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 №10**

«Деревья»

Выполнил:

студент группы 213 Файтельсон А.А.

Проверил:

ассистент кафедры ПОиАИС Овсянников А.В.

Курск, 2025

**Цель работы:**

1. Изучить принцип хранения множества значений в дереве.

2. Научиться программно реализовывать основные операции.

**Словесная постановка задачи:**

1 задача: Дерево - это связный граф без циклов. Одно из ключевых свойств дерева заключается в том, что если оно содержит n вершин, то количество его ребер всегда равно n−1. Также дерево является связным графом, то есть между любыми двумя вершинами существует путь. Проверьте на нескольких деревьях, что дерево, имеющее n вершин.

2 задача: Задача заключается в построении остового дерева неориентированного графа с использованием алгоритма обхода в глубину (DFS). При этом необходимо определить, какие ребра удаляются из графа в процессе построения остового дерева. Остовное дерево — это подграф, который содержит все вершины исходного графа и является связным ациклическим графом (деревом).

**Алгоритм решения задачи в текстуальном виде:**

**1 задача:**

1. Ввод данных графа: Задать граф в виде списка смежности или матрицы смежности. Для каждого тестового случая определить количество вершин n и список ребер.
2. Проверка количества ребер: Подсчитать количество ребер в графе. Если количество ребер не равно n−1, то граф не является деревом.
3. Проверка связности графа: Использовать алгоритм обхода графа (например, поиск в глубину, DFS) для проверки, что все вершины графа достижимы из одной произвольной вершины. Если хотя бы одна вершина недостижима, граф не является связным.
4. Вывод результата: Если оба условия выполняются, вывести сообщение, что граф является деревом. В противном случае указать, какое условие не выполнено.

**2 задача:**

1. Инициализация: Создаем пустой список для хранения ребер остовного дерева. Создаем пустой список для хранения удаленных ребер. Инициализируем массив visited, который будет отслеживать посещенные вершины.
2. Обход графа в глубину (DFS): Начинаем обход с произвольной вершины (например, с вершины 0).
3. Для каждой вершины просматриваем всех ее соседей: Если соседняя вершина еще не посещена, добавляем ребро между текущей вершиной и соседней в остовное дерево и помечаем соседнюю вершину как посещенную. Если соседняя вершина уже посещена и ребро между текущей вершиной и соседней не входит в остовное дерево, добавляем это ребро в список удаленных ребер.
4. Завершение: После завершения обхода графа выводим список ребер остовного дерева и список удаленных ребер.

**Обоснование правильности выбора алгоритма**

Алгоритм обхода в глубину (DFS) идеально подходит для построения остовного дерева, так как он позволяет эффективно находить связные компоненты графа и строить дерево, начиная с произвольной вершины. В процессе обхода мы можем легко отслеживать, какие ребра добавляются в остовное дерево, а какие ребра образуют циклы и должны быть удалены. Этот подход гарантирует, что результатом будет связный ациклический граф (дерево), содержащий все вершины исходного графа.

**Листинг программы**

package main

// 6, 2

import (

"fmt"

)

// Функция для обхода графа в глубину (DFS)

func dfs(graph map[int][]int, visited map[int]bool, node int) {

visited[node] = true

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

if !visited[neighbor] {

dfs(graph, visited, neighbor)

}

}

}

// Функция для проверки, является ли граф деревом

func isTree(n int, edges [][]int) bool {

// Создаем граф в виде списка смежности

graph := make(map[int][]int)

for \_, edge := range edges {

u, v := edge[0], edge[1]

graph[u] = append(graph[u], v)

graph[v] = append(graph[v], u)

}

// Проверяем количество ребер

if len(edges) != n-1 {

return false

}

// Проверяем связность графа с помощью DFS

visited := make(map[int]bool)

dfs(graph, visited, 1) // Начинаем обход с первой вершины

// Проверяем, все ли вершины были посещены

for i := 1; i <= n; i++ {

if !visited[i] {

return false

}

}

return true

}

// Функция для выполнения DFS и построения остовного дерева

func dfs2(graph map[int][]int, vertex int, visited map[int]bool, spanningTree \*[][]int, removedEdges \*[][]int) {

visited[vertex] = true

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

if !visited[neighbor] {

// Добавляем ребро в остовное дерево

\*spanningTree = append(\*spanningTree, []int{vertex, neighbor})

dfs2(graph, neighbor, visited, spanningTree, removedEdges)

} else {

// Проверяем, не добавлено ли ребро в остовное дерево ранее

isInSpanningTree := false

for \_, edge := range \*spanningTree {

if (edge[0] == vertex && edge[1] == neighbor) || (edge[0] == neighbor && edge[1] == vertex) {

isInSpanningTree = true

break

}

}

if !isInSpanningTree {

// Добавляем ребро в список удаленных ребер

\*removedEdges = append(\*removedEdges, []int{vertex, neighbor})

}

}

}

}

func main() {

// Примеры графов

tests := []struct {

n int

edges [][]int

}{

{4, [][]int{{1, 2}, {1, 3}, {1, 4}}}, // Дерево

{4, [][]int{{1, 2}, {2, 3}}}, // Не дерево (не связный граф)

{5, [][]int{{1, 2}, {2, 3}, {3, 4}, {4, 5}}}, // Дерево

{3, [][]int{{1, 2}, {2, 3}, {1, 3}}}, // Не дерево (цикл)

}

for i, test := range tests {

fmt.Printf("Тест %d: ", i+1)

if isTree(test.n, test.edges) {

fmt.Println("Граф является деревом.")

} else {

fmt.Println("Граф не является деревом.")

}

}

// Пример графа в виде списка смежности

graph := map[int][]int{

0: {1, 2},

1: {0, 2, 3},

2: {0, 1, 3},

3: {1, 2, 4},

4: {3},

}

// Инициализация переменных

visited := make(map[int]bool)

var spanningTree [][]int

var removedEdges [][]int

// Запуск DFS с начальной вершиной 0

dfs2(graph, 0, visited, &spanningTree, &removedEdges)

// Вывод результатов

fmt.Println("Остовное дерево:")

for \_, edge := range spanningTree {

fmt.Printf("%d - %d\n", edge[0], edge[1])

}

fmt.Println("\nУдаленные ребра:")

for \_, edge := range removedEdges {

fmt.Printf("%d - %d\n", edge[0], edge[1])

}

}