Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем та програмної інженерії

Кафедра програмної інженерії

**З В І Т**

до виконання лабораторної роботи №5

з дисципліни «Алгоритми та структури даних»

на тему Абстрактний тип даних ”Дерево”

Виконав:

студент групи СП-11

Бондар Віталій

Варіант 4(1)

Тернопіль 2022

**Мета роботи:**засвоїти теоретичний матеріал та набути практичних навичок по роботі з абстрактним типом даних “Дерево”

**Теоретичні відомості**

**АТД Дерево**

Дерево (tree) – це сукупність елементів, які називаються вузлами та відношень між цими елементами, що утворюють ієрархічну структуру дерева.

Початковий вузол дерева називається його коренем (root).

Вузли в дереві, як і елементи в списку, можуть бути даними довільного типу.

Формально, дерево можна визначити наступним чином:

1. Один вузол – є деревом. Цей же вузол також є коренем цього дерева.

2. Нехай *n* – це вузол, а *T1, T2,…,Tk* – дерева з коренями *n1, n2,…,nk* відповідно. Тоді можна побудувати нове дерево, зробивши *n* предком (ancestors) вузлів *n1, n2,…,nk.*

В цьому випадку вузол n буде коренем, а *T1, T2,…,Tk* – під деревами (subtrees) цього кореня. Вузли  *n1, n2,…,nk* називаються нащажками (descendant) вузла *n*.

3. Нульовим деревом називають дерево, яке немає вузлів.

З цього визначення випливає, що кожна вершина є в свою чергу коренем деякого піддерева. Кількість піддерев вершини має назву **ступеня**(degree) цієї вершини.

Вершина ступеню нуль має назву **кінцевої**(terminal) або **листа**(leaf).

Некінцева вершина також має назву **вершини розгалуження** (branch node).

Нащадки та предки вершини *x*, що не співпадають з нею самою, називаються власними нащадками та предками.

Якщо вершина *x*є предком *y*та не існує вершин поміж ними (тобто *x*та *y*з'єднані одним ребром), а також існують предки для *x*(тобто *x*не є коренем), то вершина *x*називається **батьком**(parent) до *y*, а *y*— **дитиною**(child) *x*.

Коренева вершина єдина не має батьків. Вершини, що мають спільного батька, називаються **братами** (siblings).

Вершини, що мають дітей, називаються **внутрішніми**(internal).

**Глибиною**вершини *x*називається довжина шляху від кореня до цієї вершини.

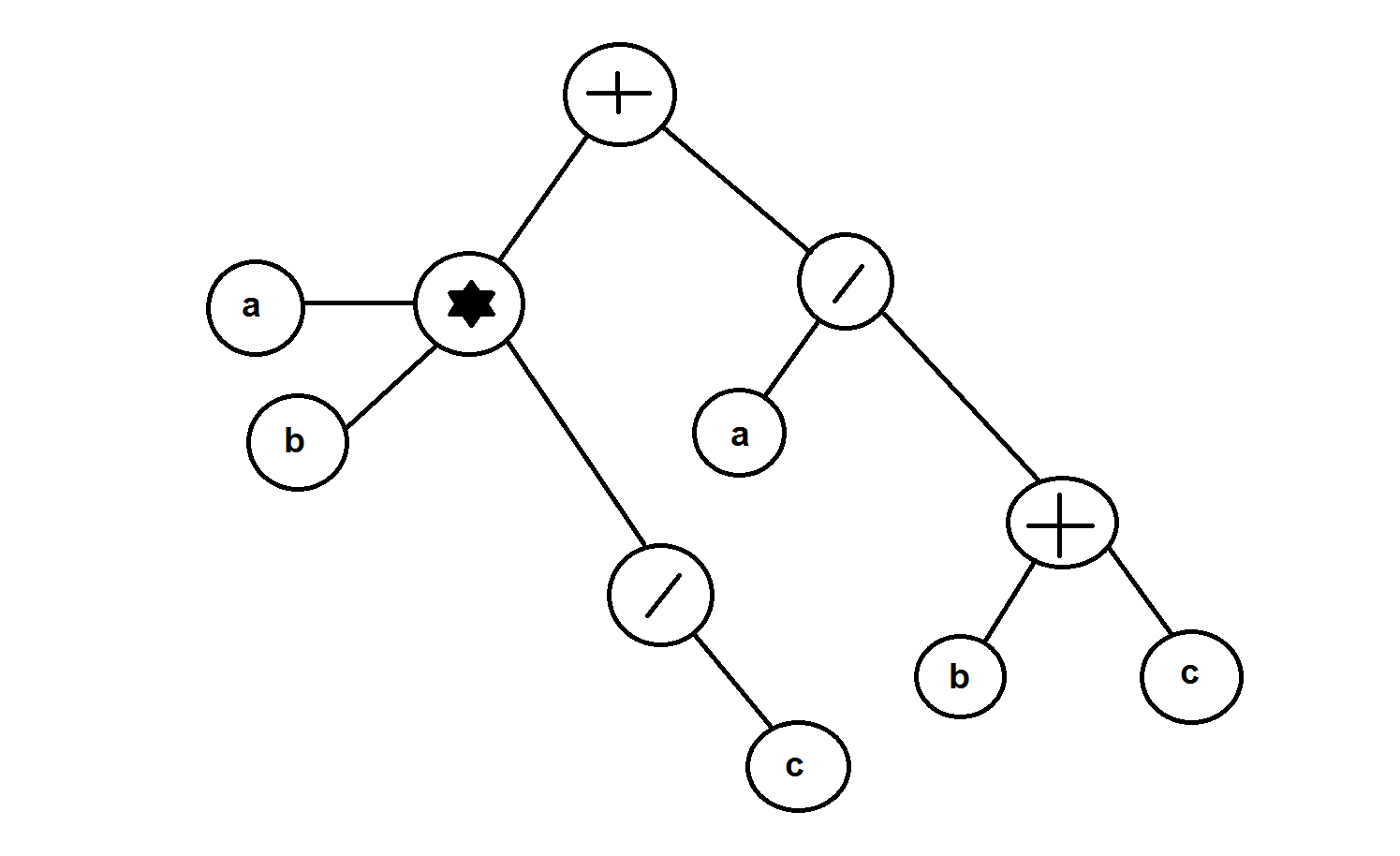
Максимальна глибина вершин дерева називається **висотою**.

**Лісом**(forest) називають множину дерев, які не перетинаються.

**Варіант 4**

**Завдання 1**

1. Побудувати графічно дерево для виразу  **(a\*b)/c+(a/(b+c))**



**Завдання 2**

2. Реалізувати (з використанням функцій АТД Tree), функції для обходу дерев та привести список обходу вузлів дерева побудованого в п.1 у прямому порядку

IDE: Microsoft Visual Studio 2019

Лістинг програми

#include <iostream>

using namespace std;

struct node

{

int value;

char label;

node\* leftchild;

node\* rightchild;

node(char lable, int value = int(), node\* leftchild = nullptr, node\* rightchild = nullptr)

{

this->value = value;

this->label = lable;

this->leftchild = leftchild;

this->rightchild = rightchild;

}

node(int value = int(), node\* leftchild = nullptr, node\* rightchild = nullptr)

{

this->value = value;

this->label = ' ';

this->leftchild = leftchild;

this->rightchild = rightchild;

}

};

class tree

{

private:

node\* Root;

public:

tree() {

this->Root = nullptr;

}

~tree(){}

void create(int value) {

this->Root = new node(value);

}

void Insert(node\* root, int value, char lable) {

if (value < root->value)

{

if (root->leftchild == nullptr)

{

root->leftchild = new node(lable, value);

}

else

{

this->Insert(root->leftchild, value, lable);

}

}

else

{

if (root->rightchild == nullptr)

{

root->rightchild = new node(lable, value);

}

else

{

this->Insert(root->rightchild, value, lable);

}

}

}

void Insert(node\* root, int value) {

if (value < root->value)

{

if (root->leftchild == nullptr)

{

root->leftchild = new node(value);

}

else

{

this->Insert(root->leftchild, value);

}

}

else

{

if (root->rightchild == nullptr)

{

root->rightchild = new node(value);

}

else

{

this->Insert(root->rightchild, value);

}

}

}

void Insert(int value, char lable) {

if (value < this->Root->value)

{

if (this->Root->leftchild == nullptr)

{

this->Root->leftchild = new node(lable, value);

}

else

{

this->Insert(this->Root->leftchild, value, lable);

}

}

else

{

if (this->Root->rightchild == nullptr)

{

this->Root->rightchild = new node(lable, value);

}

else

{

this->Insert(this->Root->rightchild, value, lable);

}

}

}

void Insert(int value) {

if (value < this->Root->value)

{

if (this->Root->leftchild == nullptr)

{

this->Root->leftchild = new node(value);

}

else

{

this->Insert(this->Root->leftchild, value);

}

}

else

{

if (this->Root->rightchild == nullptr)

{

this->Root->rightchild = new node(value);

}

else

{

this->Insert(this->Root->rightchild, value);

}

}

}

void directOrder() {

cout << "Direct order:" << endl;

if (this->Root->leftchild != nullptr)

{

this->directOrder(this->Root->leftchild);

}

cout << this->Root->value << "|" << this->Root->label << endl;

if (this->Root->rightchild != nullptr)

{

this->directOrder(this->Root->rightchild);

}

}

void directOrder(node\* root) {

if (root->leftchild != nullptr)

{

this->directOrder(root->leftchild);

}

cout << root->value << "|" << root->label << endl;

if (root->rightchild != nullptr)

{

this->directOrder(root->rightchild);

}

}

};

int main()

{

tree obj1;

int a, b, c;

cout << "Enter a: ";

cin >> a;

cout << "Enter b: ";

cin >> b;

cout << "Enter c: ";

cin >> c;

obj1.create((a \* b) / c + (a / (b + c)));

obj1.Insert(a / (b + c), '+');

obj1.Insert((a \* b) / c, '+');

obj1.Insert(b);

obj1.Insert(c);

obj1.Insert(a);

obj1.Insert(b+c, '+');

obj1.Insert(b);

obj1.Insert(c);

cout << endl;

obj1.directOrder();

return 0;

}

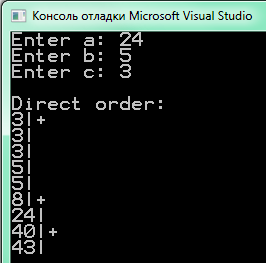


Рисунок 2 – результат виконання завдання 2

**Завдання 3**

Реалізувати (з використанням мови програмування) основні функції для АТД Tree (PARENT, LEFTMOST\_CHILD, RIGTH\_SIBLING, LABEL, ROOT).

Лістинг програми

#include <iostream>

using namespace std;

typedef char nametype;

typedef node\* TREE;

struct node {

node\* leftchild;

node\* rightchild;

nametype element;

};

int root(node\* n)

{

int left, right;

if (n->leftchild != NULL && n->rightchild != NULL)

{

left = root(n->leftchild) + 1;

right = root(n->rightchild) + 1;

if (left > right)

return right;

else

return left;

}

else if (n->leftchild == NULL)

{

return 1;

}

else if (n->rightchild == NULL)

{

return 1;

}

}

node\* determine(node\* n, node\* temp)

{

if (!(temp->leftchild == NULL && temp->rightchild == NULL))

{

if (temp->leftchild == n)

return temp;

else if (temp->rightchild == n)

return temp;

else

{

node\* r\_val = determine(n, temp->leftchild);

if (r\_val != NULL)

return r\_val;

return determine(n, temp->rightchild);

}

}

return NULL;

}

node\* PARENT(node\* n, TREE T)

{

if (T == n) {

return NULL;

}

node\* temp = T;

return determine(n, temp);

}

node\* LEFTMOST\_CHILD(node\* n, TREE T)

{

node\* temp = PARENT(n, T);

temp = temp->leftchild;

while (temp->leftchild != NULL) {

temp = temp->leftchild;

}

return temp;

}

node\* RIGHT\_SIBLING(node\* n, TREE T)

{

node\* temp = PARENT(n, T);

if (temp->leftchild == n) {

return temp->rightchild;

}

else {

return NULL;

}

}

nametype LABEL(node\* n, TREE T)

{

node\* temp = PARENT(n, T);

if (temp->leftchild == n) {

return temp->leftchild->element;

}

else

{

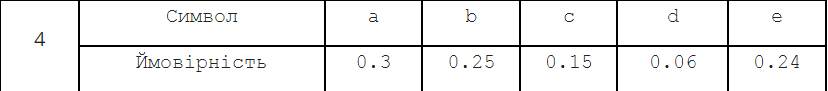
return temp->leftchild->element;

}

}

**Завдання 4,5**

Знайти код Хаффмана для повідомлення, що складається з символів [abcde] з наступними ймовірностями:



Також написати програму по кодування та декодуванню повідомлення відповідно до отриманих кодів.

Лістинг програми

#include <iostream>

#include<string>

using namespace std;

struct node

{

float value;

string label;

node\* leftchild;

node\* rightchild;

node(char lable, float value = float(), node\* leftchild = nullptr, node\* rightchild = nullptr)

{

this->value = value;

this->label = lable;

this->leftchild = leftchild;

this->rightchild = rightchild;

}

node(float value = float(), node\* leftchild = nullptr, node\* rightchild = nullptr)

{

this->value = value;

this->label = ' ';

this->leftchild = leftchild;

this->rightchild = rightchild;

}

};

class tree

{

private:

node\* Root;

public:

tree() {

this->Root = nullptr;

}

~tree(){}

void insert(node\* code, int number) {

if (number > 1)

{

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

{

for (int j = 0; j < number; j++)

{

if (code[i].value >= code[j].value)

{

swap(code[i], code[j]);

}

}

}

node\* new\_code = new node[--number];

for (int i = 0; i < number - 1; i++)

{

new\_code[i] = code[i];

}

new\_code[number - 1].value = code[number].value + code[number - 1].value;

new\_code[number - 1].label = code[number].label + code[number - 1].label;

new\_code[number - 1].leftchild = &code[number];

new\_code[number - 1].rightchild = &code[number - 1];

this->insert(new\_code, number);

}

else

{

this->Root = &code[0];

}

}

void print(node\* root) {

if (root->leftchild != nullptr)

{

this->print(root->leftchild);

}

cout << "Data = " << root->value << "|" << root->label << endl;

if (root->rightchild != nullptr)

{

this->print(root->rightchild);

}

}

void print() {

if (this->Root->leftchild != nullptr)

{

this->print(this->Root->leftchild);

}

cout << "Data = " << this->Root->value << "|" << this->Root->label << endl;

if (this->Root->rightchild != nullptr)

{

this->print(this->Root->rightchild);

}

}

void code(char value) {

node\* temp = this->Root;

while (temp->leftchild != nullptr && temp->rightchild != nullptr)

{

bool flag = true;

for (int i = 0; i < temp->leftchild->label.length(); i++)

{

if (value == temp->leftchild->label[i])

{

cout << 0;

temp = temp->leftchild;

flag = false;

break;

}

}

if (flag)

{

for (int i = 0; i < temp->rightchild->label.length(); i++)

{

if (value == temp->rightchild->label[i])

{

cout << 1;

temp = temp->rightchild;

break;

}

}

}

}

}

void coding(string str) {

cout << "Coding text: ";

for (int i = 0; i < str.length(); i++)

{

this->code(str[i]);

}

}

void decoding(string str) {

cout << "Decoding text: ";

node\* temp = this->Root;

for (int i = 0; i < str.length() + 1; i++)

{

if (temp->leftchild == nullptr && temp->rightchild == nullptr)

{

cout << temp->label;

temp = this->Root;

}

if (str[i] == '0')

{

temp = temp->leftchild;

}

else

{

temp = temp->rightchild;

}

}

}

};

int main()

{

tree Tree;

node code[5];

code[0].label = 'a';

code[0].value = 0.3;

code[1].label = 'b';

code[1].value = 0.25;

code[2].label = 'c';

code[2].value = 0.15;

code[3].label = 'd';

code[3].value = 0.06;

code[4].label = 'e';

code[4].value = 0.24;

Tree.insert(code, 5);

cout << "Tree:" << endl;

Tree.print();

cout << endl;

cout << "Code: " << endl;

cout << "Letter a: ";

Tree.code('a');

cout << endl;

cout << "Letter b: ";

Tree.code('b');

cout << endl;

cout << "Letter c: ";

Tree.code('c');

cout << endl;

cout << "Letter d: ";

Tree.code('d');

cout << endl;

cout << "Letter e: ";

Tree.code('e');

cout << endl;

string str;

cout << endl;

cout << "Enter text: ";

getline(cin, str);

Tree.coding(str);

cout << endl << endl;

str = "";

cout << "Enter code: ";

getline(cin, str);

Tree.decoding(str);

return 0;

}

**Результат**

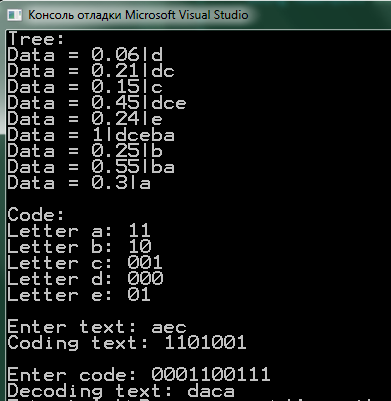
****

Рисунок 3 – результат виконання завдання 4 та 5

**Висновок.** засвоїв теоретичний матеріал та набув практичних навичок по роботі з абстрактним типом даних “Дерево”