Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра "Вычислительная техника"

Отчет

по лабораторной работе №9

по курсу “Л и ОА в ИЗ”

на тему “Поиск расстояний в графе”

Выполнили студенты группы 22ВВС1:

Агапов И.Е.

Братчиков А.А.

Приняли:

Юрова О.В.

Акифьев И.В.

Пенза 2023

### 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 поиска

расстояний, реализованную в соответствии с приведенным выше описанием.

При реализации алгоритма в качестве очереди используйте класс queue из

стандартной библиотеки С++.

3.\* Реализуйте процедуру поиска расстояний для графа, представленного

списками смежности.

**Задание 2\***

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.

2. Реализуйте процедуру поиска расстояний на основе обхода в глубину

для графа, представленного списками смежности.

3. Оцените время работы реализаций алгоритмов поиска расстояний на

основе обхода в глубину и обхода в ширину для графов разных порядков.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <iostream>

#include <queue>

using namespace std;

typedef struct Node {

int value;

struct Node\* next;

} Node;

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

Node\* create(int data)

{

// Выделение памяти под корень списка

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

tmp->value = data;

// Присваивание указателю на следующий элемент значения NULL

tmp->next = NULL;

return(tmp);

}

Node\* pushStart(int data, Node\* head)

{

Node\* tmp = create(data);

// Присваивание указателю на следующий элемент значения указателя на «голову»

// первоначального списка

tmp->next = head;

return(tmp);

}

char find\_el[256];

void spstore(int x), pop(void);

struct Node\* get\_struct(int x); // функция создания элемента

void BFS(queue <int> Q, int size, int\* vis, int\*\* mas, int S) {

Q.push(S);

vis[S] = 0;

while (!Q.empty()) {

S = Q.front();

printf("%d\t", S + 1);

Q.pop();

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

if (mas[S][i] == 1 && vis[i] == -1) {

vis[i] = vis[S] + 1;

Q.push(i);

}

}

}

return;

}

void DSF(int v, int size, int\* vis, int\*\* mas, int a) {

vis[v] = a;

printf("%d \t", v + 1);

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

if (mas[v][i] == 1 && vis[i] == -1) {

a++;

DSF(i, size, vis, mas, a);

a--;

}

}

return;

}

void DSFsps(int v, int size, int\* Num, Node\*\* last, int a) {

int w = 0;

Num[v] =a;

printf("%d \t", v + 1);

while (w < size) {

if (last[v] != NULL) {

if (Num[last[v]->value] == -1) {

a++;

DSFsps(last[v]->value, size,Num,last, a);

a--;

}

last[v] = last[v]->next;

}

w++;

}

}

void BFSsps(queue <int> Q, int size, int\* vis, Node\*\* last, int S) {

int y = 0;

Q.push(S);

vis[S]=0;

while (!Q.empty()) {

S = Q.front();

printf("%d\t", S + 1);

Q.pop();

y = last[S]->value;

while (last[S] != NULL) {

if (vis[last[S]->value] == -1) {

Q.push(last[S]->value);

vis[last[S]->value] = vis[S]+1;

}

last[S] = last[S]->next;

}

}

return;

}

int main() {

queue <int> SomeQ;

Node\*\* sps;

Node\*\* last;

int\*\* mas;

int size;

int\* Num;

int w;

srand(241324);

scanf("%d", &size);

sps = (Node\*\*)malloc(sizeof(Node\*) \* size);

last = (Node\*\*)malloc(sizeof(Node\*) \* size);

w = size;

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

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

int j = 0;

int i = 0;

setlocale(LC\_ALL, "");

for (int k = 0; k < size; k++) {

Num[k] = -1;

}

while (size > i) {

j = 0;

mas[i] = (int\*)malloc(sizeof(int) \* w);

while (w > j) {

mas[i][j] = 0;

j++;

}

i++;

}

i = 0;

j = 0;

while (size > i) {

j = i;

while (w > j) {

if (i == j) {

mas[i][j] = 0;

}

else {

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

mas[j][i] = mas[i][j];

}

j++;

}

i++;

}

//mas[1][4] = 0;

//mas[4][1] = 0;

i = 0;

j = 0;

bool fl = true;

while (size > i) {

j = 0;

while (w > j) {

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

if (fl) {

last[i] = pushStart(j, NULL);

fl = false;

}

else {

last[i] = pushStart(j, last[i]);

}

}

j++;

}

i++;

fl = true;

}

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

sps[i] = last[i];

}

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

printf("%d\t", i + 1);

while (last[i] != NULL) {

printf("%d\t", last[i]->value + 1);

last[i] = last[i]->next;

}

printf("\n");

}

printf("\n");

printf("\n");

printf("\n");

printf("\t");

for (int o = 0; o < size; o++) {

printf("%d\t", o + 1);

}

printf("\n");

for (int o = 0; o < size; o++) {

printf("%d\t", o + 1);

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

printf("%d\t", mas[o][n]);

}

printf("\n");

}

printf("\n");

printf("\n");

printf("\n");

w = 0;

scanf("%d", &w);

w--;

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

last[i] = sps[i];

}

//BFSselfqueue(SomeQ,size,Num,mas,w);

clock\_t start = clock();

BFSsps(SomeQ, size, Num, last, w);

clock\_t end = clock();

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

printf("\nВремя списки в ширину: %f сек\n", seconds);

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

printf("%d", Num[i]);

}

printf("\n");

for (int k = 0; k < size; k++) {

Num[k] = -1;

}

int a = 0;

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

last[i] = sps[i];

}

start = clock();

DSFsps(w, size, Num, last, 0);

end = clock();

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

printf("\nВремя списки в глубину: %f сек\n", seconds);

printf("\n");

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

printf("%d", Num[i]);

}

for (int k = 0; k < size; k++) {

Num[k] = -1;

}

printf("\n");

start = clock();

DSF(w, size, Num, mas, 0);

end = clock();

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

printf("\nВремя в глубину: %f сек\n", seconds);

printf("\n");

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

printf("%d", Num[i]);

}

printf("\n");

for (int k = 0; k < size; k++) {

Num[k] = -1;

}

start = clock();

BFS(SomeQ, size, Num, mas, w);

end = clock();

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

printf("\nВремя в ширину: %f сек\n", seconds);

printf("\n");

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

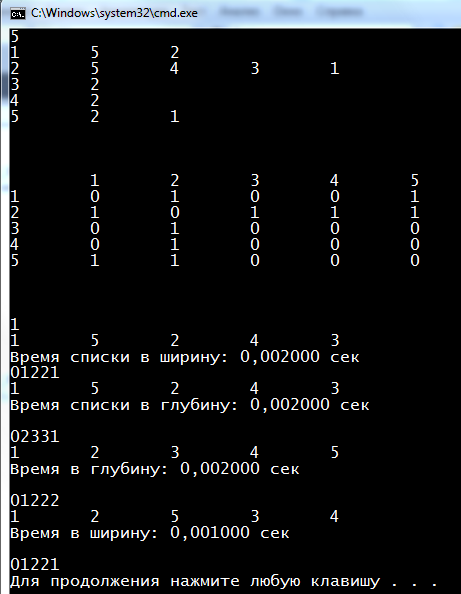
printf("%d", Num[i]);

}

printf("\n");

}

**Результат работы программы**

****

**Вывод**

### В ходе выполнения лабораторной работы были реализованы функции поиска расстояний в невзвешенном графе. Данная задача была выполнена при помощи поиска в глубину и поиска в ширину. Поиск в ширину показал явный выйгрыш в обходе графа. Данные функции были реализованы также для списков смежности.

### Были выполнены измерения скорости выполнения алгоритмов, результаты измерений представлены в данном отчете, мы изучили зависимость скорости каждой функции от размера графа.