Minimum spanning tree. Kruskal's algorithm

Given a weighted undirected graph. Required to find a subtree of this graph, which would connect all the vertices, and thus has the least weight (ie, the sum of the weights of the edges) of all. This subtree is called a minimal spanning tree, or simply the minimum core.

It will discuss some important facts related to the minimum spanning tree, then be considered Kruskal's algorithm in its simplest implementation.

Properties of the minimum spanning tree

- Minimum frame **is unique, if the weights of all edges are different** . Otherwise, there may be multiple cores minimum (specific algorithms typically receive one of the possible core).
- Minimum frame is also **the skeleton with minimal product** of edge weights.
- (it proved easy enough to replace the weight of all edges on their logarithms)
- Minimum frame is also **the skeleton of minimum weight of the heaviest edge** .
- (this follows from the validity of Kruskal's algorithm)
- **The skeleton of the maximum weight** is sought similar skeleton of minimum weight, enough to change the signs of all the ribs on the opposite and perform any of the minimum spanning tree algorithm.

Kruskal's algorithm

This algorithm has been described by Kruskal (Kruskal) in 1956

Kruskal's algorithm initially assigns each vertex in your tree, and then gradually brings these trees, combining the two in each iteration some wood some edge.Before starting the algorithm, all the edges are sorted by weight (in decreasing order). Then begins the process of unification: all edges are moving from first to last (in the sort order), and if the current edge of its ends belong to different subtrees, these subtrees are combined, and the edge is added to the answer. At the end of sorting all edges will be all the vertices belonging to the same sub-tree, and the answer is found.

The simplest implementation

This code is directly implements the algorithm described above, and runs in **O (M log N + N ^{2})** . Sort edges require O (M log N) operations. Vertices belonging to a particular subtree stored simply by using an array tree_id - it is stored for each vertex tree number to which it belongs. For each edge we in O (1) define, whether it ends belong to different trees. Finally, the union of two trees is carried out for the O (N) simply pass the array tree_id. Given that all the operations of union is N-1, we obtain the asymptotic behavior of **O (M log N + N ^{2})** .

```
int m;
vector <pair <int, pair <int,int> >> g (m); / / weight - the top one - the top 2

int cost = 0;
vector <pair <int,int>> res;

sort (g.begin (), g.end ());
vector <int> tree_id (n);
for (int i = 0; i <n; + + i)
        tree_id [i] = i;
for (int i = 0; i <m; + + i)
{
        int a = g [i]. second.first, b = g [i]. second.second, l = g [i]. first;
        if (tree_id [a]! = tree_id [b])
        {
                cost + = l;
                res.push_back (make_pair (a, b));
                int old_id = tree_id [b], new_id = tree_id [a];
                for (int j = 0; j <n; + + j)
                        if (tree_id [j] == old_id)
                                tree_id [j] = new_id;
        }
}
```

An improved

Using the data structure "system of disjoint sets" can write faster implementation of Kruskal's algorithm with the asymptotic O (Log M N) .