Пензенский государственный университет

Кафедра «Вычислительная техника»

**Отчёт**

По лабораторной работе №6

по дисциплине: «Логика и основы оптимизации в инженерных задачах»

## на тему «Поиск расстояний в графе»

Выполнили студенты группы 19ВВ2:

Муромский Д.А.

Кобзев М.И.

Приняли:

Митрохин М. А.

Юрова О.В.

Пенза 2020

**Цель работы:** освоить алгоритм поиска расстояний в графе в ширину и глубину на матрицах смежности. Закрепить работу со списками путём реализации алгоритма поиска расстояний в графе в ширину и глубину на списках смежности. Сравнить время выполнения поиска расстояний в графе с обходом в грубину и ширину.

**Практическая часть**

1. **Задание 1.1** Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.

srand(time(NULL));

//Ввод

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];

}

}

//Обнуление диагонали

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

a[i][i] = 0;

}

//Одномерные массивы

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

vis[i] = 0;

}

//Вывод

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

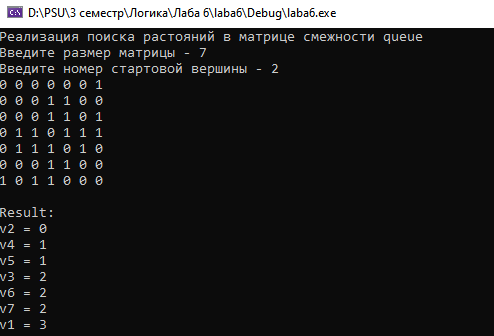
for(int j = 0; j < n; j++){

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

}

printf("\n");

}



1. **Задание 1.2** 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++.

void BFSD(int start, int\* dist, int\*\* a, int n) {

queue<int> Steck;

Steck.push(start);

dist[start] = 0;

printf("\nResult:\n");

printf("v%d = 0\n", start + 1);

int v;

while (!Steck.empty()) {

v = Steck.front();

Steck.pop();

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

if (a[v][i] && (dist[i] == -1)) {

Steck.push(i);

dist[i] = dist[v] + 1;

printf("v%d = %d\n", i + 1 , dist[i]);

}

}

}

/\*

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

printf("v%d = %d\n", i + 1, dist[i]);

}

\*/

}

void zad1(){

setlocale(LC\_ALL, "Rus");

int n, v;

printf("Реализация поиска растояний в матрице смежности queue\n");

printf("Введите размер матрицы - ");

scanf("%d", &n);

printf("Введите номер стартовой вершины - ");

scanf("%d", &v);

int \*\*a = dynamic\_array(n); //матрица смежности

int \*vis = dynamic\_array2(n); //проверка на прохождение

srand(time(NULL));

//Ввод

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];

}

}

//Обнуление диагонали

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

a[i][i] = 0;

}

//Одномерные массивы

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

vis[i] = -1;

}

//Вывод

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

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

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

}

printf("\n");

}

v--;

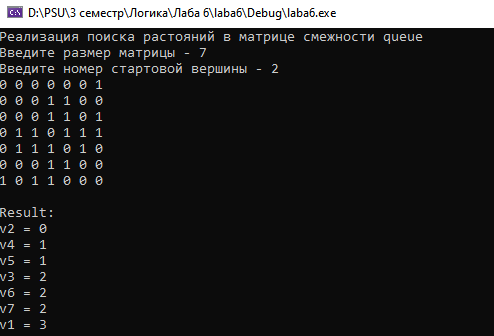
BFSD(v, vis, a, n);

dynamic\_array\_free(a, n);

dynamic\_array\_free1(vis, n);

\_getch();

}



**Задание 1.3\*** Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

void BFSDList(int start, int \*dist, List \*a, int n) {

queue<int> Steck;

Steck.push(start);

dist[start] = 0;

printf("\nResult:\n");

printf("v%d = 0\n", start + 1);

int v;

while (!Steck.empty()) {

v = Steck.front();

Steck.pop();

for (int i = 0; i < a[v].GetSize(); i++) {

if (dist[a[v][i]] == -1) {

Steck.push(a[v][i]);

dist[a[v][i]] = dist[v] + 1;

printf("v%d = %d\n", a[v][i] + 1 , dist[a[v][i]]);

}

}

}

/\*

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

printf("v%d = %d\n", i + 1, dist[i]);

}

\*/

}

void zad2() {

setlocale(LC\_ALL, "Rus");

int n, v;

printf("\n\nРеализация поиска растояний в списке смежности queue\n");

printf("Введите размер матрицы - ");

scanf("%d", &n);

printf("Введите номер вершины для прохождения - ");

scanf("%d", &v);

List \*relist = new List[n];

int \*\*a = dynamic\_array(n); //матрица смежности

int \*vis = dynamic\_array2(n); //проверка на прохождение

srand(time(NULL));

//Ввод

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];

}

}

//Обнуление диагонали

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

a[i][i] = 0;

vis[i] = -1;

}

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

printf("V%d:", i + 1);

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

if (a[i][j]) {

if (relist[i].GetSize() == 0) {

relist[i].push\_front(j);

printf(" v%d", j + 1);

}

else {

relist[i].push\_back(j);

printf(" v%d", j + 1);

}

}

}

printf("\n");

}

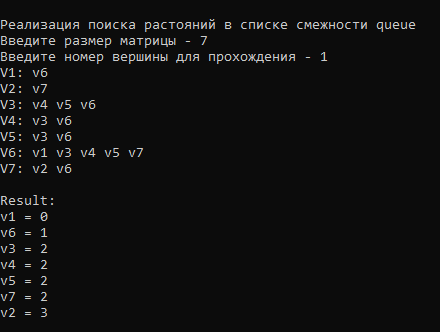
v--;

BFSDList(v, vis, relist, n);

delete[] relist;

\_getch();

}



**Задание 2.1\*** Реализуйте процедуру поиска расстояний на основе обхода в глубину.

void DFSD(int start, int\* dist, int\*\* a, int n) {

stack<int> Steck;

Steck.push(start);

dist[start] = 0;

printf("\nResult:\n");

int v;

while (!Steck.empty()) {

v = Steck.top();

Steck.pop();

printf("v%d = %d\n", v + 1, dist[v]);

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

if (a[v][i] && (dist[i]) == -1) {

Steck.push(i);

dist[i] = dist[v] + 1;

}

}

}

/\*

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

printf("v%d = %d\n", i + 1, dist[i]);

}

\*/

}

void zad3() {

setlocale(LC\_ALL, "Rus");

int n, v;

clock\_t start, end;

printf("\n\nРеализация поиска растояний в списке смежности stack\n");

printf("Введите размер матрицы - ");

scanf("%d", &n);

printf("Введите номер вершины для прохождения - ");

scanf("%d", &v);

int \*\*a = dynamic\_array(n); //матрица смежности

int \*vis = dynamic\_array2(n); //проверка на прохождение

srand(time(NULL));

//Ввод

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];

}

}

//Обнуление диагонали

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

a[i][i] = 0;

}

//Одномерные массивы

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

vis[i] = -1;

}

//Вывод

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

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

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

}

printf("\n");

}

v--;

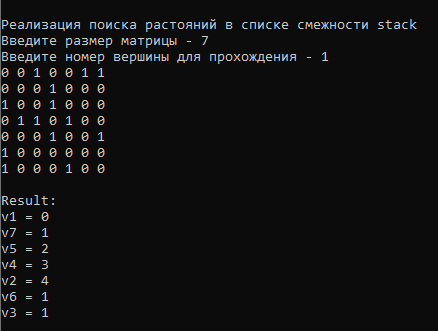
DFSD(v, vis, a, n);

dynamic\_array\_free(a, n);

dynamic\_array\_free1(vis, n);

\_getch();

}



**Задание 2.2\*** Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.

void DFSDList(int start, int\* dist, List\* a, int n) {

stack<int> Steck;

Steck.push(start);

dist[start] = 0;

printf("\nResult:\n");

int v;

while (!Steck.empty()) {

v = Steck.top();

Steck.pop();

printf("v%d = %d\n", v + 1, dist[v]);

for (int i = 0; i < a[v].GetSize(); i++) {

if (dist[a[v][i]] == -1) {

Steck.push(a[v][i]);

dist[a[v][i]] = dist[v] + 1;

}

}

}

/\*

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

printf("v%d = %d\n", i + 1, dist[i]);

}

\*/

}

void zad4() {

setlocale(LC\_ALL, "Rus");

int n, v;

printf("\n\nРеализация поиска растояний в списке смежности stack\n");

printf("Введите размер матрицы - ");

scanf("%d", &n);

printf("Введите номер вершины для прохождения - ");

scanf("%d", &v);

List\* relist = new List[n];

int\*\* a = dynamic\_array(n); //матрица смежности

int\* vis = dynamic\_array2(n); //проверка на прохождение

srand(time(NULL));

//Ввод

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];

}

}

//Обнуление диагонали

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

a[i][i] = 0;

vis[i] = -1;

}

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

printf("V%d:", i + 1);

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

if (a[i][j]) {

if (relist[i].GetSize() == 0) {

relist[i].push\_front(j);

printf(" v%d", j + 1);

}

else {

relist[i].push\_back(j);

printf(" v%d", j + 1);

}

}

}

printf("\n");

}

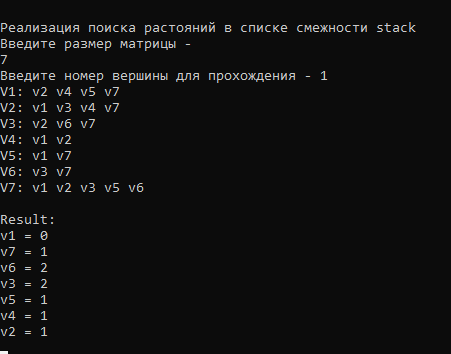
v--;

DFSDList(v, vis, relist, n);

delete[] relist;

\_getch();

}



**Задание 2.3\*** Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

void BFSD\_norec(int start, int\* dist, int\*\* a, int n) {

queue<int> Steck;

Steck.push(start);

dist[start] = 0;

int v;

while (!Steck.empty()) {

v = Steck.front();

Steck.pop();

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

if (a[v][i] && (dist[i] == -1)) {

Steck.push(i);

dist[i] = dist[v] + 1;

}

}

}

}

void DFSD\_norec(int start, int\* dist, int\*\* a, int n) {

stack<int> Steck;

Steck.push(start);

dist[start] = 0;

int v;

while (!Steck.empty()) {

v = Steck.top();

Steck.pop();

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

if (a[v][i] && (dist[i]) == -1) {

Steck.push(i);

dist[i] = dist[v] + 1;

}

}

}

}

void zvezdochki() {

setlocale(LC\_ALL, "Rus");

int n, v;

clock\_t start, end;

printf("\n\nСравнение времени поиска растояний в матрице смежности BFSD и DFSD\n");

printf("Введите размер матрицы - ");

scanf("%d", &n);

printf("Введите номер вершины для прохождения - ");

scanf("%d", &v);

int \*\*a = dynamic\_array(n); //матрица смежности

int \*vis = dynamic\_array2(n); //проверка на прохождение

srand(time(NULL));

//Ввод

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];

}

}

//Обнуление диагонали

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

a[i][i] = 0;

}

//Одномерные массивы

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

vis[i] = -1;

}

printf("\nResult BFSD: ");

v--;

start = clock();

BFSD\_norec(v, vis, a, n);

end = clock();

printf("\nUse %.2f msecond.\n", difftime(end, start));

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

vis[i] = -1;

}

printf("\nResult DFSD: ");

start = clock();

DFSD\_norec(v, vis, a, n);

end = clock();

printf("\nUse %.2f msecond.\n", difftime(end, start));

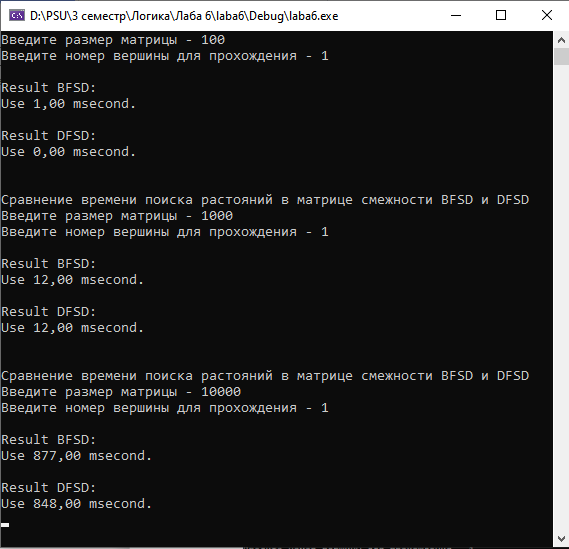
dynamic\_array\_free(a, n);

dynamic\_array\_free1(vis, n);

\_getch();

}

Результаты тестов:



**Вывод:** **Цель работы:** освоить алгоритм поиска расстояний в графе в ширину и глубину на матрицах смежности. Закрепить работу со списками путём реализации алгоритма поиска расстояний в графе в ширину и глубину на списках смежности. Сравнить время выполнения поиска расстояний в графе с обходом в грубину и ширину. В результате сравнения выяснилось , что реализации поиска в глубину работает быстрее, чем реализации поиска в ширину.