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-12
Хвостова Олександра Андріївна

Тема роботи:

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

Мета роботи:

Мета цього навчального блоку — ознайомити студентів з основами та особливостями динамічних структур даних. Це включає розуміння того, що таке динамічні структури даних, як вони використовуються для ефективного зберігання і обробки інформації, а також як реалізовувати основні операції над ними. Вивчення цих тем допоможе студентам розробляти ефективні алгоритми і структури даних, які є важливими у багатьох практичних завданнях програмування.

Теоретичні відомості:

1. Основи Динамічних Структур Даних:
    a. Вступ до динамічних структур даних: визначення та важливість
    b. Виділення пам'яті для структур даних (stack і heap)
    c. Приклади простих динамічних структур: динамічний масив
    Відео.
    https://www.youtube.com/watch?v=NyOjKd5Qruk&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=58
    Стаття. https://drukarnia.com.ua/articles/stack-ta-heap-_c5UU
    Стаття. https://acode.com.ua/urok-90-dynamichni-masyvy/
    - Статус: Ознайомлений
    - Початок опрацювання теми: 27.11.2024
    - Завершення опрацювання теми: 28.11.2024

2. Стек:
    a. Визначення та властивості стеку
    b. Операції push, pop, top: реалізація та використання
    c. Приклади використання стеку: обернений польський запис
    d. Переповнення стеку
    Стаття. https://www.geeksforgeeks.org/stack-in-cpp-stl/?ref=header_outind
    Стаття. https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/
    Стаття. https://www.geeksforgeeks.org/evaluate-the-value-of-an-arithmetic-expression-in-reverse-polish-notation-in-java/
    Стаття. https://en.wikipedia.org/wiki/Reverse_Polish_notation
    Відео. https://www.youtube.com/watch?v=gtErSTjL4vE
    Стаття.
    https://uk.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BF%D0%BE%D0%B2%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F_%D1%81%D1%82%D0%B5%D0%BA%D0%B0
    - Статус: Ознайомлений
    - Початок опрацювання теми: 26.11.2024
    - Завершення опрацювання теми: 27.11.2024

3. Черга:
    a. Визначення та властивості черги
    b. Операції enqueue, dequeue, front: реалізація та застосування
    c. Приклади використання черги: обробка подій, алгоритми планування

d. Розширення функціоналу черги: пріоритетні черги
Стаття. https://studfile.net/preview/6889815/page:4/
Відео.
https://www.youtube.com/watch?v=Yhw8NbjrSFA&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=142

- Статус: Ознайомлений
- Початок опрацювання теми: 24.11.2024
- Завершення опрацювання теми: 25.11.2024

4. Зв'язні Списки:
   a. Визначення однозв'язного та двозв'язного списку
   b. Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
   c. Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
   d. Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
   Відео.
   https://www.youtube.com/watch?v=3MfhrHWBdgo&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=138
   Відео. https://www.youtube.com/watch?v=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=139
   - Статус: Ознайомлений
   - Початок опрацювання теми: 21.11.2024
   - Завершення опрацювання теми: 23.11.2024
5. Дерева:
   a. Вступ до структури даних "дерево": визначення, типи
   b. Бінарні дерева: вставка, пошук, видалення
   c. Обхід дерева: в глибину (preorder, inorder, postorder), в ширину
   d. Застосування дерев: дерева рішень, хеш-таблиці
   e. Складніші приклади дерев: AVL, Червоно-чорне дерево
   Відео. https://www.youtube.com/watch?app=desktop&v=alxzyWswCVg&t=0s
   Відео. https://youtu.be/qBFzNW0ALxQ?si=o_QP6uPfbXygyKst
   Відео. https://www.youtube.com/watch?v=87-lAYP9KCA
   Відео. https://youtu.be/__66xMXz7wc?si=fEphxzcHedgeh3p0
   Відео. https://youtu.be/qvZGUFHWChY?si=o2_yh67M4GD5ht2F
   Відео. https://youtu.be/DB1HFCEdLxA?si=3-7GVYCgUp2paojA
   - Статус: Ознайомлений
   - Початок опрацювання теми: 15.11.2024
   - Завершення опрацювання теми: 21.11.2024
6. Алгоритми Обробки Динамічних Структур:
   a. Основи алгоритмічних патернів: ітеративні, рекурсивні
   b. Алгоритми пошуку, сортування даних, додавання та видалення елементів

   - Статус: Ознайомлений
   - Початок опрацювання теми: 12.11.2024
   - Завершення опрацювання теми: 20.11.2024

Виконання роботи:

1. Опрацювання завдання та вимог до програм та середовища:

Завдання №1. VNS Lab 10 - Task 7

- Варіант завдання: 7

- Планований час на виконання: 1 година
- Деталі завдання

Сформувати двонаправлений список. Знищити з нього перший елемент, додати елемент у кінець списку.

- Важливі деталі для врахування в імплементації програми

1. Створення списку.
2. Додавання елемента в список (у відповідності зі своїм варіантом).
3. Знищення елемента зі списку (у відповідності зі своїм варіантом).
4. Друк списку.
5. Запис списку у файл.
6. Знищення списку.
7. Відновлення списку з файлу.

Завдання №2. Algotester. Lab 5. V2

- Планований час виконання: 3 години

Lab 5v2

Обмеження: 1 сек., 256 МіБ

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

Всередині печери є пустота, пісок та каміння. Пустота позначається буквою , 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 цифер - і-й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані

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

Завдання №3-4. Algotester. Lab 78. V3

- Планований час виконання: 3 години

Lab 78v3

Обмеження: 1 сек., 256 МіБ

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.
  Реалізувати використовуючи перегрузку оператора <<

Для того щоб отримати 50% балів за лабораторну достатньо написати свою структуру.
Для отримання 100% балів ця структура має бути написана як шаблон класу, у якості параметру використати *int*.
Використовувати STL заборонено.

Завдання №5. Class Practice Work

- Планований час виконання: 3 години

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 $\Longrightarrow$ 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 №6. Self Practice Work
- Планований час виконання: 3 години

Lab 5v3

Обмеження: 1 сек., 256 МіБ

У вас є карта гори розміром $N \times M$.
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

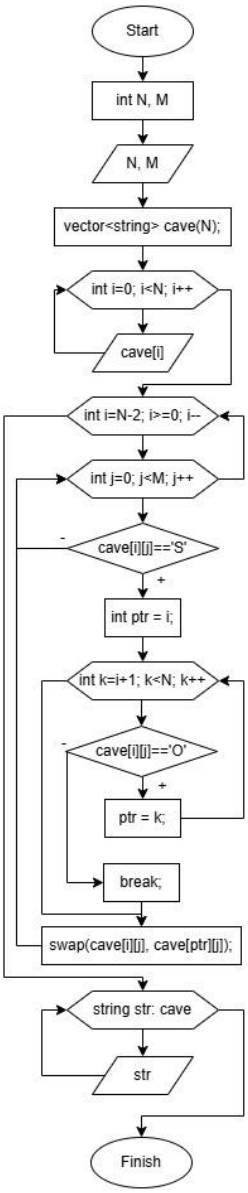
У першому рядку 2 числа N та M - розміри карти
у другому рядку 2 числа x та y - координати піку гори

Вихідні дані

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

2. Дизайн та планована оцінка часу виконання завдань:

Завдання №2. Algotester. Lab 5. V2



4. Код програм з посиланням на зовнішні ресурси:

Завдання №1. VNS Lab 10 - Task 7

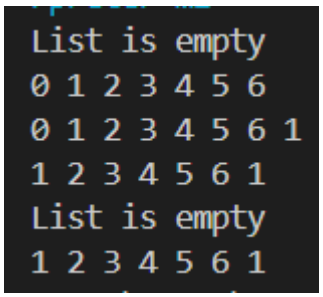
- Варіант завдання: 7

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/blob/db21ed7dfa6888a84a9c4da046edbcdf3ed4aa12/ai_12/oleksandra_khvostova/epic_6/vns_lab_10_task_oleksandra_khvostova.cpp

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

class List {
    private:
        struct Node {
            int data;
            Node* previous;
            Node* next;
            };

        Node* head;
        Node* tail;
        size_t size;

    public:
        List() : head(nullptr), tail(nullptr), size(0) {}

        void Show() const {
            if (size == 0) {
                cout << "List is empty" << endl;
                return;
            }
            Node* current = head;
            while (current != nullptr) {
                cout << current->data << " ";
                current = current->next;
            }
            cout << endl;
        }

        void Clear() {
            while (head != nullptr) {
                Node* temp = head;
                head = head->next;
                delete temp;
            }
            tail = nullptr;
            size = 0;
        }

        void PushBack(const int& value) {
            if (size == 0) {
                head = new Node {value, nullptr, nullptr};
                tail = head;
            } else {
                tail->next = new Node {value, tail, nullptr};
                tail = tail->next;
            }
            size++;
        }

```

```
55          void DeleteFirst() {
56              if (head == nullptr) {
57                  return;
58              }
59
60              Node* temp = head;
61              head = head->next;
62
63              if (head != nullptr) {
64                  head->previous = nullptr;
65              }
66
67              delete temp;
68          }
69
70          void WriteToFile(const string& filename){
71              ofstream outFile(filename);
72              if(!outFile){
73                  cerr<<"error opening "<<filename<<endl;
74                  return;
75              }
76
77              Node* current = head;
78              while(current!=nullptr){
79                  outFile<<current->data<<" ";
80                  current = current->next;
81              }
82              outFile.close();
83          }
84
85          Node* ReadFromFile (const string& filename){
86              ifstream inFile(filename);
87              if(!inFile){
88                  cerr<<"error opening "<<filename<<endl;
89                  return nullptr;
90              }
91              int value;
92              while(inFile>>value){
93                  PushBack(value);
94              }
95                  inFile.close();
96                  return head;
97              }
98      };
99
100     int main(){
101         List list;
102         list.Show();
103         for(int i=0; i<7; i++){
104             list.PushBack(i);
105         }
106         list.Show();
107         list.PushBack(1);
108         list.Show();
109         list.DeleteFirst();
110         list.Show();
111         list.WriteToFile("list.txt");
112         list.Clear();
113         list.Show();
114         list.ReadFromFile("list.txt");
115         list.Show();
116         list.Clear();
117         return 0;
118     }
```

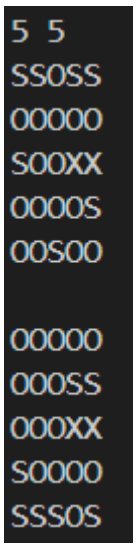
Час затрачений на виконання завдання: 2 години



Завдання №2. Algotester. Lab 5. V2
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/blob/db21ed7dfa6888a84a9c4da046edbcdf3ed4aa12/ai_12/oleksandra_khvostova/epic_6/algotester_lab_5_oleksandra_khvostova.cpp

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

int main(){
    int N, M;
    cin>>N>>M;
    vector<string> cave(N);
    for(int i=0; i<N; i++){
        cin>>cave[i];
    }
    for(int i=N-2; i>=0; i--){
        for(int j=0; j<M; j++){
            if(cave[i][j]=='S'){
                int ptr = i;
                for(int k=i+1; k<N; k++){
                    if(cave[k][j] == 'O') ptr = k;
                    else break;
                }
                swap(cave[i][j], cave[ptr][j]);
            }
        }
    }
    cout<<endl;
    for(string str: cave) cout<<str<<endl;
    return 0;
}
```

Час затрачений на виконання завдання: 2 години



Завдання №3. Algotester. Lab 78. V3

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/blob/db21ed7dfa6888a84a9c4da046edbcdf3ed4aa12/ai_12/oleksandra_khvostova/epic_6/algotester_lab_78_1_oleksandra_khvostova.cpp

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

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

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

class BinaryTree {
private:
    TreeNode* root;
    size_t treeSize;

    TreeNode* insert(TreeNode* node, int value) {
        if (node == nullptr) {
            return new TreeNode(value);
        }
        if (value < node->value) {
            node->left = insert(node->left, value);
        } else if (value > node->value) {
            node->right = insert(node->right, value);
        }
        // If value == node->value, do nothing (ignore duplicate)
        return node;
    }

    bool contains(TreeNode* node, int value) const {
        if (node == nullptr) {
            return false;
        }
        if (value == node->value) {
            return true;
        } else if (value < node->value) {
            return contains(node->left, value);
        } else {
            return contains(node->right, value);
        }
    }

    void inorder(TreeNode* node, ostream& os) const {
        if (node == nullptr) {
            return;
```

```
        return;
    }
    inorder(node->left, os);
    os << node->value << " ";
    inorder(node->right, os);
}

public:
BinaryTree() : root(nullptr), treeSize(0) {}

void insert(int value) {
    if (!contains(value)) {
        root = insert(root, value);
        treeSize++;
    }
}

bool contains(int value) const {
    return contains(root, value);
}

size_t size() const {
    return treeSize;
}

friend ostream& operator<<(ostream& os, const BinaryTree& tree) {
    tree.inorder(tree.root, os);
    return os;
}
};

int main() {
    BinaryTree tree;
    int Q;
    cin >> Q;
    string* commands = new string[Q];
    int* values = new int [Q];

    for (int i = 0; i < Q; i++) {
        cin >> commands[i];
        if (commands[i] == "insert" || commands[i] == "contains") {
            cin >> values[i];
        } else {
            values[i] = 0;
        }
    }

    for (int i = 0; i < Q; i++) {
        if (commands[i] == "insert") {
            tree.insert(values[i]);
        } else if (commands[i] == "contains") {
            if (tree.contains(values[i])) {
                cout << "Yes" << endl;
            } else {
                cout << "No" << endl;
            }
        } else if (commands[i] == "size") {
            cout << tree.size() << endl;
        } else if (commands[i] == "print") {
            cout << tree << endl;
        }
    }
    delete[]commands;
    delete[]values;
    return 0;
}
```

Завдання №4. Algotester. Lab 78. V3
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/blob/db21ed7dfa6888a84a9c4da046edbcdf3ed4aa12/ai_12/oleksandra_khvostova/epic_6/algotester_lab_78_2_oleksandra_khvostova.cpp

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

template<typename T>
struct TreeNode {
    T value;
    TreeNode* left;
    TreeNode* right;

    TreeNode(T val) : value(val), left(nullptr), right(nullptr) {}
};

template<typename T>
class BinaryTree {
private:
    TreeNode<T>* root;
    size_t treeSize;

    TreeNode<T>* insert(TreeNode<T>* node, T value) {
        if (node == nullptr) {
            return new TreeNode<T>(value);
        }
        if (value < node->value) {
            node->left = insert(node->left, value);
        } else if (value > node->value) {
            node->right = insert(node->right, value);
        }
        // якщо value == node->value, нічого не робимо (ігноруємо дублікати)
        return node;
    }

    bool contains(TreeNode<T>* node, T value) const {
        if (node == nullptr) {
            return false;
        }
        if (value == node->value) {
            return true;
        } else if (value < node->value) {
            return contains(node->left, value);
        } else {
            return contains(node->right, value);
        }
    }

    void inorder(TreeNode<T>* node, ostream& os) const {
        if (node == nullptr) {
            return;
        }
        inorder(node->left, os);
        os << node->value << " ";
        inorder(node->right, os);
    }

public:
    BinaryTree() : root(nullptr), treeSize(0) {}

    void insert(T value) {
        if (!contains(value)) {
            root = insert(root, value);
            treeSize++;
        }
    }

    bool contains(T value) const {
        return contains(root, value);
    }

    size_t size() const {
        return treeSize;
    }

    friend ostream& operator<<(ostream& os, const BinaryTree<T>& tree) {
        tree.inorder(tree.root, os);
        return os;
    }
};

int main() {
    BinaryTree<int> tree;
    int Q;
    cin >> Q;
    string* commands = new string[Q];
    int* values = new int[Q];

    for (int i = 0; i < Q; i++) {
        cin >> commands[i];
        if (commands[i] == "insert" || commands[i] == "contains") {
            cin >> values[i];
```

```
        } else {
            values[i] = 0;
        }
    }

    for (int i = 0; i < Q; i++) {
        if (commands[i] == "insert") {
            tree.insert(values[i]);
        } else if (commands[i] == "contains") {
            if (tree.contains(values[i])) {
                cout << "Yes" << endl;
            } else {
                cout << "No" << endl;
            }
        } else if (commands[i] == "size") {
            cout << tree.size() << endl;
        } else if (commands[i] == "print") {
            cout << tree << endl;
        }
    }

    delete[] commands;
    delete[] values;

    return 0;
}
```

Час затрачений на виконання завдання: 4 години

```
11

size
insert 5
insert 4
print
insert 5
print
insert 1
print
contains 5
contains 0
size
0
4 5
4 5
1 4 5
Yes
No
3
```

Завдання №5. Class Practice Work
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/blob/db21ed7dfa6888a84a9c4da046edbcdf3ed4aa12/ai_12/oleksandra_khvostova/epic_6/practice_work_task_oleksandra_khvostova.cpp

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

class List {
    private:
        struct Node {
            int data;
            Node* previous;
            Node* next;
            };

        Node* head;
        Node* tail;
        size_t size;

    public:
        List() : head(nullptr), tail(nullptr), size(0) {}

        void Show() const {
            if (size == 0) {
                cout << "List is empty" << endl;
                return;
            }
            Node* current = head;
            while (current != nullptr) {
                cout << current->data << " ";
                current = current->next;
            }
            cout << endl;
        }

        void Clear() {
            while (head != nullptr) {
                Node* temp = head;
                head = head->next;
                delete temp;
            }
            tail = nullptr;
            size = 0;
        }

```

```
        void PushBack(const int& value) {
            if (size == 0) {
                head = new Node {value, nullptr, nullptr};
                tail = head;
            } else {
                tail->next = new Node {value, tail, nullptr};
                tail = tail->next;
            }
            size++;
        }

        // Задача 1
        void Reverse() {
            Node* current = head;
            Node* temp = nullptr;

            while (current != nullptr) {
                temp = current->previous;
                current->previous = current->next;
                current->next = temp;
                current = current->previous;
            }
            if (temp != nullptr) {
                head = temp->previous;
            }
        }

        // Задача 2
        bool Compare(const List& list1, const List& list2) {
            if (list1.size != list2.size) {
                return false;
            }
            Node* current1 = list1.head;
            Node* current2 = list2.head;
            while (current1 != nullptr && current2 != nullptr) {
                if (current1->data != current2->data) {
                    return false;
                }
                current1 = current1->next;
                current2 = current2->next;
            }
            return (current1 == nullptr && current2 == nullptr);
        }

        // Задача 3
        static List AddArrays(const vector<int>& arr1, const vector<int>& arr2) {
            List result;
            int carry = 0;
            size_t i = 0;
            size_t maxLength = max(arr1.size(), arr2.size());

            while (i < maxLength || carry != 0) {
                int x = (i < arr1.size()) ? arr1[i] : 0;
                int y = (i < arr2.size()) ? arr2[i] : 0;
                int sum = carry + x + y;
                carry = sum / 10;
                result.PushBack(sum % 10);
                i++;
            }

            return result;
        }

        static List AddLists(const List& list1, const List& list2) {
            string num1_str, num2_str;
            Node* current1 = list1.head;
            Node* current2 = list2.head;

            while (current1 != nullptr) {
                num1_str += to_string(current1->data);
                current1 = current1->next;
            }
            while (current2 != nullptr) {
                num2_str += to_string(current2->data);
                current2 = current2->next;
            }

            vector<int> num1, num2;
            for (char c : num1_str) num1.push_back(c - '0');
            for (char c : num2_str) num2.push_back(c - '0');

            List result = AddArrays(num1, num2);
            return result;
        }
};
```

```cpp
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;

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

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

    TreeNode* newRoot = new TreeNode(root->val);
    newRoot->left = CreateMirrorFlip(root->right);
    newRoot->right = CreateMirrorFlip(root->left);

    return newRoot;
}

void inorderTraversal(TreeNode* root) {
    if (root == nullptr) {
        return;
    }
    inorderTraversal(root->left);
    cout << root->val << " ";
    inorderTraversal(root->right);
}

int TreeSum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }

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

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

    return root->val;
}

int main() {
    List list1;
    for (int i = 1; i < 6; i++) {
        list1.PushBack(i);
    }
    List list2;
    for (int i = 1; i < 5; i++) {
        list2.PushBack(i);
    }
    list1.Reverse(); // Задача 1
    cout << "Reversed list1: ";
    list1.Show();
    list1.Reverse();
    cout << "list1: ";
    list1.Show();
    cout << "list2: ";
    list2.Show();

    if (list1.Compare(list1, list2)) { // Задача 2
        cout << "the lists are equal" << endl;
    } else {
        cout << "the lists are not equal" << endl;
    }

    List result = List::AddLists(list1, list2); // Задача 3
    result.Show();

    // Дерево
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);
```

```
    cout << "Original tree (inorder): ";
    inorderTraversal(root);
    cout << endl;

    // Задача 4
    TreeNode* mirrorRoot = CreateMirrorFlip(root);

    cout << "Mirror flipped tree (inorder): ";
    inorderTraversal(mirrorRoot);
    cout << endl;

    TreeSum(root);

    // Задача 5
    cout << "Tree after sum operation (inorder): ";
    inorderTraversal(root);
    cout << endl;

    delete root->left->left;
    delete root->left->right;
    delete root->left;
    delete root->right->left;
    delete root->right->right;
    delete root->right;
    delete root;

    delete mirrorRoot->left->left;
    delete mirrorRoot->left->right;
    delete mirrorRoot->left;
    delete mirrorRoot->right->left;
    delete mirrorRoot->right->right;
    delete mirrorRoot->right;
    delete mirrorRoot;

    result.Clear();
    list1.Clear();
    list2.Clear();
    return 0;
}
```

Час затрачений на виконання: 4 години

```
Reversed list1: 5 4 3 2 1
list1: 1 2 3 4 5
list2: 1 2 3 4
the lists are not equal
2 4 6 8 5
Original tree (inorder): 4 2 5 1 6 3 7
Mirror flipped tree (inorder): 7 3 6 1 5 2 4
Tree after sum operation (inorder): 4 11 5 28 6 16 7
```

Завдання №6. Self Practice Work
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/blob/db21ed7dfa6888a84a9c4da046edbcdf3ed4aa12/ai_12/oleksandra_khvostova/epic_6/practice_work_self_algotester_tasks_oleksandra_khvostova.cppv

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

int main() {
    int N, M, x, y;
    cin >> N >> M >> x >> y;
    x--; y--;  // Adjust for 0-based indexing

    const int dx[] = {1, -1, 0, 0};
    const int dy[] = {0, 0, 1, -1};

    vector<vector<int>> height(N, vector<int>(M, -1));
    queue<pair<int, int>> q;

    q.push({x, y});
    height[x][y] = 0;

    while (!q.empty()) {
        auto [cx, cy] = q.front();
        q.pop();

        for (int i = 0; i < 4; ++i) {
            int nx = cx + dx[i];
            int ny = cy + dy[i];

            if (nx >= 0 && nx < N && ny >= 0 && ny < M && height[nx][ny] == -1) {
                height[nx][ny] = height[cx][cy] + 1;
                q.push({nx, ny});
            }
        }
    }
```

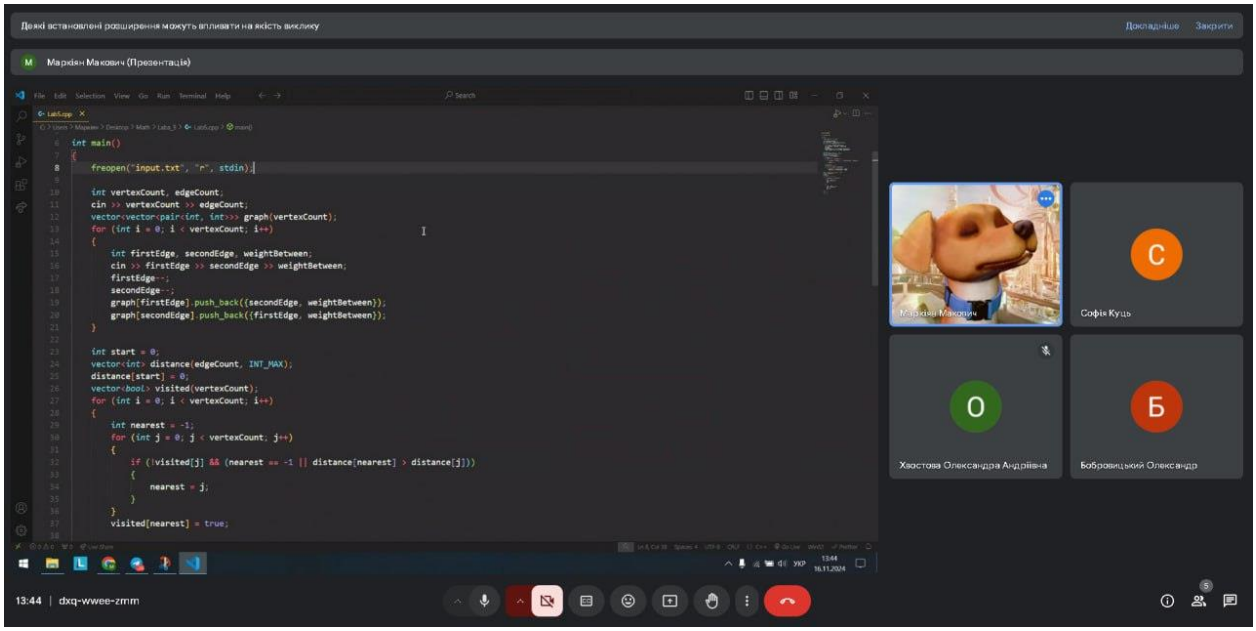
```cpp
    int maxHeight = 0;
    for (const auto& row : height) {
        for (int h : row) {
            maxHeight = max(maxHeight, h);
        }
    }

    for (const auto& row : height) {
        for (int h : row) {
            cout << maxHeight - h << " ";
        }
        cout << endl;
    }

    return 0;
}
```

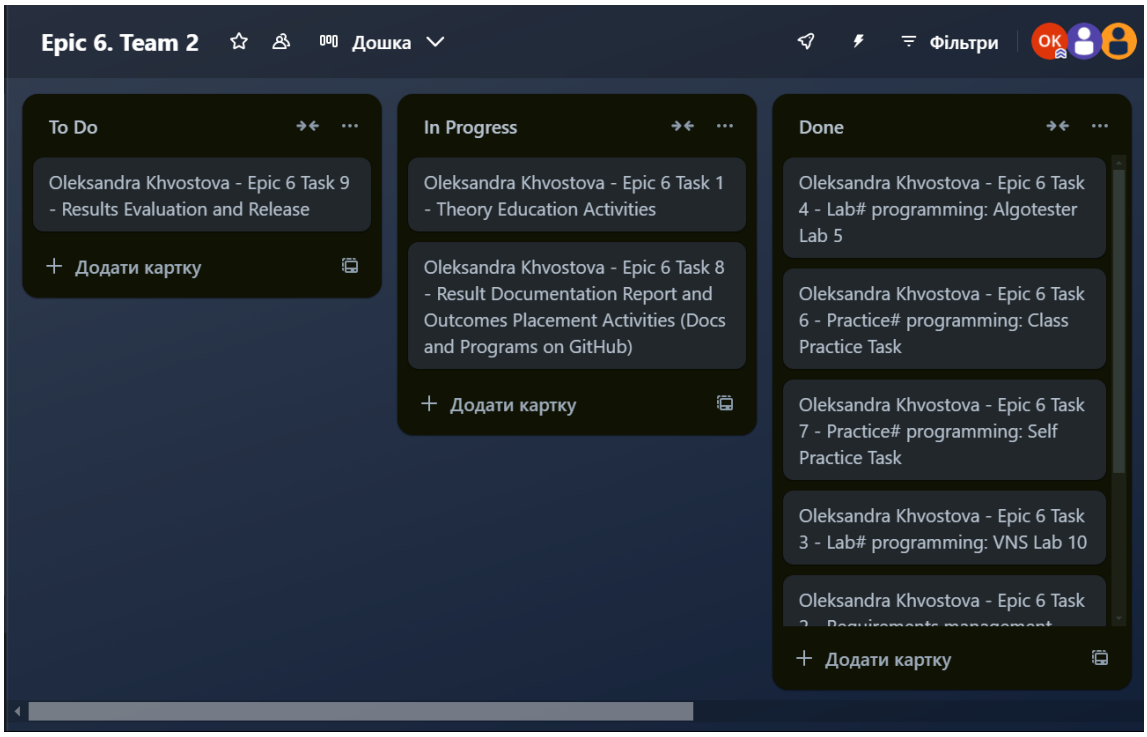
Час затрачений на виконання: 2 години

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

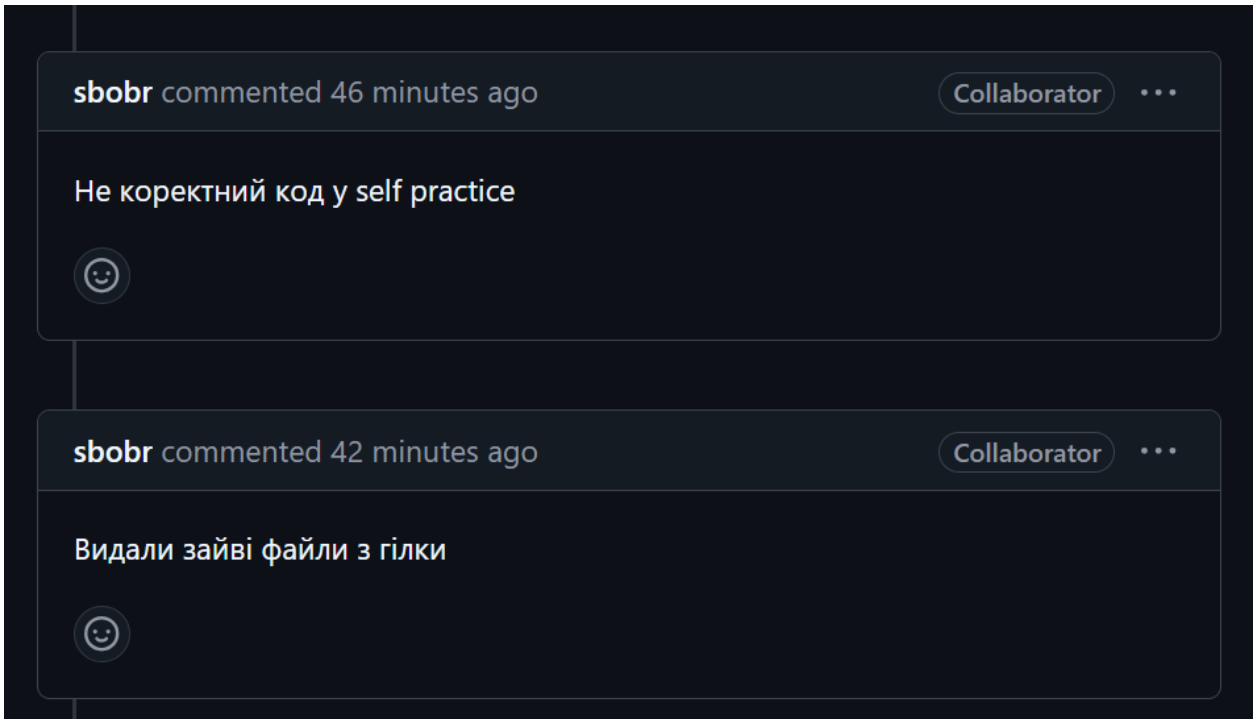
6. Кооперація з командою:

- Скрін з 1-ї зустрічі по обговоренню задач Епіку та Скрін прогресу по Трелло





Скрін з 2-му коментарями від учасників команди на пул реквесті з Ревю Роботи



Висновки:

Динамічні структури даних дозволяють ефективно керувати пам'яттю, забезпечуючи змінний розмір і можливість динамічного виділення пам'яті. Стек працює за принципом LIFO і використовується для збереження тимчасових даних або рекурсивних операцій. Черга працює за принципом FIFO і підходить для обробки подій та алгоритмів планування. Зв'язні списки дозволяють ефективно управляти пам'яттю і реалізовувати структури FIFO та LIFO. Дерева використовуються для ефективного зберігання та пошуку інформації, забезпечуючи швидкі операції вставки, пошуку і видалення елементів. Алгоритми пошуку, сортування і додавання/видалення елементів є ключовими для обробки даних у динамічних структурах даних.