Minimum spanning tree. Prim's algorithm

Given a weighted undirected graph $G$with $n$vertices and $m$edges. Required to find a subtree of this graph, which would connect all the vertices, and thus has the lowest possible weight (ie, the sum of the weights of the edges).Subtree - a set of edges connecting vertices, and every vertex can reach any other by exactly one simple way.

This subtree is called a minimal spanning tree or just **the minimum spanning tree** . Easy to understand that any framework will necessarily contain $n-1$an edge.

In **a natural setting** , this problem is as follows: there are $n$cities, and for each pair of known connection cost them dear (or know that they can not connect). You want to connect all cities so that you can get from any city to another, and thus the cost of construction of roads would be minimal.

Prim's algorithm

This algorithm is named after the American mathematician Robert Primus (Robert Prim), which opened this algorithm in 1957, however, in 1930, this algorithm has been opened by the Czech mathematician Wojtek Jarnik (Vojtěch Jarník). Furthermore, Edgar Dijkstra (Edsger Dijkstra) in 1959 as invented this algorithm independently.

Description of the algorithm

Sam **algorithm** has a very simple form. Seeking a minimal skeleton is constructed gradually by adding to it the edges one by one. Originally skeleton relies consisting of a single vertex (it can be chosen arbitrarily). Then select the minimum weight edge emanating from the vertex, and is added to the minimum spanning tree. After this framework already contains two vertices, and now sought and added an edge of minimum weight, with one end of one of the two selected vertices, and the other - on the contrary, all other than these two. And so on, i.e. whenever sought minimum weight edge with one end - is taken into the backbone of the vertex and the other end - not yet taken, and this edge is added to the skeleton (if several such edges, we can take any). This process is repeated until the frame will not yet contain all peaks (or, equivalently, $n-1$an edge).

As a result, the skeleton will be built, which is minimal. If the graph was originally not connected, then the skeleton will not be found (the number of selected edges will be less $n-1$).

Proof

Suppose that the graph $G$was connected, ie answer exists. We denote $T$the skeleton found Prim algorithm, and through S- the minimum core. Obviously, that Treally is the backbone (ie, a subtree of the graph G). We show that the weight $S$and $T$the same.

Consider the first time when Tit is added edge that is not in the best frame S. We denote this edge through $e$the ends of it - through $a$and b, as many at that time included in the skeleton vertices - through V(according to the algorithm $a \in V$, $b \notin V$or vice versa). Advantageously the skeleton Svertices $a$and $b$connected in some way P, in this way find any edge g, one end of which lies in V, and the other - no. Since Prim algorithm chose the rib $e$edges instead g, it means that the weight of the edge $g$is greater than or equal to the weight of the edge e.

Now remove from the Sedge g, and add an edge e. By just what to say, as a result of the weight of the skeleton could not increase (decrease it too could not because Sit was the best). Besides, Shas not ceased to be a skeleton (in that connection is not broken, it is easy to make sure we closed the way $P$to the ring, and then removed from one edge of the cycle).

Thus we have shown that we can choose the optimum frame Sso that it will include a rib e. Repeating this procedure as many times, we find that we can choose the optimum frame Sso that it coincided with T. Consequently, the weight of the constructed algorithm Prima Tminimal, as required.

Implementation

Time of the algorithm depends essentially on how we search the minimum of the next edge of suitable edges. There may be different approaches lead to different asymptotic behavior and different implementations.

Trivial Pursuit: algorithms for $O(nm)$and$O(n^2 + m \log n)$

If we look for an edge every time just browsing among all possible options, asymptotically be required viewing $O(m)$edges, to find among all admissible edge with the least weight. Album asymptotic behavior of the algorithm in this case will be $O(nm)$ that in the worst case there $O(n^3)$- too slow algorithm.

This algorithm can be improved if the view each time not all the edges, but only on one edge of each vertex is selected. For this example, you can sort the edges of each vertex in the order of increasing weights, and store a pointer to the first valid edge (recall allowed only those edges that are not yet in the set of selected vertices). Then, if these pointers to recalculate each time you add the ribs into the backbone, the total asymptotic behavior of the algorithm is $O(n^2 + m)$, but first need to sort all edges for

$O(m \log n)$that in the worst case (for dense graphs) gives an asymptotic $O(n^2 \log n)$.

Below we consider two slightly different algorithm: for dense and sparse graphs, received as a result significantly better asymptotic behavior.

Case of dense graphs: an algorithm for $O(n^2)$

Approach the question of the smallest search ribs on the other side: for each not yet selected will store a minimum an edge in an already selected vertex.

Then, the current step to make the selection of the minimum edge, you just see these minimum ribs each non-selected still tops - make asymptotics $O(n)$.

But now, when you add in the next frame edges and vertices of these pointers should recalculate. Note that these pointers can only decrease, ie each vertex not yet scanned or should leave it without changing the pointer, or give it the weight of the edge in the newly added vertex. Therefore, this phase can also be done over $O(n)$.

Thus, we have an option of Prim's algorithm to the asymptotic behavior $O(n^2)$.

In particular, such an implementation is particularly useful for solving the so-called **problem of Euclidean minimum spanning tree** when given $n$points on the plane, the distance between which is measured by the standard Euclidean metric, and want to find the skeleton of minimum weight connecting them all (and add new vertices anywhere elsewhere prohibited). This problem is solved by the algorithm described here in $O(n^2)$ time and $O(n)$memory, which will not achieve the Kruskal algorithm .

Implementation of Prim's algorithm for the graph given by the adjacency matrix $g[][]$:

```
// входные данные
int n;
vector < vector<int> > g;
const int INF = 1000000000; // значение "бесконечность"

// алгоритм
vector<bool> used (n);
vector<int> min_e (n, INF), sel_e (n, -1);
min_e[0] = 0;
for (int i=0; i<n; ++i) {
      int v = -1;
      for (int j=0; j<n; ++j)
            if (!used[j] && (v == -1 || min_e[j] < min_e[v]))
                  v = j;
```

```
        if (min_e[v] == INF) {
            cout << "No MST!";
            exit(0);
        }

        used[v] = true;
        if (sel_e[v] != -1)
            cout << v << " " << sel_e[v] << endl;

        for (int to=0; to<n; ++to)
            if (g[v][to] < min_e[to]) {
                min_e[to] = g[v][to];
                sel_e[to] = v;
            }
}
```

The input is the number of vertices n and the matrix $g[][]$ size $n \times n$, which marked the edge weights and stand number INF, if there is no corresponding edge.The algorithm supports three arrays: flag $\text{used}[i] = \text{true}$ means that the top i is included in the frame, the size of $\min\_e[i]$ the smallest possible weight keeps the edges from the top i, and the element $\text{sel\_e}[i]$ contains the smallest end of the rib (this is necessary to bring the edges of the reply). The algorithm makes the n steps, each of which selects a vertex v with the smallest label $\min\_e$, marks it used, and then looks at all the edges of this vertex, recounting their labels.

Case of sparse graphs: an algorithm for $O(m \log n)$

In the above algorithm can be seen finding the minimum standard operations in the set and change the values in this set. These two operations are classic, and perform many data structures, for example, implemented in C + +, red-black tree set.

Within the meaning of the algorithm is exactly the same, but now we can find the minimum edge over time $O(\log n)$. On the other hand, the time to recount n pointers now be $O(n \log n)$ even worse than in the above algorithm.

If we consider that there will be $O(m)$ conversions of pointers and $O(n)$ searches the minimum edge, then the asymptotic behavior of the total amount $O(m \log n)$ - for sparse graphs is better than both of the above algorithm, but on dense graphs, this algorithm is slower than the previous one.

Implementation of Prim's algorithm for the graph given adjacency list $g[]$:

```
// входные данные
int n;
vector < vector < pair<int,int> > > g;
const int INF = 1000000000; // значение "бесконечность"

// алгоритм
vector<int> min_e (n, INF), sel_e (n, -1);
min_e[0] = 0;
set < pair<int,int> > q;
q.insert (make_pair (0, 0));
for (int i=0; i<n; ++i) {
	if (q.empty()) {
		cout << "No MST!";
		exit(0);
	}
	int v = q.begin()->second;
	q.erase (q.begin());

	if (sel_e[v] != -1)
		cout << v << " " << sel_e[v] << endl;

	for (size_t j=0; j<g[v].size(); ++j) {
		int to = g[v][j].first,
			cost = g[v][j].second;
		if (cost < min_e[to]) {
			q.erase (make_pair (min_e[to], to));
			min_e[to] = cost;
			sel_e[to] = v;
			q.insert (make_pair (min_e[to], to));
		}
	}
}
```

The input is the number of vertices n and n the adjacency lists: $g[i]$ - a list of all edges emanating from the vertex i, as pairs (the second end of the edge weight of the edge). The algorithm supports two arrays value $\min\_e[i]$ stores the smallest possible weight from the top edge i and the element $\text{sel\_e}[i]$ contains the smallest end of the rib (this is necessary to bring the edges of the reply). Additionally, a queue q of all the nodes in increasing order of their labels $\min\_e$. The algorithm makes the n steps, each of which selects a vertex v with the smallest label $\min\_e$ (just removing it from the queue), and

then looks at all the edges of this vertex, recounting their labels (when calculated from the queue, we remove the old value, and then put back new) .

The analogy with the Dijkstra algorithm

In just two described algorithms can be traced quite clear analogy with Dijkstra's algorithm : it has the same structure ($n-1$phase, each of which is selected first optimal edge is added to the response, and then recalculated values for all not yet selected vertices). Moreover, Dijkstra's algorithm also has two options for implementation: for $O(n^2)$and $O(m \log n)$(of course we do not consider here the possibility of using complex data structures to achieve even smaller asymptotics).

If you look at Prima and Dijkstra algorithms more formally, it turns out that they are generally identical to each other except for **the weighting function** peaks: if Dijkstra each vertex supported length of the shortest path (ie the sum of the weights of some edges), then Prim's algorithm to each vertex attributed only to the minimum weight edge leading into the set of vertices already taken.

At the implementation level, this means that after adding another vertex $v$in the set of selected vertices, when we begin to view all the edges $(v, to)$of the vertex, the algorithm Prima pointer toupdated weight of the edge (v, to), and Dijkstra - mark distances $d[to]$updated sum marks $d[v]$and edge weight (v, to). Otherwise, these two algorithms can be considered identical (though they are quite different and solve the problem).

Properties of the minimum spanning tree

- **Maximum** frame can also search algorithm Prima (eg, replacing all the weights of the edges on the opposite: the algorithm does not require the non-negativity of edge weights).
- Minimum frame **is unique** , if the weights of all edges are different. Otherwise, there may be some minimum spanning tree (which one will be selected Prima algorithm depends on the order of viewing edges / vertices with the same weights / pointers)
- Minimum frame is also the core, **the minimum for the product of** all the edges (assuming that all weights are positive). In fact, if we replace the weights of all edges in their logarithms, it is easy to notice that in the algorithm will not change anything, and will be found the same edge.
- Minimum frame is the skeleton of a minimum weight **of the heaviest edge** . Most clearly this statement is clear if we consider the work of Kruskal's algorithm.

- **Criterion minimal** skeleton: the skeleton is minimal if and only if for any edge that does not belong skeleton cycle formed by adding this edge to the core, no edges harder this edge. In fact, if for some edge turned out that it is easier for some ribs formed loop, the frame can be obtained with a smaller weight (an edge is added to the skeleton, and by removing the heaviest edge of the loop). If this condition is not fulfilled for any edge, then all these edges do not improve the weight of the core when they are added.