added: 10 Jun 2008 22:03
Edited: 10 Jun 2008 22:05

A minimum spanning tree. Kruskal's algorithm

Contents [hide]

Given a weighted undirected graph. Required to find a subtree of this graph, which has linked to all its vertices, and thus has the lowest 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, and then will consider Kruskal's algorithm in its simplest implementation.

Properties of the minimum spanning tree

- The minimum spanning tree **is unique, if the weights of all edges are distinct** . Otherwise, there may be multiple cores minimum (specific algorithms typically receive one of the possible core).
- The minimum spanning tree is also a **skeleton with a minimum product of** the weights of edges. (proved it's easy enough to replace the weight of all the edges on their logarithms)
- The minimum spanning tree is also a **skeleton with a minimum weight of the heaviest edges** . (this follows from the validity of Kruskal's algorithm)
- **The skeleton of the maximum weight** is sought is similar to the skeleton of minimum weight, you need to change the signs of all edges reversed 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 its own tree, and then gradually brings these trees, combining the two at each iteration some wood some edge. Before starting the algorithm, all edges are sorted by weight (in order of decreasing). Then begins the process of unification: all edges are moving from first to last (in the sort order), and if the current edges of the ends belong to different subtrees, these subtrees are combined, and the edge is added to the answer. At the end of sorting all edges all vertices will belong to the same subtree, 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 is stored simply by using an array tree_id - in it for each vertex number is stored tree to which it belongs. For each edge we have O (1) to determine, whether it belongs to the ends of the different trees. Finally, the union of two trees is carried out for the O (N) are just passing through the array tree_id. Given that all the operations of union will be N-1, we obtain the asymptotic behavior of the **O (M log N + N ^{2})** .

```
int m;
vector <pair <int, pair <int, int>>> g (m); // Weight - the top 1 - 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 an asymptotic O (Log M N) .

Сергей • 9 месяцев назад

int old_id = tree_id[b], new_id = tree_id[a];
Здесь по всей видимости необходимо так:
int old_id = tree_id[a], new_id = tree_id[b];

justas • год назад

Почему сортировка ребер вдруг за MlogN она вроди MlogM

Андрей Кравцун → justas • 11 месяцев назад

В худшем случае M ~ N^2, следовательно, logM <= 2*logN. Тогда получим асимптотику O(M * 2 * logN). константа 2 в O-нотации несущественна, поэтому исчезает, и мы получаем асимптотику O(M*logN).