Input: Graph $G = \langle V, E \rangle$
Output: Edge set T forming a min. span tree
$\triangleright$ Here, the $MakeSet, FindSet, Union$ make use of a disjoint-set structure.

```
procedure KRUSKAL(G)
    T ← ∅                                        ▷ set of edges
    S ← ∅                                        ▷ set of disjoint sets

    for v ∈ V do
        S ← S ∪ MakeSet(v)

    for (u, v) ∈ G.E (ordered by weight) do
        set_u ← FindSet(u)
        set_v ← FindSet(v)
        if set_u ≠ set_v then
            T ← T ∪ {(u, v)}
            Union(set_u, set_v)
return T
```

Input: Graph $G = \langle V, E \rangle$
Output: E of G containing only the edges belonging to a min. span tree
▷ Connectivity can be checked via graph traversal algorithms, namely BFS or DFS.

```
procedure REVERSE_DELETE(G)
    Sort G.E in descending order
    i ← 0
    while i < |E| do
        e ← E[i]
        Delete e                          ▷ i is now at the next edge
        if G not connected then
            E[i] ← e                      ▷ Re-add edge
            i ← i + 1                     ▷ Move on to next edge
return G.E
```

Input: Graph $G = \langle V, E \rangle$
Output: Total cost c of minimum spanning tree of G

```
procedure PRIM_JARNIK(G)
    dist ← initialize to array of size |G.V|
    Q ← empty priority queue with vertex as value and weight as key
    T ← ∅

    s ← some node ∈ G.V  ▷ Here, the choice of the node does not matter
    dist[s] ← 0
    c ← 0

    for v ∈ G.V do
        dist[v] ← ∞
        if v ≠ s then
            Q.add(v, ∞)

    Q.push(s, 0)

    while Q ≠ ∅ do
        u ← Q.remove_min()
        c ← c + dist[u]

        if dist[u] = ∞ then
            return ∞                    ▷ Spanning tree does not exist

        for e = (u, v) ∈ u.edges do
            if e.weight < dist[v] then
                dist[v] ← w
                Q.update_priority(v, w)
return c
```