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 №8**

«Представление графов с помощью динамических структур данных»

Выполнил:

студент группы 213 Файтельсон А.А.

Проверил:

ассистент кафедры ПОиАИС Овсянников А.В.

Курск, 2025

**Цель работы:** Изучить различные способы представления графа в памяти

компьютера.

**Словесная постановка задачи:**

Необходимо найти все разрезы заданного графа. Разрез определяется как множество ребер, соединяющих вершины из двух непересекающихся подмножеств P1​ и P2​, на которые разбивается множество вершин графа..

**Алгоритм решения задачи в текстуальном виде:**

1. Представить граф в виде списков смежности.
2. Перебрать все возможные разбиения множества вершин на два непустых подмножества P1​ и P2​.
3. Для каждого разбиения:
4. Проверить все ребра графа.
5. Если ребро соединяет вершину из P1​ с вершиной из P2​, добавить его в разрез.
6. Вывести найденные разрезы.

**Обоснование правильности выбора алгоритма**

Алгоритм перебирает все возможные разбиения множества вершин, что гарантирует нахождение всех разрезов. Использование списков смежности позволяет эффективно проверять наличие ребер между вершинами из разных подмножеств.

**Листинг программы**

**package main**

**import (**

**"fmt"**

**)**

**// Graph представляет граф в виде списков смежности**

**type Graph struct {**

**AdjacencyList map[int][]int**

**}**

**// NewGraph создает новый граф**

**func NewGraph() \*Graph {**

**return &Graph{AdjacencyList: make(map[int][]int)}**

**}**

**// AddEdge добавляет ребро в граф**

**func (g \*Graph) AddEdge(u, v int) {**

**g.AdjacencyList[u] = append(g.AdjacencyList[u], v)**

**g.AdjacencyList[v] = append(g.AdjacencyList[v], u)**

**}**

**// FindCuts находит все разрезы графа**

**func (g \*Graph) FindCuts() {**

**vertices := make([]int, 0, len(g.AdjacencyList))**

**for v := range g.AdjacencyList {**

**vertices = append(vertices, v)**

**}**

**n := len(vertices)**

**for mask := 1; mask < (1 << n); mask++ { // Перебираем все возможные разбиения**

**P1, P2 := []int{}, []int{}**

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

**if mask&(1<<i) != 0 {**

**P1 = append(P1, vertices[i])**

**} else {**

**P2 = append(P2, vertices[i])**

**}**

**}**

**// Находим разрез для текущего разбиения**

**cut := make(map[[2]int]bool)**

**for \_, u := range P1 {**

**for \_, v := range g.AdjacencyList[u] {**

**if contains(P2, v) {**

**edge := [2]int{u, v}**

**if edge[0] > edge[1] {**

**edge[0], edge[1] = edge[1], edge[0]**

**}**

**cut[edge] = true**

**}**

**}**

**}**

**// Выводим разрез**

**fmt.Printf("P1: %v, P2: %v -> Cut: ", P1, P2)**

**for edge := range cut {**

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

**}**

**fmt.Println()**

**}**

**}**

**// contains проверяет, содержится ли элемент в массиве**

**func contains(arr []int, val int) bool {**

**for \_, v := range arr {**

**if v == val {**

**return true**

**}**

**}**

**return false**

**}**

**func main() {**

**// Создаем граф**

**graph := NewGraph()**

**graph.AddEdge(1, 2)**

**graph.AddEdge(2, 3)**

**graph.AddEdge(3, 4)**

**graph.AddEdge(4, 1)**

**graph.AddEdge(2, 4)**

**// Находим все разрезы**

**graph.FindCuts()**

**}**