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 19

Виконав студент: ІП-11 Лисенко Андрій Юрійович

Перевірила: Вітковська Ірина Іванівна

Київ 2021

**Лабораторна робота №6**

**Дерева**

**Мета роботи**

Вивчити особливості організації і обробки дерев.

**Індивідуальне завдання**

**Варіант 19**

Побудувати бінарне дерево, елементами якого є слова. Знайти у ньому значення слова, введеного з клавіатури, визначивши номер рівня.

**Постановка задачі**

Для побудови бінарного дерева використуємо алгоритм побудови повного дерева з масиву даних. У даному випадку масив даних – це масив слів, які вводить користувач. Для пошуку елемента використуємо не рекурсивний алгоритм. Ми зможемо легко інкрементувати/декрементувати рівень на якому розташовується задане слово для пошуку. Таким чином виконаємо задачу.

**Побудова математичної моделі**

Складемо таблицю змінних

|  |  |  |  |
| --- | --- | --- | --- |
| Змінна | Тип | Ім’я | Призначення |
| Масив слів | *Vector* | *Vec* | Зберігатиме масив слів для побудови дерева |
| Дерево | *Tree* | *Tree* | Структура даних для зберігання слів |
| Шукане слово | *String* | *Prompt* | зберігатиме слово яке потрібно знайти у дереві |
| Лівий та правий нащадок | *Node\** | *Left, right* | Покажчик на елемент нащадка |

**Псевдокод**

**Основна програма**

input vec

tree = new Tree()

tree.create\_tree()

input prompt

tree.search(prompt)

**Метод *Tree: create\_tree(vec, from, to)***

If size == 0: return NULL

node = new Node()

size\_left = size / 2

size\_right = size – size\_left – 1

node.left = create\_tree(vec, from + 1, size\_left)

node.right = create\_tree(vec, from + 1 + size\_left, size\_right)

return node

**Метод *Tree: search(prompt)***

current = root

stop = false

level = 0

stack = empty stack

while not stop:

while current != NULL:

stack.push(current)

level += 1

current = current.left

if stack is empty:

Output “Word not found”

Else:

Level -= 1

Current = stack.top()

If current.word == prompt:

Return “Word found at level {level}”

Current = current.right

stack.pop()

**Метод Matrix get\_average**

Counter = 0

for element in matrix:

counter += element

return counter / (rows \* cols)

**Код на С++**

**main.cpp**

#include "Node.h"

#include "Tree.h"

int main() {

std::string word = " ";

std::vector<std::string> vec;

std::cout << "Enter words (enter q to quit):" << std::endl;

while (word != "q") {

std::cin >> word;

if (word != "q")

vec.push\_back(word);

}

Tree tree;

tree.create\_tree(vec);

std::string prompt;

std::cout << "Enter prompt: ";

std::cin >> prompt;

tree.search(prompt);

}

**Node.h**

#pragma once

#include <string>

#include <iostream>

struct Node {

std::string word;

Node \*left{}, \*right{};

explicit Node(const std::string& word) { this->word = word; }

};

**Tree.h**

#pragma once

#include <vector>

#include "Node.h"

class Tree {

Node \*root;

public:

Tree(): root(nullptr) {}

void create\_tree(const std::vector<std::string>& vec);

Node\* create\_tree(const std::vector<std::string>& vec, unsigned long long from, unsigned long long size);

void search(const std::string& prompt);

};

**Tree.cpp**

#include "Tree.h"

#include <stack>

void Tree::create\_tree(const std::vector<std::string>& vec) {

root = create\_tree(vec, 0, vec.size());

}

Node\* Tree::create\_tree(const std::vector<std::string>& vec, unsigned long long from, unsigned long long size) {

if (size == 0)

return nullptr;

Node\* nodeptr = new Node(vec[from]);

unsigned long long size\_left = size / 2;

unsigned long long size\_right = size - size\_left - 1;

nodeptr->left = create\_tree(vec, from + 1, size\_left);

nodeptr->right = create\_tree(vec, from + 1 + size\_left, size\_right);

return nodeptr;

}

void Tree::search(const std::string& prompt) {

Node\* current = root;

bool stop = false;

int level = 0;

std::stack<Node\*> stack;

while (!stop) {

while (current != nullptr) {

stack.push(current);

++level;

current = current->left;

}

if (stack.empty()) {

std::cout << "Word \"" << prompt << "\" not found\n";

stop = true;

}

else {

--level;

current = stack.top();

if (current->word == prompt) {

std::cout << "The word \"" << prompt << "\" found on level " << level << "\n";

return;

}

current = current->right;

stack.pop();

}

}

}

**Висновок**

Підчас виконання цієї роботи, я розробив практичні навички роботи з такою структурою даних, як бінарні дерева, а саме створення повного дерева з масиву елементів, та не рекурсивний пошук у глибину дерева. Я використував не рекурсивний підхід, тому що так, на мою думку, легше знайти рівень шуканого слова. Отже, я вивчив особливості організації і обробки бінарних дерев.