МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО. СИСТЕМА НЕПЕРЕСЕКАЮЩИХСЯ МНОЖЕСТВ

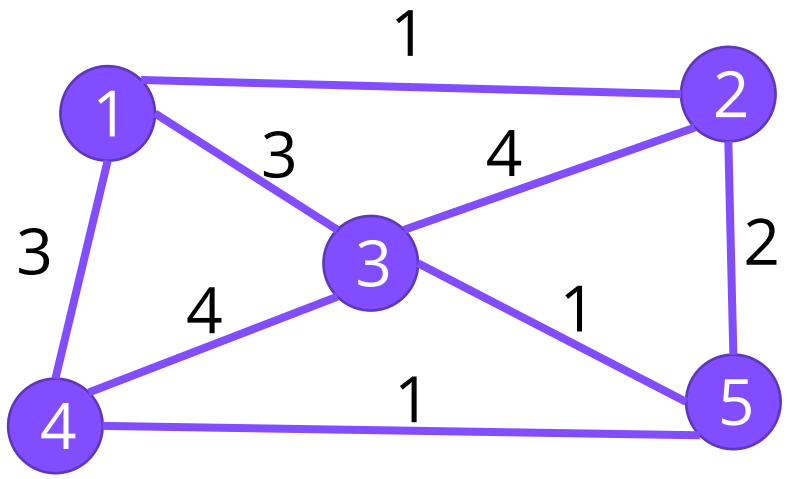
Школа::Кода

Олимпиадное программирование

2020-2021 Таганрог

Минимальное остовное дерево

- Минимальное остовное дерево (мин. остов, англ. Minimum spanning tree) – такое подмножество рёбер в взвешенном ненаправленном графе, что между любой парой вершин существует ровно один простой путь, а суммарный вес рёбер минимален.



Алгоритм Прима

- Пусть изначально в мин. остов включена одна вершина. Её можно выбрать произвольно.
- Далее в мин. остов добавляется ребро с наименьшим весом, исходящее из этой вершины, и вершина, к которой оно ведёт.
- Теперь рассматриваются все рёбра, которые соединяют вершины, входящие в мин. остов, с вершинами, которые ещё в него не входят. Из этих рёбер выбирается минимальное и добавляется в мин. остов, как и вершина, к которой оно ведёт.
- Предыдущий шаг повторяется, пока существует ребро, удовлетворяющее всем требованиям.

Реализация $O(N^2)$

```
#define INF 1e9
#define UNDEF -1

struct Node {
    vector<pair<int, int>> to;
    bool is_in_mst = false;
    int min_el = INF;
    int sel_e = UNDEF;
};
vector<Node> g;

struct Edge
{
    int u;
    int v;
    int w;

    Edge(int u, int v, int w)
        :u(u), v(v), w(w)
    {}
};
vector<Edge> e;
```

```
int Prim_mst() {
    double sum = 0;
    g[0].min_el = 0;
    for (int i = 0; i < g.size(); i++) {
        int v = UNDEF;
        for (int j = 0; j < g.size(); j++) {
            if (!g[j].is_in_mst && (v == UNDEF
                || g[j].min_el < g[v].min_el))
                v = j;
        }
        g[v].is_in_mst = true;
        if (g[v].sel_e != UNDEF) {
            e.push_back(Edge(v, g[v].sel_e, g[v].min_el));
            sum += g[v].min_el;
        }
        for (int j = 0; j < g[v].to.size(); j++) {
            int u = g[v].to[j].first;
            float edge_length = g[v].to[j].second;
            if (edge_length < g[u].min_el) {
                g[u].min_el = edge_length;
                g[u].sel_e = v;
            }
        }
    }
    return sum;
}
```

Алгоритм Краскала

- Каждая вершина представляется как отдельное дерево.
- Рёбра сортируются в порядке неубывания весов.
- Рассматривается каждое ребро.
- Если ребро соединяет вершины из разных деревьев, то оно добавляется в минимальный остов, а эти деревья объединяются.
- После перебора всех рёбер все вершины окажутся принадлежащими одному дереву, которое будет являться мин. остовом.

Система непересекающихся множеств

- Система непересекающихся множеств (СНМ, англ. disjoint-set-union, DSU) – структура данных которая позволяет администрировать множество элементов, разбитое на непересекающиеся подмножества.
- При инициализации СНМ все множества (деревья) системы состоят из 1 элемента (вершины), который является представителем (корнем) своего множества.
- Также при инициализации предком каждой вершины считается она сама.

CHM (2)

Дерево идентифицируется по своему корню. То есть, чтобы узнать, какому дереву принадлежит вершина, нужно найти корень этого дерева. Если предком вершины является она сама – то она и есть корень. Иначе можно узнать, какому дереву принадлежит предок вершины. Логично, что вершина всегда принадлежит тому же дереву, что её предок, поэтому эта операция уместна. Для предка корень дерева определяется таким же способ, что говорит о рекурсивности данного алгоритма.

Данная реализация определения дерева, которому принадлежит вершина, выполняется за глубину дерева, так как в ней рассматриваются все вершины на пути от заданной до корня. Этот показатель можно улучшить, если после каждой такой операции в качестве предка вершины запоминать найденный корень дерева, которому она принадлежит. Тогда асимптотика этой операции будет O(1).

СНМ (3)

- Если корни деревьев, которым принадлежат вершины, различаются, то и деревья являются различными. Если же корни совпадают, то и деревья совпадают.
- Объединение деревьев осуществляется путём добавления связи между корнями этих деревьев. При этом корень большего дерева становится предком корня меньшего дерева, к размеру поддерева корня большего дерева добавляется размер поддерева корня меньшего дерева. Такой выбор корня нового дерева обусловлен необходимостью балансировки дерева, чтобы оно не превращалось в «бамбук», что негативно сказывалось бы на времени выполнения операции определения дерева, которому принадлежит вершина.

Реализация CHM

```
struct DSU
{
    struct Node
    {
        int size;
        int parent;

        Node() {}
        Node(int i)
            :parent(i), size(1)
        {}
    };

    vector<Node> tree;

    DSU(int size)
    {
        tree.resize(size);
        for (int i = 0; i < size; ++i)
            tree[i] = i;
    }

    int get_parent(int node)
    {
        if (tree[node].parent == node)
            return node;
        return tree[node].parent = get_parent(tree[node].parent);
    }

    bool unite(int node1, int node2)
    {
        node1 = get_parent(node1);
        node2 = get_parent(node2);
        if (node1 == node2)
            return false;
        if (tree[node1].size > tree[node2].size)
            swap(node1, node2);
        tree[node1].parent = node2;
        tree[node2].size += tree[node1].size;
        return true;
    }
};
```

Реализация алгоритма Крускала O(M*log(M)

```
struct Edge
{
    int v;
    int u;
    int w;

    Edge(int v, int u, int w)
        :v(v), u(u), w(w)
    {}

    friend bool operator<(const Edge& e1, const Edge& e2)
    {
        return e1.w < e2.w;
    }
};
vector<Edge> e;
```

```
int Kruskal_mst(int n, vector<int>& mst)
{
    sort(e.begin(), e.end());
    DSU dsu(n);
    int res = 0;
    for (int i = 0; i < e.size(); ++i)
        if (dsu.unite(e[i].v, e[i].u))
        {
            mst.push_back(i);
            res += e[i].w;
        }
    return res;
}
```