Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Программирование на языках высокого уровня

Тема: Алгоритм Краскала

Выполнил:

студент гр. 3331506/70401 Коновалов В.А.

Преподаватель: Ананьевский М.С.

« » _______2020 г.

Санкт-Петербург

2020 г.

Оглавление

1. Формулировка задачи, которую решает алгоритм

Алгоритм Краскала – эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Также алгоритм используется для нахождения некоторых приближений для задачи Штейнера.

Остовное дерево – ациклический связный подграф данного связного неориентированного графа, в который входят все его вершины.

Минимальное остовное дерево имеет (V - 1) ребер, где V - количество вершин в данном графе.

2. Словесное описание алгоритма

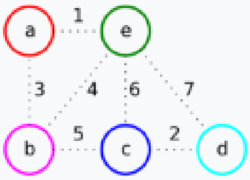
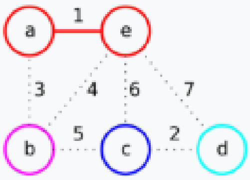
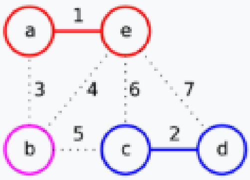
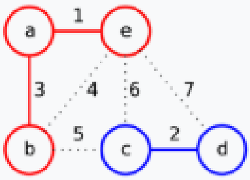
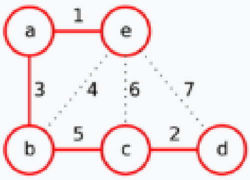
1. Сортировать все ребра в порядке убывания их веса.

2. Выбрать самое маленькое ребро. Проверить, образует ли оно цикл с сформированным до сих пор остовным деревом. Если цикл не сформирован, добавить это ребро. Иначе, отказаться от него.

3. Повторять шаг № 2, пока в связующем дереве не появятся ребра (V-1).

Пример:

Рёбра *(в порядке их просмотра)*	ae	cd	ab	be	bc	ec	ed
Веса рёбер	1	2	3	4	5	6	7

Изображение	Описание
	Первое ребро, которое будет рассмотрено — **ae**, так как его вес минимальный. Добавим его к ответу, так как его концы соединяют вершины из разных множеств (**a** — красное и **e** — зелёное). Объединим красное и зелёное множество в одно (красное), так как теперь они соединены ребром.
	Рассмотрим следующие ребро — **cd**. Добавим его к ответу, так как его концы соединяют вершины из разных множеств (**c** — синее и **d** — голубое). Объединим синее и голубое множество в одно (синее), так как теперь они соединены ребром.
	Дальше рассмотрим ребро **ab**. Добавим его к ответу, так как его концы соединяют вершины из разных множеств (**a** — красное и **b** — розовое). Объединим красное и розовое множество в одно (красное), так как теперь они соединены ребром.
	Рассмотрим следующие ребро — **be**. Оно соединяет вершины из одного множества, поэтому перейдём к следующему ребру **bc** Добавим его к ответу, так как его концы соединяют вершины из разных множеств (**b** — красное и **c** — синее). Объединим красное и синее множество в одно (красное), так как теперь они соединены ребром.
	Рёбра **ec** и **ed** соединяют вершины из одного множества, поэтому после их просмотра они не будут добавлены в ответ Всё рёбра были рассмотрены, поэтому алгоритм завершает работу. Полученный граф — минимальное остовное дерево

3. Реализация алгоритма

Листинг программы с реализацией алгоритма Краскала приведен ниже:

```
#include <iostream>

using namespace std;


class Graf // создаем класс граф
{
private:
    int MaxNodes; // максимальное кол-во узлов
    int* nodes; // массив узлов для "раскраски"
    int num_peak; // кол-во вершин
    int num_edge; // кол-во ребер
    int Last_peak; // цвет последней "окрашеной" вершины

    struct knot // структура узла
    {
        int v1;
        int v2;
        int weight;
    }*knots;
public:
    Graf();
    void InitGraf(); // функция задающая граф
    void sort(); // сортировка ребер по возрастанию
    int GetColor(int n); // получает "цвет" ребра,n - номер вершины
    void OutTree(); // вывод дерева на экран

};
```

```cpp
Graf::Graf() // конструктор
{
	MaxNodes = 50;
	nodes = new int[MaxNodes];
	knots = new knot[MaxNodes];
	num_peak = 0;
	num_edge = 0;
	Last_peak = 0;
}


void Graf::InitGraf() // функция, задающая граф
{
	setlocale(LC_ALL, "Russian");
	cout << "Ввидите число вершин" << endl;
	cin >> num_peak;
	cout << endl
		<< "Введите число ребер" << endl;
	cin >> num_edge;


	for (int i = 0; i < num_peak; i++) // задаем начальные "цвета" вершинам
		nodes[i] = -1 - i;
	cout << endl
		<< "Колличество ребер : " << num_edge << endl
		<< "Введите их в формате : вершина 1, вершина 2, вес"
		<< endl;
	for (int i = 0; i < num_edge; i++)
	{
		cout << endl << "Вершина 1 = "; cin >> knots[i].v1;
		cout << "Вершина 2 = "; cin >> knots[i].v2;
		cout << "Вес = "; cin >> knots[i].weight;
	}
	cout << endl << "Граф задан" << endl;
}


void Graf::sort() // фунция, сортирующая ребра графа по весу, начиная с наименьшего
{
	knot tmp; // обьект типа узел

	for (int i = 0; i < num_edge - 1; i++) // пузырьковая сортировка
		for (int j = 0; j < num_edge - 1; j++)
			if (knots[j].weight > knots[j + 1].weight)
			{
				tmp = knots[j];
				knots[j] = knots[j + 1];
				knots[j + 1] = tmp;
			}

}
```

```
int Graf::GetColor(int n) // функция, получающая "цвет" ребра; параметры: n - номер вершины
{
    if (nodes[n] < 0) // если "цвет" вершины с номером n - отрицательный, то...
    {
        Last_peak = n; // "цвет" последней окрашенной вершины
        return nodes[Last_peak]; // возвращаем его
    }
    else
    {
        int color;
        color = GetColor(nodes[n]); // получаем "цвет" вершины с номером n
        nodes[n] = Last_peak; // "окрашиваем" его в "цвет" вершины, с которой объединяем
        return color; // возвращаем "цвет"
    }
}
```

```
void Graf::OutTree() // функция, выводящая дерево на экран
{
    sort();
    setlocale(LC_ALL, "Russian");

    cout << "Минимальное остовное дерево состоит из ребер с весами " << endl; // сортируем ребра по возрастанию
    for (int i = 0; i < num_edge; i++)
    {
        int color1 = GetColor(knots[i].v2); // получаем "цвет" второй вершины
        int color2 = GetColor(knots[i].v1); // получаем "цвет" первой вершины

        if (color2 != color1) // если ребро соединяет вершины различных "цветов", то...
        {
            nodes[Last_peak] = knots[i].v2; // ..."перекрашиваем" вершины в "цвет" ребра
            cout << endl // добавляем вершину в минимальное остовное дерево
                << knots[i].weight;
            cout << "\n";
        }
    }
}
```

```cpp
int main()
{
    setlocale(LC_ALL, "Russian");
    Graf graf;
    int c = 1;

    while (c != 3)
    {

        cout << "Операции" << endl;
        cout << "1. Задать граф" << endl;
        cout << "2. Построить дерево" << endl;
        cout << "3. Выход" << endl;
        cout << ">> " << endl;

        cin >> c;

        switch (c)
        {
        case 1:
            graf.InitGraf();
            break;

        case 2:
            graf.OutTree();
            break;

        case 3:
            break;

        default:
            cout << endl << "Неверный выбор" << endl;
            break;
        }
    }
    return(1);
}
```

4. Анализ алгоритма

4.1 Математическое описание алгоритма

Пусть задан связный неориентированный граф G=(V,E) с весами рёбер $f(e)$. Предполагается, что веса всех рёбер различны (если это не так, то можно упорядочить рёбра сначала по весу, а потом по номеру).

Алгоритм Крускала основан на следующих двух свойствах задачи:

- **Минимальное ребро графа**. Если $e*$ – единственное ребро графа с минимальным весом, то оно принадлежит минимальному остовному дереву.
- **Схлопывание фрагментов**. Пусть F – фрагмент минимального остовного дерева графа G, а граф G' получен из G склеиванием вершин, принадлежащих F. Тогда объединение F и минимального остовного дерева графа G' даёт минимальное остовное дерево исходного графа G.

В начале работы алгоритма каждая вершина графа G является отдельным фрагментом.

На каждом шаге из рёбер, ещё не рассмотренных на предыдущих шагах, выбирается ребро с минимальным весом. Если оно соединяет два различных фрагмента, то оно добавляется в минимальное остовное дерево, а фрагменты склеиваются. В противном случае это ребро отбрасывается.

4.2 Последовательная сложность алгоритма

Время работы алгоритма складывается из сортировки рёбер и поддержания информации о фрагментах. В базовом варианте рёбра вначале сортируются за время $m \ln n$ (например, с помощью быстрой сортировки), затем просматриваются в порядке увеличения веса за время $O(m)$, при этом для хранения информации о текущих фрагментах используется система непересекающихся множеств с общим временем работы $O(m\alpha(m,n))$. Итоговая сложность алгоритма $O(m \ln n)$.

Как видно, наибольшую сложность имеет этап сортировки, при этом большая часть рёбер сортируется напрасно: они всё равно будут отброшены, как принадлежащие одному фрагменту. Использование инкрементальной быстрой сортировки (англ. IQS: Incremental Quick Sort) позволяет снизить затраты на сортировку, так что среднее время работы алгоритма составляет $O(m + n \ln 2n)$.

В случае, если рёбра графа изначально отсортированы по весу рёбер, сложность алгоритма снижается до $O(m\alpha(m,n))$.

5. Применение алгоритма

- разработка сетей (к примеру, чтобы соединить n городов в единую телефонную сеть с минимальной суммарной стоимостью соединений);
- производство печатных плат (по аналогии с сетью: мы хотим соединить n контактов проводами с минимальной суммарной стоимостью);
- минимальное остовное дерево может использоваться для визуализации многоаспектных, многомерных данных, например, для отображения их взаимосвязи;
- наука, и в частности биология, используют многомерные данные для группировки объектов, растений, животных. Минимальное остовное дерево позволяет разбивать их на взаимосвязанные классы, четко отслеживая близкие по строению и характеристикам группы.

6. Заключение

В ходе работы был описан и реализован алгоритм Краскала для нахождения минимального остовного дерева. Для анализа данного алгоритма были приведены математическое описание данного алгоритма, последовательная сложность, а также области его применения.

7. Список литературы

1. Joseph. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proc. AMS. 1956. Vol 7, No. 1. C. 48-50

2. Белоусов А. И., Ткачев С. Б. Дискретная математика. — М.: МГТУ, 2006. — 744 с.

3. Кузнецов О.П., Адельсон-Вельский Г.М. Дискретная математика для инженера. - М.: Энергоатомиздат , 1988.