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
Ореньчук Юрій Миколайович

Львів 2024

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

Мета: Вивчити динамічні структури та попрактикуватися з алгоритмами обробки динамічних структур.

Теоретичні відомості:

Тема №1: Зв’язні списки та дерева

Тема №2: Основи динамічних структур даних

Тема №3: Приклади використання черги

Індивідуальний план опрацювання теорії:

Тема №1: https://www.youtube.com/watch?v=HKfj0l7ndbc

Тема №2: https://www.youtube.com/watch?v=426CYzQC86M

Тема №3: https://www.youtube.com/watch?v=juqhvOyMoeI

Виконання роботи:

Завдання №1: VNS Lab 10 Variant 19

2. Постановка 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.

Завдання №2: Algotester Lab 5 Variant 2

Lab 5v2

Limits: 1 sec., 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 0 , 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.

Input

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.

Output

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

Завдання №3: Algotester Lab 7-8 Variant 2

- **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 <<

Input

Ціле число Q - кількість запитів.
У наступних рядках Q запитів у зазначеному в умові форматі.

Output

Відповіді на запити у зазначеному в умові форматі.

Завдання №4: Practice Work Task

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

Дизайн та планована оцінка часу виконання завдань:

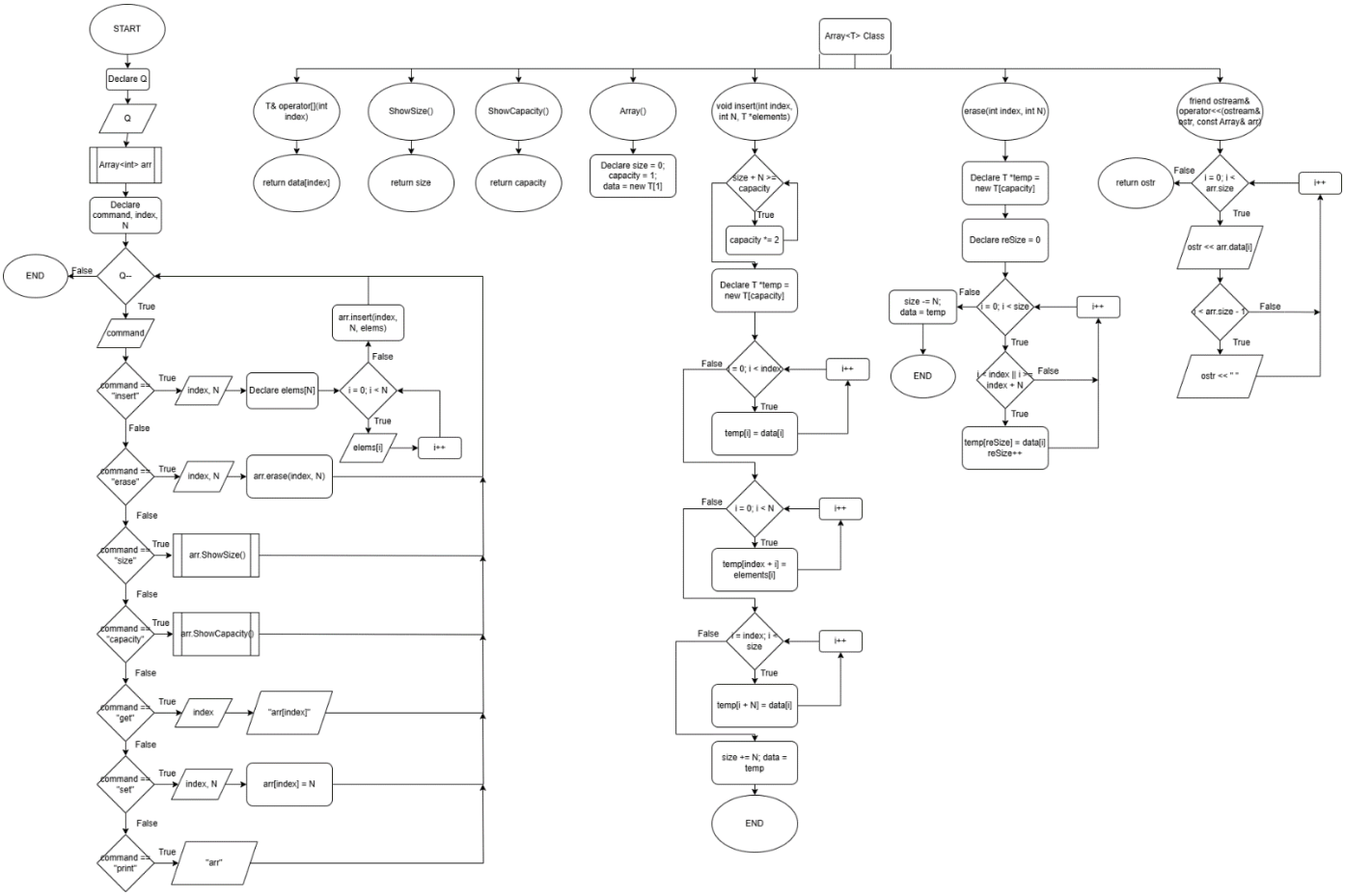
Завдання №1: VNS Lab 10 Variant 19

Планований час: 2 год

Завдання №2: Algotester Lab 5 Variant 2

Планований час: 1 год

Завдання №3: Algotester Lab 7-8 Variant 2
Планований час: 3 год



Завдання №4: Practice Work Task

Планований час: 2 год

Код програм з посиланням на зовнішгі ресурси:

Завдання №1: VNS Lab 10 Variant 19

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

struct Node {
    char* data;
    Node* prev;
    Node* next;
    Node(const char* val) {
        data = new char[strlen(val) + 1];
        strcpy(data, val);
        prev = next = nullptr;
    }
    ~Node() {
        delete[] data;
    }

};

Node* createList() {
    return nullptr;
}

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

void addToBeginning(Node*& head, const char* value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    if (head) head->prev = newNode;
    head = newNode;
}
```

```cpp

void deleteByIndex(Node*& head, const vector<int>& indexes) {
    if (!head) {
        cout << "List is empty\n";
        return;
    }

    vector<int> sortedIndexes = indexes;
    sort(sortedIndexes.rbegin(), sortedIndexes.rend());

    int cnt = 0;
    for (int index : sortedIndexes) {
        Node* curr = head;
        int currIndex = 0;

        while (curr && currIndex < index) {
            curr = curr->next;
            currIndex++;
        }

        if (!curr) {
            cout << "Index " << index << " is out of range.\n";
            continue;
        }

        if (curr->prev) curr->prev->next = curr->next;
        if (curr->next) curr->next->prev = curr->prev;

        if (curr == head) head = curr->next;

        delete curr;
        cnt++;
    }
    cout << "Deleted " << cnt << " elements\n";
}

```

```cpp
void writeListToFile(Node* head, const char* filename) {
    ofstream file(filename);
    if (!file.is_open()) {
        cout << "Failed to open file\n";
        return;
    }
    Node* current = head;
    while (current) {
        file << current->data << endl;
        cout << "Writing node: " << current->data << endl;
        current = current->next;
    }

    file.close();
    cout << "Successfully wrote List into file\n";
}

void destroyList(Node*& head) {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
    cout << "Deleted the List\n";
}
```

```
104  Node* recoverList(const char* filename) {
105      ifstream file(filename);
106      if (!file.is_open()) {
107          cout << "Failed to open file\n";
108          return nullptr;
109      }
110
111      Node* head = nullptr;
112      Node* tail = nullptr;
113      char buffer[256];
114      while (file.getline(buffer, 256)) {
115          Node* newNode = new Node(buffer);
116          if (!head) {
117              head = tail = newNode;
118          } else {
119              tail->next = newNode;
120              newNode->prev = tail;
121              tail = newNode;
122          }
123      }
124
125      file.close();
126      return head;
127  }
128
129  int main(){
130      Node* head = createList();
131      char* filename = "storage.txt";
132
133      addToBeginning(head, "Node1");
134      addToBeginning(head, "Node2");
135      addToBeginning(head, "Node3");
136      addToBeginning(head, "Node4");
137
138      printList(head);
139
140      vector<int> indexes = {1, 0};
141      deleteByIndex(head, indexes);
142
143      writeListToFile(head, filename);
144      destroyList(head);
145      head = recoverList(filename);
146      printList(head);
147
148      destroyList(head);
149
150      return 0;
151  }
```

Завдання №2: Algotester Lab 5 Variant 2

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

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

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

    char type;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cin >> type;
            if (type == 'S' || type == 'O' || type == 'X') {
                cave[i][j] = type;
            }
        }
    }
    for (int j = 0; j < M; j++) {
        for (int i = N - 2; i >= 0; i--) {
            if (cave[i][j] == 'S') {
                queue<int> sandQueue;
                sandQueue.push(i);

                while (!sandQueue.empty()) {
                    int x = sandQueue.front();
                    sandQueue.pop();

                    if (x + 1 < N && cave[x + 1][j] == 'O') {
                        cave[x + 1][j] = 'S';
                        cave[x][j] = 'O';
                        sandQueue.push(x + 1);
                    }
                }
            }
        }
    }

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

    return 0;
}
```

Завдання №3: Algotester Lab 7-8 Variant 2

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

template< typename T >
class Array {
    private:
    T* data;

    public:
    int size, capacity;

    Array() {
        size = 0;
        capacity = 1;
        data = new T[1];
    }

    void insert(int index, int N, T *elements) {
        while (size + N >= capacity) {
            capacity *= 2;
        }

        T *temp = new T[capacity];

        for (int i = 0; i < index; i++) {
            temp[i] = data[i];
        }

        for (int i = 0; i < N; i++) {
            temp[index + i] = elements[i];
        }

        for (int i = index; i < size; i++) {
            temp[i + N] = data[i];
        }

        size += N;
        data = temp;
    }

```

```
41      void erase(int index, int N) {
42          T *temp = new T[capacity];
43          int reSize = 0;
44
45          for (int i = 0; i < size; i++) {
46              if (i < index || i >= index + N) {
47                  temp[reSize] = data[i];
48                  reSize++;
49              }
50          }
51          size -= N;
52          data = temp;
53      }
54
55      T& operator[](int index) {
56          return data[index];
57      }
58
59      int ShowSize() {
60          return size;
61      }
62
63      int ShowCapacity() {
64          return capacity;
65      }
66
67      friend ostream& operator<<(ostream& ostr, const Array& arr) {
68          for (int i = 0; i < arr.size; i++) {
69              ostr << arr.data[i];
70              if (i < arr.size - 1) {
71                  ostr << " ";
72              }
73          }
74          return ostr;
75      }
76
77      ~Array() {
78          delete[] data;
79      }
80  };
```

```cpp
int main(){
    int Q;
    cin >> Q;
    Array<int> arr;

    string command;
    int index, N;
    while (Q--){
        cin >> command;
        if (command == "insert"){
            cin >> index >> N;
            int elems[N];

            for (int i = 0; i < N; i++) {
                cin >> elems[i];
            }

            arr.insert(index, N, elems);

        } else if (command == "erase"){
            cin >> index >> N;
            arr.erase(index, N);

        } else if (command == "size"){
            cout << arr.ShowSize() << endl;

        } else if (command == "capacity"){
            cout << arr.ShowCapacity() << endl;

        } else if (command == "get"){
            cin >> index;
            cout << arr[index] << endl;

        } else if (command == "set"){
            cin >> index >> N;
            arr[index] = N;

        } else if (command == "print"){
            cout << arr << endl;
        }
    }

    return 0;
```

Завдання №4: Practice Work Task

Завдання 1-3:

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

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

// Task 1: Reverse the list
Node* Reversal(Node* main) {
    Node* prev = nullptr;
    Node* curr = main;
    Node* next = nullptr;

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

    return prev;
}

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

void deleteList(Node* &head) {
    Node* current = head;
    Node* nextNode;

    while (current != nullptr) {
        nextNode = current->next;
        delete current;
        current = nextNode;
    }
```

```
44
45      head = nullptr;
46  }
47
48  //Task 2: Compare lists
49  bool Compare(Node* h1, Node* h2) {
50
51      while (h1 != nullptr && h2 != nullptr) {
52          if (h1->data != h2->data) {
53              return false;
54          }
55          h1 = h1->next;
56          h2 = h2->next;
57      }
58
59      return h1 == nullptr && h2 == nullptr;
60  }
61
62  // Task 3: Add numbers
63  Node* Add(Node* n1, Node* n2){
64      Node* res = nullptr;
65      Node* temp = nullptr;
66      int strg = 0;
67
68      while (n1 != nullptr || n2 != nullptr || strg > 0) {
69          int sum = strg;
70          if (n1 != nullptr) {
71              sum += n1->data;
72              n1 = n1->next;
73          }
74          if (n2 != nullptr) {
75              sum += n2->data;
76              n2 = n2->next;
77          }
78
79          strg = sum / 10;
80          int data = sum % 10;
```

```
82          Node* newNode = new Node(data);
83
84          if (res == nullptr) {
85              res = newNode;
86              temp = newNode;
87          } else {
88              temp->next = newNode;
89              temp = temp->next;
90          }
91      }
92
93      return res;
94  }
95
96  int main() {
97      // Task 1
98      Node* main = new Node(4);
99      main->next = new Node(3);
100     main->next->next = new Node(2);
101     main->next->next->next = new Node(1);
102
103     cout << "Original list: ";
104     Print(main);
105
106     main = Reversal(main);
107
108     cout << "Reversed list: ";
109     Print(main);
110
111     deleteList(main);
112
113     // Task 2
114     Node* head1 = new Node(1);
115     head1->next = new Node(2);
116     head1->next->next = new Node(3);
117     head1->next->next->next = new Node(4);
118
119     Node* head2 = new Node(1);
120     head2->next = new Node(2);
121     head2->next->next = new Node(3);
122     head2->next->next->next = new Node(5);
```

```
123
124     if (Compare(head1, head2)) {
125         cout << "Lists are equal\n";
126     } else {
127         cout << "Lists are different\n";
128     }
129
130     deleteList(head1);
131     deleteList(head2);
132
133     // Task 3
134     Node* n1 = new Node(9);
135     n1->next = new Node(7);
136     n1->next->next = new Node(3);
137
138     Node* n2 = new Node(6);
139     n2->next = new Node(4);
140     n2->next->next = new Node(8);
141
142     Node* res = Add(n1, n2);
143
144     cout << "Result: ";
145     Print(res);
146
147     deleteList(n1);
148     deleteList(n2);
149
150     return 0;
151 }
```

Завдання 4-5:

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

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

// Task 4: Create a mirrored tree
TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) {
        return nullptr;
    }

    TreeNode* tempNode = new TreeNode(root->data);

    tempNode->left = create_mirror_flip(root->right);
    tempNode->right = create_mirror_flip(root->left);

    return tempNode;
}

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

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

    deleteTree(root->left);
    deleteTree(root->right);

    delete root;
}

```

```cpp
// Task 5: Find the sum of tree
void tree_sum(TreeNode* root) {
    if (root == nullptr) {
        return;
    }

    int sum = 0;

    if (root->left != nullptr) {
        sum += root->left->data;
    }

    if (root->right != nullptr) {
        sum += root->right->data;
    }

    if (sum > 0) {
        root->data = sum;
    }

    tree_sum(root->left);
    tree_sum(root->right);
}

int main(){
    // Task 4:
    TreeNode* root = new TreeNode(20);
    root->left = new TreeNode(15);
    root->right = new TreeNode(27);
    root->left->left = new TreeNode(12);
    root->left->right = new TreeNode(18);
    root->right->left = new TreeNode(25);
    root->right->right = new TreeNode(30);

    cout << "Original tree: ";
    Print(root);
    cout << endl;

    TreeNode* Mirrored = create_mirror_flip(root);

    cout << "Mirrored tree: ";
    Print(Mirrored);
    cout << endl;
```

```
88
89      deleteTree(root);
90      deleteTree(Mirrored);
91
92      // Task 5:
93      tree_sum(root);
94
95      cout << "Sum of the tree: ";
96      Print(root);
97
98      return 0;
99  }
```

Результат виконання завдань, тестування та фактично затрачений час:

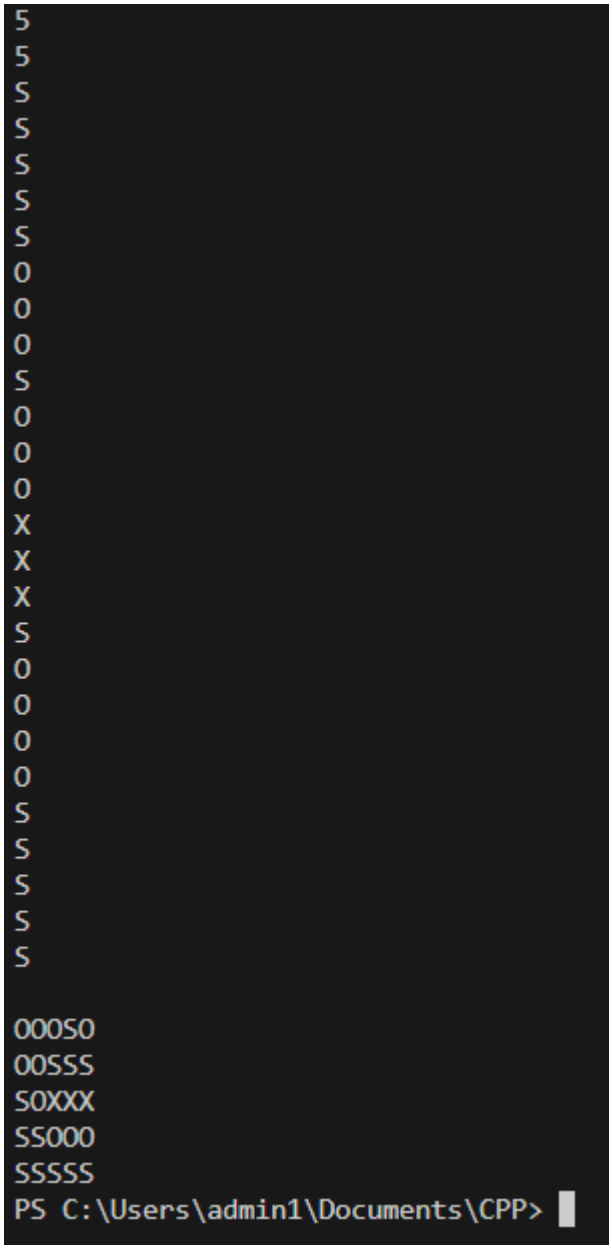
Завдання №1: VNS Lab 10 Variant 19

Фактично затрачений час: 1.5 год

```
Node4 Node3 Node2 Node1
Deleted 2 elements
Writing node: Node2
Writing node: Node1
Successfully wrote List into file
Deleted the List
Node2 Node1
Deleted the List
PS C:\Users\admin1\Documents\CPP\epics>
```

Завдання №2: Algotester Lab 5 Variant 2

Фактично затрачений час: 45 хв



a day ago	Lab 5v2 - Lab 5v2	C++ 23	Accepted	0.070	2.500	1936181

Завдання №3: Algotester Lab 7-8 Variant 2

Фактично затрачений час: 3 год

```
5
insert
0
3
64
37
129
size
3
get
1
37
print
64 37 129
capacity
4
PS C:\Users\admin1\Documents\CPP>
```

an hour ago	Lab 78v2 - Lab 78v2	C++ 23	Accepted	0.006	1.203	1936707

Завдання №4: Practice Work Task

Фактично затрачений час: 2 год

```
Original list: 4 3 2 1
Reversed list: 1 2 3 4
Lists are different
Result: 5 2 2 1
PS C:\Users\admin1\Documents\CPP>
```

```
Original tree: 12 15 18 20 25 27 30
Mirrored tree: 30 27 25 20 18 15 12
Sum of the tree: 12 30 18 42 25 55 30
PS C:\Users\admin1\Documents\CPP\epics>
```

Висновки: Я навчився користуватися динамічними структурами та попрактикувався з алгоритмами обробки динамічних структур.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/631