МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе №6

по дисциплине «Структуры и алгоритмы обработки данных»
по теме «основные алгоритмы работы с графами»

Выполнил:
Студент группы ИКБО-13-22 Руденко Алексей Дмитриевич

Проверил: ассистент Муравьёва Е.А.

МОСКВА 2023 г.

Практическая работа № 6

Цель работы

Получение практических навыков по выполнению операций над структурой данных граф.

Задание 1

Ответьте на вопросы и выполните упражнения:

1) Дайте определения понятиям: ориентированный граф, неориентированный граф, взвешенный граф, связный граф, центр графа, диаметр графа, матрица смежности.

Ориентированный граф: Граф, у которого каждое ребро имеет определенное направление. То есть связь между вершинами в таком графе имеет определенное направление, как направление движения по стрелке.

Неориентированный граф: Граф, у которого ребра не имеют направления. В нем связь между вершинами представлена просто наличием ребра, но без определенного направления.

Взвешенный граф: Граф, в котором каждое ребро имеет ассоциированное с ним значение (вес, стоимость, длина и т. д.). Эти значения указывают на некоторую характеристику связи между вершинами.

Связный граф: Граф, в котором существует путь между любой парой вершин. Все вершины графа связаны между собой.

Центр графа: Вершина или несколько вершин графа, расстояние от которых до всех остальных вершин в графе минимально. Центр графа определяется как вершина(ы) с минимальной эксцентриситетом.

Диаметр графа: Максимальная длина среди кратчайших путей между всеми парами вершин в графе. Это самое длинное из всех кратчайших расстояний в графе.

Матрица смежности: Матрица, которая представляет собой квадратную матрицу, используемую для представления связей в графе. В неориентированном графе матрица смежности симметрична, а в

ориентированном графе может быть несимметричной. Элементы матрицы указывают на наличие (или отсутствие) ребра между вершинами, а при взвешенных графах значения элементов отражают веса соответствующих ребер.

2) Что такое остовное дерево графа?

Остовное дерево графа — это подграф, который содержит все вершины из исходного графа, но является деревом, то есть не содержит циклов и при этом остается связным. Остовное дерево охватывает все вершины исходного графа, обеспечивая связность между ними, но при этом не содержит лишних ребер для образования циклов.

Для графа может существовать несколько остовных деревьев, но **минимальное остовное дерево** обладает минимальной суммой весов всех его ребер (в случае взвешенных графов), то есть оно является остовным деревом с минимальным общим весом ребер.

3) Какое количество ребер в остовном графе?

(по сути ответил в вопросе выше)

Количество ребер в остовном графе зависит от количества вершин и связности исходного графа.

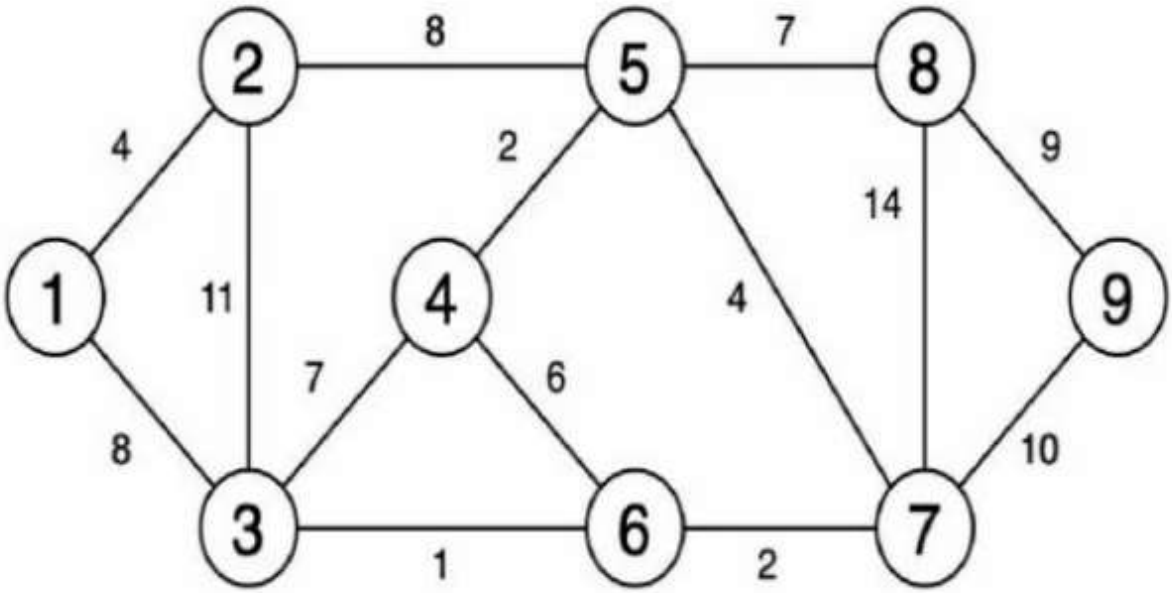
Для **связного неориентированного** графа с **n** вершинами минимальное количество ребер в остовном дереве равно **n – 1**. Это свойство верно для простых неориентированных графов без петель или кратных ребер.

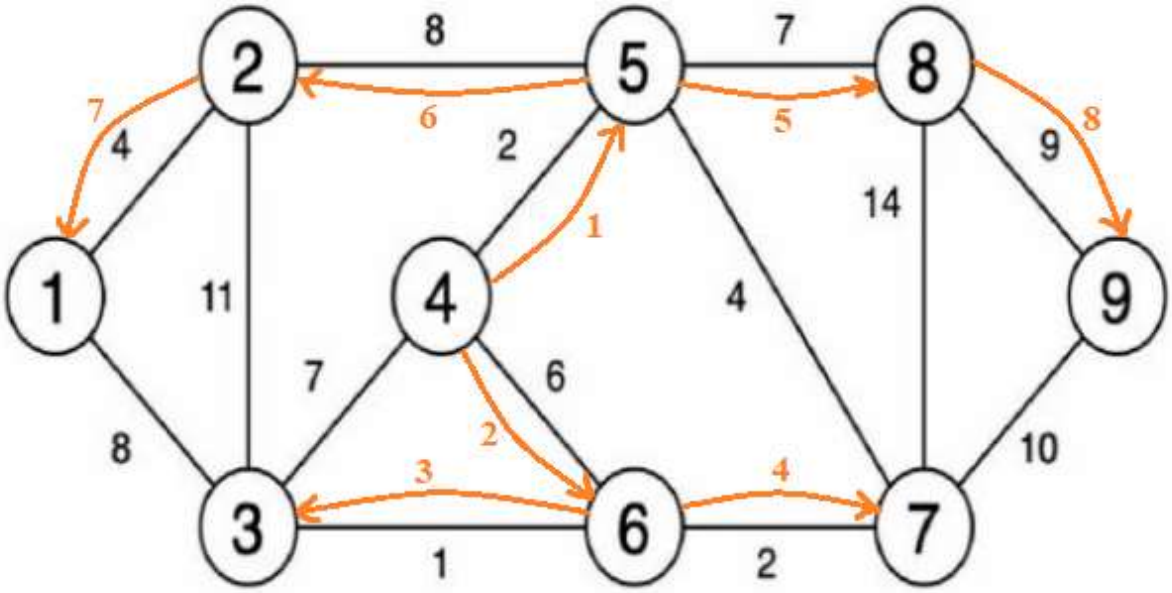
Для **ориентированных** графов с n вершинами количество ребер в остовном дереве может быть больше, так как оно зависит от структуры и направлений ребер в графе. Ориентированные графы также могут иметь минимальное остовное дерево с **n – 1** ребром, если они удовлетворяют условиям, например, если граф является ациклическим.

4) Постройте остовное дерево, используя алгоритм Прима. Стартовая вершина – 4.

Суть алгоритма в том, что мы добавляем вершины при помощи

минимально доступного ребра (список рёбер составляет из всех уже добавленных вершин).



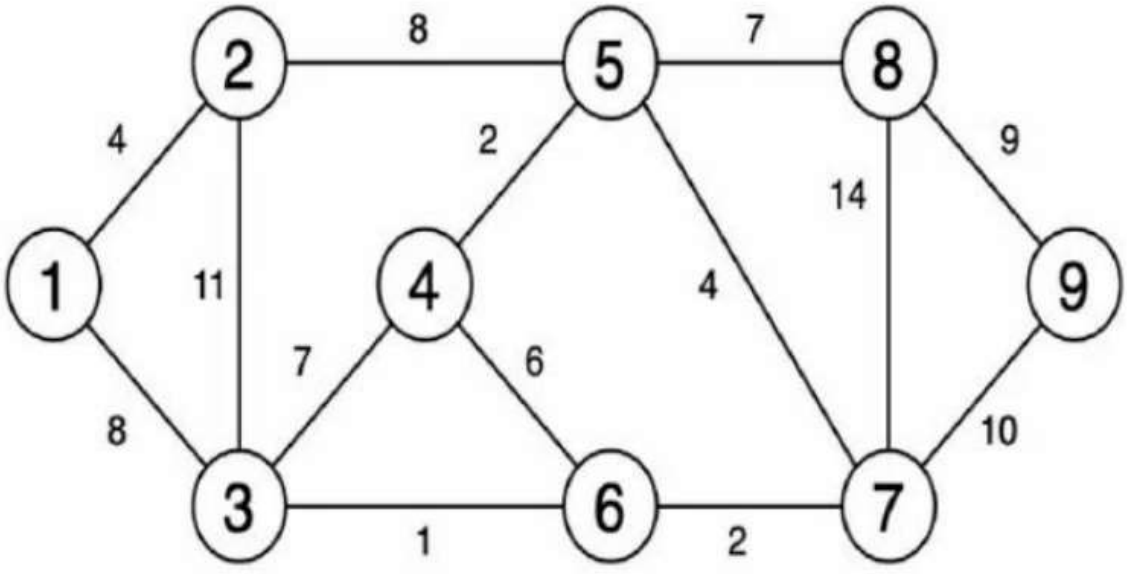


5) Что такое кратчайший путь в графе?

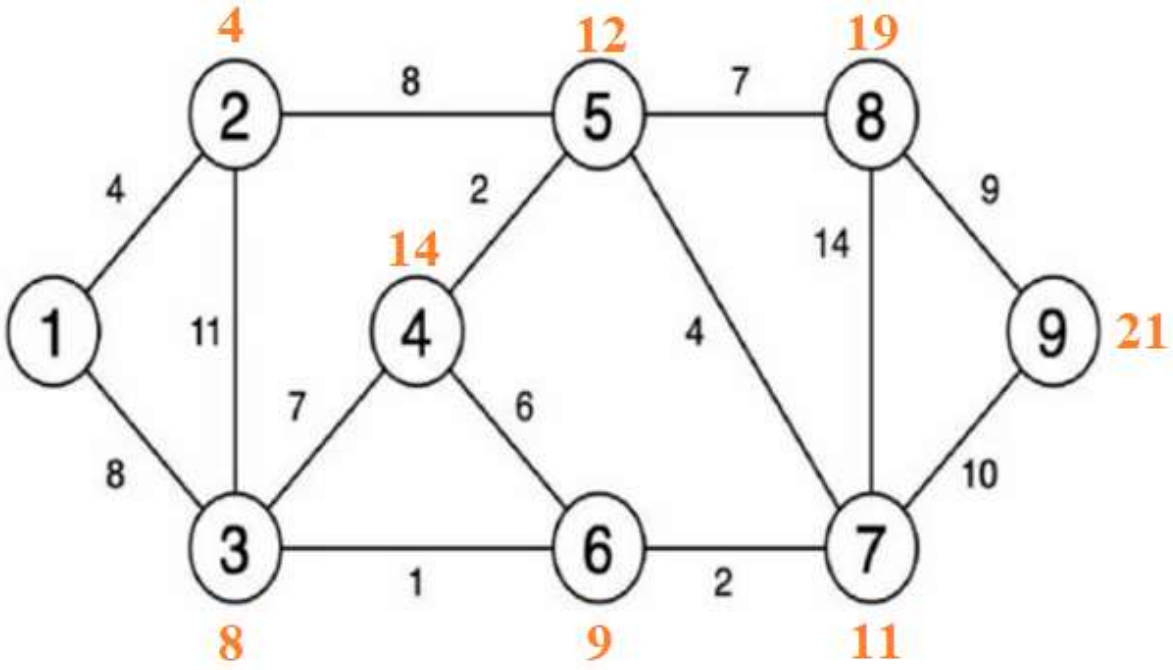
Кратчайший путь в графе — это путь между двумя вершинами, такой, что сумма весов рёбер на этом пути минимальна. Вес ребра может представлять собой физическое расстояние, стоимость, время и т. д., в зависимости от контекста задачи.

7) Найдите кратчайший путь от вершины 1 до вершины 9, используя

алгоритм Дейкстры.



Кратчайший путь от 1 до 9: **1-->3-->6-->7-->9**. Минимальное расстояние: **21**



7) В чем отличие алгоритма Дейкстры от алгоритма Флойда-Уоршалла. Какова вычислительная сложность каждого алгоритма по времени и памяти.

Алгоритм Дейкстры применяется для нахождения кратчайшего пути от одной вершины до всех остальных в графе с неотрицательными весами.

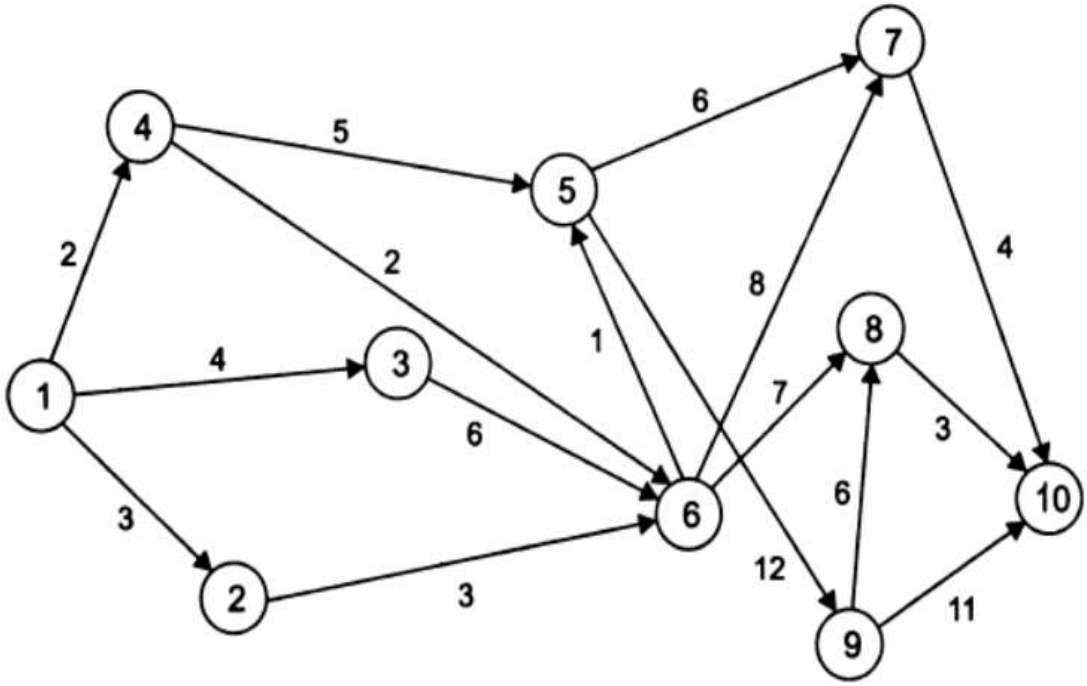
Алгоритм Флойда-Уоршалла применяется для нахождения кратчайших

путей между всеми парами вершин в графе и работает как для положительных, так и отрицательных весов ребер, но не обнаруживает отрицательные циклы.

Вычислительная сложность:

- Алгоритм Дейкстры – по времени: **O((V+E)logV)**, где V — количество вершин, E — количество рёбер. Если использовать очередь с приоритетами на базе кучи, то асимптотика улучшится. По памяти: O(V + E)
- Алгоритм Флойда-Уоршала: **O(V^3)**, где V — количество вершин. Сложность Флойда-Уоршала высока из-за использования **тройного вложенного цикла** (реализация цикла в задании) для обработки всех вершин графа. Память: O(V^2) (хранение матрицы расстояний).

8) Обойдите граф, используя метод поиска а) в ширину; б) в глубину. Стартовая вершина – 1.



a) Порядок обхода **в ширину**: 1-->2-->3-->4-->6-->5-->7-->8-->9-->10

b) Порядок обхода **в глубину**: 1-->2-->6-->5-->7-->10-->9-->8-->3-->4

Задание 2

Разработать класс «Граф», обеспечивающий хранение и работу со структурой данных «граф», в соответствии с вариантом индивидуального

Задания:

Реализовать метод ввода графа с клавиатуры, наполнение графа осуществлять с помощью метода добавления одного ребра.

Реализовать методы, выполняющие задачи, определенные вариантом индивидуального задания.

Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

Разработать программу, демонстрирующую работу всех методов класса.

Произвести тестирование программы на одном из графов, предложенных в таблице.

Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Решение:

Вариант 26

26	Список смежных вершин	Вывести все цепочки в графе, используя метод поиска в ширину. Составить программу нахождения кратчайших путей в графе заданным методом «Естественное слияние». Вывести пути, полученные методом.

Рисунок 1 – условие варианта

Листинг 1 – Реализация задания 2

```
#include <iostream>
#include <list>
#include <queue>
#include <vector>

struct Edge {
    int destination;
    int weight;

    Edge(int dest, int w) : destination(dest), weight(w) {}
};

class Graph {
private:
    int vertices;
    bool isDirected;
    std::vector<std::list<Edge>> adjacencyList;

public:
    Graph(int v, bool directed = false) : vertices(v), isDirected(directed),
adjacencyList(v) {}

    void addEdge(int u, int v, int weight) {
        if (u >= 1 && u <= vertices && v >= 1 && v <= vertices) {
            adjacencyList[u - 1].push_back(Edge(v, weight));
            if (!isDirected) {
                adjacencyList[v - 1].push_back(Edge(u, weight));
            }
        }
        else {
            std::cout << "Неверные индексы вершин!" << std::endl;
        }
    }

    void inputGraph() {
        int edges;
        std::cout << "Введите количество ребер: ";
        std::cin >> edges;

        for (int i = 0; i < edges; ++i) {
            int u, v, weight;
            std::cout << "Введите вершины u, v и вес: ";
            std::cin >> u >> v >> weight;
            addEdge(u, v, weight);
        }
    }

    void printChains() {
        for (int i = 1; i <= vertices; ++i) {
            std::cout << "Цепи, начинающиеся из вершины " << i << ": ";
            BFS(i);
            std::cout << std::endl;
        }
    }

    void BFS(int start) {
        if (start < 1 || start > vertices) {
            std::cout << "Неверная начальная вершина!" << std::endl;
            return;
        }

        std::vector<bool> visited(vertices, false);
        std::vector<int> distance(vertices, INT_MAX);
```

```
        std::queue<int> queue;

        queue.push(start);
        visited[start - 1] = true;
        distance[start - 1] = 0;

        while (!queue.empty()) {
            int current = queue.front();
            queue.pop();

            std::cout << current << " ";

            for (const Edge& edge : adjacencyList[current - 1]) {
                if (!visited[edge.destination - 1]) {
                    visited[edge.destination - 1] = true;
                    distance[edge.destination - 1] = distance[current - 1] +
edge.weight;
                    queue.push(edge.destination);
                }
            }
        }

        std::cout << "\nВесы ребер от вершины " << start << ": ";
        for (int i = 0; i < vertices; ++i) {
            if (distance[i] == INT_MAX) {
                std::cout << "-1 ";
            }
            else {
                std::cout << distance[i] << " ";
            }
        }
        std::cout << std::endl;
    }

    void naturalMerge() {
        for (int i = 1; i <= vertices; ++i) {
            std::cout << "Кратчайшие пути из вершины " << i << ":" << std::endl;
            BFS(i);
            std::cout << std::endl;
        }
    }

    void printResultingTree() {
        std::cout << "\nВывод дерева:\n";

        for (int i = 1; i <= vertices; ++i) {
            std::cout << i << ": ";
            for (const Edge& edge : adjacencyList[i - 1]) {
                if (edge.destination >= 1 && edge.destination <= vertices) {
                    std::cout << "(" << edge.destination << ", " << edge.weight
<< ") ";
                }
                else {
                    std::cout << "(Неверный пункт назначения, " << edge.weight <<
") ";
                }
            }
            std::cout << std::endl;
        }
    }
};

int main()
{
    setlocale(LC_ALL, "Rus");
```

```
    bool isDirected;
    std::cout << "Направленный ли граф? (Введите 1 если направленный, 0 если
двунаправленный): ";
    std::cin >> isDirected;

    int vertices;
    std::cout << "Введите количество вершин: ";
    std::cin >> vertices;

    Graph graph(vertices, isDirected);

    graph.inputGraph();

    std::cout << "Введите количество ребер: " << std::endl;
    graph.printChains();

    std::cout << "\nКратчайшие пути с использованием естественного слияния:" <<
std::endl;
    graph.naturalMerge();

    graph.printResultingTree();

    return 0;
}
```

Тестирование программы

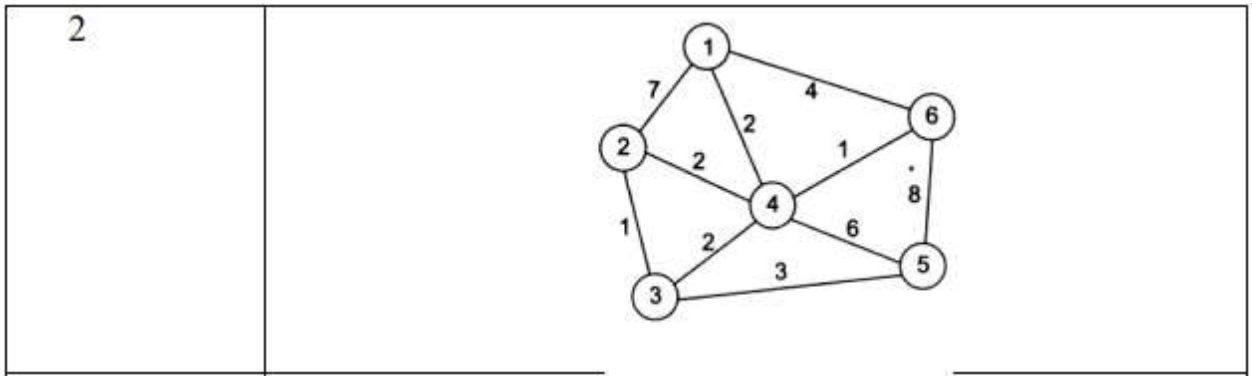


Рисунок 2 – Тестирование на графе (неориентированном)

```
Направленный ли граф? (Введите 1 если направленный, 0 если двунаправленный): 0
Введите количество вершин: 6
Введите количество ребер: 10
Введите вершины u, v и вес: 2 1 7
Введите вершины u, v и вес: 1 6 4
Введите вершины u, v и вес: 1 4 2
Введите вершины u, v и вес: 2 3 1
Введите вершины u, v и вес: 2 4 2
Введите вершины u, v и вес: 3 5 3
Введите вершины u, v и вес: 4 3 2
Введите вершины u, v и вес: 4 6 1
Введите вершины u, v и вес: 6 5 8
Введите вершины u, v и вес: 4 5 6
Введите количество ребер:
Цепи, начинающиеся из вершины 1: 1 2 6 4 3 5
Весы ребер от вершины 1: 0 7 8 2 12 4

Цепи, начинающиеся из вершины 2: 2 1 3 4 6 5
Весы ребер от вершины 2: 7 0 1 2 4 11

Цепи, начинающиеся из вершины 3: 3 2 5 4 1 6
Весы ребер от вершины 3: 8 1 0 2 3 11

Цепи, начинающиеся из вершины 4: 4 1 2 3 6 5
Весы ребер от вершины 4: 2 2 2 0 6 1

Цепи, начинающиеся из вершины 5: 5 3 6 4 2 1
Весы ребер от вершины 5: 12 4 3 6 0 8

Цепи, начинающиеся из вершины 6: 6 1 4 5 2 3
Весы ребер от вершины 6: 4 11 3 1 8 0


Кратчайшие пути с использованием естественного слияния:
Кратчайшие пути из вершины 1:
1 2 6 4 3 5
Весы ребер от вершины 1: 0 7 8 2 12 4

Кратчайшие пути из вершины 2:
2 1 3 4 6 5
Весы ребер от вершины 2: 7 0 1 2 4 11

Кратчайшие пути из вершины 3:
3 2 5 4 1 6
Весы ребер от вершины 3: 8 1 0 2 3 11

Кратчайшие пути из вершины 4:
4 1 2 3 6 5
Весы ребер от вершины 4: 2 2 2 0 6 1

Кратчайшие пути из вершины 5:
5 3 6 4 2 1
Весы ребер от вершины 5: 12 4 3 6 0 8

Кратчайшие пути из вершины 6:
6 1 4 5 2 3
Весы ребер от вершины 6: 4 11 3 1 8 0
```

Рисунок 3 – Тестирование метода естественного слияния на двунаправленном графе

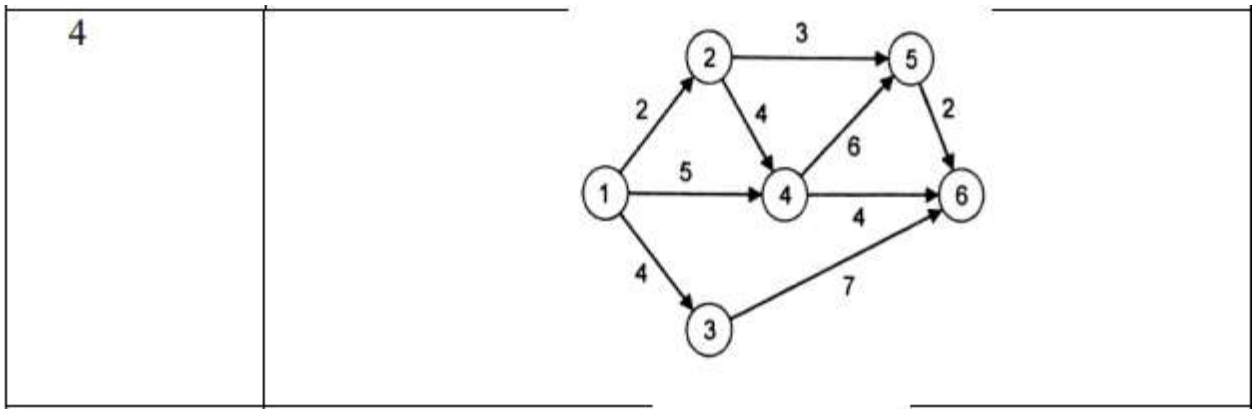


Рисунок 4 – Тестирование на графе (ориентированном)

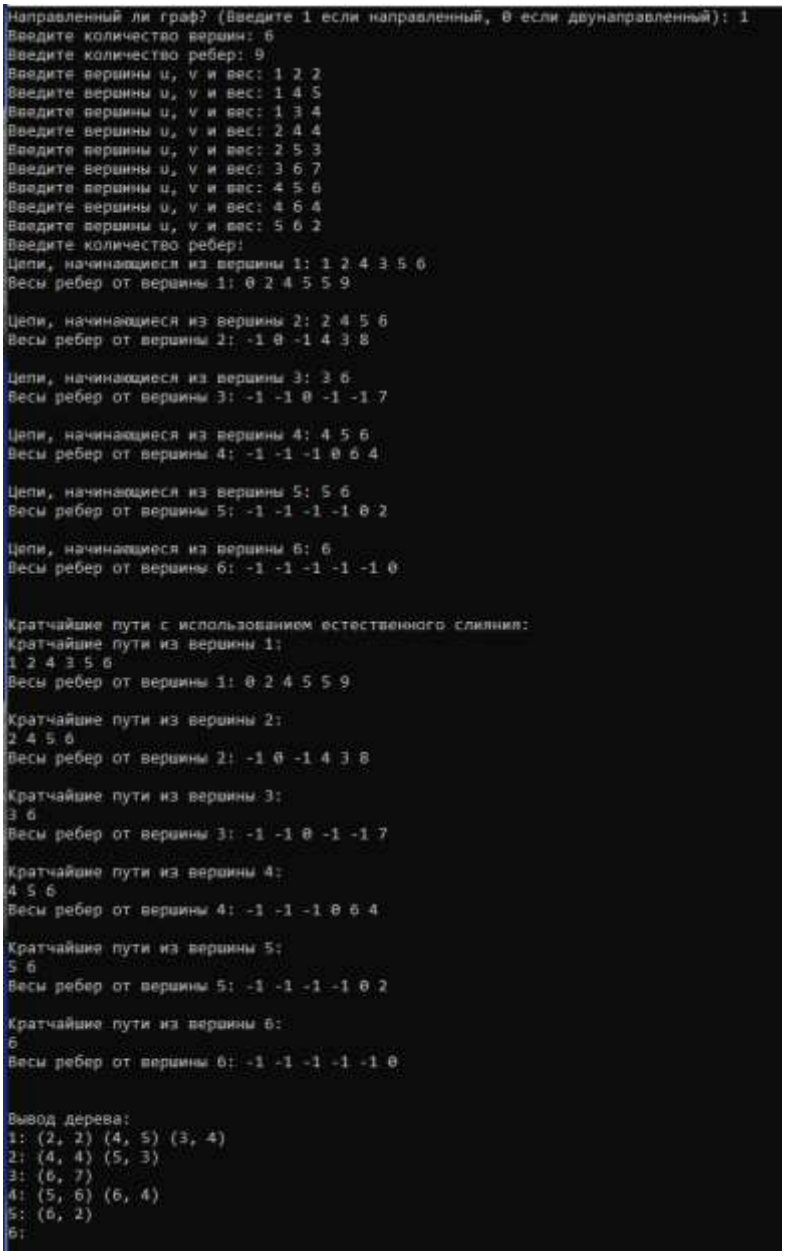


Рисунок 5 - Тестирование метода естественного слияния на направленном графе

Вывод

При выполнении задач ознакомились с графами как структурой для хранения связей и работой с ними. Графы являются важной структурой данных и имеют широкое применение в компьютерной науке и информационных технологиях. При помощи графов можно изображать множество разных систем связей от транспортировки товаров до графического представления работы циклических программ.

Главное преимущество графов - удобство поиска связей между объектами. При таком изображении связей можно использовать множество алгоритмов, которые позволяют оптимизировать пути от одной точки к другой, оптимизировать всю путевую систему или построить самые оптимизированные маршруты.

Обобщая результаты, графы можно считать, одним из самых удобных способов изображения систем взаимодействий объектов, которые можно использовать в различных сферах аналитики данных.