**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра автоматизованих систем обробки інформації**

**і управління**

**Звіт**

з лабораторної роботи № 1 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

**«Прикладні задачі теорії графів ч.1»**

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2021

Зміст

1. Мета лабораторної роботи 3
2. ЗаВдання 4
3. Виконання 8
   1. Псевдокод алгоритму 8
   2. Програмна реалізація алгоритму 8
      1. Приклад роботи 10
   3. Розв’язання задачі вручну 10

Висновок 12

Критерії оцінювання 13

# Мета лабораторної роботи

Мета роботи – вивчити основні прикладні алгоритми на графах та способи їх імплементації.

# ЗаВдання

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм задачі на графах за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму на будь-якій мові програмування для довільного графа, передбачити введення розмірності графа та введення даних графа вручну чи випадковим чином.

Для самостійно обраного графа (розмірності не менше 9 вершин) розв’язати задану за варіантом задачу вручну.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти програмне та ручне розв’язання задачі.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Задача** | **Алгоритм** | **Тип графу** | **Спосіб задання графу** |
| 1 | Обхід графу | DFS | Неорієнтований | Матриця суміжності |
| 2 | Обхід графу | BFS | Неорієнтований | Матриця суміжності |
| 3 | Пошук маршруту у графі | Террі | Неорієнтований | Матриця суміжності |
| 4 | Пошук відстані між вершинами графа | Хвильовий | Неорієнтований | Матриця суміжності |
| 5 | Пошук найкоротшого шляху між парою вершин | Дейкстри | Орієнтований | Матриця вагів |
| 6 | Пошук найкоротшого шляху між парою вершин | Беллмана-Форда | Орієнтований | Матриця вагів |
| 7 | Побудова мінімальних покриваючих дерев | Прима | Неорієнтований | Матриця вагів |
| 8 | Побудова мінімальних покриваючих дерев | Крускала | Неорієнтований | Матриця вагів |
| 9 | Побудова мінімальних покриваючих дерев | Борувки | Неорієнтований | Матриця вагів |
| 10 | Побудова Ейлерового циклу | За означенням | Неорієнтований | Матриця суміжності |
| 11 | Побудова Ейлерового циклу | Флері | Неорієнтований | Матриця суміжності |
| 12 | Побудова Гамільтонового циклу | Пошук із поверненнями | Неорієнтований | Матриця суміжності |
| 13 | Обхід графу | DFS | Неорієнтований | Матриця інцидентності |
| 14 | Обхід графу | BFS | Неорієнтований | Матриця інцидентності |
| 15 | Пошук маршруту у графі | Террі | Неорієнтований | Матриця інцидентності |
| 16 | Пошук відстані між вершинами графа | Хвильовий | Неорієнтований | Матриця інцидентності |
| 17 | Пошук найкоротшого шляху між парою вершин | Дейкстри | Орієнтований | Матриця вагів |
| 18 | Пошук найкоротшого шляху між парою вершин | Беллмана-Форда | Орієнтований | Матриця вагів |
| 19 | Побудова мінімальних покриваючих дерев | Прима | Неорієнтований | Матриця вагів |
| 20 | Побудова мінімальних покриваючих дерев | Крускала | Неорієнтований | Матриця вагів |
| 21 | Побудова мінімальних покриваючих дерев | Борувки | Неорієнтований | Матриця вагів |
| 22 | Побудова Ейлерового циклу | За означенням | Неорієнтований | Матриця інцидентності |
| 23 | Побудова Ейлерового циклу | Флері | Неорієнтований | Матриця інцидентності |
| 24 | Побудова Гамільтонового циклу | Пошук із поверненнями | Неорієнтований | Матриця інцидентності |
| 25 | Обхід графу | DFS | Неорієнтований | Матриця суміжності |
| 26 | Обхід графу | BFS | Неорієнтований | Матриця суміжності |
| 27 | Пошук маршруту у графі | Террі | Неорієнтований | Матриця суміжності |
| 28 | Пошук відстані між вершинами графа | Хвильовий | Неорієнтований | Матриця суміжності |
| 29 | Пошук найкоротшого шляху між парою вершин | Дейкстри | Орієнтований | Матриця вагів |
| 30 | Пошук найкоротшого шляху між парою вершин | Беллмана-Форда | Орієнтований | Матриця вагів |
| 31 | Побудова мінімальних покриваючих дерев | Прима | Неорієнтований | Матриця вагів |
| 32 | Побудова мінімальних покриваючих дерев | Крускала | Неорієнтований | Матриця вагів |
| 33 | Побудова мінімальних покриваючих дерев | Борувки | Неорієнтований | Матриця вагів |
| 34 | Побудова Ейлерового циклу | За означенням | Неорієнтований | Матриця суміжності |
| 35 | Побудова Ейлерового циклу | Флері | Неорієнтований | Матриця суміжності |
| 36 | Побудова Гамільтонового циклу | Пошук із поверненнями | Неорієнтований | Матриця суміжності |

# Виконання

## Псевдокод алгоритмуСнимок экрана 2021-05-06 в 10.56.00.png

## Програмна реалізація алгоритму

**import** java.util.\*;

**public** **class** PrimHeap {

**public** **static** **long** mst(**List**<Edge>[] edges, **int**[] pred) {

**int** n = edges.length;

**Arrays**.fill(pred, -1);

**boolean**[] vis = **new** **boolean**[n];

**int**[] prio = **new** **int**[n];

**Arrays**.fill(prio, **Integer**.MAX\_VALUE);

    prio[0] = 1;

**Queue**<QItem> q = **new** **PriorityQueue**<QItem>();

    q.add(**new** QItem(0, 0));

**long** res = 0;

**while** (!q.isEmpty()) {

      QItem cur = q.poll();

**int** u = cur.u;

**if** (vis[u])

**continue**;

      vis[u] = **true**;

      res += cur.prio;

**for** (Edge e : edges[u]) {

**int** v = e.t;

**if** (!vis[v] && prio[v]>e.cost)

        {

          prio[v] = e.cost;

          pred[v] = u;

          q.add(**new** QItem(prio[v], v));

        }

      }

    }

**return** res;

  }

**static** **class** Edge {

**int** t, cost;

**public** Edge(**int** t, **int** cost) {

**this**.t = t;

**this**.cost = cost;

    }

  }

**static** **class** QItem **implements** **Comparable**<QItem> {

**int** prio;

**int** u;

**public** QItem(**int** prio, **int** u) {

**this**.prio = prio;

**this**.u = u;

    }

**public** **int** compareTo(QItem q) {

**return** prio < q.prio ? -1 : prio > q.prio ? 1 : 0;

    }

  }

*// Usage example*

**public** **static** **void** main(**String**[] args) {

**int**[][] cost = { { 0, 2, 3 }, { 3 , 0, 2 }, { 2, 3, 0 } };

**int** n = cost.length;

**List**<Edge>[] edges = **new** **List**[n];

**for** (**int** i = 0; i < n; i++) {

      edges[i] = **new** **ArrayList**<Edge>();

**for** (**int** j = 0; j < n; j++) {

**if** (cost[i][j] != 0) {

          edges[i].add(**new** Edge(j, cost[i][j]));

        }      }    }

**long** res = mst(edges, **new** **int**[n]);System.out.println("**\n**");

**System**.out.println("**\n**Имеет минимальное остовное дерево массой: ");

**System**.out.println(res);

  }

}

### Приклад роботи

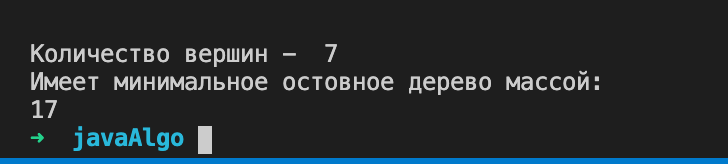
На рисунках 3.1 і 3.2 показані приклади роботи програми для графів на 7 і 15 вершин відповідно.

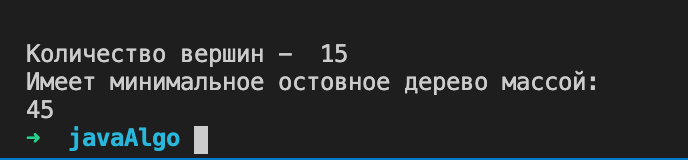
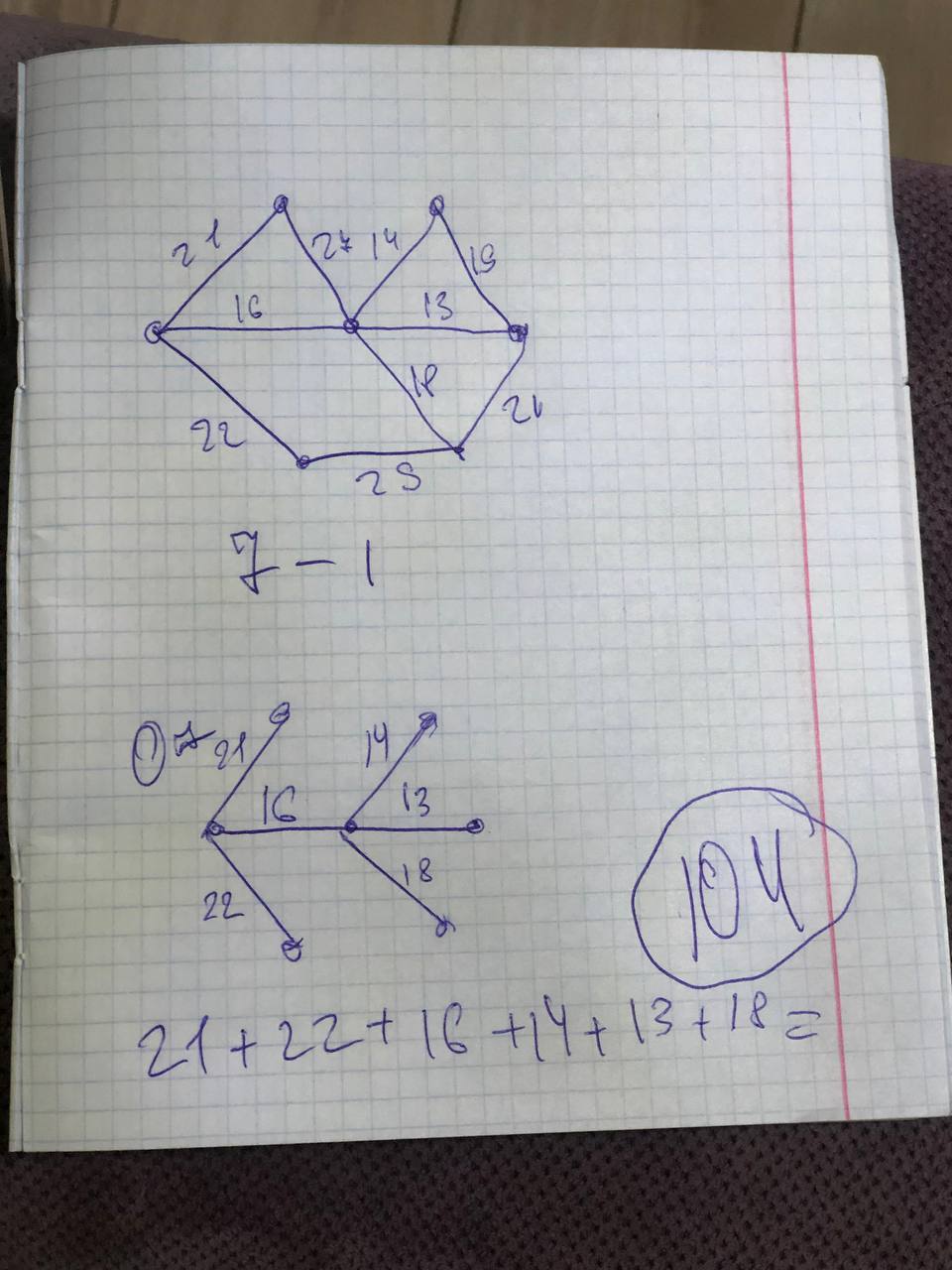
Рисунок 3.1

Рисунок 3.2 –

## Розв’язання задачі вручну

На рисунку 3.3 наведено розв’язання задачі … вручну.

Висновок

При виконанні даної лабораторної роботи я ознакомился с алгоритмом Прима. Алгоритм Прима - это алгоритм минимального остовного дерева, что принимает граф в качестве входных данных и находит подмножество ребер этого графа, который формирует дерево, включающее в себя каждую вершину, а также имеет минимальную сумму весов среди всех деревьев, которые могут быть сформированы из графа.

Как работает алгоритм Прима

Он подпадает под класс алгоритмов, называемых *«жадными» алгоритмами*, которые находят локальный оптимум в надежде найти глобальный оптимум.

Мы начинаем с одной вершины и продолжаем добавлять ребра с наименьшим весом, пока не достигнем нашей цели.

Шаги для реализации **алгоритма Прима**следующие:

1. Инициализируйте минимальное остовное дерево с произвольно выбранной вершиной.
2. Найдите все ребра, которые соединяют дерево с новыми вершинами, найдите минимум и добавьте его в дерево.
3. Продолжайте повторять шаг 2, пока не получите минимальное остовное дерево.

Критерії оцінювання

У випадку здачі лабораторної роботи до 15.03.2021 включно максимальний бал дорівнює – 5. Після 15.03.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 10%;
* програмна реалізація алгоритму – 50%;
* розв’язання задачі вручну – 20%;
* відповідь на 3 теоретичні питання по темі роботи 15%
* висновок – 5%.