

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

Виконав:

Студент групи ІІІ-12

Стик Назарій Олегович

Тема роботи:

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

Мета роботи:

Ознайомлення з основами роботи з динамічними структурами даних, такими як черга, стек, списки та дерево. Вивчення їхньої внутрішньої організації, ключових властивостей і методів доступу. Розгляд алгоритмів для роботи з динамічними структурами, зокрема для додавання, видалення, пошуку та сортування елементів. Вивчення ефективності алгоритмів обробки динамічних структур та їх застосування в програмуванні. Особлива увага приділяється практичному використанню стандартних бібліотек для реалізації цих структур і створенню власних оптимізованих реалізацій.

Теоретичні відомості:

У даній роботі розглядаються основи роботи з динамічними структурами даних, такими як черга, стек, списки та дерево. Черга є структурою даних типу FIFO (перший увійшов — перший вийшов), яка використовується для організації послідовності задач або процесів. Стек реалізує принцип LIFO (останній увійшов — перший вийшов) і широко застосовується для управління викликами функцій та обробки рекурсій. Списки забезпечують зручну роботу з динамічною кількістю елементів завдяки зв'язкам між вузлами, а дерева представляють ієрархічні структури даних, які ефективно застосовуються для пошуку, сортування та організації даних. Розглядаються алгоритми роботи з динамічними структурами, включаючи додавання, видалення, пошук та оновлення елементів. Вивчаються різні способи реалізації динамічних структур за допомогою стандартних бібліотек мов програмування, а також створення власних реалізацій для забезпечення високої ефективності та адаптивності коду. Окрему увагу приділено оцінці часової та просторової складності алгоритмів для визначення їхньої оптимальності у різних сценаріях використання.

Джерела:

- Декілька відео на YouTube:
 - Динамічні структури - <https://www.youtube.com/watch?v=A3ZUpyrnCbM&t=1s>
 - Основи динамічного розподілу пам'яті - <https://www.youtube.com/watch?v=udfbq4M2Kfc>
 - Стек - https://www.youtube.com/watch?v=GBST5uQ_yos
- Певну інформацію брав на сайтах:
 - Динамічні структури - <https://studfile.net/preview/7013685/page:10/>

- Основи динамічного розподілу пам'яті - <https://studfile.net/preview/9189217/page:15/>
 - Стек - <https://itproger.com/ua/spravka/cpp/stack>
-
- Також вивчив багато інформації за допомогою ChatGPT.

Виконання роботи:

- **Завдання №1:** Theory Education Activities

Очікувано часу: **4 дні.**

Витрачено часу: **3 дні.**

- **Завдання №2:** Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)

Очікувано часу: **1 година.**

Витрачено часу: **1 година.**

- **Завдання №3:** Lab# programming: VNS Lab 10 (Variant 19)

Умова:

19.Записи в лінійному списку містять ключове поле типу `*char` (рядок символів). Сформувати двонаправлений список. Знищити `K` елементів із заданими номерами. Додати `K` елементів у початок списку.

Код:

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct Node
6  {
7      char* data;
8      Node* next;
9      Node* prev;
10 };
11
12 Node* CreateNewNode(const char* str)
13 {
14     // 1. Виділяємо пам'ять
15     Node* newNode = new Node;
16
17     // 2. Копіюємо рядок у вузол
18     newNode->data = new char[strlen(str) + 1];
19     strcpy(newNode->data, str);
20
21     // 3. Ініціалізуємо вказівники
22     newNode->next = nullptr;
23     newNode->prev = nullptr;
24
25     // 4. Повертаємо адресу нового вузла
26     return newNode;
27 }
28
29 void AddNodeToFront(Node*& head, const char* str)
30 {
31     // 1. Створимо новий вузол
32     Node* newNode = CreateNewNode(str);
33
34     // 2. Якщо список не порожній – оновлюємо вказівники
35     if (head != nullptr)
36     {
37         head->prev = newNode;
38         newNode->next = head;
39     }
40
41     // Змінюємо голову списку
42     head = newNode;
43 }
44
45 void DeleteNode(Node*& head, int index)
46 {
47     if (head == nullptr)
48     {
49         return;
50     }
51
52     Node* current = head;
53     int currentIndex = 0;
54
55     while (current != nullptr && index > 0)
56     {
57         current = current->next;
58         currentIndex++;
59     }
60 }
```

```
61
62     if (current == nullptr)
63     {
64         return;
65     }
66
67     if (current->prev != nullptr)
68     {
69         current->prev->next = current->next;
70     }
71     else
72     {
73         head = current->next;
74     }
75     if (current->next != nullptr)
76     {
77         current->next->prev = current->prev;
78     }
79
80     delete[] current->data;
81     delete current;
82 }
83
84 void DeleteKNodes(Node*& head, int K, const int* indices)
85 {
86     for (int i = 0; i < K; i++)
87     {
88         DeleteNode(head, indices[i]-1);
89     }
90 }
91
92 void AddKNodes(Node*& head, int K, const char** data)
93 {
94     for (int i = 0; i < K; i++)
95     {
96         AddNodeToFront(head, data[i]);
97     }
98 }
99
100 int main()
101 {
102     int K; // Кількість елементів щоб видалити та додати
103     int* indices; // Масив індексів елементів щоб видалити
104     Node* head = nullptr;
105
106     AddNodeToFront(head, "Hello");
107     AddNodeToFront(head, "World");
108     AddNodeToFront(head, "How are you?");
109     AddNodeToFront(head, "I am fine");
110     AddNodeToFront(head, "Thank you");
111
112     cout << "First list: ";
113     Node* current = head;
114     while (current != nullptr)
115     {
116         cout << current->data << " ";
117         current = current->next;
118     }
119 }
```

```

118     }
119     cout << endl;
120
121     cout << "Enter the number of elements to delete and add: ";
122     cin >> K;
123     indices = new int[K];
124     cout << "Enter the indices of elements to delete: ";
125     for (int i = 0; i < K; i++)
126     {
127         cin >> indices[i];
128     }
129
130     DeleteKNodes(head, K, indices);
131
132     const char* newData[] = {"Good", "Morning", "My", "Friend"};
133     AddKNodes(head, K, newData);
134
135     cout << "Second list: ";
136     current = head;
137     while (current != nullptr)
138     {
139         cout << current->data << " ";
140         current = current->next;
141     }
142     cout << endl;
143
144     delete[] indices;
145     while (head != nullptr)
146     {
147         Node* temp = head;
148         head = head->next;
149         delete[] temp->data;
150         delete temp;
151     }
152
153     return 0;
154 }

```

Результат:

```

First list: Thank you I am fine How are you? World Hello
Enter the number of elements to delete and add: 2
Enter the indices of elements to delete: 2 3
Second list: Morning Good Thank you I am fine How are you? World Hello
PS D:\VS Code\Projects>

```

Очікувано часу: **1 година.**

Витрачено часу: **2 години.**

- **Завдання №4:** Lab# programming: Algotester Lab 5 (Variant 2)

Умова:

Lab 5v2

Обмеження: 1 сек., 256 MiB

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N , ширина - M .
Всередині печери є пустота, пісок та каміння. Пустота позначається буквою X , пісок S і каміння O .
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 - i -й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані

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

Обмеження

$1 \leq N, M \leq 1000$
 $|row_i| = M$
 $row_i \in \{X, S, O\}$

Приклади

Вхідні дані (<i>stdin</i>)	Вихідні дані (<i>stdout</i>)
5 5 SSOSS OOOOO SOOXX OOOOS OOSOO	OOOOO OOOSS OOOXX SOOOO SSSOS

Посилання:

<https://algotester.com/uk/ContestProblem/DisplayWithEditor/135602>

Код:

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main()
7  {
8      int M; // висота
9      int N; // ширина
10     do
11     {
12         cin >> M >> N;
13     } while (!(N >= 1 && N <= 1000 && M >= 1 && M <= 1000));
14
15     vector<vector<char>> cave(M, vector<char>(N));
16
17     for (int i = 0; i < M; i++)
18     {
19         for (int j = 0; j < N; j++)
20         {
21             char material;
22             cin >> material;
23             if (material == 'S' || material == 'O' || material == 'X')
24             {
25                 cave[i][j] = material;
26             }
27         }
28     }
29
30     for (int i = M - 2; i >= 0; i--) // M-2, бо останній рядок не обробляється
31     {
32         for (int j = 0; j < N; j++)
33         {
34             if (cave[i][j] == 'S')
35             {
36                 int k = 1;
37                 while (k + 1 < M && cave[k + 1][j] == 'O')
38                 {
39                     cave[k + 1][j] = 'S';
40                     cave[k][j] = 'O';
41                     k++;
42                 }
43             }
44         }
45     }
46     cout << endl;
47
48     for (int i = 0; i < M; i++)
49     {
50         for (int j = 0; j < N; j++)
51         {
52             cout << cave[i][j];
53         }
54         cout << endl;
55     }
56 }
```

Результат:

```
3 3
SSS
OXO
OOX

OSO
OXS
SOX
PS D:\VS Code\Projects>
```

Очікувано часу: 1 година.

Витрачено часу: 1 година.

- **Завдання №5:** Lab# programming: Algotester Lab 7-8 (Variant 1,3)
Варіант – 1
Lab 78v1

Обмеження: 1 сек., 256 MiB

Ваше завдання - власноруч реалізувати структуру даних "Двотязний 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 написано 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 елемента**
Ідентифікатор - *get*
Ви отримуєте ціле число - *index*, індекс елемента.
Ви виводите значення елемента за індексом.
- **Модифікація значення i -го елемента**
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.

Посилання:

<https://algotester.com/uk/ContestProblem/DisplayWithEditor/135607>

Код:

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4
5  struct Node
6  {
7      int data;
8      Node* prev;
9      Node* next;
10
11      // Конструктор узла
12      Node(int val)
13      {
14          data = val;
15          prev = nullptr;
16          next = nullptr;
17      }
18  };
19
20  struct DoublyLinkedList
21  {
22      Node* head;
23      Node* tail;
24      int size;
25      DoublyLinkedList()
26      {
27          head = nullptr;
28          tail = nullptr;
29          size = 0;
30      }
31
32
33      void insert(int index, const std::vector<int>& values)
34      {
35          if (index < 0 || index > size)
36          {
37              return;
38          }
39
40          Node* current = head;
41          for (int i = 0; i < index && current; ++i)
42          {
43              current = current->next;
44          }
45
46          for (int value : values)
47          {
48              Node* new_node = new Node(value);
49
50              if (!head)
51              {
52                  head = tail = new_node;
53              }
54              else if (current == head)
55              {
56                  // начало
57                  new_node->next = head;
58                  head->prev = new_node;
59                  head = new_node;
60              }
61              else if (!current)
62              {
63                  // конец
64                  new_node->prev = tail;
65                  tail->next = new_node;
66                  tail = new_node;
67              }
68              else
69              {
70                  // середина
71                  new_node->prev = current->prev;
72                  new_node->next = current;
73                  current->prev->next = new_node;
74                  current->prev = new_node;
75              }
76
77              size++;
78          }
79      }
80
81      void erase(int index, int n)
82      {
83          if (index < 0 || index >= size) return;
84
85          Node* current = head;
86          for (int i = 0; i < index; ++i)
87          {
88              current = current->next;
89          }
90      }
91  }
```

```

87     }
88
89     for (int i = 0; i < n && current; ++i)
90     {
91         Node* to_delete = current;
92         current = current->next;
93
94         if (to_delete->prev) to_delete->prev->next = to_delete->next;
95         if (to_delete->next) to_delete->next->prev = to_delete->prev;
96         if (to_delete == head) head = to_delete->next;
97         if (to_delete == tail) tail = to_delete->prev;
98
99         delete to_delete;
100         size--;
101     }
102 }
103
104 int get(int index)
105 {
106     if (index < 0 || index >= size) return -1;
107
108     Node* current = head;
109     for (int i = 0; i < index; ++i)
110     {
111         current = current->next;
112     }
113     return current->data;
114 }
115
116 // Змінити значення за індексом
117 void set(int index, int value)
118 {
119     if (index < 0 || index >= size) return;
120
121     Node* current = head;
122     for (int i = 0; i < index; ++i)
123     {
124         current = current->next;
125     }
126     current->data = value;
127 }
128
129 int getSize()
130 {
131     return size;
132 }
133
134 void print()
135 {
136     Node* current = head;
137     while (current)
138     {
139         std::cout << current->data << " ";
140         current = current->next;
141     }
142     std::cout << std::endl;
143 }
144 };
145
146 int main()
147 {
148     DoublyLinkedList list;
149     int Q;
150     std::cin >> Q;
151
152     while (Q--)
153     {
154         std::string command;
155         std::cin >> command;
156
157         if (command == "insert")
158         {
159             int index, n;
160             std::cin >> index >> n;
161             std::vector<int> values(n);
162             for (int i = 0; i < n; ++i)
163             {
164                 std::cin >> values[i];
165             }
166             list.insert(index, values);
167         }
168         else if (command == "erase")
169         {
170             int index, n;
171             std::cin >> index >> n;
172             list.erase(index, n);
173         }
174         else if (command == "size")
175         {
176             std::cout << list.getSize() << std::endl;
177         }
178         else if (command == "get")
179         {
180             int index;
181             std::cin >> index;
182             std::cout << list.get(index) << std::endl;
183         }
184         else if (command == "set")
185         {
186             int index, value;
187             std::cin >> index >> value;
188             list.set(index, value);
189         }
190         else if (command == "print")
191         {
192             list.print();
193         }
194     }
195
196     return 0;
197 }
198

```

Результат:

```
9
insert
0
5
1 2 3 4 5

insert
2
3
7 7 7

print
1 2 7 7 7 3 4 5

erase
1 2

print
1 7 7 3 4 5

size
6

get
3
3

set
3 13

print
1 7 7 13 4 5
PS D:\VS Code\Projects>
```

Очікувано часу: **1 година.**

Витрачено часу: **2 години.**

Варіант – 2

Lab 78v3

Обмеження: 1 сек., 256 MiB

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 <<

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.

Посилання:

<https://algotester.com/uk/ContestProblem/DisplayWithEditor/135609>

Код:

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct Node
6  {
7      int data;
8      Node* left;
9      Node* right;
10
11     Node(int val)
12     {
13         data = val;
14         left = nullptr;
15         right = nullptr;
16     }
17 };
18
19
20 struct BinarySearchTree
21 {
22     Node* root;
23     int size;
24
25     BinarySearchTree()
26     {
27         root = nullptr;
28         size = 0;
29     }
30
31     void insert(int value)
32     {
33         if (!contains(value))
34         {
35             root = insertRec(root, value);
36             size++;
37         }
38     }
39
40     Node* insertRec(Node* node, int value)
41     {
42         if (node == nullptr)
43         {
44             return new Node(value);
45         }
46
47         if (value < node->data)
48         {
49             node->left = insertRec(node->left, value);
50         }
51         else
52         {
53             node->right = insertRec(node->right, value);
54         }
55
56         return node;
57     }
58
59     bool contains(int value)
60     {
61         return containsRec(root, value);
62     }
63
64     bool containsRec(Node* node, int value)
65     {
66         if (node == nullptr)
67         {
68             return false;
69         }
70         if (node->data == value)
71         {
72             return true;
73         }
74         if (value < node->data)
75         {
76             return containsRec(node->left, value);
77         }
78         else
79         {
80             return containsRec(node->right, value);
81         }
82     }
83 }
```

```

83
84     int getSize()
85     {
86         return size;
87     }
88
89     void print()
90     {
91         printRec(root);
92         cout << endl;
93     }
94
95     void printRec(Node* node)
96     {
97         if (node == nullptr)
98         {
99             return;
100         }
101         printRec(node->left);
102         cout << node->data << " ";
103         printRec(node->right);
104     }
105 };
106
107 int main() {
108     BinarySearchTree tree;
109     int Q;
110     cin >> Q;
111
112     while (Q--)
113     {
114         string command;
115         cin >> command;
116
117         if (command == "insert")
118         {
119             int value;
120             cin >> value;
121             tree.insert(value);
122         }
123         else if (command == "contains")
124         {
125             int value;
126             cin >> value;
127             if (tree.contains(value))
128             {
129                 cout << "Yes" << endl;
130             }
131             else
132             {
133                 cout << "No" << endl;
134             }
135         }
136         else if (command == "size")
137         {
138             cout << tree.getSize() << endl;
139         }
140         else if (command == "print")
141         {
142             tree.print();
143         }
144     }
145
146     return 0;
147 }

```

Результат:

```

11
size
0
insert 5
insert 4
print
4 5
insert 5
print
4 5
insert 1
print
1 4 5
contains 5
Yes
contains 0
No
size
3
PS D:\VS Code\Projects>

```

Очікувано часу: 1 година.

Витрачено часу: 2 години.

- Завдання №6: Practice# programming: Class Practice Task

Задача №1:

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;

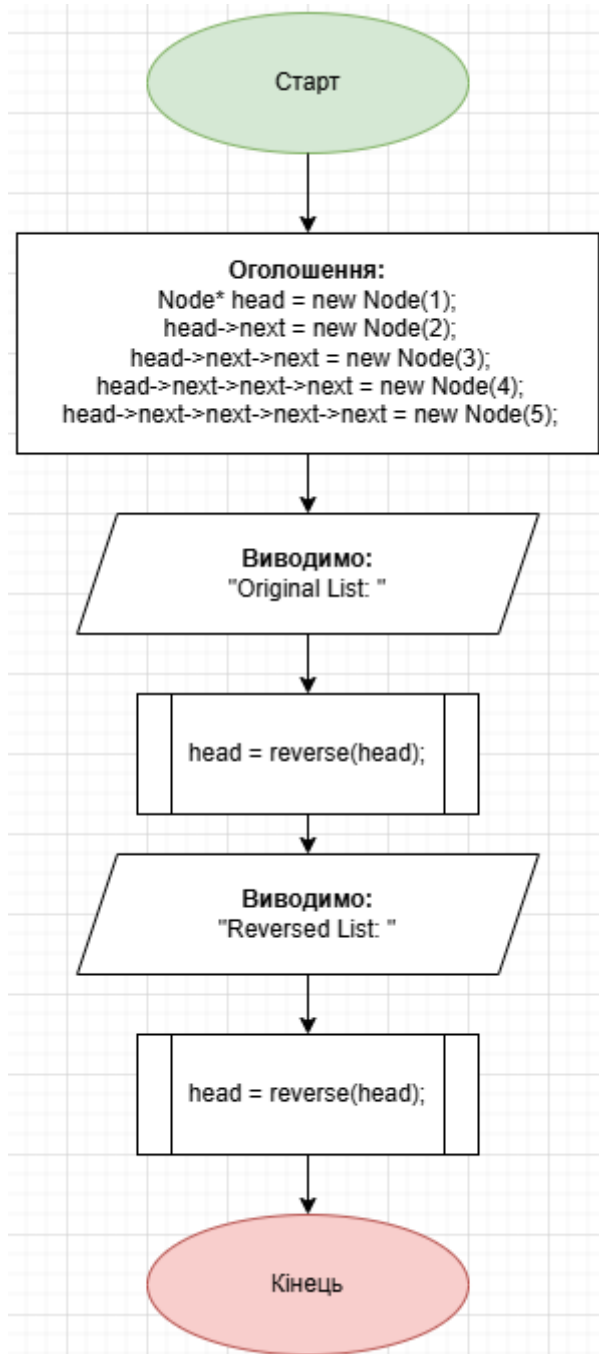
Код:

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node
5 {
6     int data;
7     Node* next;
8
9     Node(int val)
10    {
11        data = val;
12        next = nullptr;
13    }
14 };
15
16 Node* reverse(Node* head)
17 {
18     Node* prev = nullptr;
19     Node* current = head;
20     Node* next = nullptr;
21
22     while (current != nullptr)
23     {
24         next = current->next;
25         current->next = prev;
26         prev = current;
27         current = next;
28     }
29
30     return prev;
31 }
32
33 void printList(Node* head)
34 {
35     Node* temp = head;
36     while (temp != nullptr)
37     {
38         cout << temp->data << " ";
39         temp = temp->next;
40     }
41     cout << endl;
42 }
43
44 int main()
45 {
46     Node* head = new Node(1);
47     head->next = new Node(2);
48     head->next->next = new Node(3);
49     head->next->next->next = new Node(4);
50     head->next->next->next->next = new Node(5);
51
52     cout << "Original List: ";
53     printList(head);
54
55     head = reverse(head);
56     cout << "Reversed List: ";
57     printList(head);
58
59     return 0;
60 }
61
```

Результат:

```
Original List: 1 2 3 4 5  
Reversed List: 5 4 3 2 1  
PS D:\VS Code\Projects>
```

Блок-схема:



Задача №2:

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**.

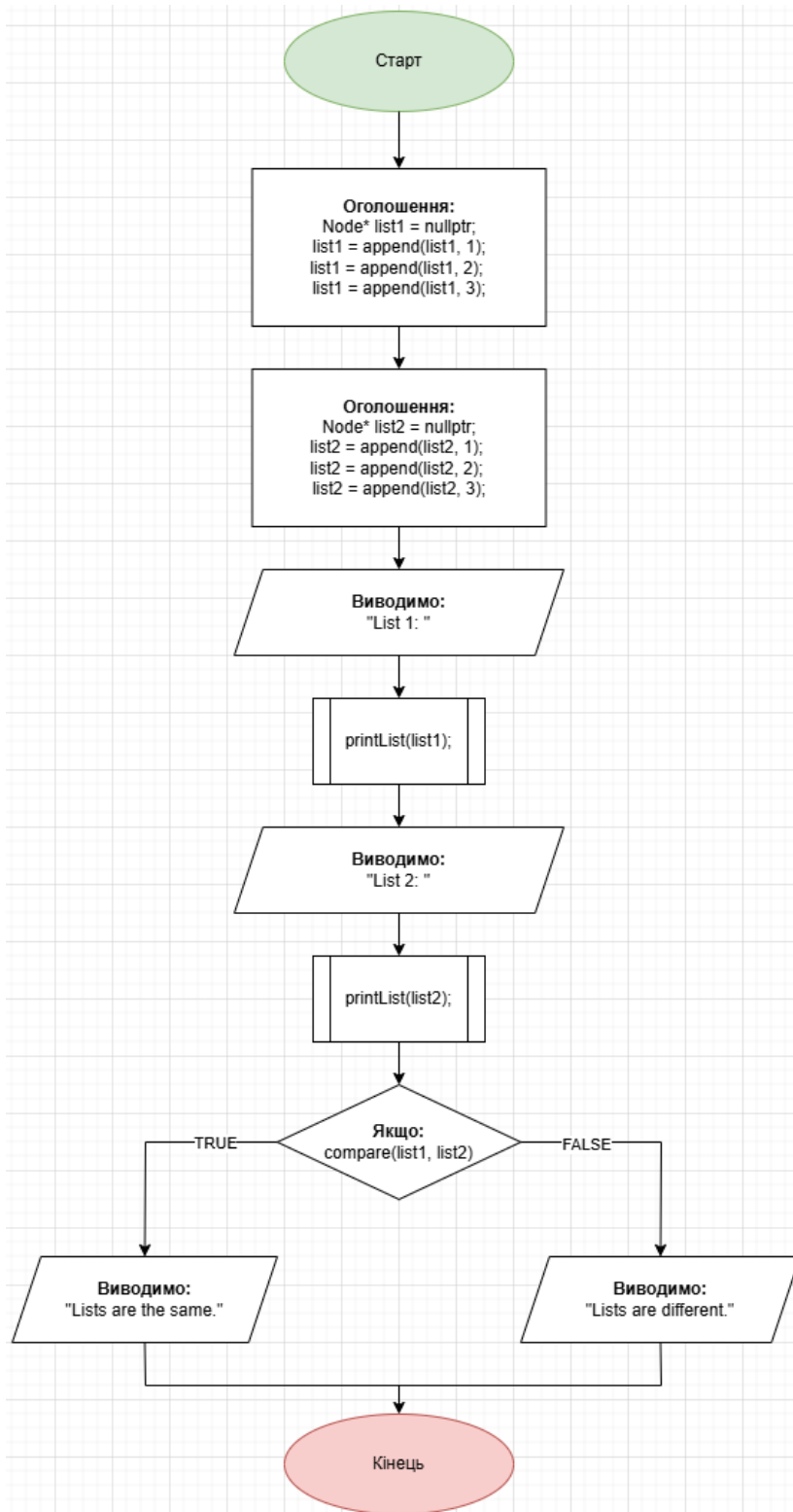
Код:

```
1  #include <iostream>
2  using namespace std;
3
4  struct Node
5  {
6      int data;
7      Node* next;
8
9      Node(int val)
10     {
11         data = val;
12         next = nullptr;
13     }
14 };
15
16 bool compare(Node* h1, Node* h2)
17 {
18     while (h1 != nullptr && h2 != nullptr)
19     {
20         if (h1->data != h2->data)
21         {
22             return false;
23         }
24         h1 = h1->next;
25         h2 = h2->next;
26     }
27
28     return h1 == nullptr && h2 == nullptr;
29 }
30
31 Node* append(Node* head, int value)
32 {
33     if (head == nullptr)
34     {
35         return new Node(value);
36     }
37     Node* temp = head;
38     while (temp->next != nullptr)
39     {
40         temp = temp->next;
41     }
42     temp->next = new Node(value);
43     return head;
44 }
45
46 void printList(Node* head)
47 {
48     Node* temp = head;
49     while (temp != nullptr)
50     {
51         cout << temp->data << " ";
52         temp = temp->next;
53     }
54     cout << endl;
55 }
56
57 int main()
58 {
59     Node* list1 = nullptr;
60     list1 = append(list1, 1);
61     list1 = append(list1, 2);
62     list1 = append(list1, 3);
63
64     Node* list2 = nullptr;
65     list2 = append(list2, 1);
66     list2 = append(list2, 2);
67     list2 = append(list2, 3);
68
69     cout << "List 1: ";
70     printList(list1);
71     cout << "List 2: ";
72     printList(list2);
73
74     if (compare(list1, list2))
75     {
76         cout << "Lists are the same." << endl;
77     }
78     else
79     {
80         cout << "Lists are different." << endl;
81     }
82
83     return 0;
84 }
```

Результат:

```
List 1: 1 2 3
List 2: 1 2 3
Lists are the same.
PS D:\VS Code\Projects>
```

Блок-схема:



Задача №3:

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 списку (напр. $379 \Rightarrow 9 \rightarrow 7 \rightarrow 3$);
- функція повертає новий список, передані в функцію списки не модифікуються.

Код:

```
1  #include <iostream>
2  using namespace std;
3
4  struct Node
5  {
6      int data;
7      Node* next;
8
9      Node(int val)
10     {
11         data = val;
12         next = nullptr;
13     }
14 };
15
16 Node* add(Node* n1, Node* n2)
17 {
18     Node* result = nullptr;
19     Node* tail = nullptr;
20     int carry = 0;
21
22     while (n1 != nullptr || n2 != nullptr || carry > 0)
23     {
24         int sum = carry;
25
26         if (n1 != nullptr)
27         {
28             sum += n1->data;
29             n1 = n1->next;
30         }
31
32         if (n2 != nullptr)
33         {
34             sum += n2->data;
35             n2 = n2->next;
36         }
37
38         carry = sum / 10;
39         int digit = sum % 10;
40
41         Node* newNode = new Node(digit);
42
43         if (result == nullptr)
44         {
45             result = newNode;
46             tail = newNode;
47         }
48         else
49         {
50             tail->next = newNode;
51             tail = newNode;
52         }
53     }
54     return result;
55 }
56
57 Node* reverse(Node* head)
58 {
59     Node* prev = nullptr;
60     Node* current = head;
61     while (current != nullptr)
62     {
63         Node* next = current->next;
64         current->next = prev;
65         prev = current;
66         current = next;
67     }
68     return prev;
69 }
70
71
72 Node* append(Node* head, int value)
73 {
74     if (head == nullptr)
75     {
76         return new Node(value);
77     }
78     Node* temp = head;
79     while (temp->next != nullptr)
80     {
81         temp = temp->next;
82     }
83     temp->next = new Node(value);
84     return head;
85 }
86
```

```

86
87 void printList(Node* head)
88 {
89     Node* temp = head;
90     while (temp != nullptr)
91     {
92         cout << temp->data;
93         temp = temp->next;
94     }
95     cout << endl;
96 }
97
98 int main()
99 {
100     Node* num1 = nullptr;
101     num1 = append(num1, 9);
102     num1 = append(num1, 7);
103     num1 = append(num1, 3);
104     num1 = append(num1, 1);
105
106     Node* num2 = nullptr;
107     num2 = append(num2, 9);
108     num2 = append(num2, 4);
109     num2 = append(num2, 8);
110     num2 = append(num2, 5);
111
112     cout << "Number 1: ";
113     printList(num1);
114     cout << "Number 2: ";
115     printList(num2);
116
117     num1 = reverse(num1);
118     num2 = reverse(num2);
119
120     Node* sum = add(num1, num2);
121
122     sum = reverse(sum);
123
124     cout << "Sum: ";
125     printList(sum);
126
127     return 0;
128 }
129

```

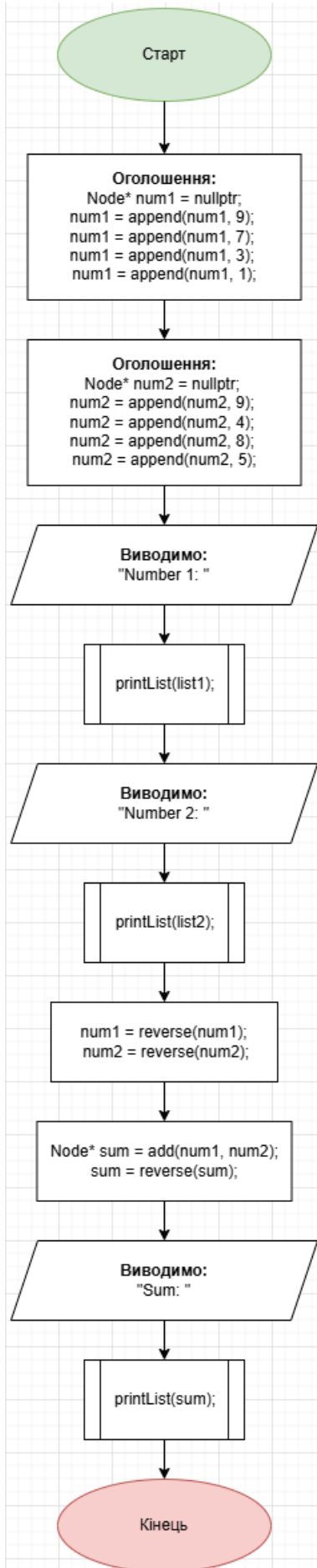
Результат:

```

Number 1: 9731
Number 2: 9485
Sum: 19216
PS D:\VS Code\Projects>

```

Блок-схема:



Задача №4:

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

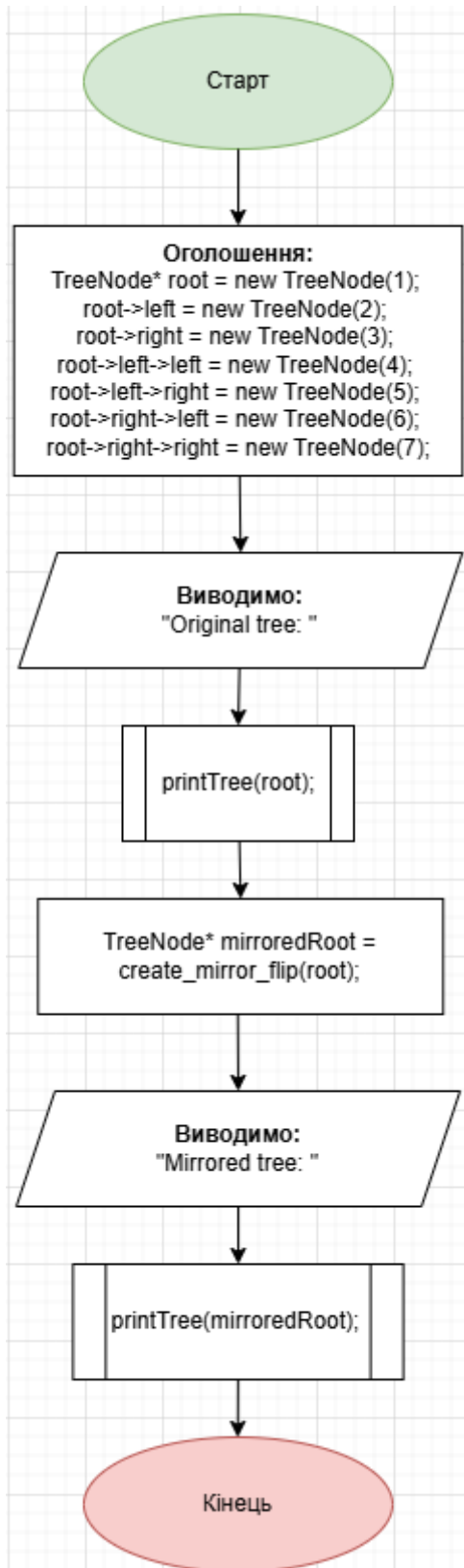
Код:

```
1  #include <iostream>
2  using namespace std;
3
4  struct TreeNode
5  {
6      int data;
7      TreeNode* left;
8      TreeNode* right;
9
10     TreeNode(int val)
11     {
12         data = val;
13         left = nullptr;
14         right = nullptr;
15     }
16 };
17
18 TreeNode* create_mirror_flip(TreeNode* root)
19 {
20     if (root == nullptr)
21     {
22         return nullptr;
23     }
24
25     TreeNode* mirroredNode = new TreeNode(root->data);
26
27     mirroredNode->left = create_mirror_flip(root->right);
28     mirroredNode->right = create_mirror_flip(root->left);
29
30     return mirroredNode;
31 }
32
33 void printTree(TreeNode* root)
34 {
35     if (root == nullptr)
36     {
37         return;
38     }
39     cout << root->data << " ";
40     printTree(root->left);
41     printTree(root->right);
42 }
43
44 int main()
45 {
46     TreeNode* root = new TreeNode(1);
47     root->left = new TreeNode(2);
48     root->right = new TreeNode(3);
49     root->left->left = new TreeNode(4);
50     root->left->right = new TreeNode(5);
51     root->right->left = new TreeNode(6);
52     root->right->right = new TreeNode(7);
53
54     cout << "Original tree: ";
55     printTree(root);
56     cout << endl;
57
58     TreeNode* mirroredRoot = create_mirror_flip(root);
59
60     cout << "Mirrored tree: ";
61     printTree(mirroredRoot);
62     cout << endl;
63
64     return 0;
65 }
```

Результат:

```
Original tree: 1 2 4 5 3 6 7  
Mirrored tree: 1 3 7 6 2 5 4  
PS D:\VS Code\Projects> █
```

Блок-схема:



Задача №5:

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

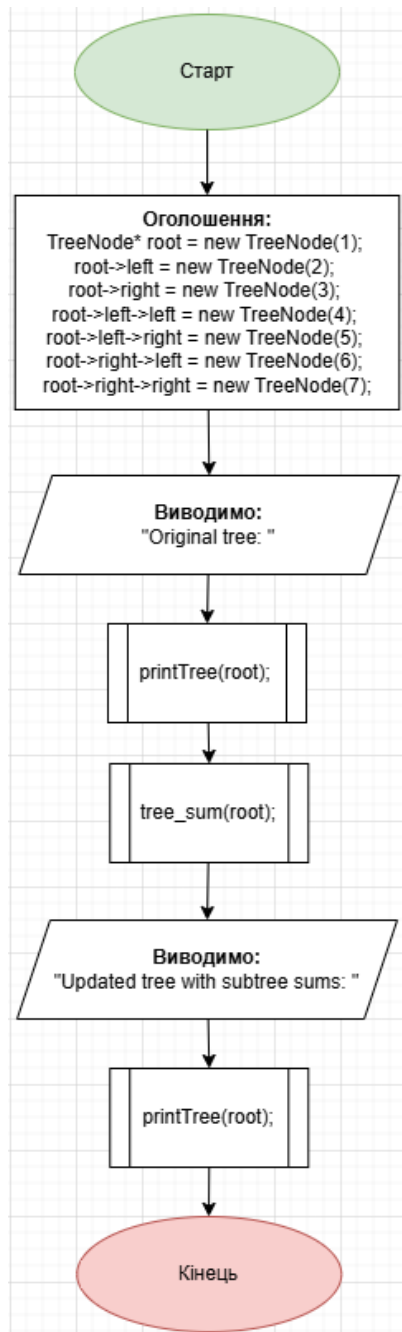
Код:

```
1  #include <iostream>
2  using namespace std;
3
4  struct TreeNode
5  {
6      int data;
7      TreeNode* left;
8      TreeNode* right;
9
10     TreeNode(int val)
11     {
12         data = val;
13         left = nullptr;
14         right = nullptr;
15     }
16 };
17
18 int calculateSubtreeSum(TreeNode* root)
19 {
20     if (root == nullptr)
21     {
22         return 0;
23     }
24
25     if (root->left == nullptr && root->right == nullptr)
26     {
27         return root->data;
28     }
29
30     int leftSum = calculateSubtreeSum(root->left);
31     int rightSum = calculateSubtreeSum(root->right);
32
33     root->data = leftSum + rightSum;
34
35     return root->data;
36 }
37
38 void tree_sum(TreeNode* root) {
39     calculateSubtreeSum(root);
40 }
41
42 void printTree(TreeNode* root)
43 {
44     if (root == nullptr)
45     {
46         return;
47     }
48     cout << root->data << " ";
49     printTree(root->left);
50     printTree(root->right);
51 }
52
53 int main()
54 {
55     TreeNode* root = new TreeNode(1);
56     root->left = new TreeNode(2);
57     root->right = new TreeNode(3);
58     root->left->left = new TreeNode(4);
59     root->left->right = new TreeNode(5);
60     root->right->left = new TreeNode(6);
61     root->right->right = new TreeNode(7);
62
63     cout << "Original tree: ";
64     printTree(root);
65     cout << endl;
66
67     tree_sum(root);
68
69     cout << "Updated tree with subtree sums: ";
70     printTree(root);
71     cout << endl;
72
73     return 0;
74 }
```


Результат:

```
Original tree: 1 2 4 5 3 6 7
Updated tree with subtree sums: 22 9 4 5 13 6 7
PS D:\VS Code\Projects>
```

Блок-схема:



Очікувано часу: **2 години.**

Витрачено часу: **4 години.**

- Завдання №7: Practice# programming: Self Practice Task

Цікава гра

Обмеження: 2 сек., 256 MiB

Мале Бісеня та Дракон полюбляють проводити дозвілля разом. Сьогодні вони грають в одну дуже цікаву гру.

У них є дошка, що складається з n рядків та m стовпців, всі клітинки якої білі.

Гравці по черзі вибирають одну білу клітинку та зафарбовують її в чорний колір. Бісеня ходить першим. Гравець, який не може зробити хід, тобто на початку ходу якого вся дошка чорна, програє.

Погостривши зубки, Бісеня зрозуміло, що у Дракона велика перевага, адже він двоголовий, а, як то кажуть, «одна голова добре, а дві — краще». Тому воно просить вас допомогти. Вам потрібно сказати за заданими n та m , хто виграє у цій напруженій грі.

Вхідні дані

У єдиному рядку задані два цілих числа n та m — розміри дошки.

Вихідні дані

Єдине слово — `Imp`, якщо переможе Бісеня, та `Dragon`, якщо переможе Дракон.

Обмеження

$1 \leq n, m \leq 100$.

Приклади

Вхідні дані (<i>stdin</i>)	Вихідні дані (<i>stdout</i>)
7 4	Dragon

Посилання: <https://algotester.com/uk/ArchiveProblem/DisplayWithEditor/20074>

Код:

```
1  #include <iostream>
2
3  int main()
4  {
5      int m, n;
6      std::cin >> m >> n;
7
8      if((m * n) % 2 == 0)
9      {
10         std::cout << "Dragon" << std::endl;
11     }
12     else
13     {
14         std::cout << "Imp" << std::endl;
15     }
16
17     return 0;
18 }
```

Результат:

```
7 4
Dragon
PS D:\VS Code\Projects>
```

Очікувано часу: **30 хвилин.**

Витрачено часу: **15 хвилин.**

Pull-Request: https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/508

Висновок: У результаті виконання цієї роботи я ознайомився з основами роботи з динамічними структурами даних, такими як черга, стек, списки та дерево. Я вивчив їхню внутрішню організацію, ключові властивості і методи доступу, а також алгоритми для додавання, видалення, пошуку та сортування елементів. Особливу увагу я приділив практичному використанню стандартних бібліотек і створенню власних оптимізованих реалізацій. Отримані знання допоможуть мені ефективно використовувати динамічні структури даних у програмуванні.