MST Algorithms

MST-KRUSKAL

MST-KRUSKAL(G, w)
1. A ←∅
2. for each vertex v ∈ V[G]
3. do MAKE-SET(v)
4. sort the edges of E into nondecreasing order by weight w
5. for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6. do if FIND-SET(u) ≠ FIND-SET(v)
7. then A ← A ∪ {(u, v)}
8. UNION(u, v)
9. return A

The implementation of Kruskal’s algorithm uses a disjoint-set data structure to maintain several disjoint sets of elements. The definitions of disjoint-set operations are listed in page 499 of the textbook.

We assume that the input undirected graph G = (V,E) is connected. Thus |E| ≥ |V| − 1.

Let us first argue that Kruskal’s algorithm finishes with a tree. Indeed, Kruskal’s algorithm does not create cycles as we never add edges whose endpoints are already connected by the existing edges. And if we assume, for a contradiction, that Kruskal’s algorithm stops before it has a connected graph, then there must be a cut (S, $\bar{S}$) that no selected edge crosses. However, edges crossing (S, $\bar{S}$) do exist, since otherwise G would not be connected. The lightest of these edges crossing (S, $\bar{S}$) would have been selected by the algorithm.

The blue rule ensures that the output of Kruskal’s algorithm is a subset of a minimum spanning tree, so it is a minimum spanning tree.

Running time: O(|E| log |E|) - since sorting is O(|E| log |E|), and all the disjoint set operations are O(log |V|) each (actually, they are faster).
MST-PRIM

MST-PRIM(G,w,r)

1. for each vertex \( u \in V[G] \)
2. do \( \text{key}[u] \leftarrow \infty \)
3. \( \pi[u] \leftarrow \text{NIL} \)
4. \( \text{key}[r] \leftarrow 0 \)
5. \( Q \leftarrow V[G] \)
6. while \( Q \neq \emptyset \)
7. do \( u \leftarrow \text{EXTRACT-MIN}(Q) \)
8. for each \( v \in \text{Adj}[u] \)
9. do if \( v \in Q \) and \( w(u,v) < \text{key}[v] \)
10. then \( \pi[v] \leftarrow u \)
11. \( \text{key}[v] \leftarrow w(u,v) \)
12. \( \text{DECREASE-KEY}(Q, v, \text{key}[v]) \)

The implementation of Prim’s algorithm uses a min-priority queue \( Q \), containing vertices \( v \) using \( \text{key}[v] \) as the key-value. The performance of Prim’s algorithm depends on how we implement \( Q \).

We assume that the input undirected graph \( G = (V,E) \) is connected.

It is an invariant of the algorithm that, if \( \text{key}[v] < \infty \), then the edge from \( v \) to \( \pi[v] \) is a minimum-weight edge among the edges with one endpoint \( v \) and the other among the vertices not in \( Q \).

Moreover, as long as \( Q \neq \emptyset \), there always exists a vertex \( v \in Q \) with \( \text{key}[v] < \infty \). Indeed, if we reach the situation that for all \( v \in Q \), \( \text{key}[v] = \infty \), then there is no edge with one endpoint in \( Q \) and one outside \( Q \), contradicting the fact that \( G = (V,E) \) is connected.

Therefore Prim’s algorithm adds \( |V| - 1 \) edges, connecting all the vertices to \( r \) using the edges defined by \( \pi[] \), and thus it outputs a tree.

The blue rule ensures that the output of Prim’s algorithm is a subset of a minimum spanning tree, so it is a minimum spanning tree.

**Running time:** \( O(|E| \log |V|) \) - since there are \( O(|E|) \) \( \text{DECREASE-KEY}() \) operations and each can be done in \( O(\log |V|) \) with binary heaps. Fibonacci heaps achieve \( O(|E|) \) running time for all the \( \text{DECREASE-KEY}() \) operations, and the running time becomes \( O(|V| \log |V| + |E|) \), with \( O(\log |V|) \) enough to do each of the \( |V| \) \( \text{EXTRACT-MIN}(Q) \) operations.