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

Звіт

про виконання лабораторних та практичних робіт блоку № 6
На тему: «Програмування: алгоритм, програма, код. Системи числення. Двійкова система числення. Розробка та середовище розробки програми.»
з дисципліни: «Основи програмування»
до:
Практичних Робіт до блоку № 6

Виконав:
Студент групи ШІ-11
Мартин Максим

Львів 2024

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

Мета: Засвоїти основи роботи з динамічними структурами даних, такими як черга, стек, списки та дерева. Ознайомитися з алгоритмами їх обробки для розв'язання різноманітних задач.

Теоретичні відомості та джерела:

Динамічні структури

-Урок №89. Динамічне виділення пам’яті;
-Урок №90. Динамічні масиви;

-Стек;
C++ • Теорія • Урок 58 • Стек, Куча, Статична пам'ять
https://www.youtube.com/watch?v=B3VHHfMW0Pg

-Черга;
#4
#5

-Списки:
#1
#2
Урок #133
Урок #134
Урок #135

-Дерева:
#3
C++ • Теорія • Урок 144 • ADT • Бінарне дерево

Виконання роботи:
VNS Lab 10v11:

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

struct Node {
 char* key;
 Node* prev;
 Node* next;

 Node(const char* k) : prev(nullptr), next(nullptr) {
  key = new char[strlen(k) + 1];
  strcpy(key, k);
 }

 ~Node() {
  delete[] key;
 }
};

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

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

 void addAtPosition(const char* key, int position) {
  Node* newNode = new Node(key);

  if (position == 0) {
   newNode->next = head;
   if (head) {
    head->prev = newNode;
   }
   head = newNode;
   if (!tail) {
    tail = newNode;
   }
   return;
  }

  Node* current = head;
  for (int i = 0; i < position - 1 && current; ++i) {
   current = current->next;
  }

  if (!current) {
   cout << "Position " << position << " exceeds list size.\n";
   delete newNode;
   return;
  }
```

```cpp
  newNode->next = current->next;
  newNode->prev = current;

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

  current->next = newNode;
 }

 void deleteByKey(const char* key) {
  Node* current = head;
  while (current) {
   if (strcmp(current->key, key) == 0) {
    if (current->prev) {
     current->prev->next = current->next;
    } else {
     head = current->next;
    }
    if (current->next) {
     current->next->prev = current->prev;
    } else {
     tail = current->prev;
    }
    delete current;
    cout << "Element with key \"" << key << "\" deleted.\n";
    return;
   }
   current = current->next;
  }
  cout << "Element with key \"" << key << "\" not found.\n";
 }

 void printList() {
  Node* current = head;
  cout << "List: ";
  while (current) {
   cout << current->key << " ";
   current = current->next;
  }
  cout << endl;
 }

```

```
 ~DoublyLinkedList() {
  Node* current = head;
  while (current) {
   Node* nextNode = current->next;
   delete current;
   current = nextNode;
  }
 }
};

int main() {
 DoublyLinkedList list;

 list.addAtPosition("first", 0);
 list.addAtPosition("second", 1);
 list.addAtPosition("third", 2);
 list.printList();

 list.deleteByKey("second");
 list.printList();

 list.addAtPosition("new", 1);
 list.printList();

 return 0;
}
```

```
List: first second third
Element with key "second" deleted.
List: first third
List: first new third
```

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

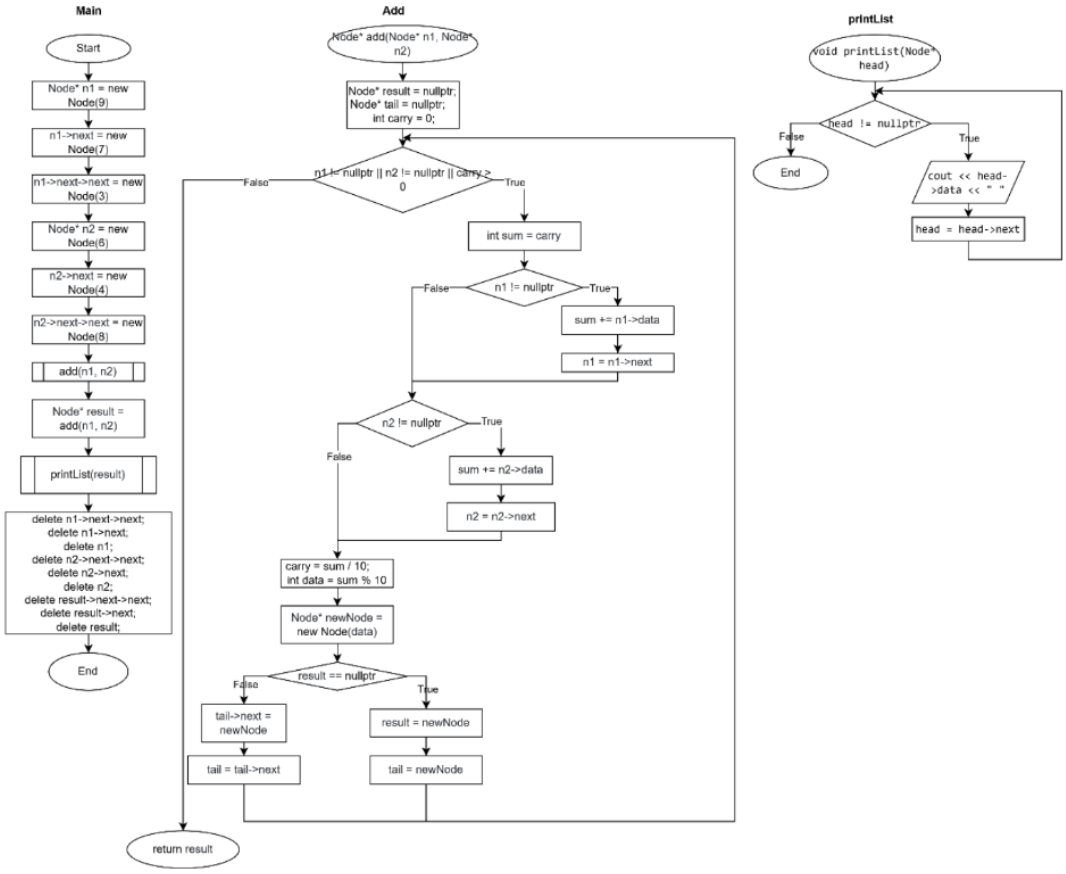
Node* add(Node *n1, Node *n2);

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 ⟹ 9→7→3);
- функція повертає новий список, передані в функцію списки не модифікуються.

Блок схема до завдання Class Practice Work – Task 3

Додавання великиких чисел



Код до задач 1-3

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

struct Node {
 int data;
 Node* next;

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

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

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

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

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* result = nullptr;
 Node* tail = 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;

  Node* newNode = new Node(digit);
  if (result == nullptr) {
```

```
    Node* newNode = new Node(digit);
    if (result == nullptr) {
     result = newNode;
     tail = result;
    } else {
     tail->next = newNode;
     tail = tail->next;
    }
   }
   return result;
  }

  int main() {
   Node* list1 = new Node(1);
   list1->next = new Node(2);
   list1->next->next = new Node(3);

   Node* list2 = new Node(1);
   list2->next = new Node(2);
   list2->next->next = new Node(3);

   cout << "Original list 1: ";
   printList(list1);
   list1 = reverse(list1);
   cout << "Reversed list 1: ";
   printList(list1);

   cout << "Comparison of lists: ";
   if (compare(list1, list2)) {
    cout << "Lists are equal." << endl;
   } else {
    cout << "Lists are not equal." << endl;
   }

   Node* num1 = new Node(9);
   num1->next = new Node(7);
   num1->next->next = new Node(3);

   Node* num2 = new Node(4);
   num2->next = new Node(6);
   num2->next->next = new Node(5);

   cout << "Number 1: ";
   printList(num1);
   cout << "Number 2: ";
   printList(num2);

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

   return 0;
  }
```

```
Original list 1: 1 2 3
Reversed list 1: 3 2 1
Comparison of lists: Lists are not equal.
Number 1: 9 7 3
Number 2: 4 6 5
Sum of numbers: 3 4 9
```

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

Код до задач 4 -5

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

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

// Функція для створення дзеркального дерева
TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) {
        return nullptr;
    }
    TreeNode* newRoot = new TreeNode(root->data);
    newRoot->left = create_mirror_flip(root->right);
    newRoot->right = create_mirror_flip(root->left);
    return newRoot;
}

// Функція для обчислення суми піддерев
int treeSum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }
    if (root->left == nullptr && root->right == nullptr) {
        return root->data;
    }
    int leftSum = treeSum(root->left);
    int rightSum = treeSum(root->right);
    root->data = leftSum + rightSum;
    return root->data;
}

// Функція для друку дерева
void printTree(TreeNode* root) {
    if (root == nullptr) {
        return;
    }
    cout << root->data << " ";
    printTree(root->left);
    printTree(root->right);
}
```

```cpp
int main() {
    // Початкове дерево
    /*
        Original tree:
              1
             /   \
            2     3
           / \   / \
          4   5 6   7
    */

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

    // Дзеркальне дерево
    TreeNode* mirroredTree = create_mirror_flip(root);
    cout << "Mirrored tree: ";
    printTree(mirroredTree);
    cout << endl;

    // Обчислення суми піддерев
    treeSum(root);
    cout << "Tree after calculating sum in each root: ";
    printTree(root);
    cout << endl;

    return 0;
}

```

```
Original tree: 1 2 4 5 3 6 7
Mirrored tree: 1 3 7 6 2 5 4
Tree after calculating sum in each root: 22 9 4 5 13 6 7
PS C:\Users\Maks\Documents\ai_programming_playground_2024\ai_11\maksym_martyn\epic_6>
```

Algotester Lab 5v2:

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.

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

using namespace std;

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

  vector<vector<char>> arr(N, vector<char>(M));

  for (int i = 0; i < N; i++) {
    string v;
    cin >> v;
    for (int j = 0; j < M; j++) {
      arr[i][j] = v[j];
    }
  }

  for (int j = 0; j < M; j++) {
    for (int i = N - 2; i >= 0; i--) {
      if (arr[i][j] == 'S') {
        int k = i;
        while (k + 1 < N && arr[k + 1][j] == 'O') {
          swap(arr[k][j], arr[k + 1][j]);
          k++;
        }
      }
    }
  }

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

  return 0;
}

```

```
3 3
SOX
OOX
XXO
OOX
SOX
XXO
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
19 hours ago	C++ 23	Accepted	0.037	2.031	View

Showing 1 to 1 of 1 rows

Algotester Lab 7-8 v1:

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 <<

Реалізація програми з структурою

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

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

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

struct BinarySearchTree {
    TreeNode* root;

    BinarySearchTree() : root(nullptr) {}

    void insert(int value) {
        root = insert(root, value);
    }

    void print() const {
        print(root);
        std::cout << std::endl;
    }

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

    int size() const {
        return size(root);
    }

private:
    TreeNode* insert(TreeNode* node, int value) {
        if (node == nullptr) {
            return new TreeNode(value);
        }
        if (value < node->data) {
            node->left = insert(node->left, value);
        } else if (value > node->data) {
            node->right = insert(node->right, value);
        }
        return node;
    }

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

    void print(TreeNode* node) const {
        if (node != nullptr) {
            print(node->left);
            std::cout << node->data << " ";
            print(node->right);
        }
    }

    int size(TreeNode* node) const {
        if (node == nullptr) return 0;
        return 1 + size(node->left) + size(node->right);
    }
};

int main() {
    BinarySearchTree myTree;
    int N;
    std::cin >> N;

    for (int i = 0; i < N; ++i) {
        std::string operation;
        std::cin >> operation;

        if (operation == "insert") {
            int value;
            std::cin >> value;
            myTree.insert(value);
        } else if (operation == "print") {
            myTree.print();
        } else if (operation == "contains") {
            int value;
            std::cin >> value;
            std::cout << (myTree.contains(value) ? "Yes" : "No") << std::endl;
        } else if (operation == "size") {
            std::cout << myTree.size() << std::endl;
        }
    }

    return 0;
}
```

```
6
insert 10
insert 5
insert 15
print
5 10 15
contains 10
Yes
size
3
```

Реалізація програми з класом

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

template <typename T>
class TreeNode {
public:
 T data;
 TreeNode* left;
 TreeNode* right;

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

template <typename T>
class BinarySearchTree {
public:
 BinarySearchTree() : root(nullptr) {}

 void insert(T value) {
  root = insert(root, value);
 }

 void print() const {
  print(root);
  std::cout << std::endl;
 }

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

 int size() const {
  return size(root);
 }

private:
 TreeNode<T>* root;

 TreeNode<T>* insert(TreeNode<T>* node, T value) {
  if (node == nullptr) {
   return new TreeNode<T>(value);
  }
  if (value < node->data) {
   node->left = insert(node->left, value);
  } else if (value > node->data) {
   node->right = insert(node->right, value);
  }
  return node;
 }

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

 void print(TreeNode<T>* node) const {
  if (node != nullptr) {
   print(node->left);
   std::cout << node->data << " ";
   print(node->right);
  }
 }

 int size(TreeNode<T>* node) const {
  if (node == nullptr) return 0;
  return 1 + size(node->left) + size(node->right);
 }
};

int main() {
 BinarySearchTree<int> myTree;
 int N;
 std::cin >> N;

 for (int i = 0; i < N; ++i) {
  std::string operation;
  std::cin >> operation;

  if (operation == "insert") {
   int value;
   std::cin >> value;
   myTree.insert(value);
  } else if (operation == "print") {
   myTree.print();
  } else if (operation == "contains") {
   int value;
   std::cin >> value;
   std::cout << (myTree.contains(value) ? "Yes" : "No") << std::endl;
  } else if (operation == "size") {
   std::cout << myTree.size() << std::endl;
  }
 }

 return 0;
}
```

```
4
insert 10
insert 52
insert 52
contains 7
No
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
21 hours ago	C++ 23	Accepted	0.008	1.434	View
21 hours ago	C++ 23	Accepted	0.008	1.605	View

Showing 1 to 2 of 2 rows

Self Practice

Найпростіші запити

Задано масив a із n цілих чисел. Потрібно відповісти на m запитів, кожен з яких одного із двох типів:

1. знайти суму елементів масиву на проміжку від l до r включно,
2. додати число d до i-го елементу масиву.

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

//Найпростіші запити

int main() {
 int n, q; // розмір масиву і к-сть запитань
 cin >> n >> q;

 vector<int> arr(n + 1);
 vector<int> prefix(n + 1, 0);

 for (int i = 1; i <= n; i++) {
  cin >> arr[i];
  prefix[i] = prefix[i - 1] + arr[i];
 }

 vector<int> results;

 while (q--) {
  int type;
  cin >> type;
  //типи запитань 1) l - r ( префіксний[l] - префіксний[r] )   2) до елемента масиву + число
  if (type == 1) {
   int l, r;
   cin >> l >> r;
   results.push_back(prefix[r] - prefix[l - 1]);

  } else if (type == 2) {
   int idx, value;
   cin >> idx >> value;
   int diff = value;
   arr[idx] += diff;

   for (int i = idx; i <= n; i++) {
    prefix[i] += diff;
   }
  }
 }

 if (results.empty()) {
  cout << "There were no requests for calculating the amount." << endl;
 } else {
  for (int result : results) {
   cout << result << endl;
  }
 }

 return 0;
}
```

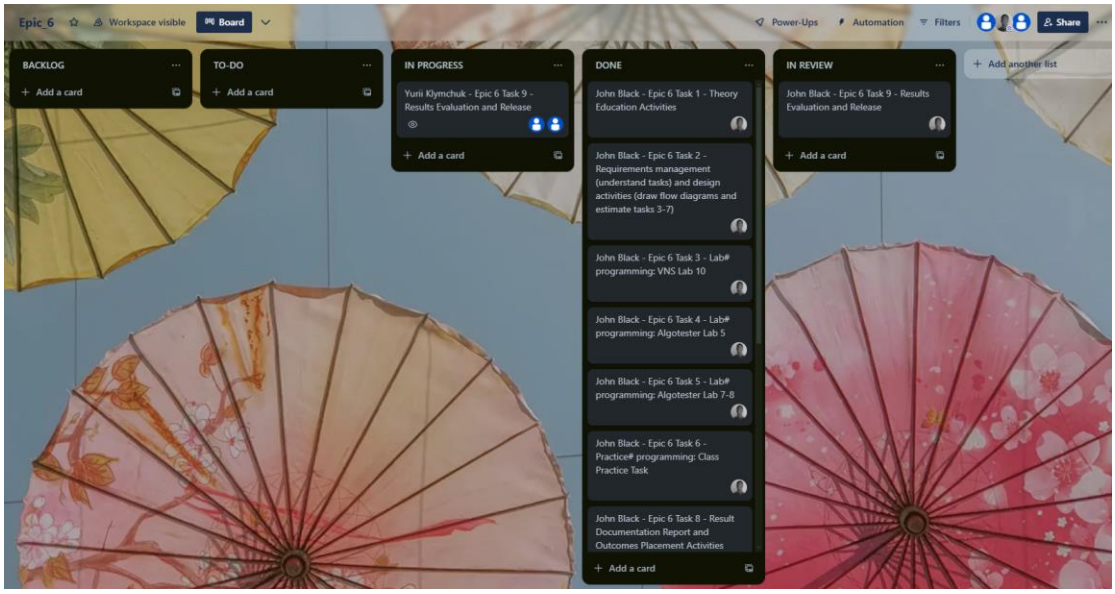
```
4 2
5 6 7 8
2 1 1
1 1 3
19
PS C:\Users\Maks\Documents\ai_programming_playground_2024\ai_11\maksym_martyn\epic_6>
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
декілька секунд тому	C++ 23	Зараховано	1.533	3.031	Перегляд
9 хвилин тому	C++ 23	Зараховано	1.533	2.730	Перегляд
годину тому	C++ 23	Зараховано	1.567	2.664	Перегляд
годину тому	C++ 23	Ліміт часу 22	2.010	2.414	Перегляд
годину тому	C++ 23	Ліміт часу 18	2.010	2.324	Перегляд
годину тому	C++ 23	Ліміт часу 18	2.013	2.348	Перегляд
годину тому	C++ 23	Ліміт часу 18	2.014	2.465	Перегляд
22 дні тому	C++ 23	Неправильна відповідь 1	0.002	0.980	Перегляд

Showing 1 to 8 of 8 rows

Робота з командою:

Налаштування Trello



Висновки:

Отже, в межах цього епіка я зрозумів, що таке списки, дерева та як їх реалізовувати в коді. Практикувався з записом даних у файли, а також покращив роботу з масивами та алгоритмами.