Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

# по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «поиск расстояния в графе»

**Выполнил:**

студент группы 20ВВ3

Духнов Олег

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

Поиск расстояний в графе.

# 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++.

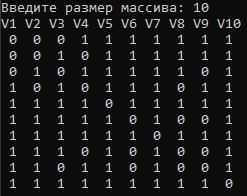
**3.**\* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

# 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.

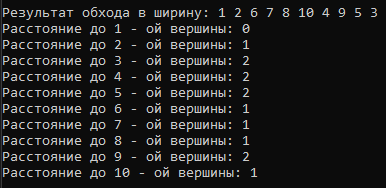
# Результат работы программы

**Задание 1.1.** Результаты работы программы показаны на рисунке 1.



**Рисунок 1 – Результат работы программы**

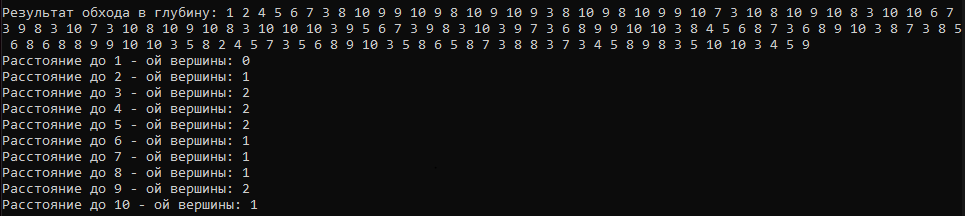
**Задание 1.2.** Результат обхода в ширину и результат поиска расстояния.



**Рисунок 2 – Результат работы программы**

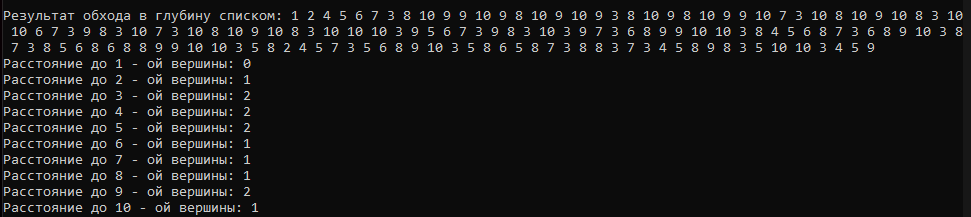
**Задание 1.3.** Реализация процедуры поиска расстояний для графа, представленного списками смежности.

**Задание 2.1.** Реализация процедуры поиска расстояний на основе обхода в глубину.



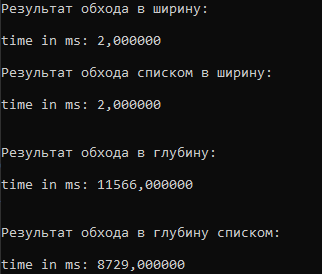
**Рисунок 3 – Результат работы программы**

**Задание 2.2.** Реализация процедуры поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.



**Рисунок 4 – Результат работы программы**

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



**Рисунок 5 – Результат работы программы**

**Вывод:** яизучил алгоритм поиска расстояний в графе.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <stdbool.h>

#include <conio.h>

#include <time.h>

#include <stack>

#include <queue>

using namespace std;

typedef struct Node

{

int vertex;

Node\* Next;

};

typedef struct Graf

{

int num;

Node\*\* Array;

};

Node\* init\_node(int meaning)

{

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

node->vertex = meaning;

node->Next = NULL;

return node;

}

Graf\* init\_graf(int n)

{

Graf\* graf = (Graf\*)malloc(sizeof(Graf));

graf->num = n;

graf->Array = (Node\*\*)malloc(n \* sizeof(Node\*));

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

graf->Array[i] = NULL;

return graf;

}

void add\_spisok(int i, int j, Graf\* graf)

{

Node\* NewNode = init\_node(j);

if (graf->Array[i] == NULL)

{

graf->Array[i] = NewNode;

NewNode = NULL;

}

Node\* temp = graf->Array[i];

while (temp->Next != NULL)

temp = temp->Next;

temp->Next = NewNode;

NewNode = init\_node(i);

if (graf->Array[j] == NULL)

{

graf->Array[j] = NewNode;

NewNode = NULL;

}

temp = graf->Array[j];

while (temp->Next != NULL)

temp = temp->Next;

temp->Next = NewNode;

}

void print(Graf\* graf)

{

for (int i = 0; i < graf->num; i++)

{

Node\* temp = graf->Array[i];

printf("%d - ая вершина: ", i + 1);

while (temp != NULL)

{

printf("%d ", temp->vertex + 1);

temp = temp->Next;

}

printf("\n");

}

}

int\*\* create\_arr(int n)

{

int s = 1;

int\*\* Arr;

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

srand(time(NULL));

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

{

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

Arr[i][i] = 0;

for (int j = s; j < n; j++)

{

if (rand() % 100 > 60)

{

Arr[i][j] = 0;

}

else {

Arr[i][j] = 1;

}

}

s++;

}

s = 1;

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

{

for (int j = s; j < n; j++)

Arr[j][i] = Arr[i][j];

s++;

}

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

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

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

{

printf("\n");

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

{

printf("%2d ", Arr[i][j]);

}

}

return Arr;

}

void BFSD(int n, int\* dist, int x, int\*\* Arr)

{

queue<int> que;

que.push(x);

dist[x] = 0;

while (!que.empty())

{

x = que.front();

que.pop();

printf("%d ", x + 1);

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

{

if (Arr[x][i] == 1 && dist[i] == -1)

{

que.push(i);

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

}

}

}

}

void DFS(int n, int\*\* Arr, int x, int\* dist, int k)

{

dist[x] = k;

printf("%d ", x + 1);

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

{

if (Arr[x][i] == 1 && dist[i] == -1)

DFS(n, Arr, i, dist, k + 1);

if (Arr[x][i] == 1 && dist[i] > k)

DFS(n, Arr, i, dist, k + 1);

}

}

void bfsd\_spisok(Graf\* graf, int n, int x, int\* dist)

{

queue<int> que;

que.push(x);

dist[x] = 0;

while (!que.empty())

{

x = que.front();

que.pop();

printf("%d ", x + 1);

Node\* temp = graf->Array[x];

while (temp)

{

if (dist[temp->vertex] == -1)

{

que.push(temp->vertex);

dist[temp->vertex] = dist[x] + 1;

}

temp = temp->Next;

}

}

}

void dfs\_spisok(Graf\* graf, int\* dist, int x, int k)

{

dist[x] = k;

printf("%d ", x + 1);

Node\* temp = graf->Array[x];

while (temp)

{

if (dist[temp->vertex] == -1)

dfs\_spisok(graf, dist, temp->vertex, k + 1);

if (dist[temp->vertex] > k)

dfs\_spisok(graf, dist, temp->vertex, k + 1);

temp = temp->Next;

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

clock\_t start, stop, st, end;

int n, x, \*\* t;

printf("Введите размер массива: ");

scanf("%d", &n);

t = create\_arr(n);

printf("\nВведите, из какой вершины вы хотите начать: ");

scanf("%d", &x);

while ((x <= 0) || (x > n))

{

printf("Такой вершины нет! Введите другую вершину: ");

scanf("%d", &x);

}

x--;

printf("\n\n");

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

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

dist[i] = -1;

printf("\nРезультат обхода в ширину: ");

start = clock();

BFSD(n, dist, x, t);

stop = clock();

float res = stop - start;

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

{

if (dist[i] != -1)

printf("\nРасстояние до %d - ой вершины: %d ", i + 1, dist[i]);

else

printf("\nДо вершины %d - нет расстояния", i + 1);

}

printf("\n\ntime in ms: %f", res);

Graf\* graf = init\_graf(n);

printf("\n");

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

{

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

{

if (t[i][j] == 1 && i > j)

add\_spisok(i, j, graf);

}

}

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

dist[i] = -1;

printf("\nРезультат обхода списком в ширину: ");

start = clock();

bfsd\_spisok(graf, n, x, dist);

stop = clock();

float resul = stop - start;

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

{

if (dist[i] != -1)

printf("\nРасстояние до %d - ой вершины: %d ", i + 1, dist[i]);

else

printf("\nДо вершины %d - нет расстояния", i + 1);

}

printf("\n\ntime in ms: %f", resul);

printf("\n\n");

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

dist[i] = -1;

printf("\nРезультат обхода в глубину: ");

start = clock();

DFS(n, t, x, dist, 0);

stop = clock();

float result = stop - start;

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

{

if (dist[i] != -1)

printf("\nРасстояние до %d - ой вершины: %d ", i + 1, dist[i]);

else

printf("\nДо вершины %d - нет расстояния", i + 1);

}

printf("\n\ntime in ms: %f", result);

printf("\n\n");

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

dist[i] = -1;

printf("\nРезультат обхода в глубину списком: ");

start = clock();

dfs\_spisok(graf, dist, x, 0);

stop = clock();

float resultat = stop - start;

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

{

if (dist[i] != -1)

printf("\nРасстояние до %d - ой вершины: %d ", i + 1, dist[i]);

else

printf("\nДо вершины %d - нет расстояния", i + 1);

}

printf("\n\ntime in ms: %f", resultat);

\_getch();

}