Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

Кафедра программного обеспечения и администрирования информационных систем

Направление подготовки математическое обеспечение и администрирование информационных систем

Форма обучения очная

**Отчет**

**по лабораторной работе №9**

«Реализация базовых алгоритмов на графах.»

Выполнил:

студент группы 213 Файтельсон А.А.

Проверил:

ассистент кафедры ПОиАИС Овсянников А.В.

Курск, 2025

**Цель работы:** Изучить основные алгоритмы теории графов

**Словесная постановка задачи:**

1. Задача заключается в определении всех вершин графа, находящихся на фиксированном расстоянии d от заданной начальной вершины v0​. Граф представлен в виде списка смежности. Расстояние между вершинами определяется как количество ребер в кратчайшем пути между ними. Необходимо разработать алгоритм и реализовать его программно.
2. Для заданных двух вершин графа необходимо найти все существенные вершины относительно этих двух вершин. Существенными называются такие вершины, которые принадлежат хотя бы одному пути, идущему от одной из заданных вершин к другой.
3. Требуется найти все точки сочленения в заданном графе. Точка сочленения — это вершина, удаление которой приводит к увеличению количества компонент связности графа.

**Алгоритм решения задачи в текстуальном виде:**

**1 задача:**

1. Инициализация: Создать пустой список для хранения вершин, находящихся на расстоянии. Создать множество для отслеживания посещенных вершин.
2. Реализация рекурсивного обхода в глубину (DFS): Функция принимает текущую вершину, текущее расстояние и множество посещенных вершин. Если текущее расстояние равно d, добавить вершину в список результатов. Если текущее расстояние больше d, прекратить дальнейший обход из этой вершины. Для каждой смежной вершины, если она еще не посещена, вызвать DFS с увеличенным расстоянием.
3. Запуск DFS: Начать обход из начальной вершины с расстоянием 0.
4. Возврат результата: После завершения обхода вернуть список вершин, находящихся на расстоянии d.

**2 задача:**

1. Инициализация данных : Граф задается в виде списка смежности. Задаются начальная и конечная вершины.
2. Поиск всех путей между вершинами :Используется алгоритм поиска в глубину (DFS) для обхода графа. Для каждой вершины, достижимой из начальной, проверяется, можно ли достичь конечной вершины через нее.
3. Определение существенных вершин : В процессе обхода графа с помощью DFS фиксируются все вершины, которые участвуют в каком-либо пути от начальной до конечной вершины. Эти вершины добавляются в множество существенных вершин.
4. Результат : После завершения обхода графа выводится список всех существенных вершин.

**3 задача:**

1. Инициализация: Запускаем DFS из каждой непосещенной вершины. Для каждой вершины храним время входа (tin) и минимальное время достижимости (low).
2. DFS:Для текущей вершины v устанавливаем tin[v] и low[v] равными текущему времени.Для каждой смежной вершины u: Если u еще не посещена, рекурсивно запускаем DFS из u. После возврата из рекурсии обновляем low[v] как минимум из low[v] и low[u]. Если low[u] >= tin[v] и v не является корнем, то v — точка сочленения. Если v — корень и имеет более одного потомка, то v — точка сочленения. Если u уже посещена и не является родителем v, обновляем low[v] как минимум из low[v] и tin[u].
3. Результат: После завершения DFS все вершины, помеченные как точки сочленения, являются искомыми.

**Обоснование правильности выбора алгоритма**

Для решения задачи был выбран модифицированный алгоритм обхода графа в глубину (DFS), так как он позволяет эффективно исследовать все пути графа, отслеживая текущее расстояние от начальной вершины. DFS особенно подходит для данной задачи, поскольку он естественным образом поддерживает рекурсивное увеличение расстояния при переходе к соседним вершинам. Алгоритм имеет временную сложность O(V+E), где V — количество вершин, а E — количество ребер, что делает его оптимальным для большинства графов.

**Листинг программы**

package main

import "fmt"

type vertexInfo struct {

vertex int

currentDistance int

visited map[int]bool

d int

graph map[int][]int

}

func dfs(info vertexInfo) []int {

if info.currentDistance == info.d {

return []int{info.vertex}

}

if info.currentDistance > info.d {

return nil

}

info.visited[info.vertex] = true

var result []int

for \_, neighbor := range info.graph[info.vertex] {

if !info.visited[neighbor] {

newInfo := vertexInfo{

vertex: neighbor,

currentDistance: info.currentDistance + 1,

visited: info.visited,

d: info.d,

graph: info.graph,

}

res := dfs(newInfo)

result = append(result, res...)

}

}

return result

}

// Функция для поиска существенных вершин

func findEssentialVertices(graph map[int][]int, start, end int) map[int]bool {

visited := make(map[int]bool)

essential := make(map[int]bool)

var dfs func(current int)

dfs = func(current int) {

// Если вершина уже посещена, выходим

if visited[current] {

return

}

// Помечаем вершину как посещенную

visited[current] = true

// Если текущая вершина совпадает с конечной, помечаем её как существенную

if current == end {

essential[current] = true

return

}

// Рекурсивно обходим соседние вершины

for \_, neighbor := range graph[current] {

dfs(neighbor)

// Если соседняя вершина является существенной, то текущая тоже существенная

if essential[neighbor] {

essential[current] = true

}

}

}

// Запускаем DFS из начальной вершины

dfs(start)

return essential

}

func findVerticesAtDistance(graph map[int][]int, startVertex int, d int) []int {

visited := make(map[int]bool)

info := vertexInfo{

vertex: startVertex,

currentDistance: 0,

visited: visited,

d: d,

graph: graph,

}

return dfs(info)

}

func findArticulationPoints(graph map[int][]int, n int) []int {

visited := make([]bool, n)

tin := make([]int, n)

low := make([]int, n)

isArticulation := make([]bool, n)

timer := 0

articulationPoints := []int{}

var dfs func(int, int)

dfs = func(v, p int) {

visited[v] = true

tin[v] = timer

low[v] = timer

timer++

children := 0

for \_, to := range graph[v] {

if to == p {

continue

}

if visited[to] {

low[v] = min(low[v], tin[to])

} else {

dfs(to, v)

low[v] = min(low[v], low[to])

if low[to] >= tin[v] && p != -1 {

isArticulation[v] = true

}

children++

}

}

if p == -1 && children > 1 {

isArticulation[v] = true

}

}

for i := 0; i < n; i++ {

if !visited[i] {

dfs(i, -1)

}

}

for i := 0; i < n; i++ {

if isArticulation[i] {

articulationPoints = append(articulationPoints, i)

}

}

return articulationPoints

}

func min(a, b int) int {

if a < b {

return a

}

return b

}

func main() {

graph := map[int][]int{

0: {1, 2},

1: {0, 3, 4},

2: {0, 5},

3: {1},

4: {1, 6},

5: {2},

6: {4},

}

startVertex := 0

d := 2

verticesAtDistanceD := findVerticesAtDistance(graph, startVertex, d)

fmt.Println(verticesAtDistanceD) // Вывод: [3 4 5]

// Пример графа в виде списка смежности

graph = map[int][]int{

0: {1},

1: {2},

2: {3},

3: {4},

4: {0},

}

start := 0

end := 4

// Находим существенные вершины

essentialVertices := findEssentialVertices(graph, start, end)

// Выводим результат

fmt.Println("Существенные вершины:")

for vertex := range essentialVertices {

fmt.Println(vertex)

}

// Пример графа в виде списка смежности

graph = map[int][]int{

0: {1, 2}, // 0

1: {0, 2, 3}, // 1

2: {0, 1}, // 2

3: {1, 4}, // 3

4: {3}, // 4

}

n := len(graph)

articulationPoints := findArticulationPoints(graph, n)

fmt.Println("Точки сочленения:", articulationPoints)

}