s** |

## 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

WebCrawler eshte nje mekanizem i cili ploteson keto principe:

1-Merr nje faqe baze URL.

2-Ben parsimin e te gjitha URL-ve qe ndodhen aty.

3-Riperterit procesin per URL e gjetura.

package seminar11;

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/Kevin Dule/Desktop/seminar10.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/Kevin Dule/Desktop/seminar10.txt"));

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

}

} 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/Kevin Dule/Desktop/seminar10.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/Kevin Dule/Desktop/seminar10.txt"));

getURLs("http://fshn.edu.al ",0);

}

}

Per kete deyre kemi perdorur strukturen HashSet qe jane instanca te HashMap(HashTable) e cila gezon keto veti:

1-Objektet ne HashSet futen sipas nje celesi te caktuar.

2-Nuk permban vlera te duplikuara vetem unike.

Kompleksiteti i algoritmit eshte:

Ne kompleksitetin e tij ndikon loopa for each e cila ne rastin me te keq ka kompleksitet o(n). Gjithshtu brenda kesaj loop-e kemi nje therritje rekrusive e cila varet shume nga depth qe do zgjedhim. Ne varesi te depth, ekzekutimi i algoritmit mund te zgjase dhe me shume se nje nate. Kompleksiteti i ketij algoritmi eshte o(n^depth). Perfundimisht algoritmi jone ka kompleksitet o(n^(depth\*n)).