МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Институт информационных технологий

Кафедра информационных технологий и экономической информатики

ОТЧЕТ

по лабораторной работе №5

Авторы отчета С.М. Панов ПрИ-201

подпись инициалы, фамилия группа

А.Д. Казбеков ПрИ-201

подпись инициалы, фамилия группа

В.И. Кочетков ПрИ-201

подпись инициалы, фамилия группа

Отчет защищен \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

дата оценка

Челябинск 2024 г.

**Цель работы:** понять и реализовать алгоритмы работы с графами, а так же разработать редактор графов.

**Задание.**

1. **Редактор графов.** Реализация основной части.
2. **Работа транспортной сети.** Визуализация алгоритма.
3. **Построение остовного дерева.** Визуализация алгоритма.
4. **Поиск кратчайшего пути между вершинами.** Визуализация.

Стек технологий:

1. Javascript
2. HTML
3. CSS

**Задание 1**

Разработать полный редактор графов. Обязательно реализовать визуальную часть добавления/удаления/изменения узлов и ребер графа. Реализовать операции сохранения/загрузки графа в файл.

Задана матрица смежности или инцидентности (по выбору) (например, в csv файле или другие, главное иметь возможность посмотреть).

Реализовать программу, которая демонстрирует работу обхода взвешенного графа в ширину и глубину.

Показать по шагам процесс обхода графа (реализовать визуальную демонстрацию совместно с текстовым описанием шагов).

Для реализации данного задания была использована библиотека Vis.js для работы с графами.

Основные параметры для графа. (см. рис. 1.1)

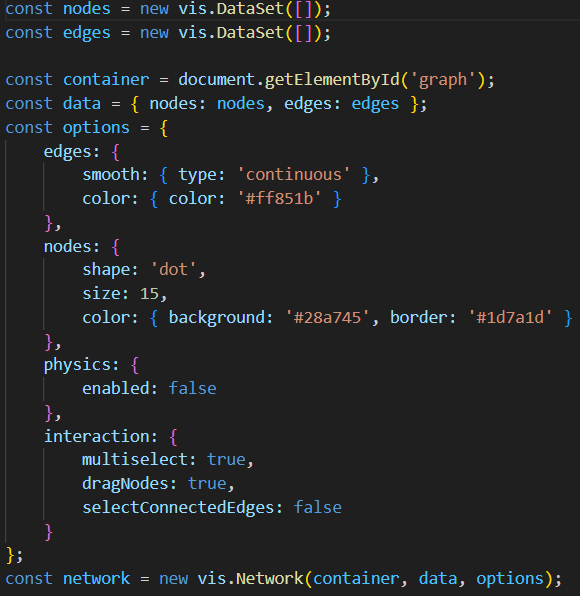


Рис. 1.1 Основные параметры для графа.

Метод для создания узлов(nodes) и ребер(edges), работает путем того, что в созданный DataSet помещаются id узла, с кем связан. (см. рис. 1.2-1.3)

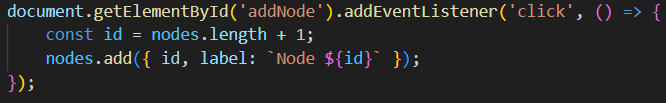


Рис. 1.2 Создание узла(node).

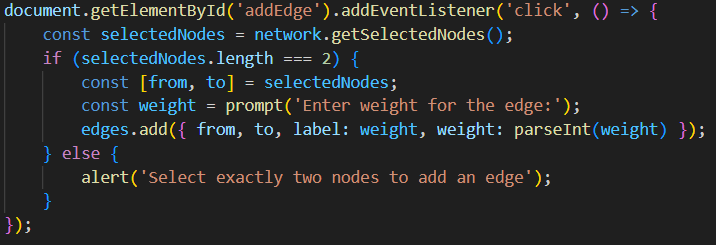


Рис. 1.3 Создание грани, где указывается откуда – куда, вес.

Далее описан метод обхода дерева в глубину и ширину. (см. рис. 1.4-1.5).

Суть работы dfs в том, что мы двигаемся от начальной вершины, все глубже и глубже, пока не дойдем до заданной точки, если мы не находим ее, то возвращаемся в исходную вершину и начинаем спуск уже по другой ветви дерева.



Рис. 1.4 Метод обхода графа в глубину.

Суть работы bfs в том, что мы посещаем каждого соседа поочередно, пока не найдем заданную вершину. Т.е., идя от заданной вершины мы поочередно посещаем каждого соседа, используя связный список и концепцию первым вошел первым вышел. Сначала посещая ближайшую вершину, затем ее соседей продолжая это до тех пор, пока очередь не закончится или не найдем заданную вершину.



Рис. 1.5 Метод обхода графа в ширину.

**Задание 2**

Реализовать программу, которая демонстрирует работу транспортной сети, и реализует алгоритм поиска максимального потока через транспортную сеть.

Обязательно реализовать визуальную часть добавления/удаления/изменения узлов и ребер транспортной сети. Реализовать операции сохранения/загрузки транспортной сети в файл.

Показать по шагам процесс поиска максимального потока через транспортную сеть (реализовать визуальную демонстрацию совместно с текстовым описанием шагов).

Для реализации использовался алгоритм Диница, сейчас будет описана его суть “на пальцах”.

1. "Рисуем карту" по уровням: Мы не смотрим на всю систему труб сразу. Мы как будто рисуем карту, где каждая труба показывает, сколько еще воды она может пропустить, и все трубы выстроены в "лесенку" по расстоянию от источника. То есть, сначала трубы, которые прямо от источника, потом те, которые от них, и так далее.
2. "Запускаем максимум воды": Мы пытаемся пропустить по этой "лесенке" как можно больше воды сразу, пока хотя бы одна труба не будет заполнена до предела. Когда хотя бы одна труба "забивается", мы понимаем, что больше воды по этой "лесенке" не пропустить.
3. "Убираем заполненные трубы": как только труба заполнилась, мы ее как будто убираем из нашей "лесенки", так как она больше не пропустит воды.
4. "Рисуем новую карту": после того, как мы "запустили" и "убрали" трубы в первом круге, мы опять "рисуем карту", но уже с учетом того, сколько воды прошло и какие трубы заполнились.
5. Повторяем, пока не наполнится бассейн: Мы продолжаем рисовать "лесенки", запускать воду, убирать заполненные трубы, пока не увидим, что воду больше нельзя перекачать, потому что до бассейна больше не доходит.

Ниже описан код алгоритма Диница. (см. рис. 2.1)

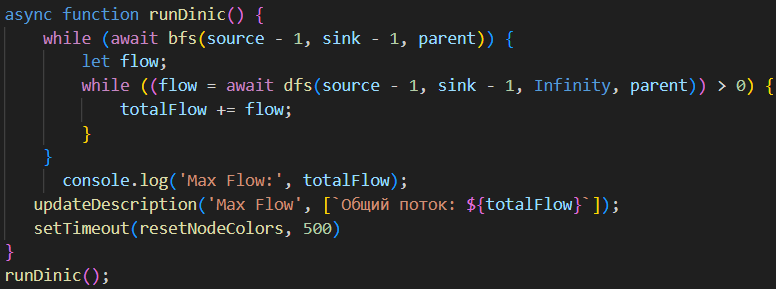


Рис. 2.1 Реализация алгоритма Диница.

Ниже представлена реализация визуализации решения алгоритма Диница. (см. рис. 2.2)

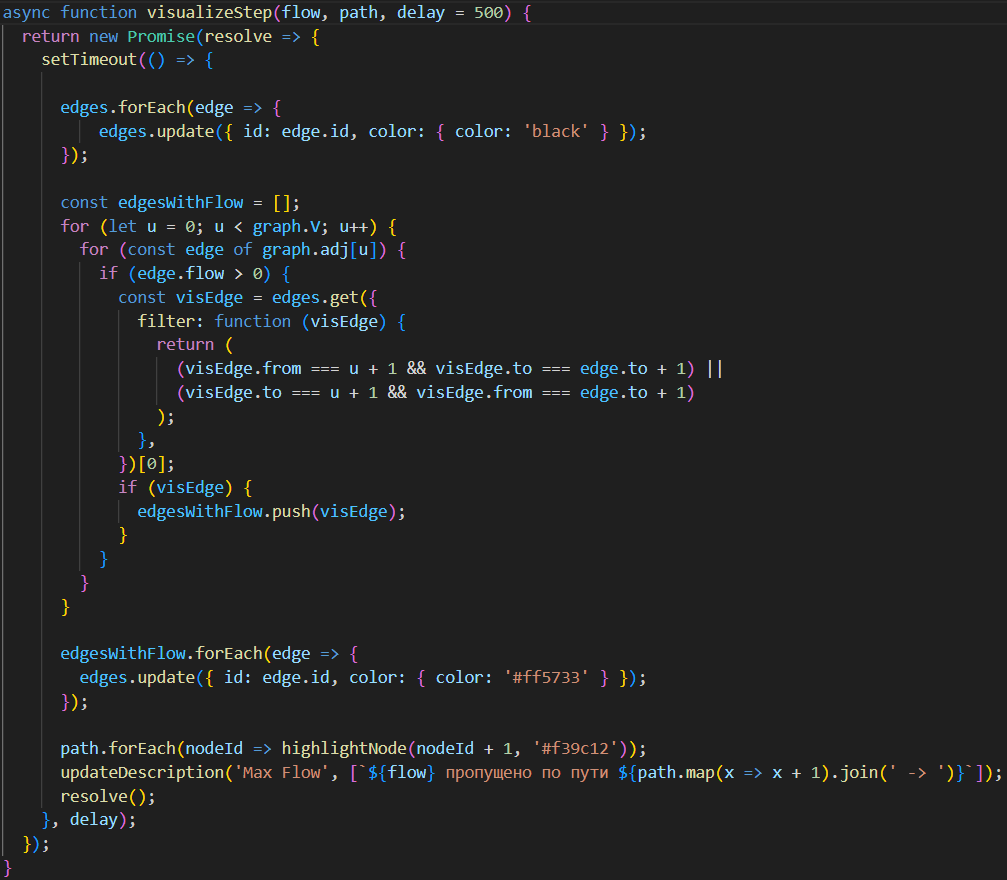


Рис. 2.2 Визуализация алгоритма Диница с максимальным потоком.

**Задание 3**

Реализовать программу, которая демонстрирует работу алгоритма построения минимального остовного дерева.

Обязательно реализовать визуальную часть добавления/удаления/изменения узлов и ребер транспортной сети. Реализовать операции сохранения/загрузки графа и остовного дерева в файл.

Метод для создания графа, т.е. граф из библиотеки Vis преобразуется в граф класса. (см. рис. 3.1)

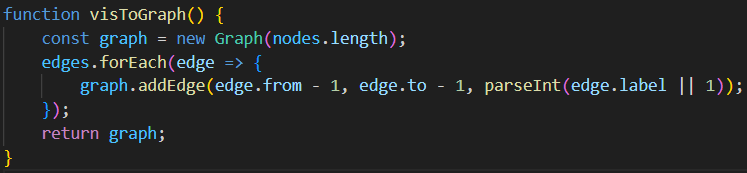


Рис. 3.1 Метод преобразования графа.

Метод для создания остовного дерева. (см. рис. 3.2). Остовное дерево – это дерево, которое получается из сходного с минимальным количеством ребер, все циклы удаляются, все вершины соединены. Суть работы метода Прима: суть алгоритма Прима заключается в постепенном построении минимального остовного дерева (MST), начиная с произвольной вершины и жадно добавляя ребра с минимальным весом, которые связывают уже построенную часть дерева с оставшимися вершинами.

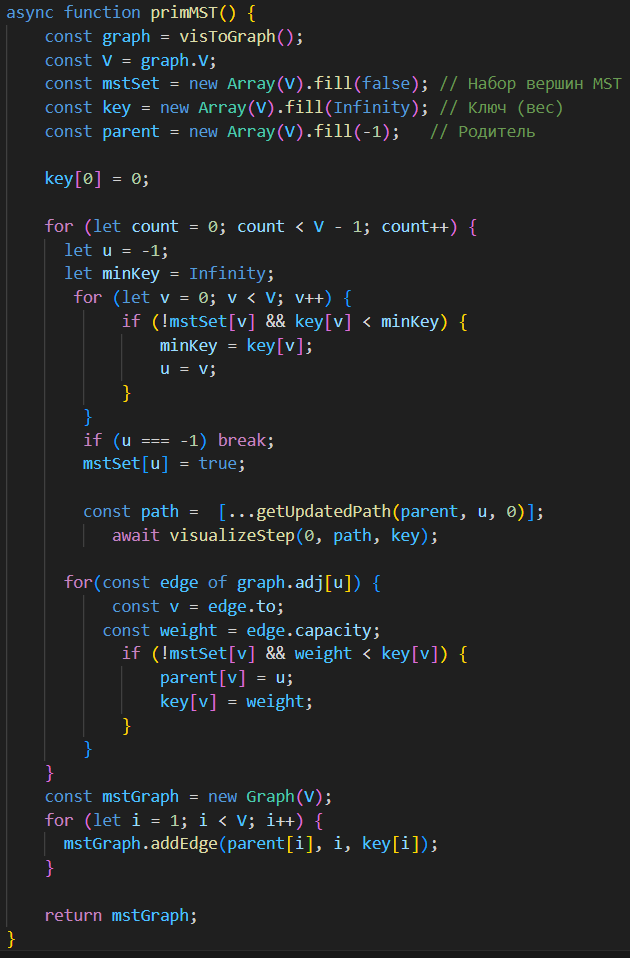


Рис. 3.2 Метод для создания остовного дерева.

**Задание 4**

Реализовать поиск кратчайшего пути между двумя вершинами графа. Вершины выбирает пользователь. (реализовать визуальную демонстрацию совместно с текстовым описанием шагов).

Реализация была выполнена с помощью алгоритма Дейкстры. Суть алгоритма заключается в том, что — это как прокладывать маршрут на карте, всегда выбирая самый дешёвый путь, пока не найдёте самый дешёвый путь от вашего города до всех остальных.

Инициализация: Создайте таблицу расстояний dist для всех вершин. Изначально расстояние до источника установите в 0, а до всех остальных вершин в бесконечность (или какое-то большое значение).

Создайте множество посещенных вершин visited (изначально пустое).

Создайте приоритетную очередь queue, которая будет хранить вершины и их текущие расстояния до источника. Добавьте в очередь начальную вершину с расстоянием 0.

Основной цикл:

Пока приоритетная очередь не пуста:

Извлеките из очереди вершину u с минимальным расстоянием (по текущим значениям dist). Если вершина u уже посещена (содержится в visited), пропустите ее и переходите к следующей итерации. Пометьте вершину u как посещенную (добавьте в visited). Для каждого соседа v вершины u: Рассчитайте расстояние от источника до вершины v через вершину u: alt = dist[u] + weight(u, v), где weight(u, v) — вес ребра от u до v. Если alt меньше, чем текущее расстояние до v (dist[v]): обновите расстояние dist[v] до alt. Обновите приоритет вершины v в очереди (или добавьте её заново, если её не было). Перейдите к следующей итерации цикла. (см. рис. 4.1)



Рис. 4.1

Ниже представлен метод для создания пути. (см. рис. 4.2)

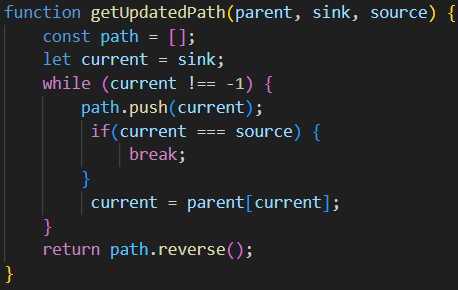


Рис. 4.2