

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

Отчёт

по лабораторной работе №7
по курсу “Логика и основы алгоритмизации в инженерных задачах”
на тему “Обход графа в глубину”

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 обхода в глубину графа при помощи матрицы смежности и списков смежности. Выполнить лабораторное задание.

Лабораторное задание

Задание 1

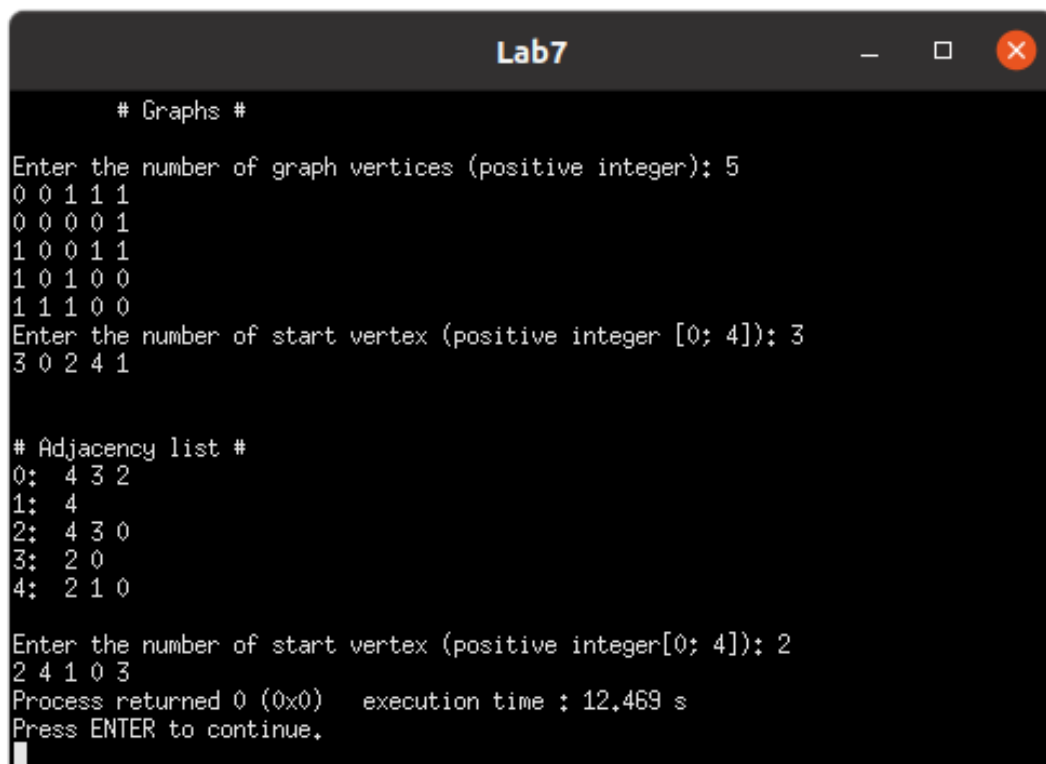
1. Сгенерировать (используя генератор случайных чисел) матрицу смежности для неориентированного графа G . Вывести матрицу на экран.
2. Для сгенерированного графа осуществить процедуру обхода в глубину, реализованную в соответствии с приведенным в методическом указании алгоритме.
3. Реализовать процедуру обхода в глубину для графа, представленного списками смежности.

Пояснительный текст к программе

В программе инициализирован двумерный массив $graph[m][n]$, отвечающий за хранение матрицы смежности неориентированного графа. А также структуры $node$ и $Graph$, отвечающие за реализацию списков смежности заданного графа. Функции обхода в глубину DFS и $ListDFS$, согласно реализации алгоритма, являются рекурсивными.

Результаты работы программы

Построение матрицы и списков смежности. Выполнение обхода в глубину.

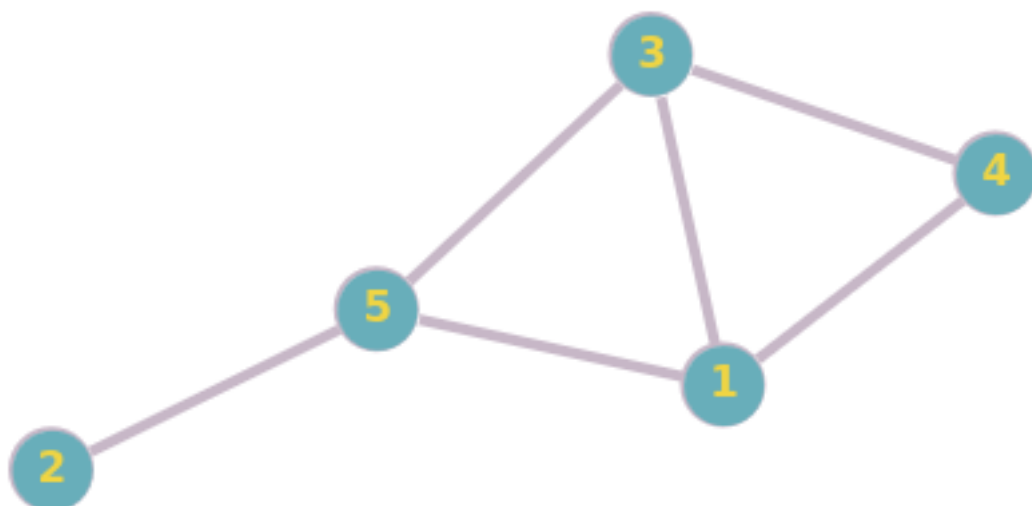


```
# Graphs #
Enter the number of graph vertices (positive integer): 5
0 0 1 1 1
0 0 0 0 1
1 0 0 1 1
1 0 1 0 0
1 1 1 0 0
Enter the number of start vertex (positive integer [0; 4]): 3
3 0 2 4 1

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

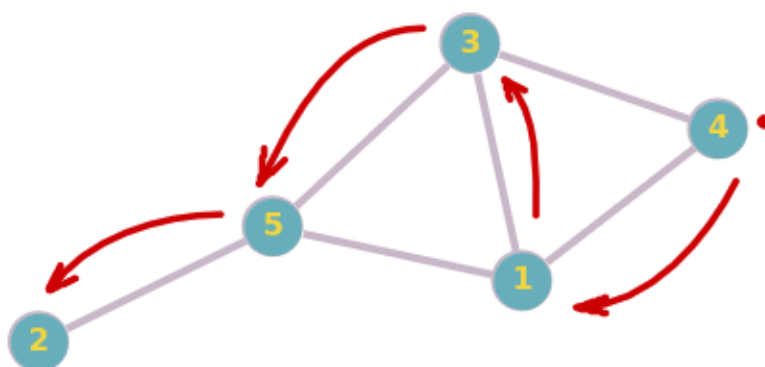
Enter the number of start vertex (positive integer[0; 4]): 2
2 4 1 0 3
Process returned 0 (0x0)   execution time : 12.469 s
Press ENTER to continue.
```

Граф, сгенерированный по матрице смежности.



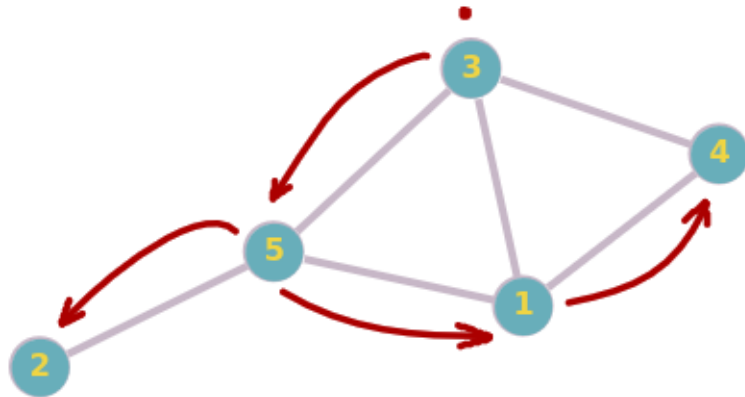
Обход графа в глубину по матрице инцидентности.

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



Обход графа в глубину по спискам инцидентности.

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



Вывод

В ходе выполнения данной лабораторной работы были получены навыки реализации на языке Си алгоритма поиска в глубину в неориентированном графе, представленном матрицей и списками смежности.

Ссылка на *GitHub* репозиторий с лабораторной работой

<https://github.com/KulakhmetovS/Lab7>

Приложение А

Листинг программы

Файл main.c

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

int** Creategraph(int **, int);
void DFS(int **, int *, int, int);

int size;

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

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

int main()
{
    int i, j, v, num;
    int **graph = NULL, *visited = NULL, *List = NULL;

    printf("\t# Graphs #\n\n");
    printf("Enter the number of graph vertices (positive integer): ");
    scanf("%d", &size);
    num = size - 1;

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

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

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

```

// <----- ! ----->
DFS(graph, visited, size, v);
// <----- ! ----->
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, v);

free(graph);
free(visited);

return 0;
}

void DFS(int **graph, 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)) DFS(graph, visited, size, i);
    }
}

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;
}

// <----- 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 v)
{
    visited[v] = 1;
    printf("%d ", v);
    struct node* temp = graph->adjLists[v];
    while (temp)
    {
        if(visited[temp->vertex] == 0)
        {
            ListDFS(graph, visited, temp->vertex);
        }
        else
        {
            temp = temp -> next;
        }
    }
}

```