

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 ШІ-11

Станько Олег Ігорович

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

Мета: зрозуміти основи побудови та використання таких структур даних, як черга, стек, списки та дерева, освоїти алгоритми, які дозволяють ефективно маніпулювати та обробляти ці структури.

### Теоретичні відомості

1. Основи Динамічних Структур Даних
2. Стек
3. Черга
4. Зв'язні Списки
5. Дерева
6. Алгоритми Обробки Динамічних Структур

### індивідуальний план опрацювання теорії

1. Основи Динамічних Структур Даних  
[Data Structures Tutorial - GeeksforGeeks](#)  
Ознайомився з виділенням пам'яті для структур даних (stack і heap)  
Витрачено 1 годину
2. **Стек**  
[Stack Data Structure - GeeksforGeeks](#)  
Вивчив принцип роботи стеку, його застосування та основні операції (push, pop, top).  
Витрачено 1 годину.
3. **Черга**  
[Queue Data Structure - GeeksforGeeks](#)  
Ознайомився з принципами роботи черги, її застосуванням та основними операціями (enqueue, dequeue, front).  
Витрачено 1 годину.
4. **Зв'язні Списки**  
[Linked List Data Structure - GeeksforGeeks](#)  
Вивчив основи зв'язних списків, різновиди (однозв'язний, двозв'язний) та основні операції (вставка, видалення, пошук).  
Витрачено 2 години.
5. **Дерева**  
[Linked List Data Structure - GeeksforGeeks](#)  
Ознайомився з деревами, їх видами (бінарні дерева, дерева пошуку) та основними операціями (вставка, видалення, пошук).  
Витрачено 2 години.
6. **Алгоритми Обробки Динамічних Структур**  
[Algorithms Tutorial - GeeksforGeeks](#)  
Вивчив алгоритми обробки динамічних структур даних, зокрема сортування та пошук.  
2 години.

### Виконання роботи:

## 1. Опрацювання завдання та вимог до програм.

### VNS LAB 10

Записи в лінійному списку містять ключове поле типу `*char` (рядок символів).

Сформуванати двонаправлений список. Знищити елемент із заданим ключем. Додати К елементів у початок списку.

### Algotester Lab 5

*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  $O$ , пісок  $S$  і каміння  $X$ ;

Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.

Ваше завдання сказати як буде виглядати печера після землетрусу.

## Вхідні дані

У першому рядку 2 цілих числа  $N$  та  $M$  - висота та ширина печери

У  $N$  наступних рядках стрічка `rowi` 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.

### Algotester Lab 7-8

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.

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

**Для отримання 100% балів ця структура має бути написана як шаблон класу, у якості параметру використати `int`. Використовувати STL заборонено.**

## Class Practice Work

*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;

`bool compare(Node *h1, Node *h2);`

*Умови задачі:*

- використовувати цілочисельні значення в списку;
- реалізувати функцію, яка ітеративно проходить 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*.

`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.

`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

## Self-practice task

**Одного осіннього дня ти, студенте, сидиш на лекції, яку викладач одностонно бубнить собі під ніс. Вигляд з вікна приковує до себе твій погляд — надворі з дерев опадає листя.**

Вигляд з вікна описується прямокутником, що складається з  $n \times m$  клітинок. Кожна клітинка прямокутника порожня (позначається крапкоюю) або містить листок з дерева (позначається зірочкоюю). Листя падає вертикально вниз.

Знайди вигляд з вікна після того, як усе листя опаде. Уважаємо, що нове листя у вікні не з'явиться.

Вхідні дані

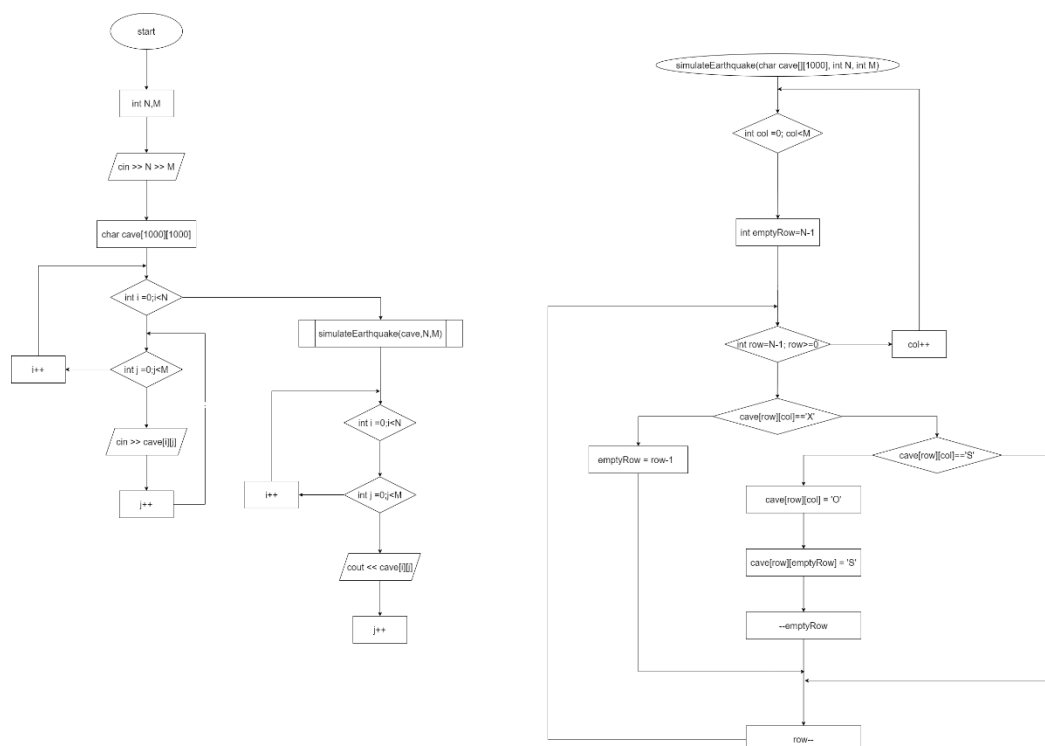
У першому рядку задано два цілі числа  $n$  і  $m$  — розміри прямокутника.

У наступних  $n$  рядках задано прямокутник розміру  $n \times m$  — опис вигляду з вікна.

Вихідні дані

В  $n$  рядках виведи вигляд з вікна після опадання листя.

Блок-схема



Коди:

VNS LAB 10

```

1  #include <iostream>
2  #include <cstring>
3  #include <cstdarg>
4  #include <fstream>
5
6  using namespace std;
7
8  struct Node {
9      char* data;
10     Node* next;
11     Node* prev;
12
13     Node(const char* value) : data(strdup(value)), next(nullptr), prev(nullptr) {}
14 };
15
16 void createll(Node*& head) {
17     cout << "Creating the list..." << endl;
18     const char* values[] = {"Node 1", "Node 2", "Node 3", "Node 4", "Node 5", "Node 6"};
19
20     for (int i = 0; i < 6; i++) {
21         Node* newNode = new Node(values[i]);
22
23         if (head == nullptr) {
24             newNode->prev = nullptr;
25             head = newNode;
26         } else {
27             Node* tmp = head;
28             while (tmp->next != nullptr) {
29                 tmp = tmp->next;
30             }
31             tmp->next = newNode;
32             newNode->prev = tmp;
33         }
34     }
35 }
36
37 void deleteByData(Node*& head, const char* data) {
38     cout << "Deleting element: " << data << " from the list..." << endl;
39     if (head == nullptr) return;
40
41     Node* tmp = head;
42     while (tmp != nullptr) {
43         if (strcmp(tmp->data, data) == 0) {
44             Node* toDelete = tmp;
45             if (toDelete == head) {
46                 head = toDelete->next;
47                 if (head != nullptr) head->prev = nullptr;
48             } else {
49                 if (toDelete->prev != nullptr) {
50                     toDelete->prev->next = toDelete->next;
51                 }
52                 if (toDelete->next != nullptr) {
53                     toDelete->next->prev = toDelete->prev;
54                 }
55             }
56             free(toDelete->data);
57             delete toDelete;
58             return;
59         }
60         tmp = tmp->next;
61     }
62 }
63
64 void printll(Node* head) {
65     cout << "Printing the list..." << endl;
66     if (head == nullptr) {
67         cout << "The list is empty." << endl;
68         return;
69     }
70
71     Node* tmp = head;
72     while (tmp != nullptr) {
73         cout << tmp->data << " ";
74         tmp = tmp->next;
75     }
76     cout << endl;
77 }
78
79 void addKElementsToBeginning(Node*& head, int K, ...) {
80     cout << "Adding " << K << " elements to the beginning of the list..." << endl;
81
82     va_list args;
83     va_start(args, K);
84
85     for (int i = 0; i < K; i++) {
86         const char* data = va_arg(args, const char*);
87         Node* newNode = new Node(data);
88         newNode->next = head;
89         newNode->prev = nullptr;
90
91         if (head != nullptr) {
92             head->prev = newNode;
93         }
94         head = newNode;
95     }
96     va_end(args);
97 }

```

```

98
99 void writeLLToFile(Node* head, const char* filename) {
100     cout << "Writing the list to file: " << filename << "..." << endl;
101     ofstream outFile(filename);
102     if (!outFile) {
103         cerr << "Error: Cannot open the file." << endl;
104         exit(1);
105     }
106
107     Node* tmp = head;
108     while (tmp != nullptr) {
109         outFile << tmp->data << endl;
110         tmp = tmp->next;
111     }
112 }
113
114 void deleteLL(Node*& head) {
115     cout << "Deleting the list..." << endl;
116     while (head != nullptr) {
117         Node* nextNode = head->next;
118         free(head->data);
119         delete head;
120         head = nextNode;
121     }
122 }
123
124 void fromFileToLL(Node*& head, const char* filename) {
125     cout << "Loading the list from file: " << filename << "..." << endl;
126     ifstream inFile(filename);
127     if (!inFile) {
128         cerr << "Error: Cannot open the file." << endl;
129         exit(2);
130     }
131
132     string line;
133     Node* tmp = nullptr;
134     while (getline(inFile, line)) {
135         Node* newNode = new Node(line.c_str());
136         newNode->next = nullptr;
137         newNode->prev = tmp;
138
139         if (head == nullptr) {
140             head = newNode;
141         } else {
142             tmp->next = newNode;
143         }
144         tmp = newNode;
145     }
146 }
147
148 int main() {
149     Node* head = nullptr;
150     const char* filename = "file.txt";
151
152     createLL(head);
153     printLL(head);
154
155     deleteByData(head, "Node 3");
156     printLL(head);
157
158     addKElementsToBegining(head, 3, "Node -1", "Node -2", "Node -3");
159     printLL(head);
160
161     writeLLToFile(head, filename);
162     deleteLL(head);
163     printLL(head);
164
165     fromFileToLL(head, filename);
166     printLL(head);
167
168     deleteLL(head);
169     printLL(head);
170 }
171

```

## Algo lab 5

al\_programming\_playground\_2024 > al\_11 > oleh\_stanko > epic\_6 > G+ algoalabs.cpp > ...

```
1  #include <iostream>
2
3  using namespace std;
4
5  void simulateEarthquake(char cave[][1000], int N, int M) {
6      for (int col = 0; col < M; ++col) {
7          int emptyRow = N - 1;
8
9          for (int row = N - 1; row >= 0; --row) {
10             if (cave[row][col] == 'X') {
11                 emptyRow = row - 1;
12             } else if (cave[row][col] == 'S') {
13                 cave[row][col] = 'O';
14                 cave[emptyRow][col] = 'S';
15                 --emptyRow;
16             }
17         }
18     }
19 }
20
21 int main() {
22     int N, M;
23     cin >> N >> M;
24
25     char cave[1000][1000];
26
27     for (int i = 0; i < N; ++i) {
28         for (int j = 0; j < M; ++j) {
29             cin >> cave[i][j];
30         }
31     }
32
33     simulateEarthquake(cave, N, M);
34
35     for (int i = 0; i < N; ++i) {
36         for (int j = 0; j < M; ++j) {
37             cout << cave[i][j];
38         }
39         cout << endl;
40     }
41
42     return 0;
43 }
44
```

## Algo lab 7\_8



```

1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 struct TreeNode {
7     int data;
8     TreeNode* left;
9     TreeNode* right;
10
11     TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}
12 };
13
14 template <class T>
15 struct binary_search_tree {
16 public:
17     binary_search_tree() : root(nullptr) {};
18
19     void insert(T value) {
20         root = insert(root, value);
21     }
22
23     bool contains(T value) {
24         return contains(root, value);
25     }
26
27     int size() {
28         return size(root);
29     }
30
31     void print() {
32         cout << *this << endl;
33     }
34
35     friend ostream& operator<<(ostream& os, const binary_search_tree<T>& tree) {
36         tree.inorderTraverse(tree.root, os);
37         return os;
38     }
39
40 private:
41     TreeNode* root;
42
43     TreeNode* insert(TreeNode* node, T value) {
44         if (node == nullptr) {
45             return new TreeNode(value);
46         }
47         if (value < node->data) {
48             node->left = insert(node->left, value);
49         } else if (value > node->data) {
50             node->right = insert(node->right, value);
51         }
52         return node;
53     }
54
55     bool contains(TreeNode* node, T value) {
56         if (node == nullptr) {
57             return false;
58         }
59         if (value == node->data) {
60             return true;
61         } else if (value < node->data) {
62             return contains(node->left, value);
63         } else {
64             return contains(node->right, value);
65         }
66     }
67
68     int size(TreeNode* node) {
69         if (node == nullptr) {
70             return 0;
71         }
72         return 1 + size(node->left) + size(node->right);
73     }
74
75     void inorderTraverse(TreeNode* node, ostream& os) const {
76         if (node != nullptr) {
77             inorderTraverse(node->left, os);
78             os << node->data << " ";
79             inorderTraverse(node->right, os);
80         }
81     }
82 };
83
84 int main() {
85     binary_search_tree<int> binary_tree;
86     int q;
87     cin >> q; // Количество команд
88     for (int i = 0; i < q; i++) {
89         string command;
90         cin >> command;
91
92         if (command == "insert") {
93             int value;
94             cin >> value;
95             binary_tree.insert(value);
96
97         } else if (command == "size") {
98             cout << binary_tree.size() << endl;
99
100         } else if (command == "contains") {
101             int value;
102             cin >> value;
103             cout << (binary_tree.contains(value) ? "Yes" : "No") << endl;
104
105         } else if (command == "print") {
106             {
107                 binary_tree.print();
108             }
109         }
110     }
111     return 0;
112 }
113

```

## Practice

```
al_programming_playground_2024 / al_1 / > dsh_danks > epic_8 > 6 practice.cpp > @ main()
1 #include <iostream>
2
3 using namespace std;
4
5 struct Node
6 {
7     int data;
8     Node* next;
9
10     Node(int value) : data(value), next(nullptr) {}
11 };
12
13 Node* reverse(Node* head)
14 {
15     Node* prev = nullptr;
16     Node* current = head;
17
18     while (current)
19     {
20         Node* nextNode = current->next;
21         current->next = prev;
22         prev = current;
23         current = nextNode;
24     }
25
26     return prev;
27 }
28
29 bool compare(Node* h1, Node* h2) {
30     while (h1 != nullptr && h2 != nullptr) {
31         if (h1->data != h2->data) {
32             return false;
33         }
34         h1 = h1->next;
35         h2 = h2->next;
36     }
37     return h1 == nullptr && h2 == nullptr;
38 }
39
40 Node* add(Node* n1, Node* n2) {
41     Node* result = nullptr;
42     Node* tail = nullptr;
43     int carry = 0;
44
45     while (n1 != nullptr || n2 != nullptr || carry != 0) {
46         int val1 = (n1 != nullptr) ? n1->data : 0;
47         int val2 = (n2 != nullptr) ? n2->data : 0;
48
49         int sum = val1 + val2 + carry;
50         carry = sum / 10;
51         int digit = sum % 10;
52
53         Node* newNode = new Node(digit);
54         if (result == nullptr) {
55             result = newNode;
56             tail = newNode;
57         } else {
58             tail->next = newNode;
59             tail = newNode;
60         }
61
62         if (n1 != nullptr) {
63             n1 = n1->next;
64         }
65         if (n2 != nullptr) {
66             n2 = n2->next;
67         }
68     }
69
70     return result;
71 }
72
73 void printlist(Node* head) {
74     while (head != nullptr) {
75         cout << head->data << " ";
76         head = head->next;
77     }
78     cout << endl;
79 }
80
81 struct TreeNode {
82     int data;
83     TreeNode* left;
84     TreeNode* right;
85
86     TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}
87 };
88
89 TreeNode* create_mirror_flip(TreeNode* root) {
90     if (!root) {
91         return nullptr;
92     }
93
94     TreeNode* newRoot = new TreeNode(root->data);
95     newRoot->left = create_mirror_flip(root->right);
96     newRoot->right = create_mirror_flip(root->left);
97
98     return newRoot;
99 }
100
101 int tree_sum(TreeNode* root)
102 {
103     if (!root)
104     {
105         return 0;
106     }
107
108     int leftSum = tree_sum(root->left);
109     int rightSum = tree_sum(root->right);
110
111     root->data += leftSum + rightSum;
112     return root->data;
113 }
```

```

114
115 void printTree(TreeNode* root) {
116     if (!root) return;
117     printTree(root->left);
118     cout << root->data << " ";
119     printTree(root->right);
120 }
121
122 int main() {
123     // Работа с одноязычными списками
124     Node* list1 = new Node(2);
125     list1->next = new Node(4);
126     list1->next->next = new Node(3);
127
128     Node* list2 = new Node(5);
129     list2->next = new Node(6);
130     list2->next->next = new Node(4);
131
132     cout << "List 1: ";
133     printList(list1);
134
135     cout << "List 2: ";
136     printList(list2);
137
138     Node* sumList = add(list1, list2);
139     cout << "Sum of List 1 and List 2: ";
140     printList(sumList);
141
142     cout << "Are List 1 and List 2 equal? " << (compare(list1, list2) ? "Yes" : "No") << endl;
143
144     Node* reversedList1 = reverse(list1);
145     cout << "Reversed List 1: ";
146     printList(reversedList1);
147
148     TreeNode* root = new TreeNode(4);
149     root->left = new TreeNode(2);
150     root->left->right = new TreeNode(3);
151     root->left->left = new TreeNode(1);
152
153     root->right = new TreeNode(6);
154     root->right->left = new TreeNode(5);
155     root->right->right = new TreeNode(7);
156
157     cout << "\nOriginal Tree: ";
158     printTree(root);
159
160     TreeNode* mirroredRoot = create_mirror_flip(root);
161     cout << "\nMirrored Tree: ";
162     printTree(mirroredRoot);
163
164     tree_sum(root);
165     cout << "\nTree after subtree sums (Preorder): ";
166     printTree(root);
167
168     return 0;
169 }
170
171

```

Self practice work

```

1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main() {
7      int n, m;
8      cin >> n >> m;
9
10     string st = "";
11     int s[100];
12
13     for (int i = 0; i < m; i++) {
14         s[i] = 0;
15     }
16
17     for (int i = 0; i < n; i++) {
18         string a;
19         cin >> a;
20         for (int j = 0; j < m; j++) {
21             if (a[j] == '*') {
22                 s[j]++;
23             }
24         }
25     }
26
27     for (int i = n; i > 0; i--) {
28         for (int j = 0; j < m; j++) {
29             if (s[j] == i) {
30                 s[j]--;
31                 st += '*';
32             } else {
33                 st += '.';
34             }
35         }
36         cout << st << endl;
37         st = "";
38     }
39
40     return 0;
41 }

```

Робота з командою



Зустрічалися в зумі 19 листопада

## Висновок:

Протягом виконання цього епіку я ознайомився з основами динамічних структур даних та їх основними операціями, такими як вставка, видалення та пошук. Перш за все, я вивчив принципи виділення пам'яті для різних структур даних, зокрема для стеку та черги. Оволодів технікою роботи з цими структурами, зокрема операціями push, pop, top для стека і enqueue, dequeue, front для черги.

Далі я вивчив зв'язні списки, зокрема однозв'язні та двозв'язні, а також їх основні операції. Це дозволило мені краще зрозуміти їх структуру і застосування. Окремо я ознайомився з деревами, зокрема з бінарними деревами і деревами пошуку, що є важливою частиною роботи з динамічними структурами.

Завершенням навчання стало вивчення алгоритмів обробки динамічних структур, таких як сортування та пошук, що допомогло мені краще зрозуміти, як ефективно працювати з цими структурами в реальних задачах.

У результаті я значно поглибив свої знання в області динамічних структур даних і алгоритмів їх обробки, що є важливим кроком для подальшого вивчення програмування.