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.»

до:
Практичних Робіт до блоку № 3

Виконала:
Студентка групи ШІ-12
Іванів Христина Вікторівна

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

Мета роботи: Ознайомитись з основними динамічними структурами даних, ознайомитись з алгоритмами їх обробки.

Теоретичні відомості:

1. Теоретичні відомості з переліком важливих тем:
    1) Тема №1: Основи Динамічних Структур Даних:
    2) Тема №2: Стек
    3) Тема №3: Черга
    4) Тема №4: Зв'язні Списки
    5) Тема №5: Дерева
    6) Тема №6: Алгоритми Обробки Динамічних Структур

2. Індивідуальний план опрацювання теорії:

- Тема №1: Основи Динамічних Структур Даних
    - Джерела Інформації:
        - Лекції О. Пшеничного
        - Практичні М. Фаріон
        - Уроки 58 з курсу C++ теорія з каналу «Блоган»
    - Що опрацьовано:
        - Лекції О. Пшеничного
        - Практичні М. Фаріон
        - Уроки 58 з курсу C++ теорія з каналу «Блоган»
    - Статус: ознаймолена з основними динамічними структурами даних

- Тема №2: Стек
    - Джерела Інформації:
        - Лекції О. Пшеничного
        - Практичні М. Фаріон
        - Уроки 142 з курсу C++ теорія з каналу «Блоган»
    - Що опрацьовано:
        - Лекції О. Пшеничного
        - Практичні М. Фаріон
        - Уроки 142 з курсу C++ теорія з каналу «Блоган»
    - Статус: ознайомлена з стеком та роботою з ним

- Тема №3: Черга
    - Джерела Інформації:
        - Лекції О. Пшеничного
        - Практичні М. Фаріон
        - Уроки 141з курсу C++ теорія з каналу «Блоган»
    - Що опрацьовано:
        - Лекції О. Пшеничного
        - Практичні М. Фаріон
        - Уроки 141 з курсу C++ теорія з каналу «Блоган»
    - Статус: ознайомлена з чергою та її практичним застосуванням

- Тема №4: Зв'язні Списки
  - Джерела Інформації:
    - Лекції О. Пшеничного
    - Практичні М. Фаріон
    - Уроки 139, 140 з курсу C++ теорія з каналу «Блоган»
  - Що опрацьовано:
    - Лекції О. Пшеничного
    - Практичні М. Фаріон
    - Уроки 139, 140 з курсу C++ теорія з каналу «Блоган»
  - Статус: ознайомлена з однозв’язними та двозв’язними списками

- Тема №5: Дерева
  - Джерела Інформації:
    - Лекції О. Пшеничного
    - Практичні М. Фаріон
    - Уроки 144 з курсу C++ теорія з каналу «Блоган»
    - Урок #11 по C++ з ютуб каналу «Школа програмування»
  - Що опрацьовано:
    - Лекції О. Пшеничного
    - Практичні М. Фаріон
    - Урок #11 по C++ з ютуб каналу «Школа програмування»
    - Уроки 144 з курсу C++ теорія з каналу «Блоган»
  - Статус: ознайомлена з деревами, роботою з ними

- Тема №6: Алгоритми Обробки Динамічних Структур
  - Джерела Інформації:
    - Лекції О. Пшеничного
    - Практичні М. Фаріон
  - Що опрацьовано:
    - Лекції О. Пшеничного
    - Практичні М. Фаріон
  - Статус: ознайомлена з алгоритмами обробки динамічних структур

Виконання роботи:

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

Завдання №1 VNS. Лабораторна робота №10

- ***Варіант завдання: 6***

- *Деталі завдання*:
  Написати програму, у якій створюються динамічні структури й виконати їхню обробку у відповідності зі своїм варіантом.
  Для кожного варіанту розробити такі функції:
  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.

- *Важливі деталі для врахування*:

  Записи в лінійному списку містять ключове поле типу int. Сформувати двонаправлений список. Знищити з нього елемент із заданим номером, додати елемент у початок списку.

Завдання №2 VNS. Algotester task 5. V-1

- ***Варіант завдання: 1***

  - *Деталі завдання:*

У світі Атод сестри Ліна і Рілай люблять грати у гру. У них є дошка із 8-ми рядків і 8-ми стовпців. На перетині і-го рядка і j-го стовпця лежить магічна куля, яка може світитись магічним світлом (тобто у них є 64 кулі). На початку гри деякі кулі світяться, а деякі ні... Далі вони обирають N куль і для кожної читають магічне заклинання, після чого всі кулі, які лежать на перетині стовпця і рядка обраної кулі міняють свій стан (ті що світяться - гаснуть, ті, що не світяться - загораються).
Також вони вирішили трохи Вам допомогти і придумали спосіб як записати стан дошки одним числом aa із 8-ми байт, а саме:

- Молодший байт задає перший рядок матриці;
- Молодший біт задає перший стовпець рядку;
- Значення біту каже світиться куля чи ні (0 - ні, 1 - так);

Тепер їх цікавить яким буде стан дошки після виконання NN заклинань і вони дуже просять Вас їм допомогти.

- *Важливі деталі для врахування*:

Input

У першому рядку одне число a - поточний стан дошки.
У другому рядку N - кількість заклинань.
У наступних N рядках по 2 числа Ri, Ci - рядок і стовпець кулі над якою виконується заклинання.

Output

Одне число b - стан дошки після виконання N заклинань.

Constraints

$0 \le N \le 10^3 0 \le N \le 10^3$
$1 \le R_i, C_i \le 8 1 \le R_i, C_i \le 8$

$0 \le a,b < 2^{64}$

Завдання №3 Algotester task 7.8. V- 2

- ***Варіант завдання*: 2**
    - *Деталі завдання*

Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив".

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 масиву, який треба вставити. У третьому рядку N цілих чисел - масив, 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 динамічному масиві.

• Визначення кількості зарезервованої пам’яті:

Ідентифікатор – capacity

Ви не отримуєте аргументів. Ви виводите кількість зарезервованої пам’яті у динамічному масиві. Ваша реалізація динамічного масиву має мати фактор росту (Growth factor) рівний 2.

• Отримання значення i-го елементу

Ідентифікатор - get

Ви отримуєте ціле число - index, індекс елемента. Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора

• Модифікація значення і-го елементу

Ідентифікатор - set

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

• Вивід динамічного масиву на екран

Ідентифікатор - print

Ви не отримуєте аргументів. Ви виводите усі елементи динамічного масиву через пробіл. Реалізувати використовуючи перегрузку оператора

- *Важливі деталі для врахування*:

Для того щоб отримати 50% балів за лабораторну достатньо написати свою структуру.

Завдання №4 VNS. Algotester task 7.8. V- 2

- ***Варіант завдання*: 2**
    - *Деталі завдання*

Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив".

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 масиву, який треба вставити. У третьому рядку N цілих чисел - масив, 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 динамічному масиві.

• Визначення кількості зарезервованої пам’яті:

Ідентифікатор – capacity

Ви не отримуєте аргументів. Ви виводите кількість зарезервованої пам’яті у динамічному масиві. Ваша реалізація динамічного масиву має мати фактор росту (Growth factor) рівний 2.

• Отримання значення i-го елементу

Ідентифікатор - get

Ви отримуєте ціле число - index, індекс елемента. Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора

• Модифікація значення i-го елементу

Ідентифікатор - set

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

• Вивід динамічного масиву на екран

Ідентифікатор - print

Ви не отримуєте аргументів. Ви виводите усі елементи динамічного масиву через пробіл. Реалізувати використовуючи перегрузку оператора

- *Важливі деталі для врахування*:

Для отримання 100% балів ця структура має бути написана як шаблон класу, у якості

параметру використати int.

Завдання №5 Class Practice Work

- **Деталі завдання**

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;

Мета задачі

Розуміння структур даних: Реалізація методу реверсу для зв'язаних списків є чудовим способом для поглиблення розуміння зв'язаних списків як фундаментальної структури даних. Він заохочує практичний підхід до вивчення того, як структуруються пов'язані списки та як ними маніпулювати.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Перевертання пов'язаного списку вимагає логічного підходу до маніпулювання покажчиками, що є ключовим навиком у інформатиці.

Засвоїти механізми маніпуляції з покажчиками: пов'язані списки значною мірою залежать від покажчиків. Це завдання покращить навички маніпулювання вказівниками, що є ключовим аспектом у таких мовах, як C++.

Розвинути навички розв'язувати задачі: перевернути пов'язаний список непросто й вимагає творчого й логічного мислення, таким чином покращуючи свої навички розв'язування поставлених задач.

Пояснення прикладу

Спочатку ми визначаємо просту структуру ***Node*** для нашого пов'язаного списку.
Потім функція ***reverse*** ітеративно змінює список, маніпулюючи наступними покажчиками кожного вузла.
printList — допоміжна функція для відображення списку.
Основна функція створює зразок списку, демонструє реверсування та друкує вихідний і обернений списки.

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***.

Мета задачі

Розуміння рівності в структурах даних: це завдання допомагає зрозуміти, як визначається рівність у складних структурах даних, таких як зв'язані списки. На відміну від

примітивних типів даних, рівність пов’язаного списку передбачає порівняння кожного елемента та їх порядку.

Поглиблення розуміння зв’язаних списків: Порівнюючи зв’язані списки, дозволяють покращити своє розуміння обходу, фундаментальної операції в обробці зв’язаних списків.

Розуміння ефективність алгоритму: це завдання також вводить поняття ефективності алгоритму. Студенти вчаться ефективно порівнювати елементи, що є навичкою, важливою для оптимізації та зменшення складності обчислень.

Розвинути базові навики роботи з реальними програми: функції порівняння мають вирішальне значення в багатьох реальних програмах, таких як виявлення змін у даних, синхронізація структур даних або навіть у таких алгоритмах, як сортування та пошук.

Розвинути навик вирішення проблем і увага до деталей: це завдання заохочує скрупульозний підхід до програмування, оскільки навіть найменша неуважність може призвести до неправильних результатів порівняння. Це покращує навички вирішення проблем і увагу до деталей.

Пояснення прикладу

- Для пов’язаного списку визначено структуру ***Node***.
- Функція ***compare*** ітеративно проходить обидва списки одночасно, порівнюючи дані в кожному вузлі.
- Якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.
- Основна функція ***main*** створює два списки та демонструє порівняння.

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.

Мета задачі

Розуміння операцій зі структурами даних: це завдання унаочнює практичне використання списка для обчислювальних потреб. Арифметичні операції з великими числами це окремий клас задач, для якого використання списків допомагає обійти обмеження у представленні цілого числа сучасними комп’ютерами.

Поглиблення розуміння зв’язаних списків: Застосовування зв’язаних списків для арифметичних операції з великими числами дозволяє покращити розуміння операцій з обробки зв’язаних списків.

Розуміння ефективність алгоритму: це завдання дозволяє порівняти швидкість алгоритму додавання з використанням списків зі швидкістю вбудованих арифметичних операцій. Студенти вчаться розрізняти позитивні та негативні ефекти при виборі структур даних для реалізації практичних програм.

Розвинути базові навики роботи з реальними програми: арифметичні операції з великими числами використовуються у криптографії, теорії чисел, астрономії, та ін.

Розвинути навик вирішення проблем і увага до деталей: завдання покращує розуміння обмежень у представленні цілого числа сучасними комп'ютерами та пропонує спосіб його вирішення.

Бінарні дерева

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

Мета задачі

Розуміння структур даних: Реалізація методу віддзеркалення бінарного дерева покращує розуміння структури бінарного дерева, виділення пам'яті для вузлів та зв'язування їх у єдине ціле. Це один з багатьох методів роботи з бінарними деревами.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева продемонструє розгортання рекурсивного виклику.

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

Мета задачі

Розуміння структур даних: Реалізація методу підрахунку сум підвузлів бінарного дерева покращує розуміння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева демонструє розгортання рекурсивного виклику.

Завдання №6 Self Practice Algotester Task

- *Деталі завдання*:

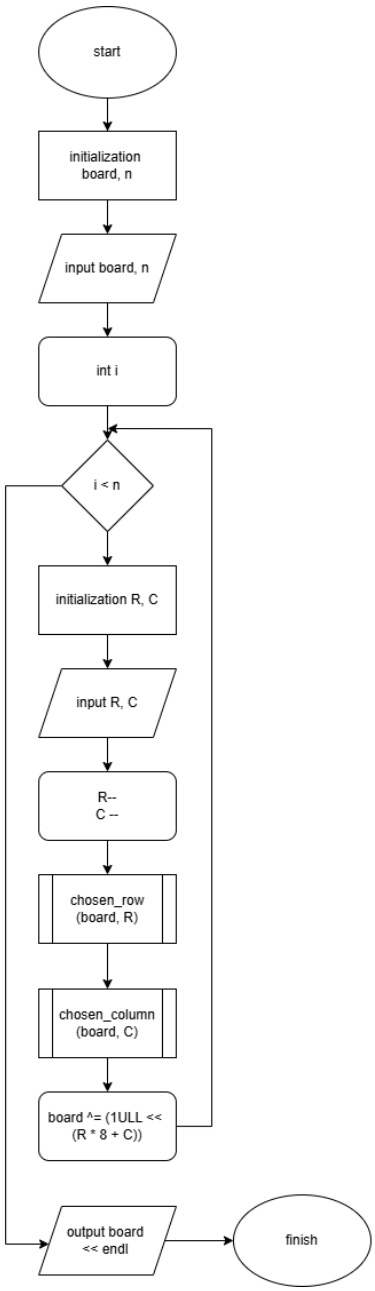
У вас є карта гори розміром N × M.
Також ви знаєте координати {x, y} , у яких знаходиться вершина гори.
Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.

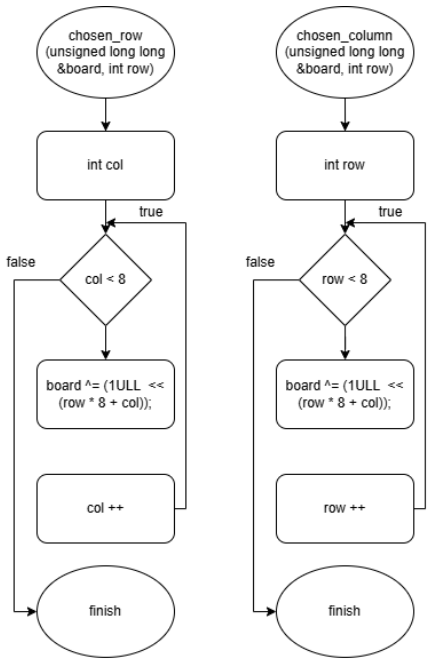
Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

Важливі деталі для врахування:
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.
Обмеження
$1 \le N, M \le 103$
$1 \le x \le N$
$1 \le y \le M$

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

Завдання №1 VNS. Лабораторна робота №10
Плановий час на реалізацію: 4години
Завдання №2 VNS. Algotester task 5. V-1





Плановий час на реалізацію: 4 години

Завдання №3 Algotester task 7.8. V- 2
Плановий час на реалізацію: 4 години

Завдання №4 Algotester task 7.8. V- 2
Плановий час на реалізацію: 4 години

Завдання №5 Class Practice Work

Плановий час на реалізацію: 6 годин

Завдання №6 Self Practice Algotester Task

Плановий час на реалізацію: 2 години

3. ***Код програми та фактичний час на реалізацію:***

Завдання №1 VNS. Лабораторна робота №10

```cpp
#include <iostream>
#include <fstream>

using namespace std;

struct Node {
    int data;
    Node* next;
    Node* previous;
};

class DoubleLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoubleLinkedList() : head(nullptr), tail(nullptr) {}

    void empty_list() {
        head = tail = nullptr;
    }

    void print_list() {
        if (head == nullptr) {
            cout << "The list is empty" << endl;
            return;
        }
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }

   void delete_by_index(unsigned int index)
    {
        Node* current = head;
        while(current)
        {
            if(index-1 == 0)
            {
                Node* curr_next = current->next;
                current->next = curr_next->next;
                curr_next->next->previous = current;
                delete curr_next;
            }
```

```
48              }
49              index--;
50              current = current->next;
51          }
52      }
53
54      void add(const int& value) {
55          if (!head) {
56              head = new Node{value, nullptr, nullptr};
57              tail = head;
58          } else {
59              tail->next = new Node{value, nullptr, tail};
60              tail = tail->next;
61          }
62      }
63
64      void delete_element(Node* node_to_delete) {
65          if (!node_to_delete) return;
66
67          if (node_to_delete == head) {
68              head = node_to_delete->next;
69              if (head != nullptr) {
70                  head->previous = nullptr;
71              }
72          } else {
73              if (node_to_delete->previous != nullptr) {
74                  node_to_delete->previous->next = node_to_delete->next;
75              }
76              if (node_to_delete->next != nullptr) {
77                  node_to_delete->next->previous = node_to_delete->previous;
78              }
79          }
80
81          delete node_to_delete;
82      }
83
84      void push_front(const int& value) {
85          if (head == nullptr) {
86              head = new Node{value, nullptr, nullptr};
87              tail = head;
88          } else {
89              Node* newNode = new Node{value, head, nullptr};
90              head->previous = newNode;
```

```
            head->previous = newNode;
            head = newNode;
        }
    }

    void save_to_file(const string& filename) {
        ofstream file(filename);
        if (!file) {
            cerr << "The file does not exist!" << endl;
            return;
        }
        Node* current = head;
        while (current != nullptr) {
            file << current->data << " ";
            current = current->next;
        }
        file.close();
        cout << "The list was saved to a file \"" << filename << "\"." << endl;
    }

    void clear_list() {
        while (head != nullptr) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        cout << "The list was destroyed." << endl;
    }

    void list_restore(const string& filename) {
        ifstream file(filename);
        if (!file) {
            cerr << "Error: opening file for reading!" << endl;
            return;
        }
        clear_list();
        int key;
        while (file >> key) {
            push_front(key);
        }
        file.close();
        cout << "The list was restored from file '" << filename << "'." << endl;
    }
    }

    ~DoubleLinkedList() {
        clear_list();
    }
};

int main() {
    DoubleLinkedList list;

    for (int value = 0; value < 10; value++) {
        list.add(value);
    }
    cout << "The list:" << endl;
    list.print_list();

    list.push_front(2);
    cout << "The list after changes:" << endl;
    list.print_list();

    list.delete_by_index(2);
    list.print_list();

    list.save_to_file("list.txt");

    list.clear_list();
    cout << "The cleared list:" << endl;
    list.print_list();

    list.list_restore("list.txt");
    cout << "List after recovery from file:" << endl;
    list.print_list();

    list.clear_list();
    cout << "List after final destruction:" << endl;
    list.print_list();

    return 0;
}
```

Фактичний час на реалізацію: 4 години

Завдання №2 VNS. Algotester task 5. V-1

```cpp
#include <iostream>

using namespace std;

void chosen_row(unsigned long long &board, int row) {
    for (int col = 0; col < 8; col++) {
        board ^= (1ULL  << (row * 8 + col));
    }
}

void chosen_column(unsigned  long long &board, int col) {
    for (int row = 0; row < 8; row++) {
        board ^= (1ULL << (row * 8 + col));
    }
}

int main() {
 unsigned long long board;
    int n;

    cin >> board >> n;

    for (int i = 0; i < n; i++) {
        int R, C;
        cin >> R >> C;

        R--;
        C--;

        chosen_row(board, R);
        chosen_column(board, C);
        board ^= (1ULL <<  (R * 8 + C));
    }

    cout << board << endl;

    return 0;
}
```

Фактичний час на реалізацію: 2 години

Завдання №3 Algotester task 7.8. V- 2

```cpp
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

enum Operation
{
    insert_arr,
    erase_arr,
    size_arr,
    capacity_arr,
    get_arr,
    set_arr,
    print_arr,
    invalid
};

Operation get_operation(const string &command)
{
    if (command == "insert")
        return insert_arr;
    if (command == "erase")
        return erase_arr;
    if (command == "size")
        return size_arr;
    if (command == "capacity")
        return capacity_arr;
    if (command == "get")
        return get_arr;
    if (command == "set")
        return set_arr;
    if (command == "print")
        return print_arr;

    return invalid;
}

class dynamic_array
{
private:
    int *data;
    int size_;
    int capacity_;

    void resize(int new_capacity)
    {
        int *new_data = new int[new_capacity];
```

```
        for (int i = 0; i < size_; i++)
        {
            new_data[i] = data[i];
        }
        delete[] data;
        data = new_data;
        capacity_ = new_capacity;
    }

public:
    dynamic_array() : size_(0), capacity_(1)
    {
        data = new int[capacity_];
    }

    ~dynamic_array()
    {
        delete[] data;
    }

    void insert(int index, int N, int *arr)
    {
        if (size_ + N > capacity_)
        {
            while (capacity_ <= size_ + N)
            {
                capacity_ *= 2;
            }
            resize(capacity_);
        }
        // move a part of the array to the right to insert by index
        for (int i = size_ - 1; i >= index; i--)
        {
            data[i + N] = data[i];
        }
        for (int i = 0; i < N; i++)
        {
            data[index + i] = arr[i];
        }
        size_ += N;
        if (size_ == capacity_)
        {
            capacity_ *= 2;
            resize(capacity_);
```

```
92              resize(capacity_);
93          }
94      }
95
96      void erase(int index, int n)
97      {
98          for (int i = index; i < size_ - n + n; i++)
99          {
100             data[i] = data[i + n];
101         }
102         size_ -= n;
103     }
104
105     int size()
106     {
107         return size_;
108     }
109
110     int capacity()
111     {
112         return capacity_;
113     }
114
115     int &operator[](int index) { return data[index]; }
116
117     friend ostream &operator<<(ostream &os, const dynamic_array &arr)
118     {
119         for (int i = 0; i < arr.size_; i++)
120         {
121             os << arr.data[i] << ' ';
122         }
123         return os;
124     }
125 };
126
127 int main()
128 {
129     int Q;
130     cin >> Q;
131     dynamic_array dynamic_arr;
132
133     for (int i = 0; i < Q; i++)
134     {
135         string option;
```

```
        string option;
        cin >> option;
        Operation operation = get_operation(option);

        switch (operation)
        {
        case insert_arr:
        {
            int index, N;
            cin >> index >> N;
            int arr[N];
            for (int i = 0; i < N; i++)
            {
                cin >> arr[i];
            }
            dynamic_arr.insert(index, N, arr);
            break;
        }
        case erase_arr:
        {
            int index, n;
            cin >> index >> n;
            dynamic_arr.erase(index, n);
            break;
        }
        case size_arr:
        {
            cout << dynamic_arr.size() << endl;
            break;
        }
        case capacity_arr:
        {
            cout << dynamic_arr.capacity() << endl;
            break;
        }
        case get_arr:
        {
            int index;
            cin >> index;
            cout << dynamic_arr[index] << endl;
            break;
        }
        case set_arr:
        {
            int index, value;
            cin >> index >> value;
            dynamic_arr[index] = value;
            break;
        }
        case print_arr:
        {
            cout << dynamic_arr << endl;
            break;
        }
        default:
            break;
        }
    }
}
```

Фактичний час на реалізацію: 4 години

Завдання №4 VNS. Algotester task 7.8. V- 2

```cpp
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

enum Operation
{
    insert_arr,
    erase_arr,
    size_arr,
    capacity_arr,
    get_arr,
    set_arr,
    print_arr,
    invalid
};

Operation get_operation(const string &command)
{
    if (command == "insert")
        return insert_arr;
    if (command == "erase")
        return erase_arr;
    if (command == "size")
        return size_arr;
    if (command == "capacity")
        return capacity_arr;
    if (command == "get")
        return get_arr;
    if (command == "set")
        return set_arr;
    if (command == "print")
        return print_arr;

    return invalid;
}

template <typename T = int>
class dynamic_array
{
private:
    T *data;
    int size_;
    int capacity_;

    void resize(int new_capacity)
    {
```

```
49          T *new_data = new T[new_capacity];
50          for (int i = 0; i < size_; i++)
51          {
52              new_data[i] = data[i];
53          }
54          delete[] data;
55          data = new_data;
56          capacity_ = new_capacity;
57      }
58
59  public:
60      dynamic_array() : size_(0), capacity_(1)
61      {
62          data = new T[capacity_];
63      }
64
65      ~dynamic_array()
66      {
67          delete[] data;
68      }
69
70      void insert(int index, int N, T *arr)
71      {
72          if (size_ + N > capacity_)
73          {
74              while (capacity_ <= size_ + N)
75              {
76                  capacity_ *= 2;
77              }
78              resize(capacity_);
79          }
80
81          for (int i = size_ - 1; i >= index; i--)
82          {
83              data[i + N] = data[i];
84          }
85
86          for (int i = 0; i < N; i++)
87          {
88              data[index + i] = arr[i];
89          }
90
91          size_ += N;
92          if (size_ == capacity_)
93          {
```

```
93          {
94              capacity_ *= 2;
95              resize(capacity_);
96          }
97      }
98
99      void erase(int index, int n)
100     {
101         for (int i = index; i < size_ - n; i++)
102         {
103             data[i] = data[i + n];
104         }
105         size_ -= n;
106     }
107
108     int size()
109     {
110         return size_;
111     }
112
113     int capacity()
114     {
115         return capacity_;
116     }
117
118     T &operator[](int index) { return data[index]; }
119
120     friend ostream &operator<<(ostream &os, const dynamic_array &arr)
121     {
122         for (int i = 0; i < arr.size_; i++)
123         {
124             os << arr.data[i] << ' ';
125         }
126         return os;
127     }
128 };
129
130 int main()
131 {
132     int Q;
133     cin >> Q;
134
135     dynamic_array<int> dynamic_arr;
136
137     for (int i = 0; i < Q; i++)
```

```
    for (int i = 0; i < Q; i++)
    {
        string option;
        cin >> option;
        Operation operation = get_operation(option);

        switch (operation)
        {
        case insert_arr:
        {
            int index, N;
            cin >> index >> N;
            int arr[N];
            for (int i = 0; i < N; i++)
            {
                cin >> arr[i];
            }
            dynamic_arr.insert(index, N, arr);
            break;
        }
        case erase_arr:
        {
            int index, n;
            cin >> index >> n;
            dynamic_arr.erase(index, n);
            break;
        }
        case size_arr:
        {
            cout << dynamic_arr.size() << endl;
            break;
        }
        case capacity_arr:
        {
            cout << dynamic_arr.capacity() << endl;
            break;
        }
        case get_arr:
        {
            int index;
            cin >> index;
            cout << dynamic_arr[index] << endl;
            break;
        }
        case set_arr:
        {
            int index, value;
            cin >> index >> value;
            dynamic_arr[index] = value;
            break;
        }
        case print_arr:
        {
            cout << dynamic_arr << endl;
            break;
        }
        default:
            break;
        }
    }
}
```

Фактичний час на реалізацію: 4 години

Завдання №5 Class Practice Work

```cpp
#include <iostream>

using namespace std;

struct Node
{
    int data;
    Node *next;
    Node(int value) : data(value), next(nullptr) {}
};

Node *reverse(Node *head)
{
    Node *previous = nullptr;
    Node *current = head;
    Node *next = nullptr;

    while (current != nullptr)
    {
        next = current->next;
        current->next = previous;
        previous = current;
        current = next;
    }
    return previous;
}

bool compare(Node *h1, Node *h2)
{
    while (h1 != nullptr && h2 != nullptr)
    {
        if (h1->data != h2->data)
        {
            return false;
        }
        h1 = h1->next;
        h2 = h2->next;
    }
    return h1 == nullptr && h2 == nullptr;
}

Node *add(Node *n1, Node *n2)
{
    Node dummy(0);
    Node *current = &dummy;
    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry)
```

```
    while (n1 != nullptr || n2 != nullptr || carry)
    {
        int sum = carry;
        if (n1 != nullptr)
        {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2 != nullptr)
        {
            sum += n2->data;
            n2 = n2->next;
        }
        carry = sum / 10;
        current->next = new Node(sum % 10);
        current = current->next;
    }
    return dummy.next;
}

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

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

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

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

    return new_root;
}

void tree_sum(TreeNode *root)
{
    if (root == nullptr)
        return;

```

```
93      |
94      int left_sum = 0;
95      int right_sum = 0;
96
97      if (root->left != nullptr)
98      {
99      |   left_sum += root->left->value;
100     }
101     if (root->right != nullptr)
102     {
103     |   right_sum += root->right->value;
104     }
105
106     tree_sum(root->left);
107     tree_sum(root->right);
108
109     if (!root->left && !root->right)
110     |   return;
111     root->value = left_sum + right_sum;
112 }
113
114 void printList(Node *head)
115 {
116     while (head != nullptr)
117     {
118     |   cout << head->data << " ";
119     |   head = head->next;
120     }
121     cout << endl;
122 }
123
124 void printTree(TreeNode *root)
125 {
126     if (root != nullptr)
127     {
128     |   printTree(root->left);
129     |   cout << root->value << " ";
130     |   printTree(root->right);
131     }
132 }
133
134 int main()
135 {
136     Node *head = new Node(7);
137     head->next = new Node(9);
138     head->next->next = new Node(11);
```

```
    head->next->next = new Node(11);
    head->next->next->next = new Node(12);

    cout << "Linked List with no changes: ";
    printList(head);

    head = reverse(head);
    cout << "Reversed Linked List: ";
    printList(head);

    Node* list1 = new Node(2);
    list1->next = new Node(6);
    list1->next->next = new Node(7);

    Node* list2 = new Node(3);
    list2->next = new Node(5);
    list2->next->next = new Node(8);

    cout << "Are lists equal? " << (compare(list1, list2) ? "Yes" : "No") << endl;

    list2->next->next->data = 4;
    cout << "Are lists equal after changes? " << (compare(list1, list2) ? "Yes" : "No") << endl;

    Node* num1 = new Node(5);
    num1->next = new Node(6);
    num1->next->next = new Node(7);

    Node* num2 = new Node(1);
    num2->next = new Node(2);
    num2->next->next = new Node(1);

    Node* sum = add(num1, num2);
    cout << "Sum: ";
    printList(sum);

    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(3);
    root->right = new TreeNode(4);
    root->left->left = new TreeNode(2);
    root->left->right = new TreeNode(5);

    cout << "Original Tree: ";
    printTree(root);
    cout << endl;

    TreeNode* mirroredRoot = create_mirror_flip(root);
    TreeNode* mirroredRoot = create_mirror_flip(root);
    cout << "Mirrored Tree: ";
    printTree(mirroredRoot);
    cout << endl;

    TreeNode* sumTreeRoot = new TreeNode(1);
    sumTreeRoot->left = new TreeNode(2);
    sumTreeRoot->right = new TreeNode(3);
    sumTreeRoot->left->left = new TreeNode(4);
    sumTreeRoot->left->right = new TreeNode(5);

    cout << "Tree before summing subtrees: ";
    printTree(sumTreeRoot);
    cout << endl;

    tree_sum(sumTreeRoot);
    cout << "Tree after: ";
    printTree(sumTreeRoot);
    cout << endl;

    return 0;
}
```

Фактичний час на реалізацію: 4 години

Завдання №6 Self Practice Algotester Task

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
int main() {
    int N, M, x, y;
    cin >> N >> M;
    cin >> x >> y;

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

    x--; y--;

    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 el = q.front();
        q.pop();

        for (int i = 0; i < 4; ++i) {
            int nx = el.first + dx[i];
            int ny = el.second + dy[i];

            if (nx >= 0 && nx < N && ny >= 0 && ny < M && height[nx][ny] == -1) {
                height[nx][ny] = height[el.first][el.second] + 1;
                q.push({ nx, ny });
            }
        }
    }
    int maxHeight = 0;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            maxHeight = max(maxHeight, height[i][j]);
        }
    }
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            cout << maxHeight - height[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Фактичний час на реалізацію: 2 години

4. ***Результати виконання завдань, тестування:***

Завдання №1 VNS. Лабораторна робота №10

```
The list:
0 1 2 3 4 5 6 7 8 9
The list after changes:
2 0 1 2 3 4 5 6 7 8 9
2 0 2 3 4 5 6 7 8 9
The list was saved to a file "list.txt".
The list was destroyed.
The cleared list:
The list is empty
The list was destroyed.
The list was restored from file 'list.txt'.
List after recovery from file:
9 8 7 6 5 4 3 2 0 2
The list was destroyed.
List after final destruction:
The list is empty
The list was destroyed.
```

Завдання №2 VNS. Algotester task 5. V-1

```
0
4
1 1
1 2
2 2
2 1
771
```

Завдання №3 Algotester task 7.8. V - 2

```
12

size
0

insert 0 5
251 252 253 254 255

size
5
capacity
8
print
251 252 253 254 255

get 1
252
set 1 777
get 1
777

erase 1 3

get 1
255

size
2
print
251 255
```

Завдання №4 Algotester task 7.8. V- 2

```
12

size
0

insert 0 5
251 252 253 254 255

size
5
capacity
8
print
251 252 253 254 255

get 1
252
set 1 777
get 1
777

erase 1 3

get 1
255

size
2
print
251 255
```

Завдання №5 Class Practice Work

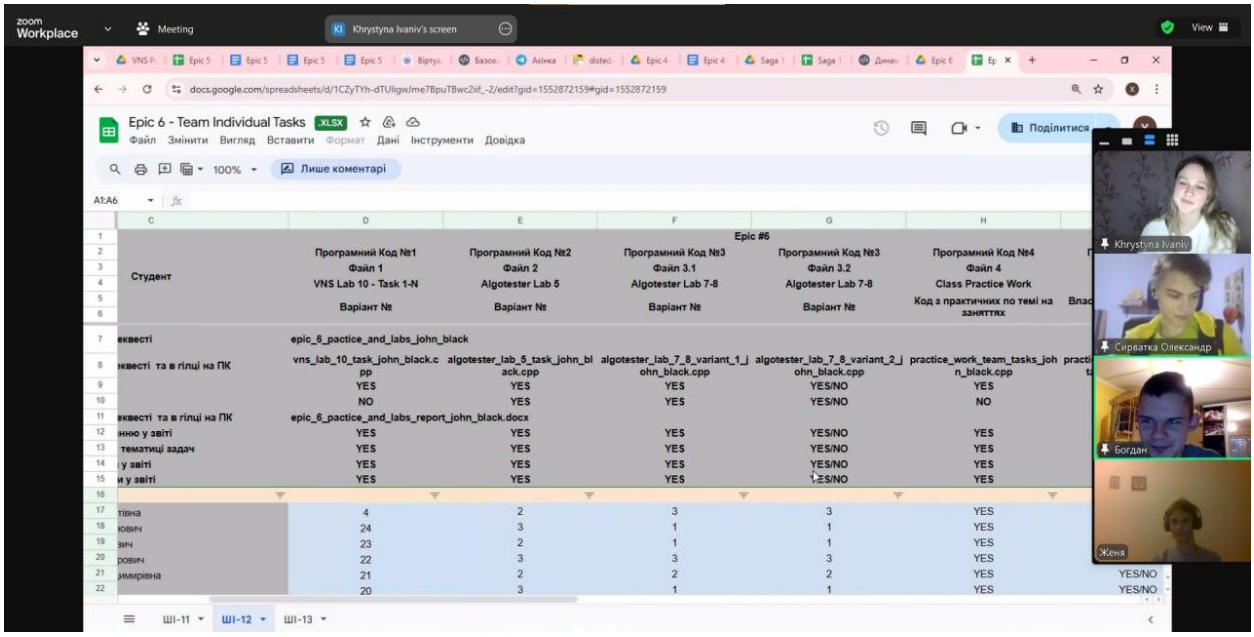
```
Linked List with no changes: 7 9 11 12
Reversed Linked List: 12 11 9 7
Are lists equal? No
Are lists equal after changes? No
Sum: 6 8 8
Original Tree: 2 3 5 1 4
Mirrored Tree: 4 1 5 3 2
Tree before summing subtrees: 4 2 5 1 3
Tree after: 4 9 5 5 3
```

Завдання №6 Self Practice Algotester Task

```
8 9 8 7 6 5 4 3 2
7 8 7 6 5 4 3 2 1
6 7 6 5 4 3 2 1 0
```

5 . Кооперація з командою:

Провели зустріч у зумі, обговорили деталі виконянна завдань



Висновок: Під час виконання епіку *я* ознайомилась з динамічними структурами (Черга, Стек, Списки, Дерево) та алгоритмами обробки динамічних структур.