Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №8

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

## на тему: "Обход графа в ширину"

Выполнили студенты гр. 22ВВП2:

Корнилов В.М.

Самофалова А.В.

Горбунов Д.А.

Приняли:

Акифьев И.В.

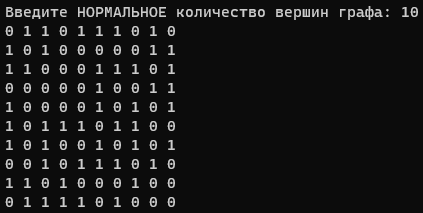
Юрова О.В.

Пенза, 2023

**Цель: Обойти граф в ширину.**

**Задание 1.**

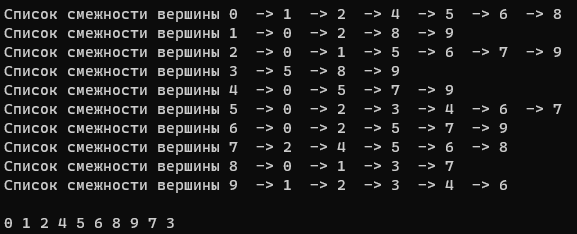
1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа G. Выведите матрицу на экран.



1. 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++.

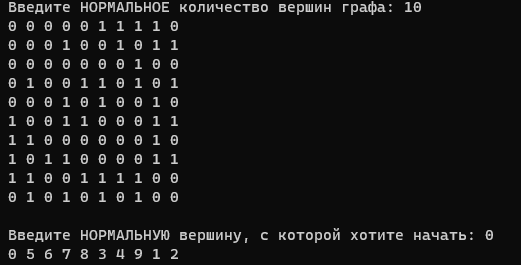


1. \* Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.



**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.



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

|  |  |  |
| --- | --- | --- |
| **Алгоритм/набор данных** | **queue** | **Наша очередь** |
| 10 | 0.000 | 0.000 |
| 100 | 0.000 | 0.001 |
| 500 | 0.007 | 0.007 |
| 1000 | 0.017 | 0.016 |
| 2000 | 0.047 | 0.049 |

На не большом количестве вершин алгоритмы работают одинаково, в пределах небольшой погрешности. При увеличении порядка графа алгоритм, использующий стандартную библиотеку queue, работает быстрее.

**Вывод:** научились обходить граф в ширину.

**ПриложениеА (Листинг)**

**File lab8.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include<windows.h>

#include<queue>

#include<time.h>

#include<iostream>

using namespace std;

typedef struct Node{

int ver;

struct Node\* next;

} Node;

// Функция добавления ребра в список

void addver(Node\*\* List, int x, int y) {

// Созданиеновогоузла

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->ver = y;

newNode->next = NULL;

// Проверка, существует ли уже ребро в списке

if (List[x] == NULL) {

List[x] = newNode;

}

else {

Node\* temp = List[x];

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

return;

}

// Функция отображения списков смежности

void vivod(Node\*\* List, int n) {

int i;

for (i = 0; i < n; ++i) {

Node\* temp = List[i];

printf("Список смежности вершины %d ", i);

while (temp) {

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

temp = temp->next;

}

printf("\n");

}

return;

}

int BFS(int\*\* A, int\* vis, int n, int v) {

queue<int> q;

q.push(v);

vis[v] = 1;

while (!q.empty()) {

v = q.front();

printf("%d ", v);

q.pop();

for (int i = 0; i < n; i++) {

if (A[v][i] == 1 && vis[i] != 1) {

q.push(i);

vis[i] = 1;

}

}

}

return 0;

}

int BFS2(Node\*\* List, int\* vis, int n, int v) {

queue<int> q;

q.push(v);

vis[v] = 1;

while (!q.empty()) {

v = q.front();

printf("%d ", v);

q.pop();

Node\* temp = List[v]; // обновляем указатель на список смежных вершин

while (temp != NULL) {

int i = temp->ver;

if (vis[i] != 1) {

q.push(i);

vis[i] = 1;

}

temp = temp->next; // переходим к следующему узлу списка

}

}

return 0;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n = 0;

int\*\* A;

int\* vis;

int\* vis1;

int v = 0;

// Создание и заполнение матрицы смежности

printf("Введите НОРМАЛЬНОЕ количество вершин графа: ");

scanf("%d", &n);

vis = (int\*)calloc(n, sizeof(int));

srand(time(NULL));

A = (int\*\*)malloc(sizeof(int\*) \* n);

for (int i = 0; i < n; i++) {

A[i] = (int\*)malloc(sizeof(int) \* n);

}

for (int i = 0; i < n; i++) {

for (int j = i; j < n; j++) {

A[i][j] = rand() % 2;

A[j][i] = A[i][j];

if (i == j) {

A[i][j] = 0;

}

}

}

// Выводматрицысмежности

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

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

}

printf("\n");

}

printf("\n");

printf("Введите НОРМАЛЬНУЮ вершину, с которой хотите начать: ");

scanf("%d", &v);

time\_t start1 = clock();

BFS(A, vis, n, v);

time\_t stop = clock();

double time2 = ((double)stop - (double)start1) / CLOCKS\_PER\_SEC;//времясортировки

printf("\nВремя обхода: %5.3f секунд", time2);

// Создание списка смежности

Node\*\* list1 = (Node\*\*)malloc(n \* sizeof(Node\*));

int i, j;

for (i = 0; i < n; ++i) {

list1[i] = NULL;

}

// Заполнение списка смежности

for (i = 0; i < n; ++i) {

for (j = 0; j < n; ++j) {

if (A[i][j] == 1) {

addver(list1, i, j);

}

}

}

// Отображениеспискасмежности

vis1 = (int\*)calloc(n, sizeof(int));

printf("\n");

printf("\n");

vivod(list1, n);

printf("\n");

BFS2(list1, vis1, n, v);

return 0;

}

**File 30BallovZaCTpls.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#include <time.h>

#include <Windows.h>

struct node

{

int ver; // полезная информация

struct node\* next; // ссылка на следующий элемент

};

struct node\* head = NULL, \* last = NULL, \* f = NULL; // указатели на первый и последний элементы списка

struct node\* get\_struct(int v)

{

struct node\* p = NULL;

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL) // выделяем память под новый элемент списка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->ver= v;

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в конец)\*/

void spstore(int d)

{

struct node\* p = NULL;

p = get\_struct(d);

if (head == NULL && p != NULL) // если списка нет, то устанавливаем голову списка

{

head = p;

last = p;

}

else if (head != NULL && p != NULL) // список уже есть, то вставляем в конец

{

last->next = p;

last = p;

}

return;

}

/\* Удаление элемента \*/

int del(void)

{

struct node\* struc = head;

struct node\* prev = NULL;

if (isEmpty() == 0)

return 0;

prev = head->next;

free(head);

head = prev;

return 0;

}

int isEmpty() {

struct node\* struc = head;

if (head == NULL)

{

return 0;

}

return 1;

}

int BFS(int\*\* A, int\* vis, int n, int v) {

spstore(v);

vis[v] = 1;

while (isEmpty() != 0) {

struct node\* struc = head;

v = head->ver;

printf("%d ", v);

del();

for (int i = 0; i < n; i++) {

if (A[v][i] == 1 && vis[i] != 1) {

spstore(i);

vis[i] = 1;

}

}

}

return 0;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n = 0;

int\*\* A;

int\* vis;

int v = 0;

// Создание и заполнение матрицы смежности

printf("Введите НОРМАЛЬНОЕ количество вершин графа: ");

scanf("%d", &n);

vis = (int\*)calloc(n, sizeof(int));

srand(time(NULL));

A = (int\*\*)malloc(sizeof(int\*) \* n);

for (int i = 0; i < n; i++) {

A[i] = (int\*)malloc(sizeof(int) \* n);

}

for (int i = 0; i < n; i++) {

for (int j = i; j < n; j++) {

A[i][j] = rand() % 2;

A[j][i] = A[i][j];

if (i == j) {

A[i][j] = 0;

}

}

}

// Вывод матрицы смежности

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

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

}

printf("\n");

}

printf("\n");

printf("Введите НОРМАЛЬНУЮ вершину, с которой хотите начать: ");

scanf("%d", &v);

time\_t start1 = clock();

BFS(A, vis, n, v);

time\_t stop = clock();

double time2 = ((double)stop - (double)start1) / CLOCKS\_PER\_SEC;

printf("\nВремя обхода: %5.3f секунд", time2);

}