Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра “Вычислительная техника”

**Отчет**

по лабораторной работе №8

по курсу “ Логика и основы алгоритмизации в инженерных задачах”

на тему “Обход графа в ширину”

Выполнили

студенты группы 22ВВП2:

Широкова И.Д.

Перкин П.О.

Приняли

Акифьев И.В.

Юрова О.В.

Пенза 2023

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру обхода в ширину, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.

**Задание 2\***

1. Для матричной формы представления графов реализуйте алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3.
2. Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

**Выполнение задания 2**

|  |  |  |
| --- | --- | --- |
| **Размер матрицы** | **Стандартная библиотека** | **Собственная реализация** |
| 100 | 0.001 | 0.001 |
| 500 | 0.009 | 0.004 |
| 1000 | 0.022 | 0.016 |
| 2000 | 0.064 | 0.057 |
| 3000 | 0.136 | 0.125 |
| 4000 | 0.238 | 0.227 |
| 5000 | 0.369 | 0.347 |
| 6000 | 0.538 | 0.531 |
| 7000 | 0.7 | 0.675 |
| 8000 | 0.909 | 0.883 |

Алгоритм с собственным списком всегда работает немного быстрее, чем, алгоритм, использующий встроенную библиотеку queue. Время выполнения всех трех алгоритмов примерно линейно масштабируется в зависимости от размера массива.

В целом, оба алгоритма обеспечивают приемлемую производительность, разница во времени выполнения между двумя алгоритмами несущественна.

**Листинг**

**#include <iostream>**

**#include <ctime>**

**#include <cstdlib>**

**#include <queue>**

**#include <fstream>**

**using namespace std;**

**struct Node {**

**int data;**

**Node\* next;**

**};**

**void printAdjacencyList(Node\*\* adjacencyList, int size) {**

**for (int i = 0; i < size; i++) {**

**cout << "Vertex " << i << " is adjacent to: ";**

**Node\* current = adjacencyList[i];**

**while (current != nullptr) {**

**cout << current->data << " ";**

**current = current->next;**

**}**

**cout << endl;**

**}**

**}**

**Node\*\* adjacencyMatrixToAdjacencyList(int\*\* matrix, int size) {**

**Node\*\* adjacencyList = new Node \* [size];**

**for (int i = 0; i < size; i++) {**

**adjacencyList[i] = nullptr;**

**for (int j = size - 1; j >= 0; j--) {**

**if (matrix[i][j] == 1) {**

**Node\* newNode = new Node;**

**newNode->data = j;**

**newNode->next = adjacencyList[i];**

**adjacencyList[i] = newNode;**

**}**

**}**

**}**

**return adjacencyList;**

**}**

**void freeAdjacencyList(Node\*\* adjacencyList, int size) {**

**for (int i = 0; i < size; i++) {**

**Node\* current = adjacencyList[i];**

**while (current != nullptr) {**

**Node\* next = current->next;**

**delete current;**

**current = next;**

**}**

**}**

**delete[] adjacencyList;**

**}**

**int\*\* createMatrix(int size) {**

**int\*\* matrix = new int\* [size];**

**for (int i = 0; i < size; i++) {**

**matrix[i] = new int[size];**

**}**

**return matrix;**

**}**

**void deleteMatrix(int\*\* matrix, int size) {**

**for (int i = 0; i < size; i++) {**

**delete[] matrix[i];**

**}**

**delete[] matrix;**

**}**

**void fillMatrixRandom(int\*\* matrix, int size) {**

**for (int i = 0; i < size; i++) {**

**for (int j = i; j < size; j++) {**

**if (i == j) {**

**matrix[i][j] = 0;**

**}**

**else {**

**int randomValue = rand() % 2;**

**matrix[i][j] = randomValue;**

**matrix[j][i] = randomValue;**

**}**

**}**

**}**

**}**

**void printMatrix(int\*\* matrix, int size) {**

**for (int i = 0; i < size; i++) {**

**for (int j = 0; j < size; j++) {**

**cout << matrix[i][j];**

**cout << ", ";**

**}**

**cout << endl;**

**}**

**}**

**int\* breadthFirstSearch(int\*\* graph, int size, int startVertex) {**

**bool\* visited = new bool[size];**

**for (int i = 0; i < size; i++) {**

**visited[i] = false;**

**}**

**int\* traversalOrder = new int[size];**

**int currentIndex = 0;**

**queue<int> q;**

**q.push(startVertex);**

**visited[startVertex] = true;**

**while (!q.empty()) {**

**int currentVertex = q.front();**

**q.pop();**

**traversalOrder[currentIndex++] = currentVertex;**

**for (int i = 0; i < size; i++) {**

**if (graph[currentVertex][i] == 1 && !visited[i]) {**

**q.push(i);**

**visited[i] = true;**

**}**

**}**

**}**

**delete[] visited;**

**return traversalOrder;**

**}**

**int\* bfsTraversal(Node\*\* adjacencyList, int size, int startVertex) {**

**bool\* visited = new bool[size];**

**for (int i = 0; i < size; i++) {**

**visited[i] = false;**

**}**

**int\* traversalOrder = new int[size];**

**int currentIndex = 0;**

**queue<int> q;**

**q.push(startVertex);**

**visited[startVertex] = true;**

**while (!q.empty()) {**

**int currentVertex = q.front();**

**q.pop();**

**traversalOrder[currentIndex++] = currentVertex;**

**Node\* current = adjacencyList[currentVertex];**

**while (current != nullptr) {**

**int neighbor = current->data;**

**if (!visited[neighbor]) {**

**q.push(neighbor);**

**visited[neighbor] = true;**

**}**

**current = current->next;**

**}**

**}**

**delete[] visited;**

**return traversalOrder;**

**}**

**void addEdge(Node\*\* adjacencyList, int vertex, int data) {**

**Node\* newNode = new Node;**

**newNode->data = data;**

**newNode->next = adjacencyList[vertex];**

**adjacencyList[vertex] = newNode;**

**}**

**struct QueueNode {**

**int data;**

**QueueNode\* next;**

**};**

**struct Queue {**

**QueueNode\* front;**

**QueueNode\* rear;**

**};**

**Queue\* createQueue() {**

**Queue\* queue = new Queue;**

**queue->front = queue->rear = nullptr;**

**return queue;**

**}**

**bool isEmpty(Queue\* queue) {**

**return queue->front == nullptr;**

**}**

**void enqueue(Queue\* queue, int data) {**

**QueueNode\* newNode = new QueueNode;**

**newNode->data = data;**

**newNode->next = nullptr;**

**if (isEmpty(queue)) {**

**queue->front = queue->rear = newNode;**

**return;**

**}**

**queue->rear->next = newNode;**

**queue->rear = newNode;**

**}**

**int dequeue(Queue\* queue) {**

**if (isEmpty(queue)) {**

**cerr << "Очередь пуста!" << endl;**

**return -1;**

**}**

**int data = queue->front->data;**

**QueueNode\* temp = queue->front;**

**queue->front = queue->front->next;**

**delete temp;**

**if (queue->front == nullptr) {**

**queue->rear = nullptr;**

**}**

**return data;**

**}**

**int\* BFScustomList(int\*\* matrix, int size, int startVertex) {**

**bool\* visited = new bool[size];**

**for (int i = 0; i < size; i++) {**

**visited[i] = false;**

**}**

**int\* traversalOrder = new int[size];**

**int currentIndex = 0;**

**Queue\* queue = createQueue();**

**enqueue(queue, startVertex);**

**visited[startVertex] = true;**

**while (!isEmpty(queue)) {**

**int currentVertex = dequeue(queue);**

**traversalOrder[currentIndex++] = currentVertex;**

**for (int i = 0; i < size; i++) {**

**if (matrix[currentVertex][i] == 1 && !visited[i]) {**

**enqueue(queue, i);**

**visited[i] = true;**

**}**

**}**

**}**

**delete[] visited;**

**delete(queue);**

**return traversalOrder;**

**}**

**int main() {**

**setlocale(LC\_ALL, "Rus");**

**srand(time(0));**

**int choice;**

**do {**

**cout << "Меню:" << endl;**

**cout << "1. генерация матрицы и поиск в ширину" << endl;**

**cout << "2. тест алгоритмов обхода в ширину" << endl;**

**cout << "3. Выход" << endl;**

**cout << "Пожалуйста, введите номер выбранного пункта: ";**

**cin >> choice;**

**switch (choice) {**

**case 1: {**

**cout << "Вы выбрали пункт 1." << endl;**

**int size;**

**cout << "\nВведите размер матрицы смежности: ";**

**cin >> size;**

**int\*\* adjacencyMatrix = createMatrix(size);**

**fillMatrixRandom(adjacencyMatrix, size);**

**printMatrix(adjacencyMatrix, size);**

**cout << "\nAdjacency List:\n";**

**Node\*\* adjacencyList = adjacencyMatrixToAdjacencyList(adjacencyMatrix, size);**

**printAdjacencyList(adjacencyList, size);**

**int startVertex;**

**cout << "Введите начальную вершину для обхода: ";**

**cin >> startVertex; cout << "\n";**

**int\* traversalOrder = breadthFirstSearch(adjacencyMatrix, size, startVertex);**

**for (int i = 0; i < size; i++) {**

**cout << traversalOrder[i] << " ";**

**}**

**cout << " ВbreadthFirstSearch\n" << endl;**

**delete[] traversalOrder;**

**int\* traversalOrder1 = bfsTraversal(adjacencyList, size, startVertex);**

**for (int i = 0; i < size; i++) {**

**cout << traversalOrder1[i] << " ";**

**}**

**cout << " bfsTraversal\n" << endl;**

**delete[] traversalOrder1;**

**int\* traversalOrder2 = BFScustomList(adjacencyMatrix, size, startVertex);**

**for (int i = 0; i < size; i++) {**

**cout << traversalOrder2[i] << " ";**

**}**

**delete[] traversalOrder2;**

**cout << " BFScustomList\n" << endl;**

**deleteMatrix(adjacencyMatrix, size);**

**break; }**

**case 2: {**

**cout << "Вы выбрали пункт 2." << endl;**

**int sizes[] = { 100, 200, 500 };**

**ofstream outputFile("graph\_traversal\_times.txt");**

**if (!outputFile.is\_open()) {**

**cerr << "Failed to open the output file." << endl;**

**return 1;**

**}**

**outputFile << "Matrix Size\tBFS Matrix (s)\tBFS List (s)\tBFS Custom (s)\n";**

**for (int i = 0; i < 3; i++) {**

**int size = sizes[i];**

**int\*\* adjacencyMatrix = createMatrix(size);**

**fillMatrixRandom(adjacencyMatrix, size);**

**Node\*\* adjacencyList = adjacencyMatrixToAdjacencyList(adjacencyMatrix, size);**

**int startVertex = 0;**

**clock\_t startTime, endTime;**

**startTime = clock();**

**int\* traversalOrder = breadthFirstSearch(adjacencyMatrix, size, startVertex);**

**endTime = clock();**

**double bfsMatrixTime = double(endTime - startTime) / CLOCKS\_PER\_SEC;**

**delete[] traversalOrder;**

**startTime = clock();**

**int\* traversalOrder1 = bfsTraversal(adjacencyList, size, startVertex);**

**endTime = clock();**

**double bfsListTime = double(endTime - startTime) / CLOCKS\_PER\_SEC;**

**delete[] traversalOrder1;**

**startTime = clock();**

**int\* traversalOrder2 = BFScustomList(adjacencyMatrix, size, startVertex);**

**endTime = clock();**

**double bfsCustomTime = double(endTime - startTime) / CLOCKS\_PER\_SEC;**

**delete[] traversalOrder2;**

**outputFile << size << "\t\t" << bfsMatrixTime << "\t\t" << bfsListTime << "\t\t" << bfsCustomTime << "\n";**

**deleteMatrix(adjacencyMatrix, size);**

**freeAdjacencyList(adjacencyList, size);**

**}**

**outputFile.close();**

**break; }**

**case 3:**

**cout << "Выход." << endl;**

**break;**

**default:**

**cout << "Неправильный выбор. Пожалуйста, введите верный номер пункта." << endl;**

**break;**

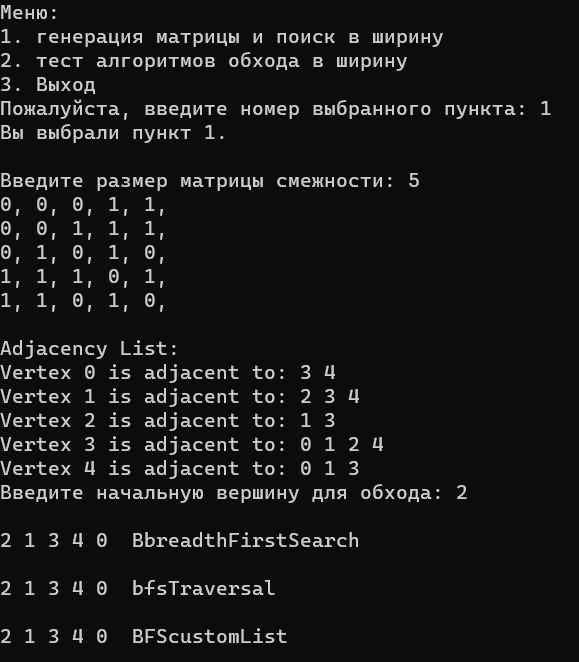
**}**

**} while (choice != 3);**

**return 0;**

**}**

**Результаты работы программы**

****

**Вывод**

В ходе выполнения данной работы был изучен обход графа в ширину, как с использованием рекурсии, так и без нее. Мы изучили способы обходить графы в ширину для различных представлений графов, включая матричную форму и списки смежности.