Федеральное агентство связи

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ПМиК

Курсовая работа
по дисциплине «Методы оптимизаций»
на тему
«Реализация алгоритма Краскала»

Выполнил:
студент гр. МГ-165
Терешков Р. В.

Проверил:
к.ф.-м.н., доцент
Рубан А. А.

Новосибирск – 2017

Теоретические сведения

Граф G(V, E) -- абстрактный объект, представляющий собой множество вершин графа (V) и набор рёбер (E), которые попарно соединяют вершины между собой.

Остовное дерево (*spanning tree*) -- дерево графа, состоящее из минимального подмножества рёбер, таких, что из любой вершины графа можно попасть в любую другую вершину двигаясь по этому дереву. **Минимальное остовное дерево** состоит из тех рёбер, которые в сумме дают минимальный возможный вес.

Алгоритм Краскала (*Kruskal's algorithm*) -- это алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Эффективная реализация данного алгоритма использует систему непересекающихся множеств лежащую в своей основе.

Система непересекающихся множеств (*disjoint-set*, *union-find data structure*) -- это структура данных, которая предназначена для отслеживания множества элементов, разбитого на непересекающиеся подмножества. При этом каждому подмножеству назначается его представитель -- один из элементов данного подмножества. Обычно определяется тремя операциями:

- MakeSet(x) -- создаёт для элемента x новое подмножество и назначает этот же элемент представителем данного подмножества;
- Find(x) -- определяет для x подмножество, к которому этот элемент принадлежит, и возвращает его представителя;
- Merge(x, y) -- объединяет подмножества, принадлежащие представителям x и y, и назначает одного из этих элементов представителем данного множества.

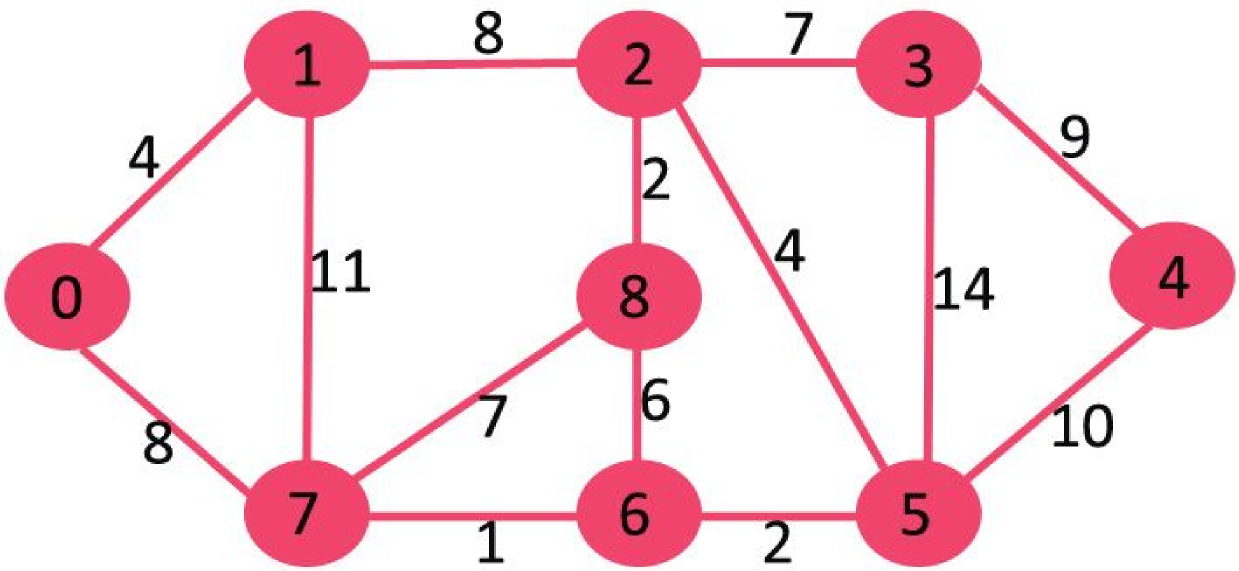
Алгоритм построения минимального остовного дерева выглядят следующим образом:

1) Сортируем по неубыванию рёбра графа по их весу.
2) Устанавливаем текущее множество рёбер пустым.
3) Из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появления цикла в нём, выбирается ребро минимального веса и добавляется к этому множеству.
4) Как только таких рёбер больше нет, алгоритм завершает свою работу.

Сложность алгоритма: *O(E x log(E))*.

Практическая часть

В качестве примера работы алгоритма Краскала рассмотрим граф следующего вида:



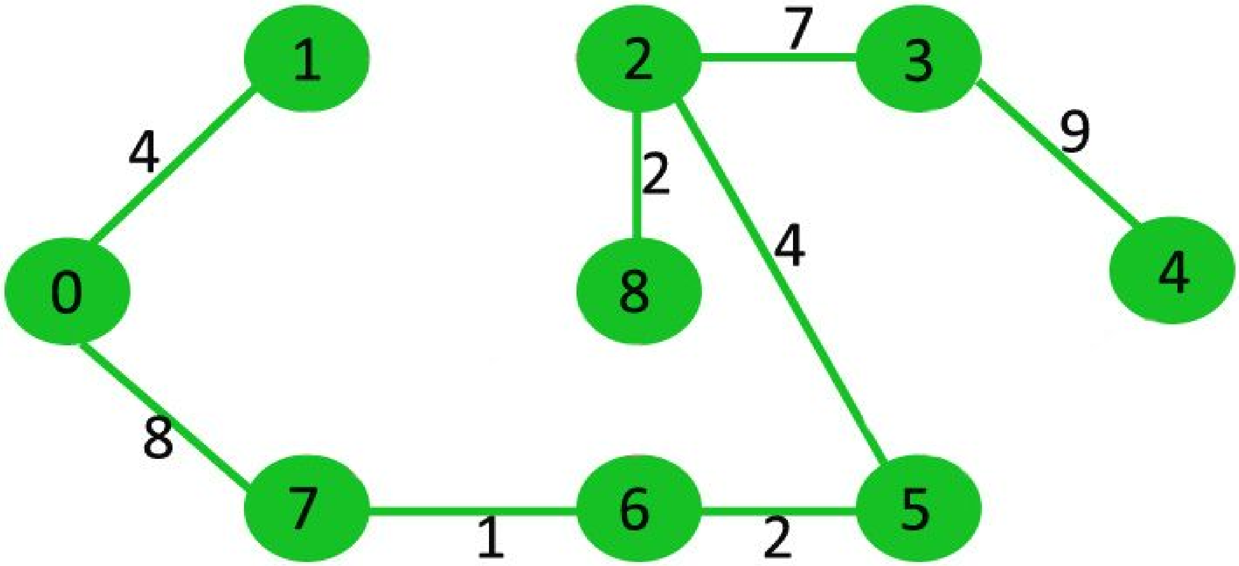
и найдём его минимальное остовное дерево.

Представим в качестве таблицы шаги работы алгоритма при выборе вершин для создания непересекающихся множеств (красным цветом помечены рёбра, которые образуют цикл в графе и не будут включены в итоговое минимальное остовное дерево):

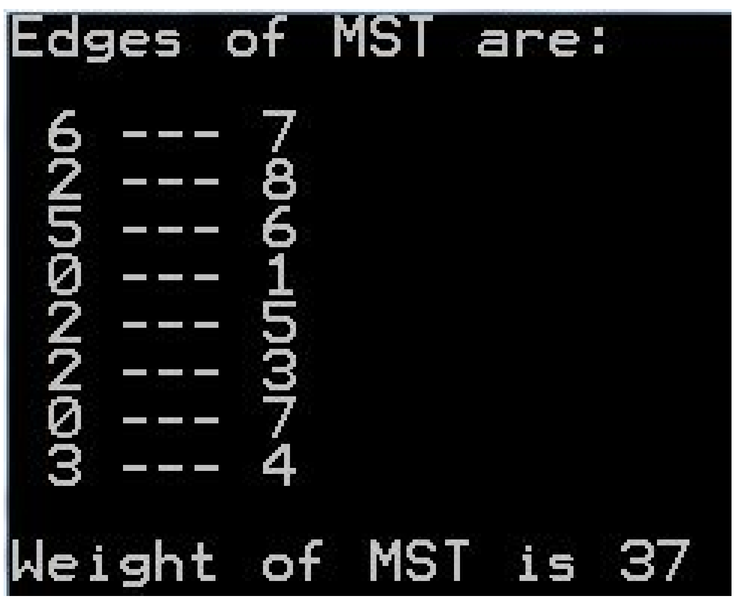
№	1	2	3	4	5	6	7	8	9	10	11	-	-	-
E	6-7	2-8	5-6	0-1	2-5	6-8	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

Как можно заметить, работу алгоритма можно завершать на том моменте, когда количество выбранных рёбер станет равное V - 1.

Итоговое минимальное остовное дерево, вес которого равен 37, будет выглядеть следующим образом:



Теперь запустим написанную на языке C++ программу представленную в листинге с этими же исходными данными и сравним полученный результат. По завершению получаем следующий вывод программы:



```
Edges of MST are:

 6 --- 7
 2 --- 8
 5 --- 6
 0 --- 1
 2 --- 5
 2 --- 3
 0 --- 7
 3 --- 4

Weight of MST is 37
```

, что является эквивалентным результатом с рассмотренным ранее.

Листинг программы

Graph.h

```
#ifndef _GRAPH_H_
#define _GRAPH_H_

#include <iostream>
#include <vector>

typedef std::pair<int, int> iPair;

struct Graph
{
	int V, E;
	std::vector<std::pair<int, iPair>> edges;

	Graph(int _v, int _e) : V(_v), E(_e) {}

	void addEdge(int _u, int _v, int _w)
	{
		edges.push_back({ _w, { _u, _v } });
	}
};

#endif
```

DisjointSet.h

```
#ifndef _DISJOINTSET_H_
#define _DISJOINTSET_H_

struct DisjointSet
{
	int *parent, *rank;
	int n;

	DisjointSet(int);
	~DisjointSet();

	int find(int _u);
	void merge(int _x, int _y);
};

#endif
```

DisjointSet.cpp

```
#include "DisjointSet.h"

DisjointSet::DisjointSet(int _n) : n(_n)
{
	parent = new int[n + 1];
	rank = new int[n + 1];
```

```
	for (int i = 0; i <= n; ++i)
	{
		parent[i] = i;
		rank[i] = 0;
	}
}

DisjointSet::~DisjointSet()
{
	delete[] parent;
	delete[] rank;
}

int DisjointSet::find(int _u)
{
	if (_u != parent[_u])
	{
		parent[_u] = find(parent[_u]);
	}

	return parent[_u];
}

void DisjointSet::merge(int _x, int _y)
{
	_x = find(_x), _y = find(_y);

	if (rank[_x] > rank[_y])
	{
		parent[_y] = _x;
	}
	else
	{
		parent[_x] = _y;
	}

	if (rank[_x] == rank[_y])
	{
		rank[_y]++;
	}
}
```

Main.cpp

```
#include <algorithm>
#include "Graph.h"
#include "DisjointSet.h"

int kruskalMST(Graph g);

int main(void)
{
	int V = 9, E = 14;
	Graph g(V, E);
```

```
	g.addEdge(0, 1, 4);
	g.addEdge(0, 7, 8);
	g.addEdge(1, 2, 8);
	g.addEdge(1, 7, 11);
	g.addEdge(2, 3, 7);
	g.addEdge(2, 8, 2);
	g.addEdge(2, 5, 4);
	g.addEdge(3, 4, 9);
	g.addEdge(3, 5, 14);
	g.addEdge(4, 5, 10);
	g.addEdge(5, 6, 2);
	g.addEdge(6, 7, 1);
	g.addEdge(6, 8, 6);
	g.addEdge(7, 8, 7);

	std::cout << "Edges of MST are: \n\n";

	int mst_wt = kruskalMST(g);

	std::cout << "\n Weight of MST is " << mst_wt << std::endl;

	getchar();
	return 0;
}

int kruskalMST(Graph g)
{
	int mst_wt = 0;
	int e_cnt = 0;

	std::sort(g.edges.begin(), g.edges.end());

	DisjointSet ds(g.V);

	for (auto it = g.edges.begin(); it != g.edges.end() && e_cnt != g.V - 1; it++)
	{
		int u = it->second.first;
		int v = it->second.second;

		int set_u = ds.find(u);
		int set_v = ds.find(v);

		if (set_u != set_v)
		{
			std::cout << " " << u << " --- " << v << std::endl;

			e_cnt++;
			mst_wt += it->first;
			ds.merge(set_u, set_v);
		}
	}

	return mst_wt;
}
```