МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ИС

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: алгоритм поиска минимального остова на основе алгоритма Краскала (Крускала).

Студентка гр. 9372 Громова Е.А.

Преподаватель Пелевин М.С.

Санкт-Петербург
2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Громова Е.А.
Группа 9372

Тема работы: алгоритм поиска минимального остова на основе алгоритма Краскала (Крускала).

Исходные данные:

Любой текстовый файл или введённый через консоль набор троек:

```
A B 3
B C 2
A C 1
```

Результат в виде отсортированных по имени пар и суммарный вес:

```
A C
B C
3
```

Максимальный размер входных данных: 50 вершин. Вершины могут быть заданы любой текстовой последовательностью без пробелов. Вес ребра ограничен интервалом от 0 до 1023 включительно.

Цель:

Продемонстрировать знания следующих вопросов:

- сортировка
- обход графов (в глубину и в ширину)
- хранение графов (списки смежности, матрицы смежности, инцидентности)
- построение системы непересекающихся множеств

ВВЕДЕНИЕ

Цель данной работы - реализовать алгоритм поиска минимального остова на основе алгоритма Краскала (Крускала).

Суть алгоритма:

В начале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершен. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса. Подробное описание алгоритма можно найти в литературе.

В данной курсовой работе алгоритм был реализован на языке программирования TypeScript с консольным интерфейсом, обрабатываемым платформой NodeJS. Также к работе были написаны тесты, использующие библиотеку jest.

РЕАЛИЗАЦИЯ АЛГОРИТМА НА ЯЗЫКЕ TYPESCRIPT

Алгоритм состоит из класса Graph, включающего в себя методы:

- **fillFromString()** - наполняет граф связями на основе строк, аналогичных приведенным в качестве примера входных данных;
- **getMinFrame()** - возвращает минимальный остов графа;
- **getMinFrameAsString()** - возвращает минимальный остов графа в виде строк в формате аналогичном приведенным в примере входным данным

```
type Connection = {
 from: string;
 to: string;
 weight: number;
};
type WeightlessConnection = Omit<Connection, 'weight'>;

export class Graph {
 constructor(connections: Connection[]) {
   this.connections = connections;
 }
 fillFromString(description: string = '') {
   const nodesDescription = description.toString().split('\n');
   for (let node of nodesDescription) {
     const parent: string = node.split(' ')[0];
     const child: string = node.split(' ')[1];
     const weight: number = parseInt(node.split(' ')[2], 10);
     // @ts-ignore
     if (![parent, child, weight].includes(undefined) && !isNaN(weight)) {
       this.connections.push({
         from: parent,
         to: child,
         weight
       });
     }
   }
 }
 connections: Connection[];
 getMinFrame() {
   const result: WeightlessConnection[] = [];
   const sortedConnections: Connection[] = [...this.connections].sort((a, b)
=> a.weight - b.weight);
   const frameOfNode: { [frameRoot: string]: string } = {};
   for (const connection of sortedConnections) {
     frameOfNode[connection.from] = connection.from;
     frameOfNode[connection.to] = connection.to;
   }
   const replaceFrame = (oldFrame: string, newFrame: string) => {
     for (const connection of sortedConnections) {
```

```
        if (frameOfNode[connection.from] == oldFrame) {
          frameOfNode[connection.from] = newFrame;
        }
        if (frameOfNode[connection.to] == oldFrame) {
          frameOfNode[connection.to] = newFrame;
        }
      }
    };
    let sumWeight: number = 0;
    for (const connection of sortedConnections) {
      const { from, to, weight } = connection;
      if (frameOfNode[from] !== frameOfNode[to]) {
        sumWeight += weight;
        result.push({ from, to });
        const parentFrame = frameOfNode[from];
        const childFrame = frameOfNode[to];
        replaceFrame(childFrame, parentFrame);
      }
    }
    return {
      pairs: result,
      weight: sumWeight
    };
  }
  getMinFrameAsString() {
    const minFrame = this.getMinFrame();
    const connections = minFrame.pairs.map(({ from, to }) => `${from}
${to}`).join('\n');

    return `${connections}\n${minFrame.weight}`;
  }
}
```

РЕАЛИЗАЦИЯ CLI

Для удобного взаимодействия с алгоритмом был реализован простейший консольный интерфейс.

```
import * as readline from 'readline';
import { Graph } from './Kruskal';
const rl = readline.createInterface({
 input: process.stdin,
 output: process.stdout,
 terminal: false
});

let input = '';

rl.on('line', (line) => {
 if (line) {
   input += '\n' + line;
 } else {
   const graph = new Graph([]);
   graph.fillFromString(input);
   const result = graph.getMinFrameAsString();
   console.log(result);
   process.exit();
 }
});
```

ТЕСТЫ

Тесты были написаны с использованием библиотеки jest.

```
import { Graph } from './Kruskal';

describe('Graph', () => {
 it("Pelevin's Ground truth", () => {
   const graph = new Graph([]);
   graph.fillFromString('A B 3\nB C 2\nA C 1');

   expect(graph.getMinFrameAsString()).toBe(`A C\nB C\n3`);
 });
 it('Tests from https://olympiads.ru/sng/9/index.shtml', () => {
   const tests = [
     {
       input: '1 2 1\n2 3 2\n3 1 3',
       output: 3
     },
     {
       input:
         '14 37 775\n3 13 422\n19 ...',
       output: 7359
     },
     {
       input:
         '100 480\n46 70 69\n20 62 324\n42 ...',
       output: 12302
     }
   ];
   for (let test of tests) {
     const graph = new Graph([]);
     graph.fillFromString(test.input);

     expect(graph.getMinFrame().weight).toBe(test.output);
   }
 });
 it('Test from
https://informatics.mccme.ru/mod/statements/view3.php?id=&chapterid=3559', ()
=> {
   const graph = new Graph([]);
   graph.fillFromString('1 2 5\n1 3 2\n2 3 4\n2 4 3\n3 4 6\n0 3 20\n0 4 10');

   expect(graph.getMinFrame().weight).toBe(19);
 });
});
```

СЛОЖНОСТЬ АЛГОРИТМА

До начала работы алгоритма необходимо отсортировать рёбра по весу, это требует $O(E \times \log(E))$ времени. После чего компоненты связности удобно хранить в виде системы непересекающихся множеств. Все операции в таком случае займут $O(E \times \alpha(E, V))$, где α — функция, обратная к функции Аккермана. Поскольку для любых практических задач $\alpha(E, V) < 5$, то можно принять её за константу, таким образом, общее время работы алгоритма Краскала можно принять за $O(E * \log(E))$.

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы был реализован алгоритм поиска минимального остова на основе алгоритма Краскала (Крускала).