Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**Звіт**

**Звіт**

**про виконання лабораторних та практичних робіт блоку № 6**

На тему: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»

***з дисципліни:*** «Мови та парадигми програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт № 6

**Виконав:**

Студент групи ШІ-11

Шляхетко Данило Віталійович

# **Тема роботи:**

Знайомство з динамічними структурами (Списки, Дерево) і алгоритмами обробки динамічних структур.

# **Мета роботи:**

Ознайомитися з динамічними структурами (Списки, Дерево) і алгоритмами обробки динамічних структур.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Класи та структури.
* Тема №2: Лінійні списки, Бінарне дерево.
* Тема №3: Алгоритмами обробки динамічних структур.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Класи та структури.
  + Джерела Інформації
    - Лекції та практичні
    - Попередні знання
  + Що опрацьовано:
    - Як працювати з структурами, класами і темплейтами в С++
  + Статус: Ознайомлений
  + Початок опрацювання теми: 07/12/2023
  + Звершення опрацювання теми: 13/12/2023
* Тема №2: Класи та структури.
  + Джерела Інформації
    - Лекції та практичні
    - Попередні знання
  + Що опрацьовано:
    - Принцип роботи лінійних списків та бінарного дерева
  + Статус: Ознайомлений
  + Початок опрацювання теми: 07/12/2023
  + Звершення опрацювання теми: 13/12/2023
* Тема №3: Алгоритмами обробки динамічних структур.
  + Джерела Інформації
    - Лекції та практичні
    - Попередні знання
  + Що опрацьовано:
    - Створення алгоритмів обробки динамічних структур в С++
  + Статус: Ознайомлений
  + Початок опрацювання теми: 07/12/2023
  + Звершення опрацювання теми: 13/12/2023

# **Виконання роботи:**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №1 VNS Lab 10

* Варіант 5

1. Написати функцію для створення списку. Функція може створювати порожній список, а потім додавати в нього елементи. 2. Написати функцію для друку списку. Функція повинна передбачати вивід повідомлення, якщо список порожній. 3. Написати функції для знищення й додавання елементів списку у відповідності зі своїм варіантом. 4. Виконати зміни в списку й друк списку після кожної зміни. 5. Написати функцію для запису списку у файл. 6. Написати функцію для знищення списку. 7. Записати список у файл, знищити його й виконати друк (при друці повинне бути видане повідомлення "Список порожній"). 8. Написати функцію для відновлення списку з файлу. 9. Відновити список і роздрукувати його. 10.Знищити список.

Записи в лінійному списку містять ключове поле типу int. Сформувати однонаправлений список. Знищити з нього К елементів, починаючи із заданого номера, додати К елементів, починаючи із заданого номера;

Завдання №4 Algotester Lab 5

* Варіант 2
* В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N, ширина - M.

Всередині печери є пустота, пісок та каміння. Пустота позначається буквою О, пісок

S і каміння X.

Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.

Ваше завдання сказати як буде виглядати печера після землетрусу.

Завдання №5 Algotester Lab 7-8

* Варіант 3
* Ваше завдання - власноруч реалізувати структуру даних "Двійкове дерево пошуку".

Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри.

Вам будуть поступати запити такого типу:

Вставка:

Ідентифікатор - insert. Ви отримуєте ціле число value - число, яке треба вставити в дерево.

Пошук:

Ідентифікатор - contains. Ви отримуєте ціле число value - число, наявність якого у дереві необхідно перевірити.

Якщо value наявне в дереві - ви виводите Yes, у іншому випадку No

Визначення розміру:

Ідентифікатор - size. Ви не отримуєте аргументів. Ви виводите кількість елементів у дереві.

Вивід дерева на екран

Ідентифікатор - print. Ви не отримуєте аргументів. Ви виводите усі елементи дерева через пробіл.

Завдання №6 Class Practice Task

* Реалізувати метод реверсу списку: Node\* reverse(Node \*head);

Умови задачі:

- використовувати цілочисельні значення в списку;

- реалізувати метод реверсу;

- реалізувати допоміжний метод виведення вхідного і обернутого списків;

bool compare(Node \*h1, Node \*h2);

Умови задачі:

- використовувати цілочисельні значення в списку;

- реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;

- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає false.

Node\* add(Node \*n1, Node \*n2);

Умови задачі:

- використовувати цифри від 0 до 9 для значень у списку;

- реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. 379 ⟹ 9→7→3);

- функція повертає новий список, передані в функцію списки не модифікуються.

TreeNode \*create\_mirror\_flip(TreeNode \*root);

Умови задачі:

- використовувати цілі числа для значень у вузлах дерева

- реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева

- функція повертає нове дерево, передане в функцію дерево не модифікується

void tree\_sum(TreeNode \*root);

Умови задачі:

- використовувати цілочисельні значення у вузлах дерева;

- реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів

- вузол-листок не змінює значення

- значення змінюються від листків до кореня дерева

Завдання №7 Self Practice Task

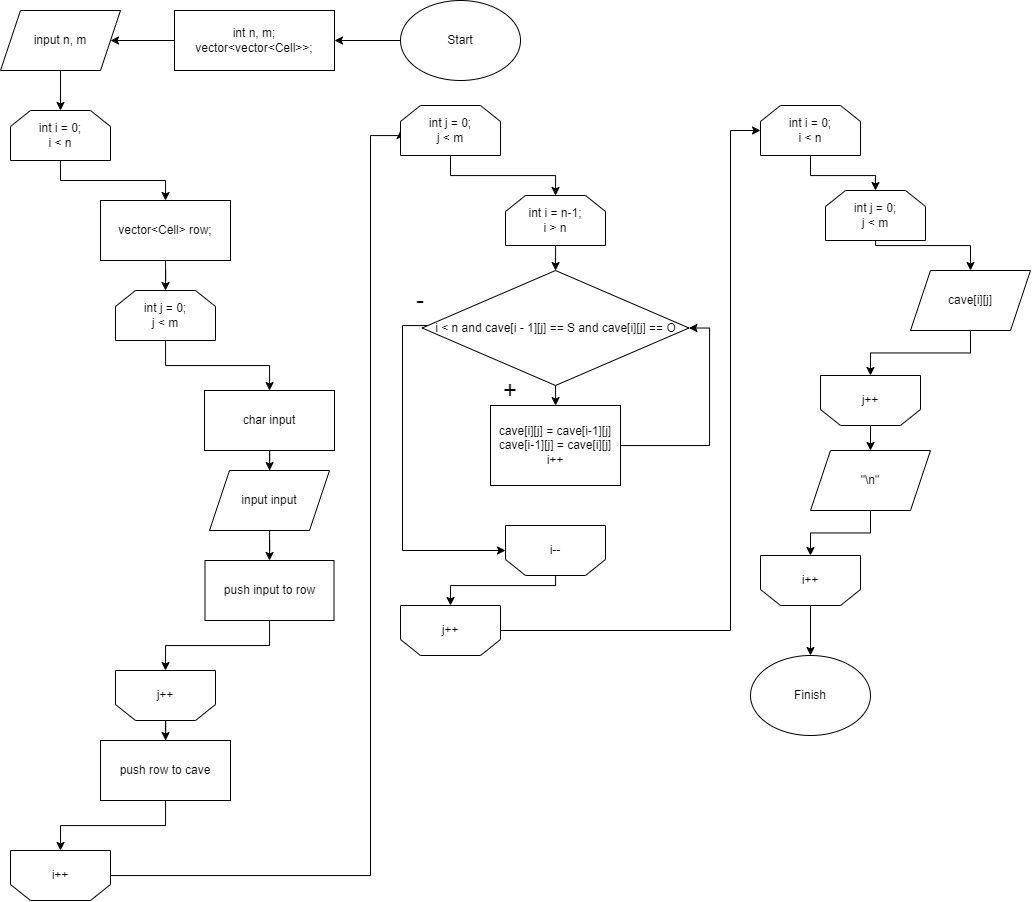
* Написати програму з використанням нового матеріалу

## **2. Дизайн та планована оцінка часу виконання завдань:**

Завдання №1 VNS Lab 10

* Запланований час: 2 год.

Завдання №4 Algotester Lab 5

* 

*Малюнок 1. Flowchart Algotester Lab 5*

* Запланований час: 1 год.

Завдання №5 Algotester Lab 7-8

* Запланований час: 2 год.

Завдання №6 Class Practice Task

* Запланований час: 3 год.

Завдання №7 Self Practice Task

* Запланований час: 1 год.

## **3. Код програм з посиланням на зовнішні ресурси:**

Завдання №1 VNS Lab 10

* <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1039/files#diff-2e1d1df8adb26ae07d4b55857612dc9bd8fcea779f628b1a36669fa7e63eb735>
* #include <iostream>
* #include <fstream>
* #include <ctime>
* using namespace std;
* const string PATH = "C:/files/university/programming-paradigms/data/linkedList.txt";
* class LinkedList
* {
* class Node
* {
* public:
* int data;
* Node \*next;
* };
* private:
* Node \*tail;
* Node \*head;
* int size = 0;
* public:
* int getSize()
* {
* return size;
* }
* Node \*getNodeByIndex(int index)
* {
* if (index < 0 || index >= size)
* {
* return NULL;
* }
* Node \*current = head;
* for (int i = 0; i < index; i++)
* {
* current = current->next;
* }
* return current;
* }
* void addElementByIndex(int value, int index)
* {
* Node \*node = new Node();
* node->data = value;
* node->next = NULL;
* if (node == NULL || index > size || index < 0)
* {
* return;
* }
* if (size == 0)
* {
* head = node;
* tail = node;
* }
* else if (index == 0)
* {
* node->next = head;
* head = node;
* }
* else if (index == size)
* {
* tail->next = node;
* tail = node;
* }
* else
* {
* Node \*current = getNodeByIndex(index - 1);
* node->next = current->next;
* current->next = node;
* }
* size++;
* }
* void removeElementByIndex(int index)
* {
* if (index < 0 || index >= size)
* {
* return;
* }
* if (index == 0)
* {
* head = head->next;
* }
* Node \*node = getNodeByIndex(index);
* Node \*previousNode = getNodeByIndex(index - 1);
* if (index == size - 1)
* {
* tail = previousNode;
* tail->next = NULL;
* }
* else
* {
* previousNode->next = node->next;
* }
* delete node;
* --size;
* }
* void addRandomElementByIndex(int index)
* {
* addElementByIndex(rand() % 100, index);
* }
* void createRandomElementsList(int size)
* {
* for (int i = 0; i < size; i++)
* {
* addRandomElementByIndex(i);
* }
* }
* void print(Node \*n)
* {
* if (n == NULL)
* {
* return;
* }
* cout << n->data << " ";
* }
* void printList(Node \*n)
* {
* if (size == 0)
* {
* cout << "List is empty\n";
* return;
* }
* if (n == NULL)
* {
* cout << "\n";
* return;
* }
* print(n);
* printList(n->next);
* };
* void printList()
* {
* printList(head);
* }
* void deleteFromSomeNumber(int index, int k)
* {
* for (int i = 0; i < k; i++)
* {
* removeElementByIndex(index);
* }
* }
* void addFromSomeNumber(int index, int k)
* {
* for (int i = 0; i < k; i++)
* {
* addRandomElementByIndex(index + i);
* }
* }
* void writeToFile(string path)
* {
* Node \*n = head;
* ofstream deleteFile(path, ios::trunc);
* deleteFile.close();
* ofstream writeFile;
* writeFile.open(path, ios\_base::app);
* if (!writeFile)
* {
* return;
* }
* while (n != NULL)
* {
* writeFile << n->data << "\n";
* n = n->next;
* }
* writeFile.close();
* }
* void deleteList()
* {
* while (head != NULL)
* {
* Node \*next = head->next;
* delete head;
* head = next;
* size--;
* }
* }
* void readFromFile(string path)
* {
* string currentLine;
* ifstream readFile;
* readFile.open(path, ios\_base::app);
* if (!readFile)
* {
* return;
* }
* int i = 0;
* while (getline(readFile, currentLine))
* {
* int parsedOutput = stoi(currentLine);
* addElementByIndex(parsedOutput, i);
* i++;
* }
* readFile.close();
* }
* };
* main()
* {
* int from;
* int k;
* LinkedList list;
* srand(time(0));
* cout << "Create List:\n";
* list.createRandomElementsList(10);
* list.printList();
* cout << "Enter from which element start deleting:\n";
* cin >> from;
* cout << "Enter number of elements to delete:\n";
* cin >> k;
* cout << "Delete elements from List:\n";
* list.deleteFromSomeNumber(from, k);
* list.printList();
* cout << "Enter from which element start adding:\n";
* cin >> from;
* cout << "Enter number of elements to add:\n";
* cin >> k;
* cout << "Add elements to List:\n";
* list.addFromSomeNumber(from, k);
* list.printList();
* cout << "Write List to file:\n";
* list.writeToFile(PATH);
* cout << "Delete List:\n";
* list.deleteList();
* list.printList();
* cout << "Read List from file:\n";
* list.readFromFile(PATH);
* list.printList();
* return 0;
* }

Завдання №4 Algotester Lab 5

* <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1039/files#diff-ea4e652dbbaee291522fbd470083de9b1ee4a9d1ad62e20f339755420190bb71>
* #include <iostream>
* #include <vector>
* using namespace std;
* enum Cell
* {
* O,
* S,
* X,
* };
* int main()
* {
* vector<vector<Cell>> cave;
* int n, m;
* cin >> n >> m;
* for (int i = 0; i < n; i++)
* {
* vector<Cell> row;
* for (int j = 0; j < m; j++)
* {
* char input;
* cin >> input;
* Cell cell = (input == 'S' ? S : (input == 'X' ? X : O));
* row.push\_back(cell);
* }
* cave.push\_back(row);
* }
* for (int j = 0; j < m; j++)
* {
* for (int i = n - 1; i > 0; i--)
* {
* while (i < n && cave[i - 1][j] == S && cave[i][j] == O)
* {
* Cell buff = cave[i][j];
* cave[i][j] = cave[i - 1][j];
* cave[i - 1][j] = buff;
* i++;
* }
* }
* }
* for (int i = 0; i < n; i++)
* {
* for (int j = 0; j < m; j++)
* {
* char output = (cave[i][j] == S ? 'S' : (cave[i][j] == X ? 'X' : 'O'));
* cout << output;
* }
* cout << "\n";
* }
* return 0;
* }

Завдання №5 Algotester Lab 7-8

* <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1039/files#diff-be81aa56c65ddbc1afb1b34131a4881140fc738a2fcf88b55c7d270777ba5573>

#include <iostream>

#include <vector>

using namespace std;

struct Input

{

string command;

int argument;

};

template <class T>

class BinaryTree

{

class Node

{

public:

T data;

Node \*left;

Node \*right;

Node \*parent;

};

private:

Node \*head = NULL;

int size = 0;

public:

int getSize()

{

return size;

}

void addElement(Node \*branchHead, Node \*node)

{

if (node == NULL)

return;

if (branchHead == NULL)

{

head = node;

size++;

return;

}

Node \*current = branchHead;

if (current->data < node->data)

{

if (current->right == NULL)

{

size++;

current->right = node;

node->parent = current;

node->left = NULL;

node->right = NULL;

return;

}

addElement(current->right, node);

}

else if (current->data > node->data)

{

if (current->left == NULL)

{

size++;

current->left = node;

node->parent = current;

node->left = NULL;

node->right = NULL;

return;

}

addElement(current->left, node);

}

}

void addElement(T value)

{

Node \*node = new Node();

node->data = value;

addElement(head, node);

}

void print(Node \*node)

{

if (node == NULL)

return;

print(node->left);

cout << node->data << " ";

print(node->right);

}

void doesContain(Node \*node, T value)

{

Node \*current = node;

if (current == NULL)

{

cout << "No";

return;

}

else

{

if (current->data == value)

{

cout << "Yes";

return;

}

else

{

if (current->data < value)

{

doesContain(current->right, value);

}

else

{

doesContain(current->left, value);

}

}

}

}

void doesContain(T value)

{

doesContain(head, value);

}

void print()

{

print(head);

}

};

int main()

{

int q;

cin >> q;

BinaryTree<int> tree;

vector<Input> inputs;

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

{

string command;

cin >> command;

Input inp;

inp.command = command;

if (command == "insert" || command == "contains")

{

int input;

cin >> input;

inp.argument = input;

}

inputs.push\_back(inp);

}

for (int i = 0; i < inputs.size(); i++)

{

string command = inputs[i].command;

if (command == "size")

{

cout << tree.getSize() << "\n";

}

else if (command == "print")

{

tree.print();

cout << "\n";

}

else if (command == "insert" || command == "contains")

{

int argument = inputs[i].argument;

if (command == "insert")

{

tree.addElement(argument);

}

else

{

tree.doesContain(argument);

cout << "\n";

}

}

}

return 0;

}

Завдання №6 Class Practice Task

* Task 1-3
* <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1039/files#diff-a5f7c0cd1cdaea6c6ac5090df29962de071d0d3868bcdd260398362fd4d2dd5e>

#include <iostream>

using namespace std;

class ReverseList

{

class Node

{

public:

int data;

Node \*next;

Node \*previous;

};

private:

Node \*tail;

Node \*head;

int size = 0;

Node \*getNodeByIndex(int index)

{

if (index < 0 || index >= size)

{

return NULL;

}

Node \*current = head;

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

{

current = current->next;

}

return current;

}

void addElementByIndex(int value, int index)

{

Node \*node = new Node();

node->data = value;

node->next = NULL;

if (node == NULL || index > size || index < 0)

{

return;

}

if (size == 0)

{

head = node;

tail = node;

}

else if (index == 0)

{

node->next = head;

head->previous = node;

head = node;

}

else if (index == size)

{

node->previous = tail;

tail->next = node;

tail = node;

}

else

{

Node \*current = getNodeByIndex(index - 1);

node->previous = current;

node->next = current->next;

current->next->previous = node;

current->next = node;

}

size++;

}

void removeElementByIndex(int index)

{

if (index < 0 || index >= size)

{

return;

}

if (index == 0)

{

head = head->next;

}

Node \*node = getNodeByIndex(index);

Node \*previousNode = getNodeByIndex(index - 1);

if (index == size - 1)

{

tail = previousNode;

tail->next = NULL;

}

else

{

previousNode->next = node->next;

}

delete node;

--size;

}

void addRandomElementByIndex(int index)

{

addElementByIndex(rand() % 100, index);

}

void addRandomDigitByIndex(int index)

{

addElementByIndex(rand() % 10, index);

}

void print(Node \*n)

{

if (n == NULL)

{

return;

}

cout << n->data << " ";

}

void printNumber(Node \*n)

{

if (n == NULL)

{

return;

}

cout << n->data;

}

void printList(Node \*n)

{

if (size == 0)

{

cout << "List is empty\n";

return;

}

if (n == NULL)

{

cout << "\n";

return;

}

print(n);

printList(n->next);

};

void printNumberList(Node \*n)

{

if (size == 0)

{

cout << "List is empty\n";

return;

}

if (n == NULL)

{

cout << "\n";

return;

}

printNumber(n);

printNumberList(n->previous);

};

void printReversedList(Node \*n)

{

if (size == 0)

{

cout << "List is empty\n";

return;

}

if (n == NULL)

{

cout << "\n";

return;

}

print(n);

printReversedList(n->previous);

};

void reverse()

{

Node \*current = head;

swap(head, tail);

Node \*temp = NULL;

while (current != NULL)

{

temp = current->next;

current->next = current->previous;

current->previous = temp;

current = temp;

}

}

bool compare(Node \*h1, Node \*h2)

{

while (h1->next != NULL || h2->next != NULL)

{

if (h1->data != h2->data)

{

return false;

}

h1 = h1->next;

h2 = h2->next;

}

if (h1->next != h2->next)

return false;

return true;

}

Node \*getHead()

{

return head;

}

ReverseList sum(Node \*h1, Node \*h2, ReverseList answer)

{

int index = 0;

int intBuffer = 0;

Node \*a = h1;

Node \*b = h2;

while (a != nullptr && b != nullptr)

{

int s = b->data + a->data + intBuffer;

if ((s) < 10)

{

answer.addElementByIndex(s, index);

intBuffer = 0;

}

else

{

answer.addElementByIndex(s - 10, index);

intBuffer = 1;

}

a = a->next;

b = b->next;

index++;

}

if (intBuffer)

{

answer.addElementByIndex(intBuffer, index);

}

return answer;

}

public:

int getSize()

{

return size;

}

void createRandomElementsList(int size)

{

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

{

addRandomElementByIndex(i);

}

}

void createRandomDigitList(int size)

{

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

{

addRandomDigitByIndex(i);

}

}

void addNewElement(int value)

{

addElementByIndex(value, 0);

}

void printList()

{

printList(head);

}

void printReversedList()

{

printReversedList(tail);

}

void printNumberList()

{

printNumberList(tail);

}

void reverseList()

{

reverse();

}

bool compareList(ReverseList list)

{

return compare(head, list.getHead());

}

ReverseList sum(ReverseList a, ReverseList b)

{

ReverseList c;

return sum(a.getHead(), b.getHead(), c);

}

};

int main()

{

cout << "Task 1\n";

ReverseList list;

list.createRandomElementsList(10);

list.printList();

list.reverseList();

list.printList();

list.reverseList();

list.printList();

list.printReversedList();

cout << "\nTask 2\n";

ReverseList newList;

newList.createRandomElementsList(10);

newList.printList();

cout << boolalpha;

cout << "list and newList: " << list.compareList(newList) << endl;

cout << "list and list: " << list.compareList(list) << endl;

cout << "\nTask 3\n";

ReverseList listA;

listA.createRandomDigitList(10);

*// listA.addNewElement(3);*

*// listA.addNewElement(4);*

*// listA.addNewElement(6);*

*// listA.addNewElement(8);*

ReverseList listB;

listB.createRandomDigitList(10);

*// listB.addNewElement(5);*

*// listB.addNewElement(7);*

*// listB.addNewElement(8);*

*// listB.addNewElement(8);*

listA.printNumberList();

listB.printNumberList();

ReverseList listC;

listC = listC.sum(listA, listB);

listC.printNumberList();

return 0;

}

* Task 4-5
* <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1039/files#diff-eff088f4a6f7bfe034aebda573a91305193d78c4e638b34e578661e52f479c43>

#include <iostream>

#include <vector>

using namespace std;

template <class T>

class BinaryTree

{

class Node

{

public:

T data;

Node \*left;

Node \*right;

Node \*parent;

};

private:

Node \*head = NULL;

int size = 0;

public:

int getSize()

{

return size;

}

void addElement(Node \*branchHead, Node \*node)

{

if (node == NULL)

return;

if (branchHead == NULL)

{

head = node;

size++;

return;

}

Node \*current = branchHead;

if (current->data < node->data)

{

if (current->right == NULL)

{

size++;

current->right = node;

node->parent = current;

node->left = NULL;

node->right = NULL;

return;

}

addElement(current->right, node);

}

else if (current->data > node->data)

{

if (current->left == NULL)

{

size++;

current->left = node;

node->parent = current;

node->left = NULL;

node->right = NULL;

return;

}

addElement(current->left, node);

}

}

void addElement(T value)

{

Node \*node = new Node();

node->data = value;

addElement(head, node);

}

void print(Node \*node)

{

if (node == NULL)

return;

print(node->left);

cout << node->data << " ";

print(node->right);

}

void doesContain(Node \*node, T value)

{

Node \*current = node;

if (current == NULL)

{

cout << "No";

return;

}

else

{

if (current->data == value)

{

cout << "Yes";

return;

}

else

{

if (current->data < value)

{

doesContain(current->right, value);

}

else

{

doesContain(current->left, value);

}

}

}

}

void doesContain(T value)

{

doesContain(head, value);

}

void print()

{

print(head);

}

Node \*mirrorTree(Node \*node)

{

if (node == NULL)

{

return node;

}

Node \*left = mirrorTree((node->left));

Node \*right = mirrorTree((node->right));

node->left = right;

node->right = left;

return node;

}

void mirrorTree()

{

mirrorTree(head);

}

Node \*sum(Node \*node)

{

if (node == NULL)

{

return node;

}

Node \*right = sum((node->right));

Node \*left = sum((node->left));

if (left != NULL && right != NULL)

{

node->data = left->data + right->data;

}

else if (right == NULL && left != NULL)

{

node->data = left->data;

}

else if (left == NULL && right != NULL)

{

node->data = right->data;

}

return node;

}

void sum()

{

sum(head);

}

};

int main()

{

cout << "Task 4\n";

BinaryTree<int> tree;

srand(time(0));

for (int i = 0; i < 10; i++)

{

tree.addElement(rand() % 100);

}

tree.print();

cout << "\n";

tree.mirrorTree();

tree.print();

cout << "\nTask 5\n";

tree.print();

cout << "\n";

tree.sum();

tree.print();

return 0;

}

Завдання №7 Self Practice Task

* <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1039/files#diff-bced0759f06698a2947f851f82a10ceae5653b9bf756c1353fb697e78d397329>

#include <iostream>

#include <vector>

using namespace std;

int main()

{

int n, m, x, y;

cin >> n >> m >> x >> y;

int map[n][m];

int maxY = (y > m - y) ? y - 1 : m - y;

int maxX = (x > n - x) ? x - 1 : n - x;

int peak = maxX + maxY;

y--;

x--;

map[x][y] = peak;

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

{

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

{

if (x - i >= 0 && x - i < n && y - j >= 0 && y - j < m)

map[x - i][y - j] = peak - i - j;

if (x + i >= 0 && x + i < n && y + j >= 0 && y + j < m)

map[x + i][y + j] = peak - i - j;

if (x + i >= 0 && x + i < n && y - j >= 0 && y - j < m)

map[x + i][y - j] = peak - i - j;

if (x - i >= 0 && x - i < n && y + j >= 0 && y + j < m)

map[x - i][y + j] = peak - i - j;

}

}

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

{

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

{

cout << map[i][j] << " ";

}

cout << "\n";

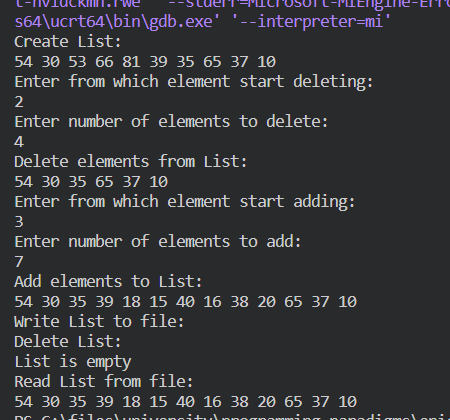
}

return 0;

}

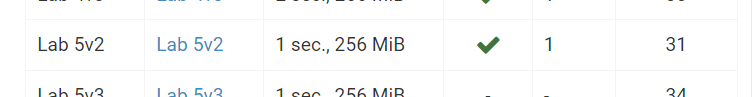
## **4. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1 VNS Lab 10

* Затрачений час: 3 год.
* 

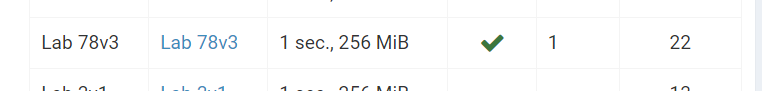
*Малюнок 2. Results VNS Lab 10*

Завдання №4 Algotester Lab 5

* Затрачений час: 40 хв.
* 

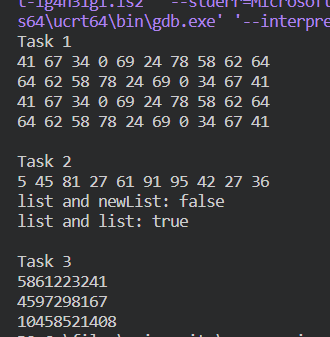
*Малюнок 3. Results Algotester Lab 5*

Завдання №5 Algotester Lab 7-8

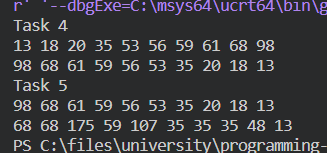
* Затрачений час: 2 год. 30 хв.
* 

*Малюнок 4. Results Algotester Lab 7-8*

Завдання №6 Class Practice Task

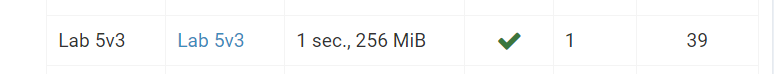
* Затрачений час: 3 год.
* 

*Малюнок 5. Class Practice Task 1-3*

* 

*Малюнок 6. Class Practice Task 4-5*

Завдання №7 Self Practice Task

* Затрачений час: 1 год.
* 

*Малюнок 7. Class Practice Task 4-5*

# **Висновки:**

Ознайомився з:

* динамічними структурами (Списки, Дерево)
* алгоритмами обробки динамічних структур.

З допомогою здобутих знань реалізував код програм заданих в поданому епіку.