Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №8  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Обход графа в ширину»

Выполнили студенты группы 22ВВВ2:  
Хоссейни Нежад С. А. С. М.

Приняли:  
Акифьев И. В.  
Митрохин М. А.

Пенза 2023

**Название**

Обход графа в ширину

**Лабораторное задание**

**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.

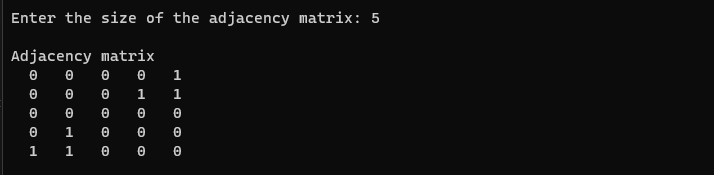
**Задание 2\***

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. Ход работы

**Ход работы**

**Задание 1**

1. Сгенерировал (используя генератор случайных чисел) матрицу смежности для неориентированного графа G. Вывел матрицу на экран.



1. Для сгенерированного графа осуществил процедуру обхода в ширину. При реализации алгоритма в качестве очереди использовал класс queue из стандартной библиотеки С++.



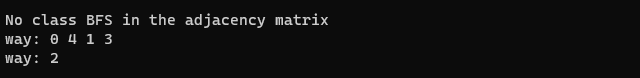
1. **\***Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.



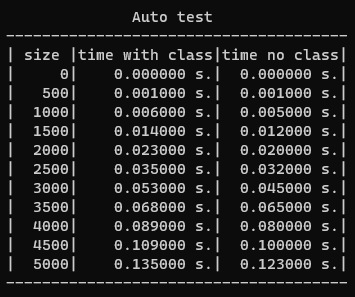


**Задание 2 \***

1. Для матричной формы представления графов реализовал алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3.



2. Оценил время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс queue и использующего очередь, реализованную самостоятельно) для графов разных порядков.



На всех размерах по времени быстрее всего работает реализация алгоритма обхода в ширину реализованная самостоятельно.

**Листинг**

* Header.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

void fill\_arr(int\*\* Arr, int size);

void print\_arr(int\*\* Arr, int size);

int find\_arr(int\* Arr, int size, int element);

struct node\* createNode(int v);

struct Graph\* createGraph(int vertices);

void addEdge(struct Graph\* graph, int src, int dest);

void printGraph(struct Graph\* graph);

struct Graph\* transformGraph(int\*\* Arr, int size);

void BFSList(struct Graph\* graph, int size);

void BfsList(struct Graph\* graph, int vertex, int\* visited);

void BFS(int\*\* Matrix, int size, int shutdown);

void Bfs(int\*\* Matrix, int size, int vertex, int\* visited, int shutdown);

void noClassBFS(int\*\* Matrix, int size, int shutdown);

void noClassBfs(int\*\* Matrix, int size, int vertex, int\* visited, int shutdown);

void autoTest();

struct nodeQueue\* create\_node(int inf);

int pop();

void push(int inf);

int queueEmpty();

* Queue.cpp

#include "Header.h"

struct nodeQueue\* head = NULL, \* last = NULL;

struct nodeQueue

{

int inf;

struct nodeQueue\* next;

};

void push(int inf)

{

struct nodeQueue\* p = NULL;

p = create\_node(inf);

if (head == NULL && p != NULL)

{

head = p;

last = p;

}

else if (head != NULL && p != NULL)

{

last->next = p;

last = p;

}

return;

}

int pop()

{

if (head == NULL)

{

printf("the list is empty\n");

return NULL;

}

struct nodeQueue\* struc = head;

head = struc->next;

int data = struc->inf;

free(struc);

return data;

}

struct nodeQueue\* create\_node(int inf)

{

struct nodeQueue\* p = NULL;

int ext = 0;

if ((p = (nodeQueue\*)malloc(sizeof(struct nodeQueue))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

p->inf = inf;

p->next = NULL;

return p;

}

int queueEmpty()

{

if (head == NULL)

return 1;

else

return 0;

}

* Lab8.cpp

#include "Header.h"

#include <queue>

#include <iostream>

using namespace std;

int main(void)

{

srand(time(NULL));

int sizeMatrix;

int\*\* Arr;

printf("Enter the size of the adjacency matrix: ");

scanf("%d", &sizeMatrix);

Arr = (int\*\*)malloc(sizeMatrix \* sizeof(int\*));

for (int i = 0; i < sizeMatrix; ++i)

{

Arr[i] = (int\*)malloc(sizeMatrix \* sizeof(int));

}

printf("\nAdjacency matrix\n");

fill\_arr(Arr, sizeMatrix);

print\_arr(Arr, sizeMatrix);

struct Graph\* graph = transformGraph(Arr, sizeMatrix);

printGraph(graph);

printf("\nBFS in the adjacency matrix");

BFS(Arr, sizeMatrix, 0);

printf("\n\nBFS in the adjacency list");

BFSList(graph, sizeMatrix);

printf("\n\nNo class BFS in the adjacency matrix");

noClassBFS(Arr, sizeMatrix, 0);

for (int i = 0; i < sizeMatrix; i++)

free(Arr[i]);

free(Arr);

autoTest();

return 0;

}

struct node

{

int vertex;

struct node\* next;

};

struct Graph

{

int numVertices;

struct node\*\* adjLists;

};

void BFS(int\*\* Matrix, int size, int shutdown)

{

int\* visited = (int\*)calloc(size, size \* sizeof(int));

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

{

if (visited[i] == 0) {

if(!shutdown)

printf("\nway: ");

Bfs(Matrix, size, i, visited, shutdown);

}

}

free(visited);

}

void Bfs(int\*\* Matrix, int size, int vertex, int\* visited, int shutdown)

{

std::queue<int> queue;

queue.push(vertex);

visited[vertex] = 1;

while (!queue.empty())

{

vertex = queue.front();

queue.pop();

if (!shutdown)

printf("%d ", vertex);

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

{

if (Matrix[vertex][i] == 1 && visited[i] == 0)

{

queue.push(i);

visited[i] = 1;

}

}

}

}

void BFSList(struct Graph\* graph, int size)

{

int\* visited = (int\*)calloc(size, size \* sizeof(int));

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

{

if (visited[i] == 0) {

printf("\nway: ");

BfsList(graph, i, visited);

}

}

free(visited);

}

void BfsList(struct Graph\* graph, int vertex, int\* visited)

{

std::queue<int> queue;

queue.push(vertex);

visited[vertex] = 1;

while (!queue.empty())

{

vertex = queue.front();

queue.pop();

printf("%d ", vertex);

struct node\* temp = graph->adjLists[vertex];

while (temp)

{

int i = temp->vertex;

if (visited[i] == 0)

{

queue.push(i);

visited[i] = 1;

}

temp = temp->next;

}

}

}

void noClassBFS(int\*\* Matrix, int size, int shutdown)

{

int\* visited = (int\*)calloc(size, size \* sizeof(int));

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

{

if (visited[i] == 0) {

if (!shutdown)

printf("\nway: ");

noClassBfs(Matrix, size, i, visited, shutdown);

}

}

free(visited);

}

void noClassBfs(int\*\* Matrix, int size, int vertex, int\* visited, int shutdown)

{

std::queue<int> queue;

push(vertex);

visited[vertex] = 1;

while (!queueEmpty())

{

vertex = pop();

if (!shutdown)

printf("%d ", vertex);

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

{

if (Matrix[vertex][i] == 1 && visited[i] == 0)

{

push(i);

visited[i] = 1;

}

}

}

}

void print\_arr(int\*\* Arr, int size)

{

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

{

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

{

printf("%3d ", Arr[i][j]);

}

printf("\n");

}

}

void fill\_arr(int\*\* Arr, int size)

{

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

{

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

{

if (i == j)

{

Arr[i][j] = 0;

continue;

}

if (rand() % 3 == 1)

{

Arr[i][j] = 1;

Arr[j][i] = 1;

}

else

{

Arr[i][j] = 0;

Arr[j][i] = 0;

}

}

}

}

void autoTest()

{

printf("\n\n Auto test");

printf("\n--------------------------------------\n");

printf("| size |time with class|time no class|\n");

for (int size = 0; size <= 5000; size += 500)

{

printf("|%6d|", size);

int\*\* Arr = (int\*\*)malloc(size \* sizeof(int\*));

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

{

Arr[i] = (int\*)malloc(size \* sizeof(int));

}

fill\_arr(Arr, size);

clock\_t start = clock();

BFS(Arr, size, 1);

clock\_t end = clock();

double time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("%12f s.", time\_spent);

start = clock();

noClassBFS(Arr, size, 1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("|%10f s.|\n", time\_spent);

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

free(Arr[i]);

free(Arr); }

printf("--------------------------------------\n");

}

struct node\* createNode(int v)

{

struct node\* newNode = (struct node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = (struct node\*\*)malloc(vertices \* sizeof(struct node\*));

for (int i = 0; i < vertices; i++)

graph->adjLists[i] = NULL;

return graph;

}

struct Graph\* transformGraph(int\*\* Arr, int size)

{

struct Graph\* graph = createGraph(size);

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

{

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

{

if (Arr[i][j] == 1)

{

addEdge(graph, i, j);

}

}

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph)

{

printf("\nAdjancecy list");

for (int v = 0; v < graph->numVertices; v++)

{

struct node\* temp = graph->adjLists[v];

printf("\n|%d|", v);

while (temp)

{

printf(" -> %d", temp->vertex);

temp = temp->next;

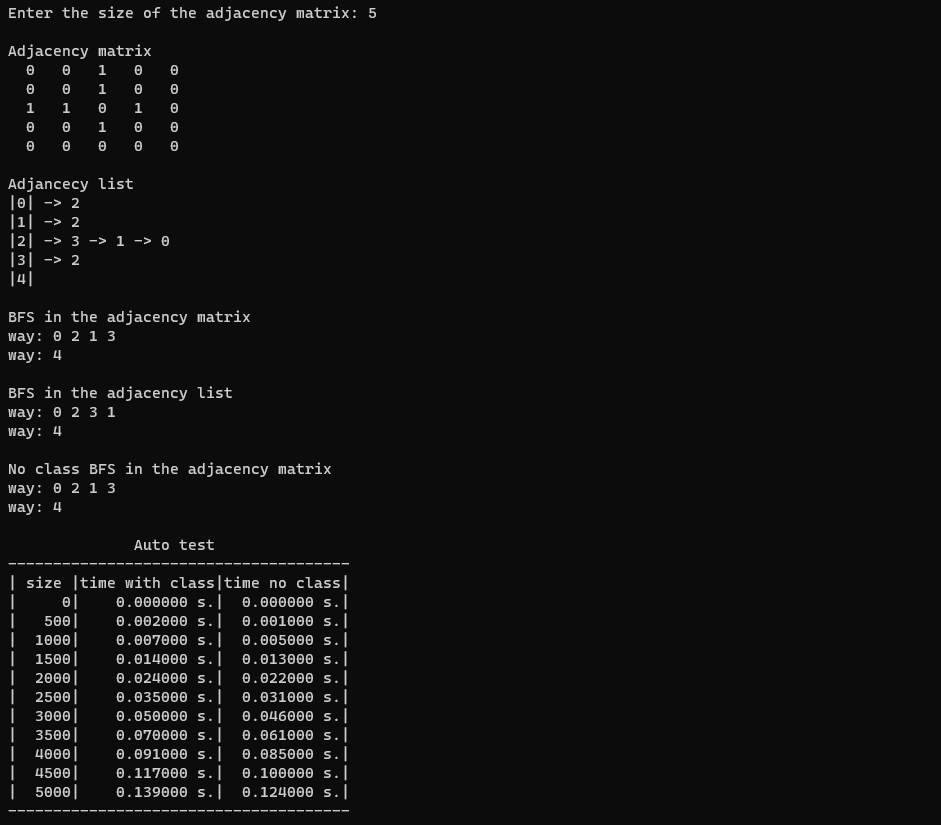
}

}

printf("\n");

}

**Результат работы программы**

****

**Вывод**

В результате выполнения лабораторной работы я научился выполнять обход графа в ширину.