Prim's Algorithm

Task: Describe another algorithm for constructing the minimal spanning tree, which is characterized by the fact that at each step of the algorithm, the subgraph is a tree. This algorithm is called Prim's Algorithm.

Vojtěch Jarník first discovered and published this algorithm in 1930. Robert Prim subsequently rediscovered and published it in 1957. It was once again rediscovered by Edsger Dijkstra in 1959.

Moral of the story

The idea behind this algorithm is very natural. We apply procedure (2) for constructing a spanning tree that we discussed before using the same queue of edges ordered by cost as in Kruskal's algorithm. The result at each step is a tree, and at the end we get a minimal spanning tree.

Prim's algorithm

Let (V,E) be a connected graph w/ an associated cost function $c: E \longrightarrow \mathbb{R}$.

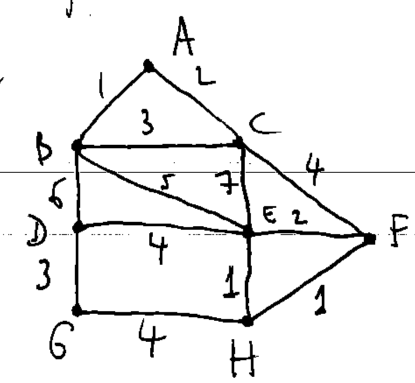
1. Start by choosing some vertex $v \in V$. Our starting subgraph is $(\{v\}, \emptyset)$.
2. List all edges in E in a queue so that the cost of the edges is non-decreasing in the queue, i.e. if $e, e' \in E$ and if $c(e) < c(e')$, then e precedes e' in the queue.

3. We identify the first edge in the queue, which has one vertex included in the current subgraph and the other vertex not included in the subgraph. We add that edge to the current subgraph as well as the vertex not already included. Since the subgraph with which we started was a tree, the resulting subgraph is a tree (we added one vertex and one edge). Continue this process until it is not possible to proceed any further, i.e. we have added all vertices in V.

The fact at each stage we have a tree, and at the end that tree contains all vertices in V guarantees that Prim's Algorithm yields a spanning tree. The fact that we choose what edge to add next by following the queue of edges ordered by cost guarantees that the tree we obtain is a minimal spanning tree of the original connected graph (V, E).

Let us illustrate Prim's Algorithm on the same graph we used for Kruskal's algorithm.

<u>Example</u> Consider

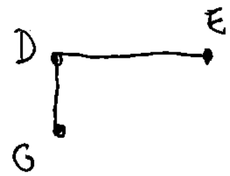


We use the same queue as before $AB, EH, FH, AC, EF, BC, DG, DE, CF, GH, BE, BD, CE$.

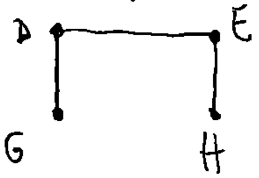
We have a choice which vertex we take to start the algorithm. Let us choose vertex D. So at step 0, we have $(\{D\}, \emptyset)$.

Step 1 We process the queue and find that the first edge in it incident to vertex D is DG. We add vertex G (not already in the subgraph) and edge DG.

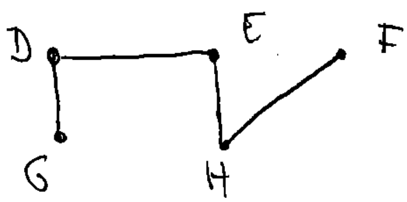
Step 2 We process the queue looking for the first edge incident to either vertex D or vertex G and find DE. We add vertex E (not already in the subgraph) and edge DE.



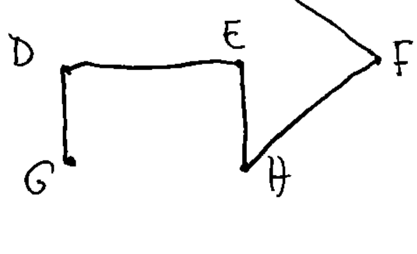
Step 3 We process the queue from the beginning again looking for the first edge incident to D, E or G and find EH. We add vertex E (not already in the subgraph) and edge EH.



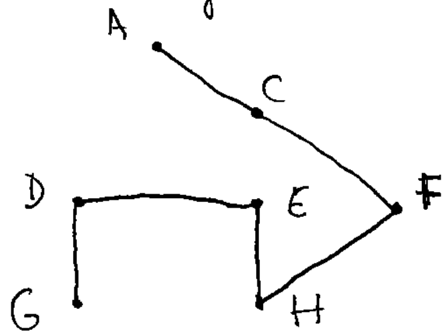
Step 4 We process the queue from the beginning again looking for the first edge incident to D, E, G or H with an endpoint not in the set {D, E, G, H} and find FH. We add vertex F and edge FH.



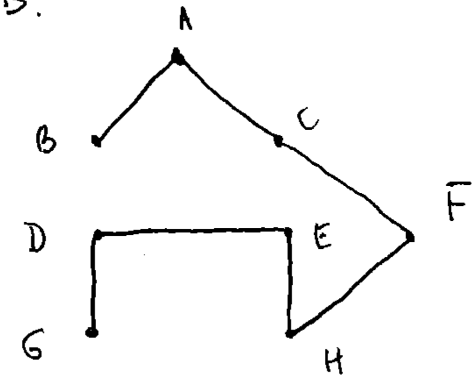
Step 5 We process the queue from the beginning looking for the first edge incident to D, E, F, G or H with the other endpoint not in {D, E, F, G, H} and find CF. We add vertex C and edge CF.



Step 6 We process the queue from the beginning looking for the first edge incident to C, D, E, F, G or H w/ the other endpoint not in {C, D, E, F, G, H} and find AC. We add vertex A and edge AC.



Step 7 We process the queue from the beginning looking for the first edge incident to A, C, D, E, F, G or H w/ the other endpoint not in {A, C, D, E, F, G, H} and find AB. We add vertex B and edge AB.



We have recovered all vertices of the original graph so the algorithm ends here. Prim's Algorithm produced the same tree as Kruskal's in this case given the same queue.