

Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра “Вычислительная техника”

Отчёт

по лабораторной работе №9
по курсу “Логика и основы алгоритмизации в инженерных задачах”
на тему “Поиск расстояний в графе”

Выполнил студент гр. 22ВВВЗ:
Кулахметов С.И.

Приняли:
к.т.н., доцент Юрова О.В.
к.э.н., доцент Акифьев И.В.

Пенза 2023

Цель работы

Реализовать алгоритм поиска расстояний между вершинами неориентированного графа при помощи обхода в ширину и обхода в глубину, выполнить лабораторное задание.

Лабораторное задание

Задание 1

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

2. Для сгенерированного графа осуществить процедуру поиска расстояний, реализованную в соответствии с приведенным в методическом указании алгоритмом. При реализации алгоритма в качестве очереди использовать класс **queue** из стандартной библиотеки C++.

3. Реализовать процедуру поиска расстояний для графа, представленного списками смежности.

Задание 2

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

2. Реализовать процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.

3. Оценить время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

Пояснительный текст к программе

Программа разделена на 3 файла, которые отвечают за выполнение различных частей лабораторной работы: 1 файл Task1.cpp — отвечает за выполнение первого задания полностью; 2 файл Lab9_Task2_1_2 — отвечает за выполнение пунктов 1 и 2 второго задания; 3 файл Task2_3 — отвечает за выполнение третьего пункта второго задания.

В первом файле существуют 2 основные функции: *BFSD* — поиск расстояний в графе, представленном матрицей смежности; *bfs* — поиск расстояний в графе, представленном списками смежности. За хранение информации о расстояниях отвечают массивы *DIST* и *dist* соответственно. В данном случае алгоритм нахождения расстояний основан на обходе в ширину. Также стоит упомянуть, что

нативно массив расстояний заполняется «-1», так как «0» или «1» могут быть расстояниями.

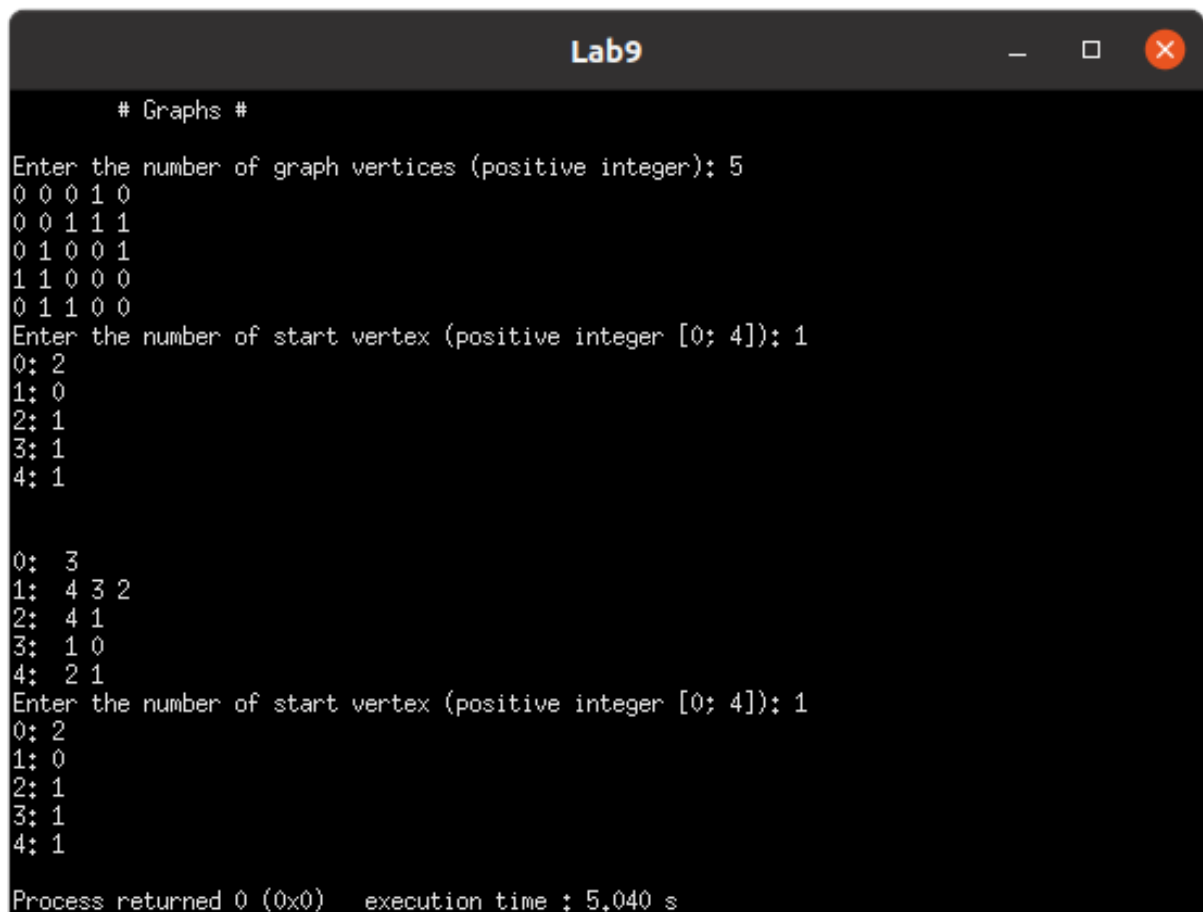
Во втором файле основными функциями являются: *DFS* — поиск расстояний в графе, представленном матрицей смежности; *ListDFS* - поиск расстояний в графе, представленном списками смежности. Алгоритм поиска расстояний основан на обходе графа в глубину.

В третьем файле представлено сравнение поиска расстояний в ширину и поиска расстояний в глубину по времени их выполнения. Для реализации задачи использовалась стандартная функция *clock()*;

Результаты работы программы

Задание 1

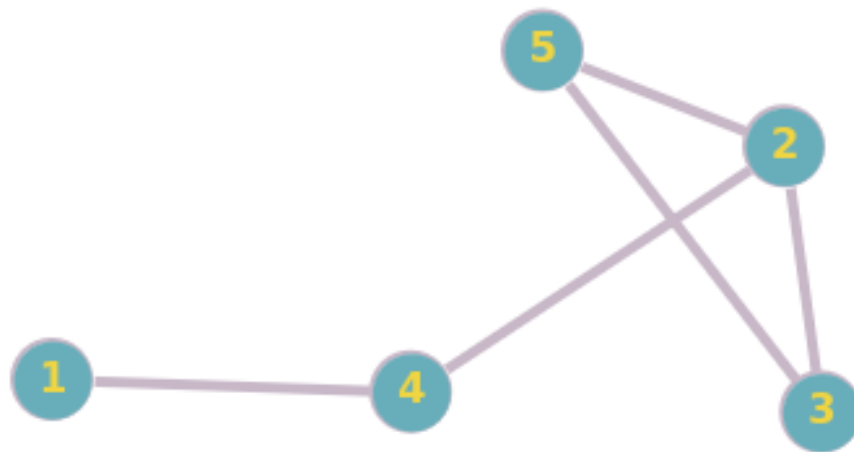
Построение матрицы и списков смежности. Выполнение поиска расстояний при помощи обхода в ширину.



```
Lab9
# Graphs #
Enter the number of graph vertices (positive integer): 5
0 0 0 1 0
0 0 1 1 1
0 1 0 0 1
1 1 0 0 0
0 1 1 0 0
Enter the number of start vertex (positive integer [0; 4]): 1
0: 2
1: 0
2: 1
3: 1
4: 1

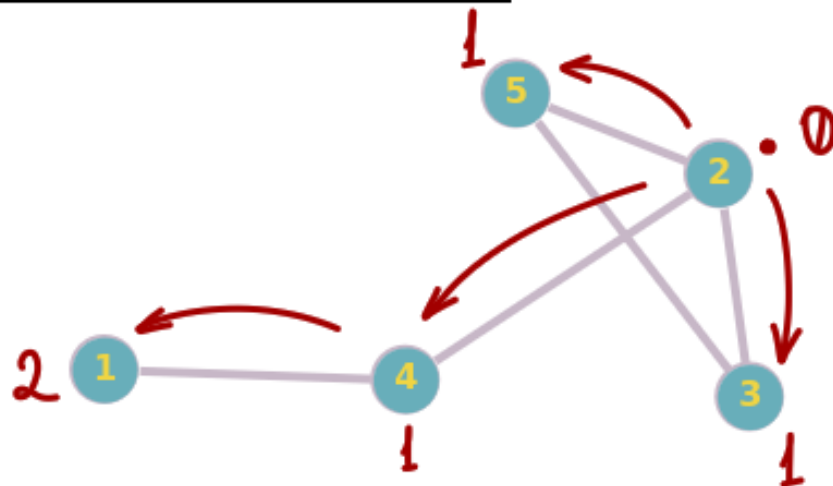
0: 3
1: 4 3 2
2: 4 1
3: 1 0
4: 2 1
Enter the number of start vertex (positive integer [0; 4]): 1
0: 2
1: 0
2: 1
3: 1
4: 1
Process returned 0 (0x0)   execution time : 5.040 s
```

Граф, сгенерированный по матрице смежности.



Поиск расстояний от заданной вершины при помощи обхода в ширину матрицы смежности.

```
Enter the number of graph vertices (positive integer): 5
0 0 0 1 0
0 0 1 1 1
0 1 0 0 1
1 1 0 0 0
0 1 1 0 0
Enter the number of start vertex (positive integer [0; 4]): 1
0: 2
1: 0
2: 1
3: 1
4: 1
```



Поиск расстояний при помощи обхода графа в ширину по спискам смежности идентичен рассмотренному только что примеру.

Задание 2

Построение матрицы и списков смежности. Выполнение поиска расстояний при помощи обхода в глубину.

```
Lab9_Task2

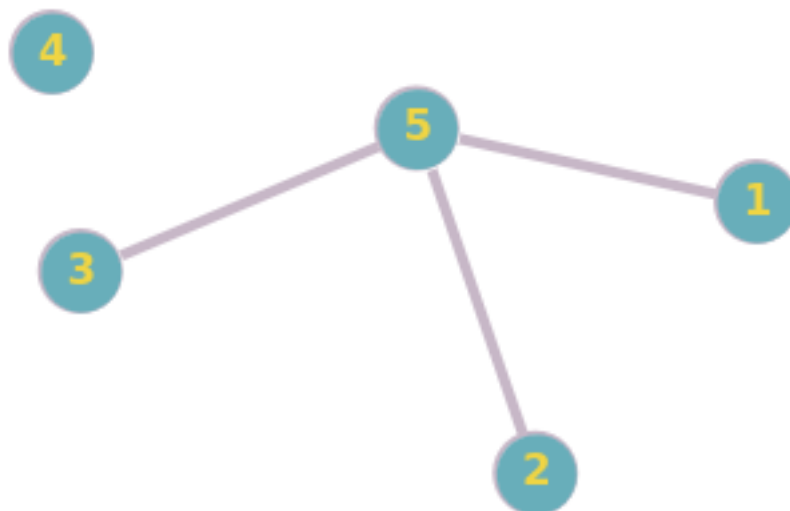
# Graphs #

Enter the number of graph vertices (positive integer): 5
0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0
1 1 1 0 0
Enter the number of start vertex (positive integer [0; 4]): 0
0 4 1 2
0: 0
1: 2
2: 2
3: -1
4: 1

# Adjacency list #
0: 4
1: 4
2: 4
3:
4: 2 1 0

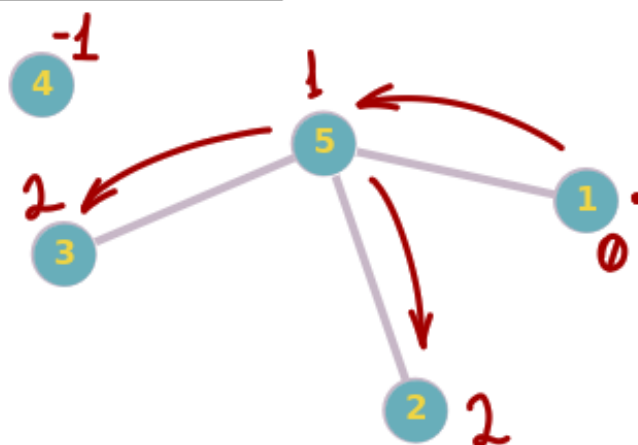
Enter the number of start vertex (positive integer[0; 4]): 0
0 4 2 1
0: 0
1: 2
2: 2
3: -1
4: 1
```

Граф, сгенерированный по матрице смежности.



Поиск расстояний от заданной вершины при помощи обхода в глубину матрицы смежности.

```
Enter the number of graph vertices (positive integer): 5
0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0
1 1 1 0 0
Enter the number of start vertex (positive integer [0; 4]): 0
0 4 1 2
0: 0
1: 2
2: 2
3: -1
4: 1
```



Поиск расстояний при помощи обхода графа в глубину по спискам смежности идентичен рассмотренному ранее примеру.

Измерение времени при нахождении расстояний путём обхода графов разных порядков в ширину и в глубину.

Количество вершин	Время обхода	
	BSF	DSF
5	0.043 с.	0.003 с.
10	0.054 с.	0.031 с.
15	0.086 с.	0.047 с.

(Замеры времени см. Приложение Б)

Вывод

В ходе выполнения данной лабораторной работы были получены навыки реализации на языке C++ алгоритма поиска расстояний от указанной вершины в неориентированном графе, представленном матрицей и списками смежности при помощи обхода в ширину и глубину. Проведено сравнение скорости работы алгоритма при поиске расстояний в графе путём обхода в ширину и глубину. В

результате можно сделать вывод, что алгоритм обхода в глубину работает быстрее. Однако, иногда он указывает более длинные расстояния от искомой вершины.

Ссылка на *GitHub* репозиторий с лабораторной работой

<https://github.com/KulakhmetovS/Lab9>

Приложение А

Листинг программы

Файл Lab9_Task1/main.cpp

```
#include <iostream>
#include <queue>
#include <cstdlib>
#include <ctime>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

int** Creategraph(int **, int); //Создание графа
void BFSD(int **, int *, int, int); //Обход в ширину

int size; //Размер графа

struct node
{
    int vertex;
    struct node* next;
};

struct node* createNode(int);

struct Graph
{
    int numVertices;
    struct node** adjLists;
    int* visited;
};

struct Graph* createGraph(int );
void addEdge(struct Graph *, int , int );
void printGraph(struct Graph *, int *);
void bfs(struct Graph *, int *, int );

int main()
{
    int i, j, v;
    int **graph = NULL, *DIST = NULL;

    printf("\t# Graphs #\n\n");
    printf("Enter the number of graph vertices (positive integer): ");
    scanf("%d", &size);

    // Creating the graph
    graph = Creategraph(graph, size);
    DIST = (int *) (malloc(sizeof(int *) * size)); // Array for visited vertices

    // Printing the matrix
    for(i = 0; i < size; i++)
    {
        DIST[i] = -1;
```



```

        for(j = 0; j < size; j++)
            printf("%d ", graph[i][j]);
        printf("\n");
    }

    int num = size - 1;

    printf("Enter the number of start vertex (positive integer [0; %d]): ", num);
    scanf("%d", &v);

    // <----- ! ----->
    BFSD(graph, DIST, size, v);
    // <----- ! ----->

    for(i = 0; i < size; i++)
        printf("%d: %d\n", i, DIST[i]);

    struct Graph* Graph = createGraph(size);
    int *dist = (int *) (malloc(sizeof(int *) * size));

    for(i = 0; i < size; i++)
    {
        dist[i] = -1;
        for(j = i; j < size; j++)
        {
            if(graph[i][j] == 1)
            {
                addEdge(Graph, i, j);
            }
        }
    }

    printf("\n\n");

    printGraph(Graph, dist);

    printf("Enter the number of start vertex (positive integer [0; %d]): ", num);
    scanf("%d", &v);

    // <----- ! ----->
    bfs(Graph, dist, v);
    // <----- ! ----->

    for(i = 0; i < size; i++)
        printf("%d: %d\n", i, dist[i] + 1);

    delete[] graph;
    delete[] DIST;
    delete[] dist;

    return 0;
}

int** Creategraph(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

```

```

// Memory allocation
graph = (int **)(malloc(sizeof(int *) * size));
for(i = 0; i < size; i++)
    graph[i] = (int *)(malloc(sizeof(int *) * size));

// Filling the matrix
for(i = 0; i < size; i++)
    for(j = i; j < size; j++)
    {
        graph[i][j] = rand() % 2;
        graph[j][i] = graph[i][j];
        if(i == j) graph[i][j] = 0;
        if(graph[i][j] == 1);
    }

return graph;
}

void BFSD(int **graph, int *DIST, int size, int v)
{
    queue<int> q;
    q.push(v);
    DIST[v] = 0;

    while(!q.empty())
    {
        v = q.front();
        q.pop();

        for(int i = 0; i < size; i++)
        {
            if((graph[v][i] == 1) && (DIST[i] == -1))
            {
                q.push(i);
                DIST[i] = DIST[v] + 1;
            }
        }
    }
}

//-----

void bfs(struct Graph* graph, int *dist, int startVertex)
{
    queue<int> q;
    q.push(startVertex);

    graph->visited[startVertex] = 1;

    while(!q.empty())
    {
        int currentVertex = q.front();
        q.pop();

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

        while(temp)

```

```

    {
        int adjVertex = temp->vertex;
        if(graph->visited[adjVertex] == 0)
        {
            graph->visited[adjVertex] = 1;
            q.push(adjVertex);
            dist[adjVertex] = dist[currentVertex] + 1;
        }
        temp = temp->next;
    }
}
}
}

```

```

struct node* createNode(int v)
{
    struct node* newNode = new struct node;
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

```

```

struct Graph* createGraph(int vertices)
{
    struct Graph* graph = new struct Graph;
    graph->numVertices = vertices;

    graph->adjLists = new struct node*[vertices];
    graph->visited = new int[vertices];

    int i;
    for (i = 0; i < vertices; i++)
    {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}

```

```

void addEdge(struct Graph* graph, int src, int dest)
{
    // Add edge from src to dest
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

```

```

void printGraph(struct Graph* graph, int *dist)
{
    int v;
    for (v = 0; v < graph->numVertices; v++)
    {
        struct node* temp = graph->adjLists[v];
        printf("%d: ", v);
    }
}

```

```

        if(temp == NULL) dist[v] = -2;

        while (temp)
        {
            printf("%d ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

```

Файл Lab9_Task2_1_2/main.c

```

#include <stdio.h>
#include <stdlib.h>

```

```

int** Creategraph(int **, int); //Создание графа
void DFS(int **, int *, int *, int, int);

```

```

int size; //Размер графа

```

```

// <----- Список смежности ----->
struct node
{
    int vertex;
    struct node* next;
};
struct node* createNode(int);
struct Graph
{
    int numVertices;
    struct node** adjLists;
};
struct Graph* createGraph(int);
void addEdge(struct Graph* , int , int);
void printGraph(struct Graph*);
// <----- ! ----->

```

```

void ListDFS(struct Graph*, int *, int *, int);

```

```

int main()
{
    int i, j, v, cnt;
    int **graph = NULL, *DIST = NULL, *dist = NULL, *visited = NULL;

    printf("\t# Graphs #\n\n");
    printf("Enter the number of graph vertices (positive integer): ");
    scanf("%d", &size);

    // Creating the graph
    graph = Creategraph(graph, size);
    DIST = (int *) (malloc(sizeof(int *) * size)); // Array for visited vertices
    dist = (int *) (malloc(sizeof(int *) * size));
    visited = (int *) (malloc(sizeof(int *) * size));
}

```

```

// Printing the matrix
for(i = 0; i < size; i++)
{
    cnt = 0;
    DIST[i] = -1;
    visited[i] = 0;
    for(j = 0; j < size; j++)
    {
        printf("%d ", graph[i][j]);
        if(graph[i][j] == 1) cnt++;
    }
    printf("\n");
    if(cnt == 0) DIST[i] = -2;
    dist[i] = DIST[i];
}

int num = size - 1;

printf("Enter the number of start vertex (positive integer [0; %d]): ", num);
scanf("%d", &v);

// <----- ! ----->
DFS(graph, DIST, visited, size, v);
// <----- ! ----->
printf("\n");

for(i = 0; i < size; i++)
    printf("%d: %d\n", i, DIST[i] + 1);

printf("\n\n");

// <----- Список смежности ----->
struct Graph* graf = createGraph(size);

for(i = 0; i < size; i++)
{
    for(j = i; j < size; j++)
    {
        if(graph[i][j] == 1)
        {
            addEdge(graf, i, j);
        }
    }
}

printf("\n# Adjacency list #\n");
printGraph(graf);
// <----- ! ----->

for(i = 0; i < size; i++)
    visited[i] = 0;

printf("\nEnter the number of start vertex (positive integer[0; %d]): ", num);
scanf("%d", &v);

// <----- ! ----->
ListDFS(graf, visited, dist, v);
// <----- ! ----->

```

```

printf("\n");

for(i = 0; i < size; i++)
    printf("%d: %d\n", i, dist[i] + 1);

free(DIST);
free(dist);
free(graph);
free(visited);

return 0;
}

int** Creategraph(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *) (malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = i; j < size; j++)
        {
            graph[i][j] = rand() % 2;
            graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            if(graph[i][j] == 1);
        }

    return graph;
}

void DFS(int **graph, int *DIST, int *visited, int size, int v)
{
    int i;
    visited[v] = 1;
    printf("%d ", v);

    for(i = 0; i < size; i++)
    {
        if((graph[v][i] == 1) && (visited[i] == 0))
        {
            DIST[i] = DIST[v] + 1;
            DFS(graph, DIST, visited, size, i);
        }
    }
}

// <----- Список смежности ----->
struct node* createNode(int v)
{
    struct node* newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
}

```

```

    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int vertices)
{
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;

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

    int i;
    for (i = 0; i < vertices; i++)
        graph->adjLists[i] = NULL;

    return graph;
}

void addEdge(struct Graph* graph, int src, int dest)
{
    // Add edge from src to dest
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

void printGraph(struct Graph* graph)
{
    int v;
    for (v = 0; v < graph->numVertices; v++)
    {
        struct node* temp = graph->adjLists[v];
        printf("%d: ", v);
        while(temp)
        {
            printf("%d ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

void ListDFS(struct Graph* graph, int *visited, int *dist, int v)
{
    visited[v] = 1;
    printf("%d ", v);
    struct node* temp = graph->adjLists[v];
    while (temp)
    {
        if(visited[temp->vertex] == 0)
        {
            dist[temp->vertex] = dist[v] + 1;
            ListDFS(graph, visited, dist, temp->vertex);
        }
        else
    }
}

```

```

    {
        temp = temp -> next;
    }
}
}

```

Файл Lab9_Task2_3/main.cpp

```

#include <iostream>
#include <queue>
#include <cstdlib>
#include <ctime>
#include <stdio.h>
#include <stdlib.h>

```

```
using namespace std;
```

```

int** Creategraph(int **, int); //Создание графа
void BFS(int **, int *, int, int); //Обход в ширину
void DFS(int **, int *, int *, int, int);

```

```
int size; //Размер графа
```

```
int main()
```

```

{
    int i, j, v, cnt;
    int **graph = NULL, *DIST = NULL, *dist = NULL, *visited = NULL;

```

```

    printf("\t# Graphs #\n\n");
    printf("Enter the number of graph vertices (positive integer): ");
    scanf("%d", &size);

```

```

    // Creating the graph
    graph = Creategraph(graph, size);
    DIST = (int *) (malloc(sizeof(int *) * size)); // Array for visited vertices
    dist = (int *) (malloc(sizeof(int *) * size));
    visited = (int *) (malloc(sizeof(int *) * size));

```

```

    // Printing the matrix
    for(i = 0; i < size; i++)
    {
        cnt = 0;
        DIST[i] = -1;
        dist[i] = DIST[i];
        visited[i] = 0;
        for(j = 0; j < size; j++)
        {
            printf("%d ", graph[i][j]);
            if(graph[i][j] == 1) cnt++;
        }
        printf("\n");
        if(cnt == 0) dist[i] = -2;
    }

```

```
int num = size - 1;
```



```

printf("Enter the number of start vertex (positive integer [0; %d]): ", num);
scanf("%d", &v);
printf("\t# BFS distance #\n");

// <----- ! ----->
unsigned int start_time = clock();
BFSD(graph, DIST, size, v);
unsigned int end_time = clock();
unsigned int search_time = end_time - start_time;
// <----- ! ----->
printf("\n");
for(i = 0; i < size; i++)
    printf("%d: %d\n", i, DIST[i]);
cout << "runtime = " << search_time / 1000.0;

printf("\n\n");

printf("Enter the number of start vertex (positive integer [0; %d]): ", num);
scanf("%d", &v);
printf("\t# DFS distance #\n");
// <----- ! ----->
start_time = clock();
DFS(graph, dist, visited, size, v);
end_time = clock();
search_time = end_time - start_time;
// <----- ! ----->
printf("\n");

for(i = 0; i < size; i++)
    printf("%d: %d\n", i, dist[i] + 1);
cout << "runtime = " << search_time / 1000.0;

delete[] visited;
delete[] dist;
delete[] graph;
delete[] DIST;

return 0;
}

int** Creategraph(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *) (malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = i; j < size; j++)
        {
            graph[i][j] = rand() % 2;
            graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
        }
}

```

```

        if(graph[i][j] == 1);
    }

    return graph;
}

void BFS(int **graph, int *DIST, int size, int v)
{
    queue<int> q;
    q.push(v);
    DIST[v] = 0;

    while(!q.empty())
    {
        v = q.front();
        printf("%d ", v);
        q.pop();

        for(int i = 0; i < size; i++)
        {
            if((graph[v][i] == 1) && (DIST[i] == -1))
            {
                q.push(i);
                DIST[i] = DIST[v] + 1;
            }
        }
    }
}

void DFS(int **graph, int *DIST, int *visited, int size, int v)
{
    int i;
    visited[v] = 1;
    printf("%d ", v);

    for(i = 0; i < size; i++)
    {
        if((graph[v][i] == 1) && (visited[i] == 0))
        {
            DIST[i] = DIST[v] + 1;
            DFS(graph, DIST, visited, size, i);
        }
    }
}

```

Приложение Б

Замеры времени поиска расстояний в графе путём обхода в ширину и глубину

```
Lab9_Task2_3
Enter the number of graph vertices (positive integer): 5
0 1 0 1 1
1 0 0 0 0
0 0 0 0 1
1 0 0 0 1
1 0 1 1 0
Enter the number of start vertex (positive integer [0; 4]): 2
# BFS distance #
2 4 0 3 1
0: 2
1: 3
2: 0
3: 2
4: 1
runtime = 0,043

Enter the number of start vertex (positive integer [0; 4]): 2
# DFS distance #
2 4 0 1 3
0: 2
1: 3
2: 0
3: 3
4: 1
runtime = 0,003
Process returned 0 (0x0)   execution time : 11,781 s
Press ENTER to continue.
```

```
Lab9_Task2_3
2: 1
3: 1
4: 0
5: 1
6: 2
7: 3
8: 2
9: 2
runtime = 0,054

Enter the number of start vertex (positive integer [0; 9]): 4
# DFS distance #
4 2 0 1 5 3 9 6 7 8
0: 2
1: 3
2: 1
3: 5
4: 0
5: 4
6: 7
7: 7
8: 8
9: 6
runtime = 0,031
Process returned 0 (0x0)   execution time : 7,377 s
Press ENTER to continue.
```

Lab9_Task2_3

```
13: 2
14: 1
runtime = 0.086

Enter the number of start vertex (positive integer [0; 14]): 7
# DFS distance #
7 3 2 0 1 5 4 8 6 9 10 12 13 11 14
0: 3
1: 4
2: 2
3: 1
4: 6
5: 5
6: 8
7: 0
8: 7
9: 9
10: 10
11: 13
12: 11
13: 12
14: 13
runtime = 0.047
Process returned 0 (0x0)   execution time : 6.151 s
Press ENTER to continue.
█
```