Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра систем штучного інтелекту

Звіт

про виконання лабораторних та практичних робіт блоку № 6

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

з дисципліни: «Основи програмування»

до:

ВНС Лабораторної Роботи № 10
Алготестер Лабораторної Роботи №5
Алготестер Лабораторної Роботи № 7-8
Практичних робіт до блоку №6

Виконала:

Студентка групи ШІ-11

Потапова Світлана Сергіївна

Львів 2024

Тема роботи: Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур

Мета роботи:

- Ознайомитися з динамічними структурами: чергою, стеком, списками, деревами; навчитися працювати з ними та застосовувати різноманітні алгоритми обробки динамічних структур

Теоретичні відомості:

1. Теоретичні відомості з переліком важливих тем:

- Тема №1. Основи динамічних структур даних
- Тема №2. Стек
- Тема №3. Черга
- Тема №4. Зв'язні списки
- Тема №5. Дерева
- Тема №6. Алгоритми обробки динамічних структур

2. Індивідуальний план опрацювання теорії:

Тема №1. Основи динамічних структур даних
- Джерела інформації:
  - https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation/
  - https://www.geeksforgeeks.org/stack-data-structure/
  - https://www.geeksforgeeks.org/heap-data-structure/
- Що опрацьовано:
  - Поняття stack, heap, прості динамічні структури
- Статус: ознайомлена

Тема №2. Стек
- Джерела інформації:
  - https://www.geeksforgeeks.org/stack-data-structure/
- Що опрацьовано:
  - Стек, операції над ним, приклади використання
- Статус: ознайомлена

Тема №3. Черга
- Джерела інформації:
  - https://www.w3schools.com/cpp/cpp_queues.asp
  - https://www.w3schools.com/cpp/cpp_deque.asp
  - https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/
- Що опрацьовано:

        - Черги, їх функціонал, використання, пріоритетні черги
    - Статус: ознайомлена

    Тема №4. Зв’язні списки
    - Джерела інформації:
        - https://www.geeksforgeeks.org/cpp-linked-list/
    - Що опрацьовано:
        - Зв’язні списки та їх види, операції над ними, використання
    - Статус: ознайомлена

    Тема №5. Дерева
    - Джерела інформації:
        - https://www.geeksforgeeks.org/binary-tree-in-cpp/
    - Що опрацьовано:
        - Бінарні дерева, обхід дерева, основні операції, застосування
    - Статус: ознайомлена

    Тема №6. Алгоритми обробки динамічних структур
    - Джерела інформації:
        - https://www.geeksforgeeks.org/recursion-algorithms/
    - Що опрацьовано:
        - Ітеративні, рекурсивні алгоритми
    - Статус: ознайомлена

Виконання роботи:

1. Опрацювання завдань та вимог до середовища:
   Завдання №1. VNS lab 10 variant 15.

   Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити K елементів з кінця списку. Додати елемент після елемента із заданим ключем.

   Завдання №2. Algotester lab 5 variant 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 O, 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.

Вхідні дані
У першому рядку 2 цілих числа N та M - висота та ширина печери
У N наступних рядках стрічка row_i яка складається з N
цифр - i-й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані
N рядків, які складаються з стрічки розміром M - стан печери після землетрусу.

Завдання №3. Algotester lab 7-8 variant 1

Ваше завдання - власноруч реалізувати структуру даних "Двозв’язний список". Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

Вам будуть поступати запити такого типу:
Вставка:
Ідентифікатор - insert
Ви отримуєте ціле число index елемента, на місце якого робити вставку.
Після цього в наступному рядку рядку написане число N - розмір списку, який треба вставити.
У третьому рядку N цілих чисел - список, який треба вставити на позицію index

Видалення:
Ідентифікатор - erase
Ви отримуєте 2 цілих числа - index, індекс елемента, з якого почати видалення та n кількість елементів, яку треба видалити.

Визначення розміру:
Ідентифікатор - size
Ви не отримуєте аргументів.
Ви виводите кількість елементів у списку.

Отримання значення i-го елементу:
Ідентифікатор - get
Ви отримуєте ціле число - index, індекс елемента.
Ви виводите значення елемента за індексом.

Модифікація значення i-го елементу:
Ідентифікатор - set

Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.

Вивід списку на екран:
Ідентифікатор - print
Ви не отримуєте аргументів. Ви виводите усі елементи списку через пробіл. Реалізувати використовуючи перегрузку оператора <<

Вхідні дані
Ціле число Q - кількість запитів.
У наступних рядках Q запитів у зазначеному в умові форматі.

Вихідні дані
Відповіді на запити у зазначеному в умові форматі.

Завдання №4. Class Practice Task

Задача №1 - Реверс списку (Reverse list)
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;

Задача №2 - Порівняння 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*.

Задача №3 – Додавання великих чисел
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 $\implies$ 9→7→3);

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

Задача №4 - Віддзеркалення дерева
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

Задача №5 - Записати кожному батьківському вузлу суму підвузлів
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 №5. Self-Practice task Algotester lab 5 variant 3

У вас є карта гори розміром N×M.
Також ви знаєте координати {x,y}, у яких знаходиться вершина гори.
Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.
Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

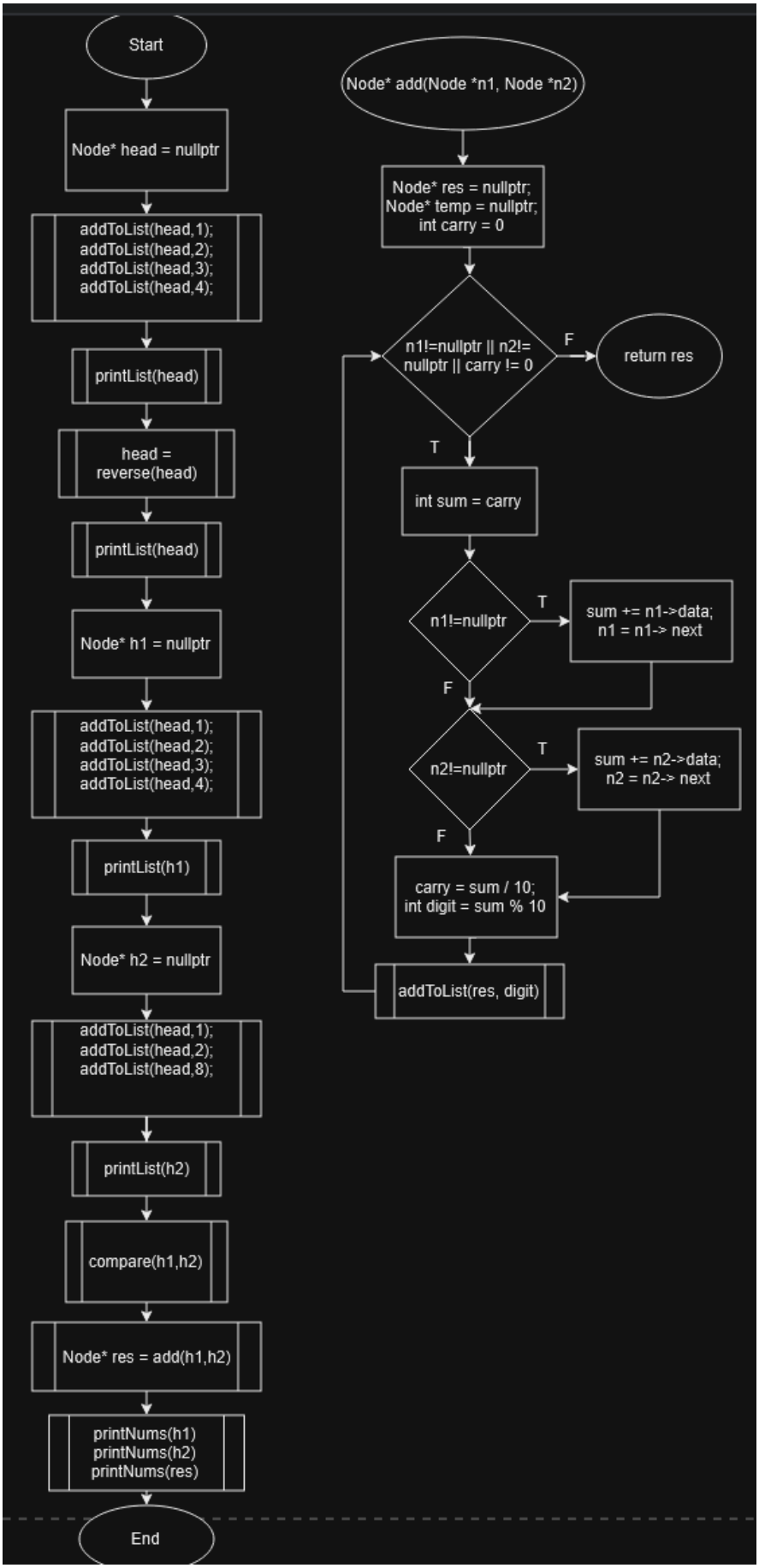
Вхідні дані

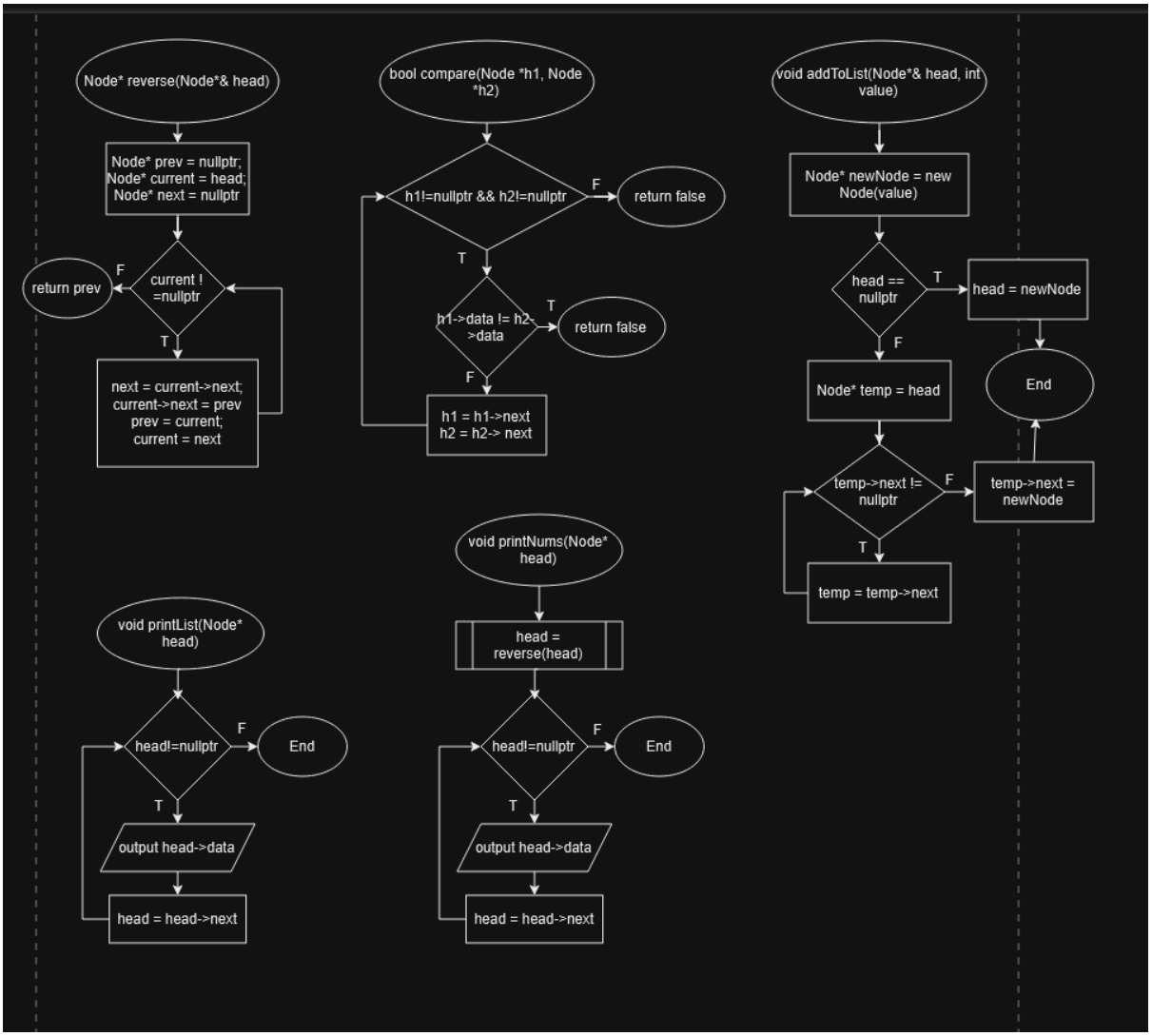
У першому рядку 2 числа N та M - розміри карти, у другому рядку 2 числа x та y - координати піку гори

Вихідні дані
N рядків по M елементів в рядку через пробіл - висоти карти.

2. Дизайн виконання завдань

Завдання №4. Class Practice Task 1-3





3. Код програм з посиланням на зовнішні ресурси

Завдання №1. VNS lab 10 variant 15.

```cpp
#include <iostream>
#include <cstring>
#include <fstream>
using namespace std;

struct Node{
    char data[100];
    Node* next;
    Node* prev;
};

void addToList(Node*& head, Node*& tail, const char* key){
    Node* newNode = new Node;
    strcpy(newNode->data, key);
    newNode->next = nullptr;
    newNode->prev = nullptr;

    if(head == nullptr){
        head = newNode;
        tail = newNode;
    } else{
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

void printList(Node* head){
    if(head == nullptr){
        cout << "The list is empty" << endl;
        return;
    }

```

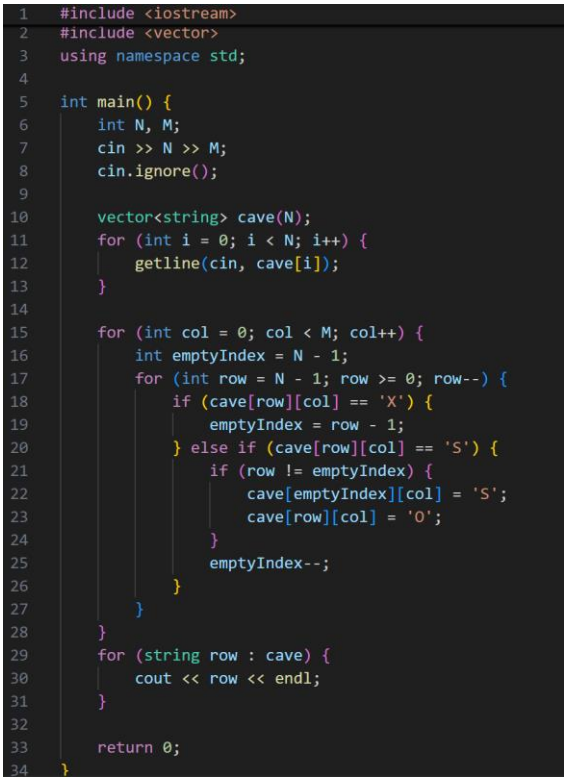
```
    while(head != nullptr){
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

void deleteElements(Node*& head, Node*& tail, int K){
    while(K>0 && tail!= nullptr){
        Node* del = tail;
        tail = tail->prev;

        if(tail != nullptr){
            tail->next = nullptr;
        } else{
            head = nullptr;
        }

        delete del;
        K--;
    }
}

void addAfterKey(Node*& head, Node*& tail, const char* key, const char* toAdd){
    Node* temp = head;

    while(temp != nullptr &&  strcmp(temp->data, key) != 0){
        temp = temp->next;
    }

    if(temp == nullptr){
        cout << key << " is not found in the list" << endl;
```

```
            return;
        }

        Node* newNode = new Node;
        strcpy(newNode->data, toAdd);

        newNode->next = temp->next;
        newNode ->prev = temp;

        if(temp->next != nullptr){
            temp->next->prev = newNode;
        } else{
            tail = newNode;
        }

        temp->next = newNode;
    }

    void writeToFile(Node*& head, const char* filename){
        ofstream f(filename);
        if (!f.is_open()){
            cout << "Error oppening file" << endl;
            return;
        }

        while(head != nullptr){
            f << head->data << " ";
            head = head->next;
        }

        f.close();
```

```
}

void deleteList(Node*& head){
    while(head != nullptr){
        Node* temp = head;
        head = head->next;
        delete temp;
    }

    cout << "The list was deleted" << endl;
}

void restoreList(Node*& head, Node*& tail, const char* filename){
    ifstream f(filename);
    if (!f.is_open()){
        cout << "Error oppening file" << endl;
        return;
    }

    char key[100];
    while(f >> key){
        addToList(head, tail, key);
    }

    f.close();
}

int main(){
    Node* head = nullptr;
    Node* tail = nullptr;
    const char* filename = "DoublyLinkedList.txt";
    int K;
    cout << "Enter K:" << endl;
    cin >> K;

    addToList(head, tail, "Paris");
    addToList(head, tail, "Amsterdam");
    addToList(head, tail, "Vienna");
    addToList(head, tail, "London");

    printList(head);

    cout << "The list after deleting elements: " << endl;
    deleteElements(head, tail, K);
    printList(head);

    cout << "The list after adding element: " << endl;
    addAfterKey(head, tail, "Paris", "Berlin");
    printList(head);

    writeToFile(head, filename);
    deleteList(head);
    printList(head);

    cout << "The list was restored: " << endl;
    restoreList(head, tail, filename);
    printList(head);
    deleteList(head);


    return 0;
}
```

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/604/files#diff-da7ccc4c40b2d525523bc768591eebe6bfb3c7973a04be4d16aafc7e4eecaca4

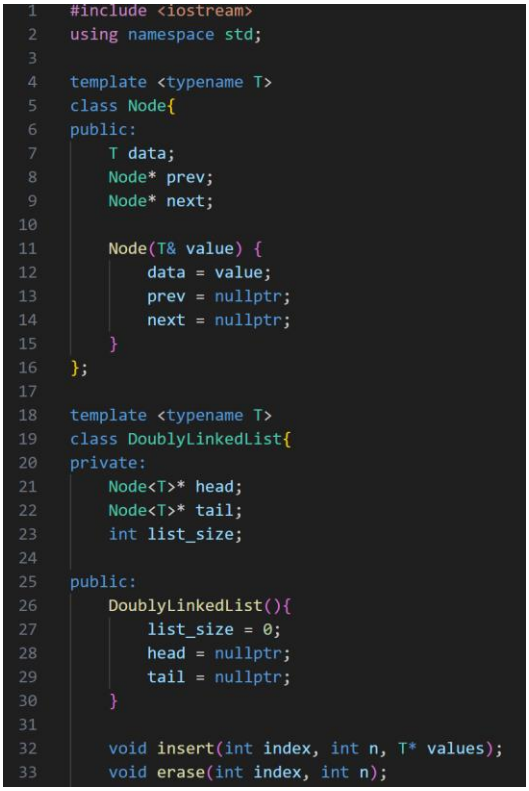
Завдання №2. Algotester lab 5 variant 2.



```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int N, M;
    cin >> N >> M;
    cin.ignore();

    vector<string> cave(N);
    for (int i = 0; i < N; i++) {
        getline(cin, cave[i]);
    }

    for (int col = 0; col < M; col++) {
        int emptyIndex = N - 1;
        for (int row = N - 1; row >= 0; row--) {
            if (cave[row][col] == 'X') {
                emptyIndex = row - 1;
            } else if (cave[row][col] == 'S') {
                if (row != emptyIndex) {
                    cave[emptyIndex][col] = 'S';
                    cave[row][col] = 'O';
                }
                emptyIndex--;
            }
        }
    }
    for (string row : cave) {
        cout << row << endl;
    }

    return 0;
}
```

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/604/files#diff-82b33139d62e28d9829670b352eb0e30669e62070997740b7c11df583e454f3b

Завдання №3. Algotester lab 7-8 variant 1



```cpp
#include <iostream>
using namespace std;

template <typename T>
class Node{
public:
    T data;
    Node* prev;
    Node* next;

    Node(T& value) {
        data = value;
        prev = nullptr;
        next = nullptr;
    }
};

template <typename T>
class DoublyLinkedList{
private:
    Node<T>* head;
    Node<T>* tail;
    int list_size;

public:
    DoublyLinkedList(){
        list_size = 0;
        head = nullptr;
        tail = nullptr;
    }

    void insert(int index, int n, T* values);
    void erase(int index, int n);
```

```
    int size();
    T get(int index);
    void set(int index, T value);
    void print();

    friend ostream& operator<<(ostream& os, const DoublyLinkedList& list) {
        Node<T>* current = list.head;

        while(current) {
            os << current->data << " ";
            current = current->next;
        }
        return os;
    }
};

template<typename T>
void DoublyLinkedList<T>::insert(int index, int n, T* values){
    Node<T>* current = head;
    Node<T>* prevNode = nullptr;

    for (int i = 0; i < index; i++) {
        prevNode = current;
        current = current->next;
    }

    for (int i = 0; i < n; ++i){
        Node<T>* newNode = new Node<T>(values[i]);

        if(!prevNode){
            newNode->next = head;
            if (head) head->prev = newNode;
                else tail = newNode;
                head = newNode;
            } else{
                newNode->next = current;
                newNode->prev = prevNode;

                prevNode->next = newNode;
                if(current) current->prev = newNode;
                else tail = newNode;
            }
            prevNode = newNode;
        }
        list_size += n;
    }

    template<class T>
    void DoublyLinkedList<T>::erase(int index, int n){
        Node<T>* current = head;

        for(int i=0; i<index; i++) {
            current = current->next;
        }

        for(int i=0; i<n; i++){
            Node<T>* temp = current;

            if(current->prev){
                current->prev->next = current->next;
            }
            else head = current->next;
```

```
96
97          if(current->next){
98              current->next->prev = current->prev;
99          }
100         else tail = current->prev;
101
102         current = current->next;
103
104         delete temp;
105     }
106     list_size -= n;
107 }
108
109 template<typename T>
110 int DoublyLinkedList<T>::size() {
111     return list_size;
112 }
113
114 template<typename T>
115 T DoublyLinkedList<T>::get(int index){
116     Node<T>* current = head;
117
118     for(int i=0; i<index; i++) {
119         current = current->next;
120     }
121
122     return current->data;
123 }
124
125 template<typename T>
126 void DoublyLinkedList<T>:: set(int index, T value){
127     Node<T>* current = head;
128
129     for(int i=0; i<index; i++){
130         current = current->next;
131     }
132
133     current->data = value;
134 }
135
136 int main(){
137     DoublyLinkedList<int> list;
138     int Q;
139     cin >> Q;
140
141     for (int i=0; i<Q; i++){
142         string option;
143         cin >> option;
144
145         if(option == "insert"){
146             int index, N;
147             cin >> index >> N;
148
149             int *values = new int[N];
150             for(int i=0; i<N; i++){
151                 cin >> values[i];
152             }
153             list.insert(index, N, values);
154
155             delete[] values;
156         } else if(option == "erase"){
157             int index, n;
158             cin >> index >> n;
159             list.erase(index, n);
```

```
160             } else if(option == "size"){
161                 cout << list.size() << endl;
162             } else if(option == "get"){
163                 int index;
164                 cin >> index;
165                 cout << list.get(index) << endl;
166             } else if(option == "set"){
167                 int index, value;
168                 cin >> index >> value;
169                 list.set(index, value);
170             } else if(option == "print"){
171                 cout << list << endl;
172             }
173 
174         }
175     }
```

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/604/files#diff-3e39bdc32da3c12f6cd70bf3e89d7423b53241ed56a363bfb815112336770d20

Завдання №4. Class Practice Task

```
1   #include <iostream>
2   using namespace std;
3   
4   struct Node{
5       int data;
6       Node* next;
7   
8       Node(int value) : data(value), next(nullptr) {}
9   };
10  
11  
12  Node* reverse(Node*& head){ // task 1
13      Node* prev = nullptr;
14      Node* current = head;
15      Node* next = nullptr;
16  
17      while(current!=nullptr){
18          next = current->next;
19          current->next = prev;
20          prev = current;
21          current = next;
22      }
23  
24      return prev;
25  }
26  
27  bool compare(Node *h1, Node *h2){ // task 2
28      while(h1!=nullptr && h2!=nullptr){
29          if(h1->data != h2->data){
30              return false;
31          }
32          h1 = h1->next;
33          h2 = h2->next;
```

```
    }
    return h1 == nullptr && h2 == nullptr;
}

void addToList(Node*& head, int value){
    Node* newNode = new Node(value);

    if(head == nullptr){
        head = newNode;
    } else{
        Node* temp = head;
        while (temp->next != nullptr){
            temp = temp->next;
        }
        temp->next = newNode;
    }

}

Node* add(Node *n1, Node *n2){ // task 3
    Node* res = nullptr;
    Node* temp = nullptr;
    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry != 0){
        int sum = carry;
        if (n1 != nullptr) {
            sum += n1->data;
            n1 = n1->next;
        }

        if (n2 != nullptr) {
            sum += n2->data;
            n2 = n2->next;
        }

        carry = sum / 10;
        int digit = sum % 10;

        addToList(res, digit);
    }

    return res;
}

void printList(Node* head){
    while(head != nullptr){
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

void printNums(Node* head){
    head = reverse(head);
    while(head != nullptr){
        cout << head->data;
        head = head->next;
    }
    cout << endl;
```

```
}

int main() {
    Node* head = nullptr;
    addToList(head, 1);
    addToList(head, 2);
    addToList(head, 3);
    addToList(head, 4);

    cout << "Task 1" << endl;
    printList(head);

    head = reverse(head);

    cout << "Reversed list: " << endl;
    printList(head);

    Node* h1 = nullptr;
    addToList(h1, 1);
    addToList(h1, 2);
    addToList(h1, 3);
    addToList(h1, 4);

    cout << endl << "Task 2" << endl;
    cout << "First list: " << endl;
    printList(h1);

    Node* h2 = nullptr;
    addToList(h2, 1);
    addToList(h2, 2);
    addToList(h2, 8);

    cout << "Second list: " << endl;
    printList(h2);

    if (compare(h1,h2)){
        cout << "The lists are equal" << endl;
    }
    else{
        cout << "The lists are not equal" << endl;
    }

    cout << endl << "Task 3" << endl;
    Node* res = add(h1, h2);

    cout << "First number: ";
    printNums(h1);
    cout << "Second number: ";
    printNums(h2);
    cout << "Sum: ";
    printNums(res);

    return 0;
}
```

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/604/files#diff-0563be18d5f8ea16a66522ffd9103060795abaad31413746afc554d22806a278

```cpp
#include <iostream>
#include <stack>
using namespace std;

struct TreeNode{
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int value): data(value), left(nullptr), right(nullptr) {};
};

void printTree(TreeNode* root) {
    if (root != nullptr) {
        cout << root->data << " ";
        printTree(root->left);
        printTree(root->right);
    }
}

TreeNode *create_mirror_flip(TreeNode *root){
    if (root == nullptr) {
        return nullptr;
    }

    TreeNode* left = create_mirror_flip(root->left);
    TreeNode* right = create_mirror_flip(root->right);

    root->left = right;
    root->right = left;

    return root;
}
```

```cpp

int tree_sum(TreeNode* root) {

    if (root == nullptr) {
        return 0;
    }

    int leftSum = tree_sum(root->left);
    int rightSum = tree_sum(root->right);

    if (root->left != nullptr || root->right != nullptr) {
        root->data += leftSum + rightSum;
    }

    return root->data;
}

int main(){
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(4);
    root->left->left = new TreeNode(8);
    root->right->left = new TreeNode(9);
    root->right->right = new TreeNode(7);

    cout << "Task 4 " << endl;
    printTree(root);

    root = create_mirror_flip(root);
    cout << endl << "Mirrotred tree: " << endl;
    printTree(root);
```

```
65
66      cout << endl << "Task 5" << endl;
67      tree_sum(root);
68
69      cout << "Tree sum: ";
70      printTree(root);
71      cout << endl;
72
73      return 0;
74  }
```

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/604/files#diff-1df3f191db53c30dd97cd1fcdc9f0062974fb56f6e234606f2cc93b1cd34197a

Завдання №5. Self-Practice task Algotester lab 5 variant 3

```cpp
#include <iostream>
#include <queue>
#include <vector>
#include <algorithm>
using namespace std;

struct Cell{
    int x, y, height;
};

int main(){
    int N, M, x, y;
    cin >> N >> M;
    cin >> x >> y;

    x--;
    y--;

    int peak_height = max(x, N - 1 - x) + max(y, M - 1 - y);

    vector<vector<int>> map(N, vector<int>(M, -1));

    vector<pair<int,int>> directions = {{-1,0}, {1,0}, {0,-1}, {0,1}};

    queue<Cell> qu;
    qu.push({x,y,peak_height});
    map[x][y] = peak_height;

    while(!qu.empty()){
        Cell current = qu.front();
        qu.pop();

        for (auto [dx, dy] : directions) {
```

```
            int X = current.x + dx;
            int Y = current.y + dy;

            if (X >= 0 && X < N && Y >= 0 && Y < M && map[X][Y] == -1) {
                map[X][Y] = current.height - 1;
                qu.push({X, Y, current.height - 1});
            }
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cout << map[i][j] << " ";
        }
        cout << endl;
    }

    return 0;

}
```

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/604/files#diff-40a9a254233406b09afe83afb0102ccf39dfcef7f2ba3c7745bac6929a0de5cf

4. **Результати виконання завдань та фактично затрачений час**

Завдання №1. VNS lab 10 variant 15

```
Enter K:
2
Paris Amsterdam Vienna London
The list after deleting elements:
Paris Amsterdam
The list after adding element:
Paris Berlin Amsterdam
The list was deleted
The list is empty
The list was restored:
Paris Berlin Amsterdam
The list was deleted
```

Планований час: 1,5 год, фактично: 1,5 год

Завдання №2. Algotester lab 5 variant 2.

```
5 6
SSOSOS
SOSSOO
OOOXOX
SOSOSO
OOSOOS
OOOSOO
OOOSOS
SOSXOX
SOSOOO
SSSOSS
```

Планований час: 40 хв, фактично: 50 хв

день тому	Lab 5v2 - Lab 5v2	C++ 23	Зараховано	0.025	1.824	1914769

Завдання №3. Algotester lab 7-8 variant 1

```
3
insert
0
3
1 2 3
erase
2
1
print
1 2
```

Планований час: 1,5 год, фактично: 2 год

20 годин тому	Lab 78v1 - Lab 78v1	C++ 23	Зараховано	0.008

Завдання №4. Class Practice Task

```
Task 1
1 2 3 4
Reversed list:
4 3 2 1

Task 2
First list:
1 2 3 4
Second list:
1 2 8
The lists are not equal

Task 3
First number: 4321
Second number: 821
Sum: 5142
```

Планований час: 1 год, фактично: 1 год

```
Task 4
1 2 8 4 9 7
Mirrotred tree:
1 4 7 9 2 8
Task 5
Tree sum: 31 20 7 9 10 8
```

Планований час: 1 год, фактично: 50 хв

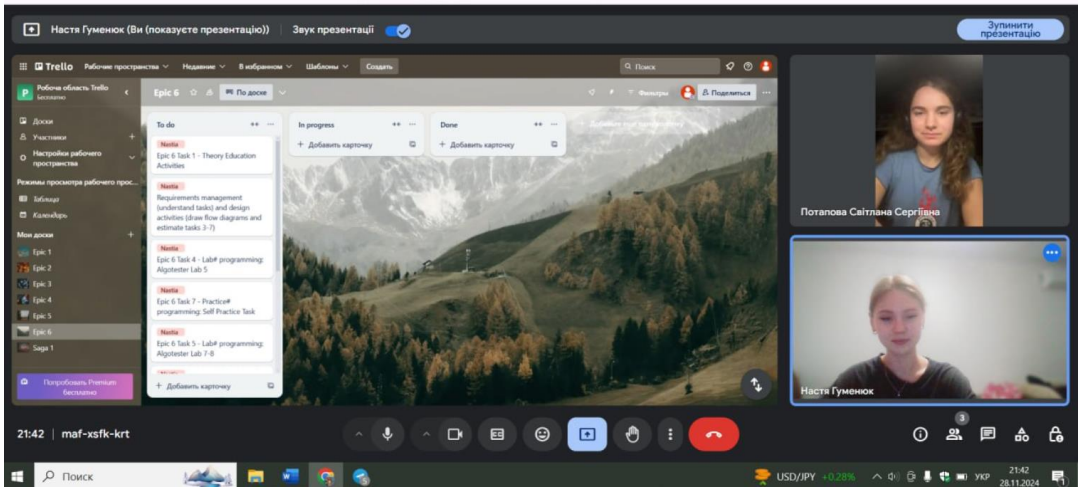
Завдання №5. Self-Practice task Algotester lab 5 variant 3

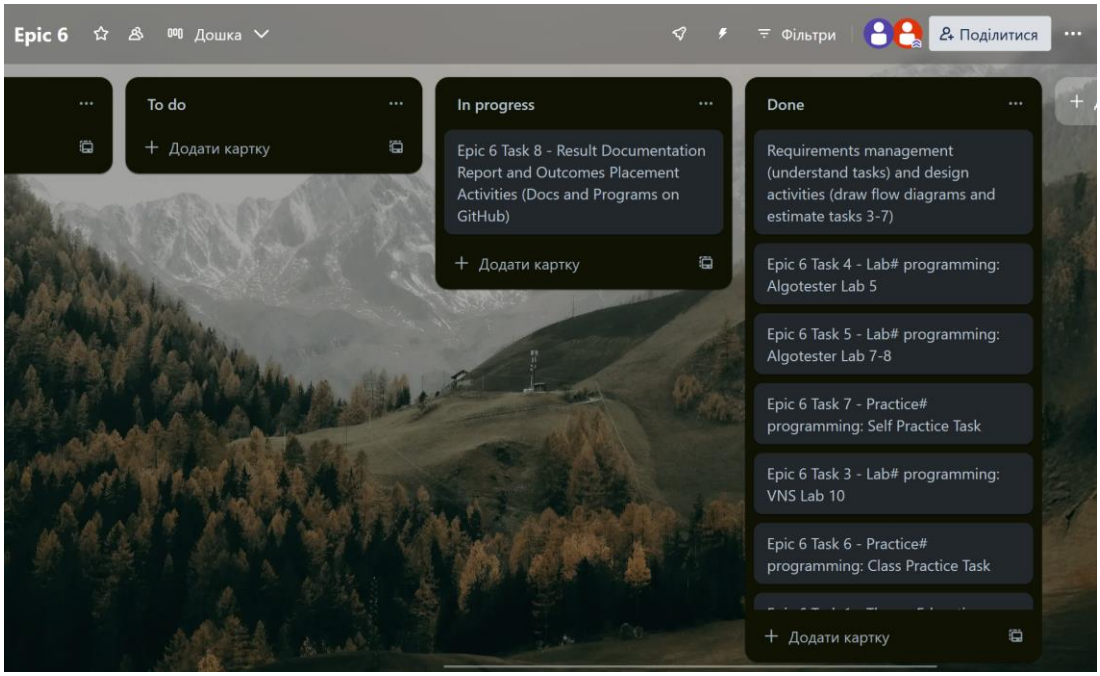
```
3 4
2 2
1 2 1 0
2 3 2 1
1 2 1 0
```

Планований час: 50 хв, фактично: 50 хв

3 години тому	Lab 5v3 - Lab 5v3	C++ 23	Зараховано	0.117	6.836	192503

5. Кооперація з командою





Висновок: Під час виконання роботи я ознайомилася з динамічними структурами та навчилася використовувати їх на практиці.