Minimum spanning tree. Kruskal's algorithm with a system of disjoint sets

Description of the problem and Kruskal's algorithm, see here .

Implementation will be discussed here using the data structure "system of disjoint sets" (DSU) , which will reach the asymptotic behavior of **O (Log M N)** .

Description

Just as in the simple version of Kruskal's algorithm, we can sort all the edges in non-decreasing weight. Then put each node in your tree (ie its set) by calling the DSU MakeSet - it will take in the amount of O (N). Loop through all the edges (in the sort order), and for each edge in O (1) we determine whether it belongs to the ends of different trees (using two calls FindSet O (1)). Finally, the union of two trees will be calling the Union - also O (1). Overall, we obtain the asymptotic behavior of O (M log N + N + M) = O (M log N).

Implementation

To reduce the amount of code and carry out all operations are not as separate functions, and directly in the code of Kruskal's algorithm.

Here we will use a randomized version of the DSU.

```
vector <int> p (n);

int dsu_get (int v) {
	return (v == p [v])? v: (p [v] = dsu_get (p [v]));
}

void dsu_unite (int a, int b) {
	a = dsu_get (a);
	b = dsu_get (b);
	if (rand () & 1)
		swap (a, b);
	if (a! = b)
		p [a] = b;
}

... in function main (): ...
```

```
int m;
vector <pair <int, pair <int,int> >> g; / / weight - the top one - the top 2
... Reading the graph ...

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

sort (g.begin (), g.end ());
p.resize (n);
for (int i = 0; i <n; + + i)
	p [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 (dsu_get (a)! = dsu_get (b)) {
		cost + = l;
		res.push_back (g [i]. second);
		dsu_unite (a, b);
	}
}
```