Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



# Звіт

про виконання

**Лабораторних та практичних робіт №6*:*** «Мови та парадигми програмування»

***з розділу***: «Лінійні та розгалужені алгоритми. Умовні оператори. Константи,

змінні»

***Виконав:***

студент групи ШІ-11 Сабадило Марко

Львів 2023

# Тема роботи:

Реалізація та використання динамічних структур даних**Мета роботи:**

* Ознайомитися з основними типами динамічних структур даних: стеком, чергою, списком, деревом.
* Вивчити основні операції над динамічними структурами даних.
* Закріпити навички реалізації алгоритмів з використанням колекцій на мові програмування С++.

Теоретичні відомості

Динамічні структури даних - це структури даних, які можуть змінюватися за розміром під час виконання програми. Вони використовуються для зберігання даних, які можуть змінюватися в процесі роботи програми.

Основні види динамічних структур даних:

Черга - це структура даних, в якій елементи додаються в один кінець і видаляються з іншого. Черги часто використовуються для моделювання черг, наприклад, в лініях до каси.

Стек - це структура даних, в якій елементи додаються і видаляються з одного кінця. Скеки часто використовуються для зберігання історії дій, наприклад, в веб-браузері.

Список - це структура даних, в якій елементи можуть бути додані в будь-яке місце. Списки часто використовуються для зберігання даних, які не мають певного порядку, наприклад, список покупок.

Дерево - це структура даних, в якій елементи можуть мати дочірні елементи. Дерева часто використовуються для зберігання даних, які мають ієрархічний порядок, наприклад, генеалогічне дерево.

Алгоритми обробки динамічних структур

Алгоритми обробки динамічних структур - це алгоритми, які працюють з динамічними структурами даних. Вони можуть використовуватися для виконання різних завдань, таких як додавання, видалення, пошук, сортування та обчислення.

Деякі поширені алгоритми обробки динамічних структур:

Додання елемента в чергу - це алгоритм, який додає елемент в кінець черги.

Видалення елемента з черги - це алгоритм, який видаляє елемент з початку черги.

Додання елемента в стек - це алгоритм, який додає елемент на вершину стека.

Видалення елемента зі стека - це алгоритм, який видаляє елемент з вершини стека.

Пошук елемента в списку - це алгоритм, який знаходить елемент в списку за заданим ключем.

Сортування списку - це алгоритм, який сортує елементи списку в певному порядку.

Обчислення суми елементів дерева - це алгоритм, який обчислює суму всіх елементів дерева.

Індивідуальний план опрацювання теорії

Тема №1: Черга

Джерела інформації:

Книга: "Структури даних і алгоритми" (Автор: В. Н. Войцеховський)

Відео: "Черга в Python" (YouTube-канал "Відеоуроки з програмування")

Стаття: "Черга в Java" (Вікіпедія)

Курс: "Структури даних і алгоритми" (Курс на сайті Coursera)

Що опрацьовано:

Поняття черги

Основні операції з чергою

Алгоритми додавання і видалення елементів з черги

Коментар 1:

Черга - це структура даних, в якій елементи додаються в один кінець і видаляються з іншого.

Основні операції з чергою - це додавання, видалення, пошук і перевірка наявності елемента в черзі.

Алгоритми додавання і видалення елементів з черги є досить простими.

Коментар 2:

Черги часто використовуються для моделювання черг, наприклад, в лініях до каси.

Черги також можуть використовуватися для реалізації деяких алгоритмів, наприклад, алгоритму Бойера-Мура.

Статус: Ознайомлений

Початок опрацювання теми: 2023-12-23

Звершення опрацювання теми: 2023-12-25

Тема №2: Стек

Джерела інформації:

Книга: "Структури даних і алгоритми" (Автор: В. Н. Войцеховський)

Відео: "Стек в Python" (YouTube-канал "Відеоуроки з програмування")

Стаття: "Стек в Java" (Вікіпедія)

Курс: "Структури даних і алгоритми" (Курс на сайті Coursera)

Що опрацьовано:

Поняття стеката організацію програмного

**Код програм:**

Завдання № 1

|  |
| --- |
| # #include <iostream> using namespace std;  int main() {  int n, m;  if (!(cin >> n >> m)) {  cerr << "Invalid input format!\n";  return 1;  }   char cave[1000][1000];   for (int i = 0; i < n; i++) {  for (int j = 0; j < m; j++) {  cin >> cave[i][j];  }  }   // Цикл для імітації землетрусу  for (int j = 0; j < m; j++) {  for (int i = n - 2; i >= 0; i--) {  if (cave[i][j] == 'S') {  int counter = 1, newRow = i;  while (i + counter < n && cave[i + counter][j] == 'O') {  newRow++;  counter++;  }   cave[i][j] = 'O';  cave[newRow][j] = 'S';  }  }  }   // Виведення результату  for (int i = 0; i < n; i++) {  for (int j = 0; j < m; j++) {  cout << cave[i][j];  }  cout << endl;  }   return 0; } |

Завдання № 2

|  |
| --- |
| #include <iostream>  using namespace std;  class DynamicArray { private:  int\* arr;  int size;  int capacity;  public:  DynamicArray() : arr(new int [1]), size(0), capacity(1) {}   ~DynamicArray() {  delete[] arr;  }   int& operator[](int index) {  return arr[index];  }   void insert(int index, int N, int\* values) {  if (index < 0 || index > size) return;  resize(N);   for (int i = size - 1; i >= index; i--) {  arr[i + N] = arr[i];  }   for (int i = 0; i < N; i++) {  arr[index + i] = values[i];  }   size += N;  resize();  }   void erase(int index, int n) {  if (index < 0 || index >= size) return;   for (int i = index + n; i < size; i++) {  arr[i - n] = arr[i];  }   size -= n;  }   int getSize() const {  return size;  }   int getCapacity() const {  return capacity;  }   friend ostream& operator<<(ostream& os, const DynamicArray& da) {  for (int i = 0; i < da.size; i++) {  os << da.arr[i] << " ";  }  return os;  }  private:  void resize(int N=0) {  while(size + N >=capacity) {  int newCapacity = capacity \* 2;  int \*temp = new int[newCapacity];  for (int i = 0; i < size; i++) {  temp[i] = arr[i];  }   delete[] arr;  arr = temp;  capacity = newCapacity;  }  } };  int main() {  int Q;  cin >> Q;   DynamicArray da;   for(int j=0;j<Q;j++) {  string operation;  cin >> operation;   if (operation == "insert") {  int index, N;  cin >> index >> N;  int\* values = new int[N];  for (int i = 0; i < N; i++) {  cin >> values[i];  }  da.insert(index, N, values); // cout << da.getSize() << endl; // Вивід розміру після вставки // cout << da.getCapacity() << endl; // Вивід ємності після вставки  delete[] values;  } else if (operation == "erase") {  int index, n;  cin >> index >> n;  da.erase(index, n);  } else if (operation == "size") {  cout << da.getSize() << endl;  } else if (operation == "capacity") {  cout << da.getCapacity() << endl;  } else if (operation == "get") {  int index;  cin >> index;  cout << da[index] << endl;  } else if (operation == "set") {  int index, value;  cin >> index >> value;  da[index] = value;  } else if (operation == "print") {  cout << da << endl;  }  }   return 0; } |

Завдання № 3

|  |
| --- |
| #include <iostream> #include <cstring>  using namespace std;  // Структура елемента двонаправленого списку struct Node {  char \*data;  Node \*prev;  Node \*next; };  // Функція для створення нового вузла Node\* createNode(const char \*str) {  Node \*newNode = new Node;  newNode->data = new char[strlen(str) + 1];  strcpy(newNode->data, str);  newNode->prev = newNode->next = nullptr;  return newNode; }  // Функція для додавання нового вузла на початок списку void addToFront(Node \*&head, const char \*str) {  Node \*newNode = createNode(str);  if (!head) {  head = newNode;  return;  }  newNode->next = head;  head->prev = newNode;  head = newNode; }  // Функція для додавання нового вузла на кінець списку void addToEnd(Node \*&head, const char \*str) {  Node \*newNode = createNode(str);  if (!head) {  head = newNode;  return;  }  Node \*temp = head;  while (temp->next) {  temp = temp->next;  }  temp->next = newNode;  newNode->prev = temp; }  // Функція для видалення елементів перед і після елемента з заданим ключем void removeAdjacentNodes(Node \*&head, const char \*key) {  Node \*current = head;  while (current) {  if (strcmp(current->data, key) == 0) {  if (current->prev) {  Node \*prevNode = current->prev;  if (prevNode->prev) {  prevNode->prev->next = current;  } else {  head = current;  }  delete prevNode;  }   if (current->next) {  Node \*nextNode = current->next;  if (nextNode->next) {  nextNode->next->prev = current;  }  delete nextNode;  }  }  current = current->next;  } }  // Функція для виведення списку void printList(Node \*head) {  Node \*temp = head;  while (temp) {  cout << temp->data << " ";  temp = temp->next;  }  cout << endl; }  // Головна функція int main() {  Node \*head = nullptr;   addToFront(head, "world");  addToFront(head, "hello");   printList(head); // Виведе: hello world  return 0; } |

Висновки Виконання лабораторної роботи дозволило ознайомитися з основними типами динамічних структур даних: стеком, чергою, списком, деревом. Були вивчені основні операції над динамічними структурами даних та закріплені навички реалізації алгоритмів на мові програмування С++. У процесі виконання лабораторної роботи були реалізовані динамічні структури даних - вектори за допомогою динамічних масивів, списки та дерева — власних вузлів. Дані колекції були згодом використанні для вирішення задач, поставлених у лабораторній роботі, що продемонструвало їх важливість та допомогло зрозуміти їх роботу зсередини.

Зображення, що містить текст, знімок екрана, дизайн, Шрифт

Автоматично згенерований опис

**Висновки**:Виконання лабораторної роботи дозволило ознайомитися з основними типами динамічних структур даних: стеком, чергою, списком, деревом. Були вивчені основні операції над динамічними структурами даних та закріплені навички реалізації алгоритмів на мові програмування С++. У процесі виконання лабораторної роботи були реалізовані динамічні структури даних - вектори за допомогою динамічних масивів, списки та дерева — власних вузлів. Дані колекції були згодом використанні для вирішення задач, поставлених у лабораторній роботі, що продемонструвало їх важливість та допомогло зрозуміти їх роботу зсередини.