Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 6 з дисципліни
«Основи програмування 2. Модульне програмування»
«Дерева»
Варіант 7

Виконав студент ІП-15, Гуменюк Олександр Володимирович
(шифр, прізвище, ім'я, по батькові)
Перевірила Вєчерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота 6

Дерева

Індивідуальне завдання

Варіант 7

7. Заданий текст. Підрахувати кількість повторень кожного слова. Побудувати дерево із слів тексту. Слова, що зустрічаються найчастіше розмістити на верхньому рівні, на інших рівнях дерева розмістити слова з меншою кількістю повторень.

Код C++

OP_Lab6.cpp

```
#include "functions.h"
#include "heap.h"


int main() {
	Heap heap = buildHeap();
	cout << "\nTree" << endl;
	heap.printHeap(heap.getRoot());

	cout << "\nTree words" << endl;
	heap.levelOrderPrint();
}
```

functions.h

```
#pragma once
#include "heap.h"
#include <iostream>
#include <string>
#include <vector>


using namespace std;

Heap buildHeap();
```

functions.cpp

```
#include "functions.h"

vector <string> splitWords(string line) {

	line = line + " ";
	vector <string> words;
	string word = "";
```

```
	for (auto letter : line) {
		if (letter == ' ') {
			if (word != "") {
				words.push_back(word);
			}
			word = "";
		}
		else {
			word += letter;
		}
	}

	return words;
}

void insertWordsInHeap(Heap &heap, string line) {
	vector <string> words = splitWords(line);

	for (auto word : words) {
		heap.insert(word);
	}

}


Heap buildHeap() {
	string line;
	size_t combinationCode = 7;
	Heap heap{};

	cout << "Input your text. ENTER for next line. CTRL+G to stop" << endl;
	cin.ignore();

	getline(cin, line);
	while (line[0] != combinationCode) {
		insertWordsInHeap(heap, line);
		getline(cin, line);
	}

	return heap;
}
```

node.h

```
#pragma once
#include <string>

using namespace std;

class Node {
	string value;
	int counter;
	Node* left;
	Node* right;
	Node* parent;
public:
	Node(string);

	string getValue();
```

```
        void setValue(string);

        int getCounter();
        void setCounter(int);
        void increaseCounter();

        Node* getLeft();
        void setLeft(Node*);

        Node* getRight();
        void setRight(Node*);

        Node* getParent();
        void setParent(Node*);

        void swap(Node*);
};
```

node.cpp

```
#include "node.h"

Node::Node(string val) {
        value = val;
        left = nullptr;
        right = nullptr;
        parent = nullptr;
        counter = 1;
}

string Node::getValue() {
        return value;
}

void Node::setValue(string val) {
        value = val;
}

int Node::getCounter() {
        return counter;
}

void Node::setCounter(int val) {
        counter = val;
}

void Node::increaseCounter() {
        counter++;
}

Node* Node::getLeft() {
        return left;
}

void Node::setLeft(Node* node) {
        left = node;
}
```

```
Node* Node::getRight() {
	return right;
}

void Node::setRight(Node* node) {
	right = node;
}

Node* Node::getParent() {
	return parent;
}

void Node::setParent(Node* node) {
	parent = node;
}

void Node::swap(Node* node) {
	string tempValue = node->getValue();
	int tempCounter = node->getCounter();
	node->setValue(value);
	node->setCounter(counter);

	value = tempValue;
	counter = tempCounter;
}
```

heap.h

```
#pragma once
#include <queue>
#include <iostream>
#include <iomanip>
#include "node.h"

class Heap {
	Node* root;

	void swapUp(Node*);
	void increasePriority(Node*);
	void levelOrderInsert(string);
	Node* findNode(string, Node*);
public:
	Node* getRoot();
	void insert(string);
	bool isEmpty();
	void printHeap(Node*, int = 0);
	void levelOrderPrint();
};
```

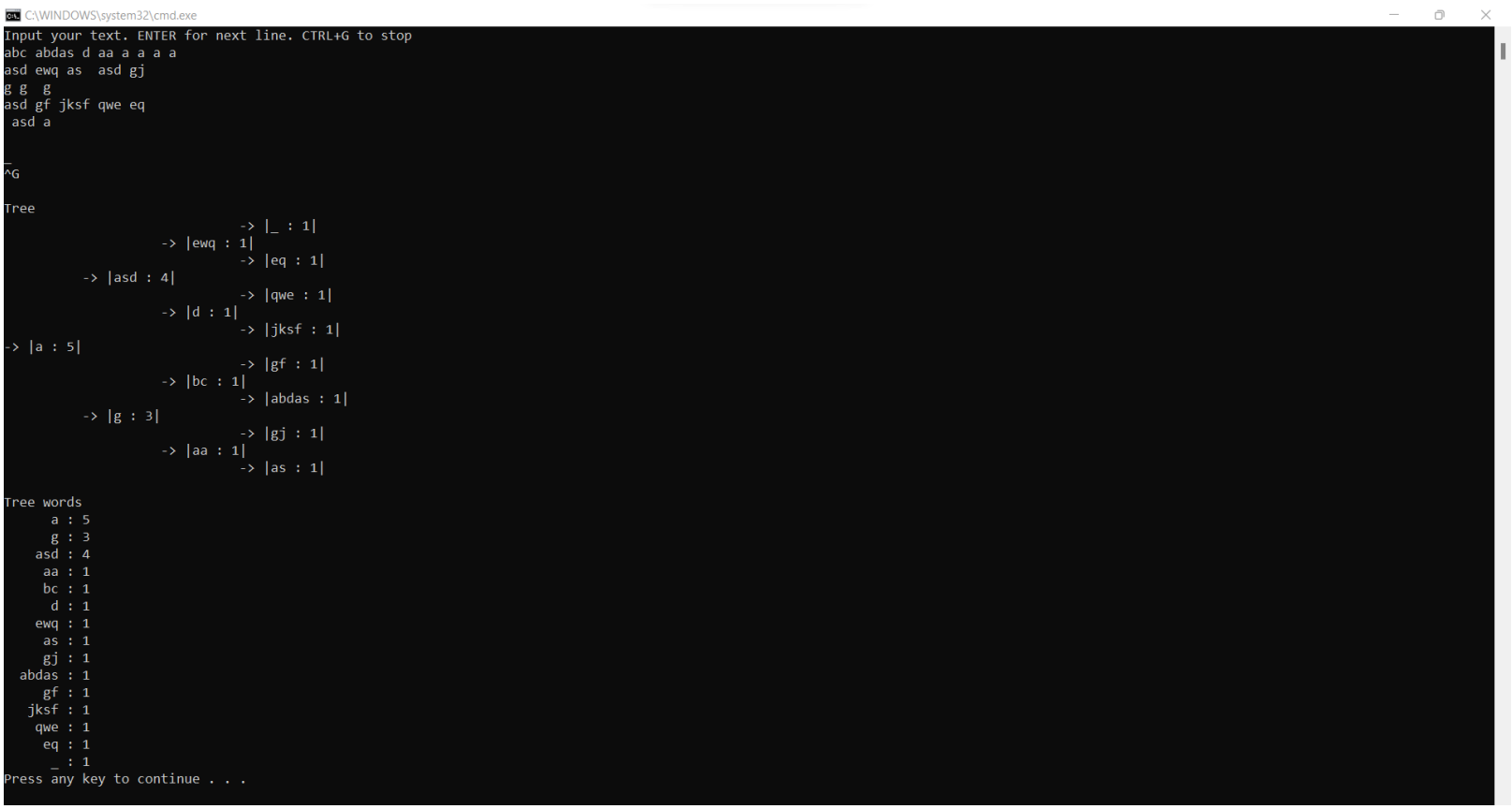
heap.cpp

```
#include "heap.h"


Node* Heap::getRoot() {
	return root;
}
```

```
bool Heap::isEmpty() {
	return root == nullptr;
}

void Heap::insert(string value) {
	if (isEmpty()) {
		root = new Node(value);
	}
	else {
		Node* node = findNode(value, root);
		if (node == nullptr) levelOrderInsert(value);
		else 			increasePriority(node);
	}
}

Node* Heap::findNode(string value, Node* node) {
	if (node != nullptr) {

		Node* leftNode = findNode(value, node->getLeft());
		Node* rightNode = findNode(value, node->getRight());

		if (node->getValue() == value) return node;
		else if (leftNode != nullptr)  return leftNode;
		else 					return rightNode;

	}
	else {
		return nullptr;
	}
}

void Heap::levelOrderInsert(string value) {
	queue<Node*> q;
	q.push(root);
	while (!q.empty()) {
		Node* temp = q.front();
		q.pop();

		if (temp->getLeft() == nullptr) {
			Node *node = new Node(value);
			node->setParent(temp);
			temp->setLeft(node);
			return;
		}
		else q.push(temp->getLeft());

		if (temp->getRight() == nullptr) {
			Node* node = new Node(value);
			node->setParent(temp);
			temp->setRight(node);
			return;
		}
		else q.push(temp->getRight());
	}
}

void Heap::swapUp(Node* node) {
```

```
	if (node->getParent() != nullptr && node->getParent()->getCounter() < node-
>getCounter()) {
		node->swap(node->getParent());
		swapUp(node->getParent());
	}
}


void Heap::increasePriority(Node* node) {
	node->increaseCounter();
	swapUp(node);
}

void Heap::printHeap(Node* node, int level) {
	if (node != nullptr) {
		printHeap(node->getRight(), level + 1);

		string spaces(10 * level, ' ');
		cout << spaces << "-> |" << node->getValue() << " : " << node-
>getCounter() << "|" << endl;

		printHeap(node->getLeft(), level + 1);
	}

}

void Heap::levelOrderPrint() {
	queue<Node*> q;
	q.push(root);
	while (!q.empty()) {
		Node* temp = q.front();
		q.pop();

		cout << setw(7) << temp->getValue() << " : " << temp->getCounter() <<
endl;

		if (temp->getLeft() != nullptr) {
			q.push(temp->getLeft());
		}

		if (temp->getRight() != nullptr) {
			q.push(temp->getRight());
		}
	}
}
```

Тестування


C:\WINDOWS\system32\cmd.exe
Input your text. ENTER for next line. CTRL+G to stop
abc abdas d aa a a a a
asd ewq as asd gj
g g g
asd gf jksf qwe eq
 asd a
_
^G
Tree
-> |_ : 1|
-> |ewq : 1|
-> |eq : 1|
-> |asd : 4|
-> |qwe : 1|
-> |d : 1|
-> |jksf : 1|
-> |a : 5|
-> |gf : 1|
-> |bc : 1|
-> |abdas : 1|
-> |g : 3|
-> |gj : 1|
-> |aa : 1|
-> |as : 1|
Tree words
a : 5
g : 3
asd : 4
aa : 1
bc : 1
d : 1
ewq : 1
as : 1
gj : 1
abdas : 1
gf : 1
jksf : 1
qwe : 1
eq : 1
_ : 1
Press any key to continue . . .