**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

Курсовая РАБОТА

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Реализация алгоритма Краскала

|  |  |  |
| --- | --- | --- |
| Студент гр. 8374 |  | Пихтовников К.С. |
| Преподаватель |  | Пелевин М.С |

Санкт-Петербург

2019

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Пихтовников К.С. | | |
| Группа 8374 | | |
| Тема работы: Реализация алгоритма Краскала**(Крускала)** | | |
|  | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент гр. 8374 |  | Пихтовников К.С. |
| Преподаватель |  | Пелевин М.С |

**Аннотация**

Алгоритм Краскала — эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

В алгоритме Краскала весь единый список ребер упорядочивается по неубыванию весов ребра. Далее ребра перебираются от ребер с меньшим весом к большему, и очередное ребро добавляется к каркасу, если оно не образовывает цикла с ранее выбранными ребрами. В частности, первым всегда выбирается одно из ребер минимального веса в графе.

Для проверки того, что выбранные ребра не образовывает цикл, будем представлять граф, как объединение нескольких компонент связности (непересекающихся множеств). В самом начале, когда ни одно ребро графа не выбрано, каждая вершина является отдельной компонентой связности. По мере добавления новых ребер компоненты связности будут объединяться, пока не получится одна общая компонента связности. При рассмотрении очередного ребра рассматриваем вершины компонент связности, соответствующих концам этого ребра. Если эти вершины совпадают, то эти вершины уже лежат в одной компоненте связности, поэтому добавление этого ребра образовывает цикл. Значит мы его не добавляем эту вершину. Если же ребро соединяет две разные компоненты связности, например, с номерами a и b, то ребро добавляется к части основного дерева, а эти две компоненты связности объединяются вместе.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Сортировка | 6 |
| 2. | Хранение графов | 7 |
| 3. | Система непересекающихся множеств | 8 |
| 4. | Описание алгоритма | 9 |
|  | Заключение | 10 |
|  | Список использованных источников | 11 |
|  |  |  |

**введение**

Цель работы: Продемонстрировать знания следующих вопросов:

- сортировка

- хранение графов

- построение системы непересекающихся множеств

Задача: Реализовать алгоритм поиска минимального остова на основе

алгоритма Краскала (Крускала).

1. **Сортировка**

В реализации алгоритма Краскала была использована сортировка вставками.

Сортировка вставками (*Insertion Sort*) — это простой алгоритм сортировки. Суть его заключается в том что, на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Стоит отметить что массив из 1-го элемента считается отсортированным.

Худшее время- O(n^2) сравнений, обменов

Лучшее время- O(n) сравнений,0 обменов

Среднее время- O(n^2) сравнений, обменов

Затраты памяти- O(n) всего, O(1) вспомогательный {\displaystyle O(n^{2})}o

**2.Хранение графов**

Для хранения графов используется массив структур. Одна структура содержит две вершины и вес ребра. Такой способ хранения напоминает список ребер, где также для каждого ребра графа хранится пара номеров вершин, которые оно соединяет.

Для графа с N ребрами такой способ хранения требует порядка N памяти, но при этом основные операции — проверка наличия ребра между двумя вершинами и перебор вершин, смежных с данной — выполняются также за время порядка N, поскольку требуют полного перебора всех списков ребер.

Иногда удобно хранить каждое ребро неориентированного графа дважды: как [u, v] и как [v, u].

**3. система непересекающихся множеств**

**Система непересекающихся множеств** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *disjoint-set*, или *union–find* *data structure*) — [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), которая позволяет администрировать множество элементов, разбитое на непересекающиеся подмножества. При этом каждому подмножеству назначается его представитель — элемент этого подмножества. Абстрактная структура данных определяется множеством трёх операций:

**MakeSet(X)** — внести в структуру новый элемент X, создать для него множество размера 1 из самого себя.  
**Find(X)** — возвратить идентификатор множества, которому принадлежит элемент X. В качестве идентификатора мы будем выбирать один элемент из этого множества — представителя множества. Гарантируется, что для одного и того же множества представитель будет возвращаться один и тот же, иначе невозможно будет работать со структурой: не будет корректной даже проверка принадлежности двух элементов одному множеству if (Find(X) == Find(Y)).  
**Unite(X, Y)** — объединить два множества, в которых лежат элементы X и Y, в одно новое.

#### Быстродействие: Работа каждой операции сильно зависит от структуры дерева, а структура дерева — от списка выполненных до того операций. Исключение составляет только MakeSet — её время работы очевидно O(1). {\displaystyle \{\mathrm {Union} ,\mathrm {Find} ,\mathrm {MakeSet} \}}

**4.Описание алгоритма**

1*.* Данные считываются из файла тройками. Пример записи в файле дан ниже

Например,

A B 3  
B C 2

A C 1

2. Перед началом выполнения алгоритма, все рёбра сортируются по весу с помощью Insention sort(сортировкой вставками) в порядке неубывания.

3.Затем начинается процесс объединения: перебираются все рёбра от первого до последнего (в порядке сортировки).

4. Перед добавлением ребра в множество, осуществляется проверка на циклы.

Если циклов не возникает, то ребро добавляется в множество.

5.Если у текущего ребра его концы принадлежат разным множествам, то эти множества объединяются, а ребро добавляется к одному из множеств.

4.По окончании перебора всех рёбер все вершины окажутся принадлежащими одному множеству.

5.Сортируем наш массив структур по имени пар и выводим его на экран

6.Также выводим на экран суммарный вес всех ребер получившегося минимального остова.

Результат для примера.

А С

В С

3

**заключение**

Цель работы была выполнена. В ходе курсового проекта получилось реализовать алгоритм Краскала. Были продемонстрированы знания следующих вопросов:

1.сортировка вставками

2.хранение графов

3.система непересекающихся множеств.

По результатам тестирования программного кода, не было найдено никаких ошибок. Минимальное остовное дерево, которое выводится на экран в виде отсортированных по имени пар вершин, строится правильно.

**список использованных источников**

1. Адитья Бхаргава “Грокаем Алгоритмы”

2.<https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BD%D0%B5%D0%BF%D0%B5%D1%80%D0%B5%D1%81%D0%B5%D0%BA%D0%B0%D1%8E%D1%89%D0%B8%D1%85%D1%81%D1%8F_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2>

3. <https://foxford.ru/wiki/informatika/sposoby-hraneniya-grafa>

4. <https://e-maxx.ru/algo/mst_kruskal>

5. <https://habr.com/ru/post/181271/>