Minimum Spanning Tree

CS 5633 Analysis of Algorithms

Computer Science
University of Texas at San Antonio

November 13, 2024

Minimum Spanning Tree

Minimum Spanning Tree (MST)

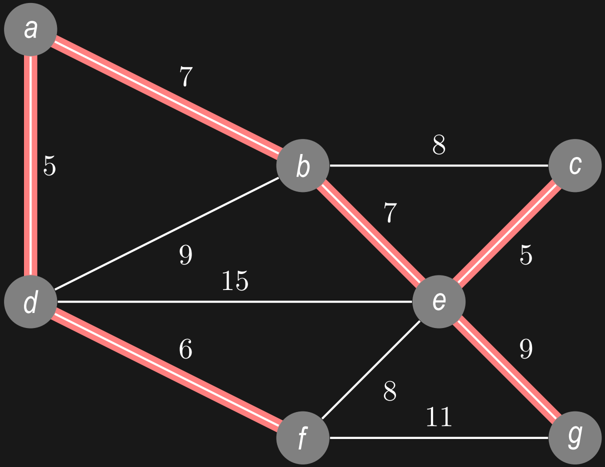
- Suppose we have a set of n locations, and we wish to build a connected network on top of them. The network should be connected (there should be a path between any two locations in the network), and subject to this constraint, we wish to build the network as cheaply as possible.
- Note that a solution to this must be a tree (if the network contains a cycle, we can remove one of the connections to obtain a cheaper network and still satisfy the connectivity constraint).

Minimum Spanning Tree (MST) cont.

- In graph theory, a tree which contains every vertex of the graph is known as a **spanning tree**.
- If we assign weights to the edges of a graph (i.e. the cost to connect two locations in the network), then a **minimum spanning tree** (MST) is a spanning tree such that the sum of the weights of the edges in the tree is minimized.

Example of MST

- The MST consists of red edges.



MST Algorithms

Growing the MST

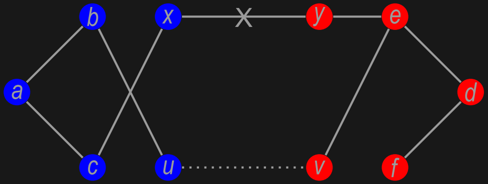
- Considering that MST is an optimization problem, a greedy algorithm strategy may work.
- Intuitively, a greedy strategy would always choose the edge with the lowest weight.
- We can then grow the MST by gradually adding low weight edges to it.

Proof of Greedy Strategy

- We need to prove that always adding the least-weight edges can give us the minimum tree.
- Theorem: Let T be a MST of $G = \{V, E\}$, and let $A \in V$. Suppose $(u, v) \in E$ is the least-weight edge connects $\{A\}$ and $\{V - A\}$. Then $(u, v) \in T$.
- Proof: We will prove by contradiction.
  1. Assume (u, v) is no in T. Let blue vertices be in $\{A\}$, red vertices be in $\{V - A\}$.
  2. One of u and v is blue and one is red because (u, v) connect A and $\{V - A\}$.
  3. Follow the path in T from u to v, and remove an edge $\{x, y\}$ that connects a red vertex to a blue vertex. Such an edge much exist because this path begin from a blue vertex and ends at a red vertex.

Proof of Greedy Strategy cont.

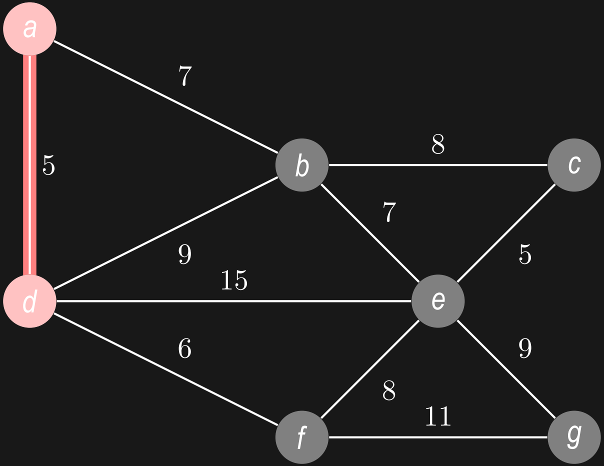
- Theorem: Let T be a MST of $G = \{V, E\}$, and let $A \in V$. Suppose $(u, v) \in E$ is the least-weight edge connects $\{A\}$ and $\{V - A\}$. Then $(u, v) \in T$.
- Proof: continuing from previous page,
  4. We can then add $\{u, v\}$ to T to construct T'. T' is a spanning tree as it connect the red vertices and blue vertices through $\{u, v\}$, while red (blue) vertices are connected to each other using the original edges in T.
  5. Because $\{u, v\}$ has lower weight than $\{x, y\}$, T' is smaller than T, which contradicts the premise that T is the minimum spanning tree. Therefore, the assumption that $\{u, v\}$ is not in T is wrong.



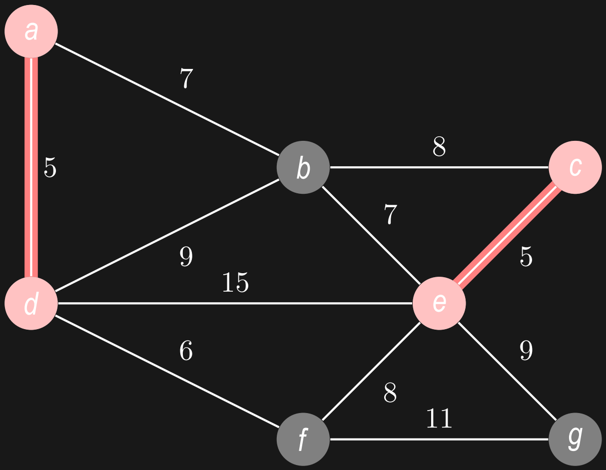
Kruskal's Algorithm

1. Invented by Kruskal in 1956.
2. Maintain a tree T.
3. Repeat the following steps until all edges are examined.
   3.1 Pick the least weight edge (u, v) from the remaining edges $\{E - T\}$ of the graph.
   3.2 If (u, v) causes a cycle in T discard it. Otherwise, add (u, v) to T.

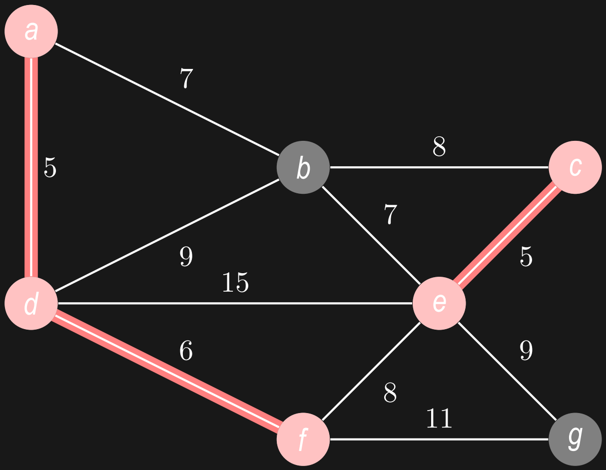
An Example of Kruskal's Algorithm



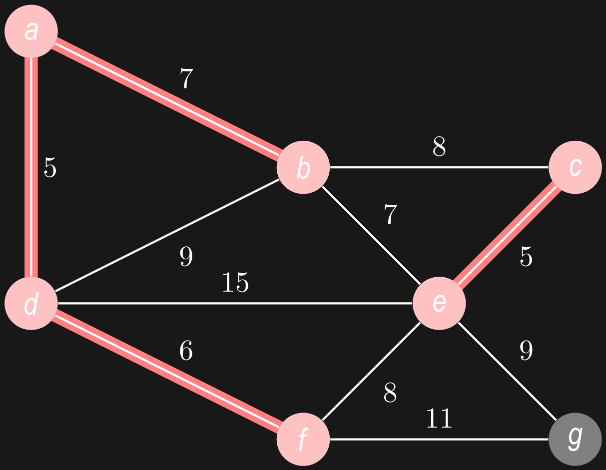
An Example of Kruskal's Algorithm



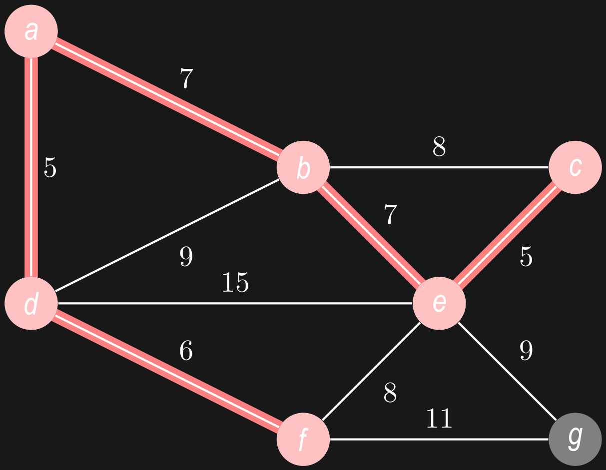
An Example of Kruskal's Algorithm



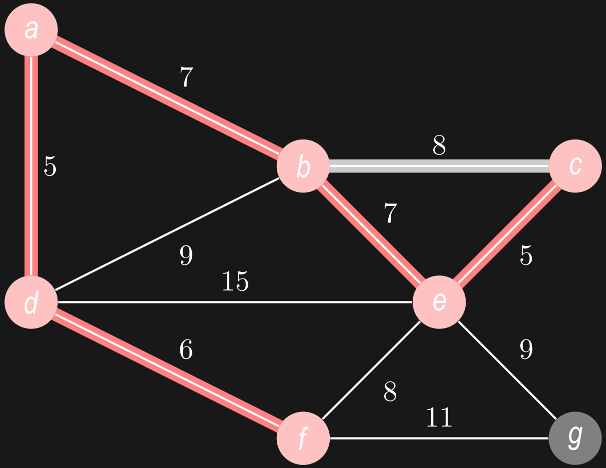
An Example of Kruskal's Algorithm



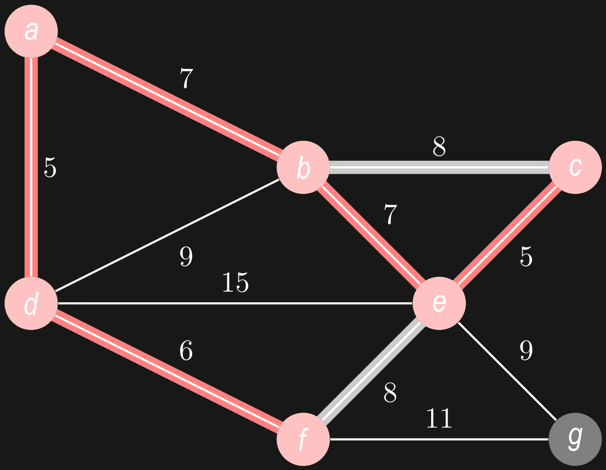
An Example of Kruskal’s Algorithm



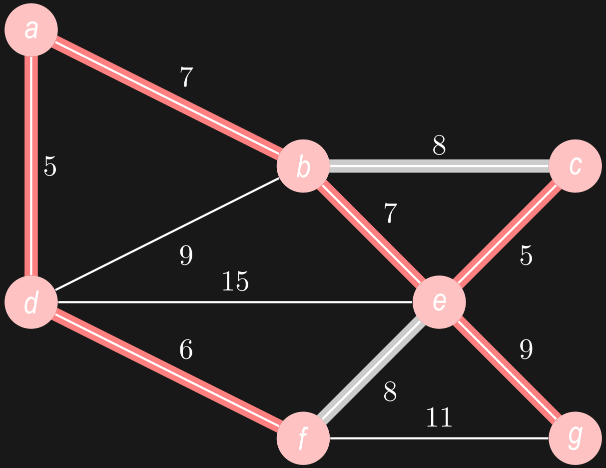
An Example of Kruskal's Algorithm



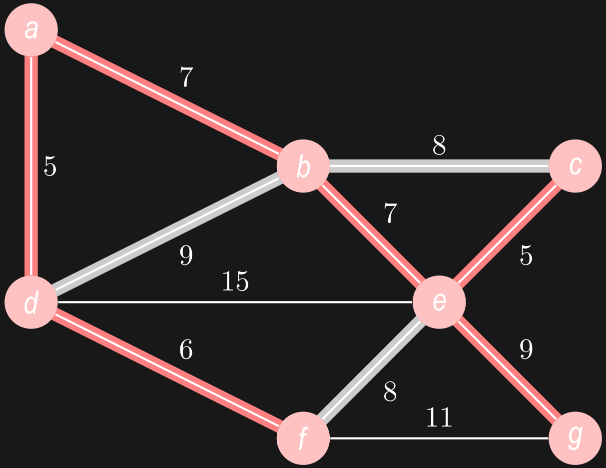
An Example of Kruskal's Algorithm



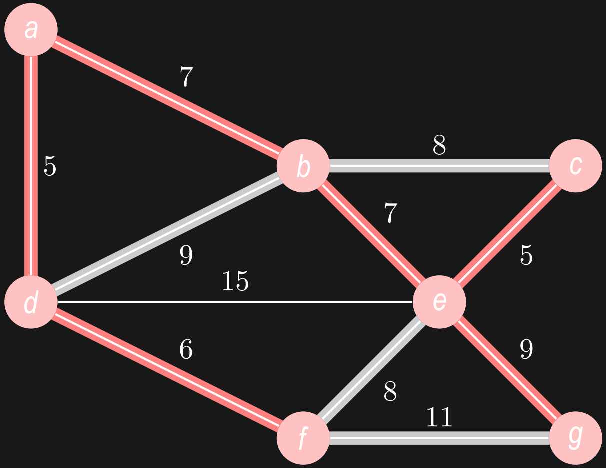
An Example of Kruskal's Algorithm



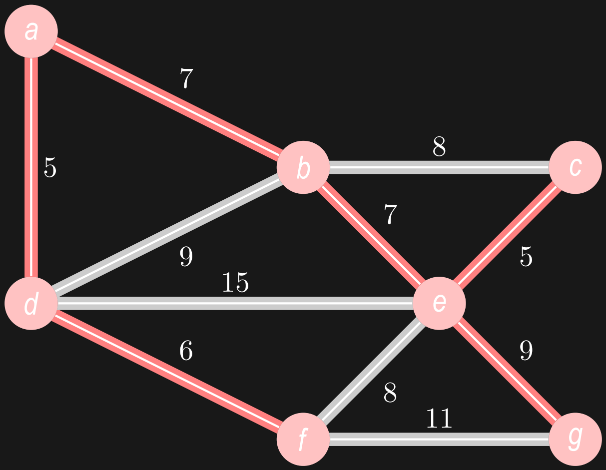
An Example of Kruskal's Algorithm



An Example of Kruskal's Algorithm



An Example of Kruskal's Algorithm



Implementing Kruskal's Algorithm

- The difficult part of Kruskal's algorithm is to detect whether adding an edge creates a cycle.
- Usually, cycles are detected with DFS search with a cost of $O(|E| + |V|)$.
- Given Kruskal's algorithm requires examine very edge, the total cost with DFS is $O(|E|^2 + |E||V|)$.
- To reduce this cost, Kruskal's algorithm is usually implemented with discrete sets.
  - Because Kruskal's algorithm may construct several trees along the way, each discrete set is used to represent one such tree.
  - If an edge connects two vertices within the same discrete set, this edge mush create a cycle in that set and should be discarded.

Pseudo-code of Kruskal's Algorithm

Algorithm 1: Kruskal's algorithm with discrete sets.

```
Function MST_Kruskal(graph G(V, E))
    T = empty set;
    for each u ∈ V do
        // Initially each vertex is a tree by itself represented by its own
          discrete set;
        MAKE_SET(u);
    Sort(E); // In-place sort the edges based on weight from low to high;
    for each edge (u, v) ∈ E do
        // Examine the least-weight edge;
        if FIND_SET(u) ≠ FIND_SET(v) then
            T.add((u, v));
            Merge u's set and v's set into one set;
```

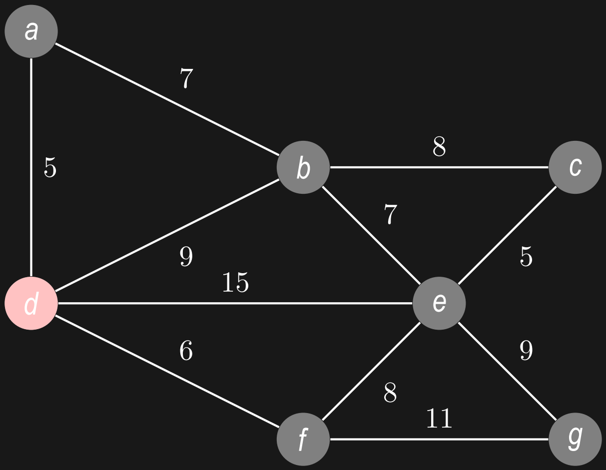
Run-time of Kruskal's Algorithm

- Creating the initial sets takes $O(|V|)$ time.
- Sorting the edges takes $O(|E| \lg |E|)$ time.
- Examining the edges takes $O(|E| \lg(|V|))$ time. Consider using tree sets. Each discrete set find requires $O(\lg(|V|))$ time, and each merge requires $O(1)$ time.
- Therefore, the total cost of Kruskal's algorithm is $O(|E| \lg |E|)$.
- Since $|E| < |V|^2$, we have $\lg |E| < 2 \lg |V|$. The total run-time of Kruskal's algorithm is then $O(|E| \lg |E|) = O(|E| \lg |V|)$.

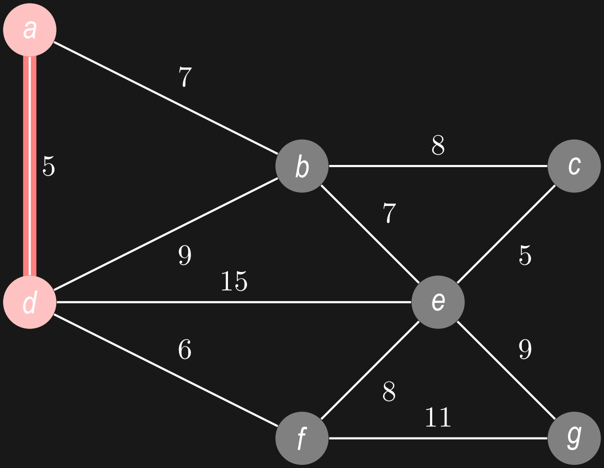
Prim's Algorithm

1. First discovered in 1930 by Jarnik, rediscovered in 1957 by Prim and in 1959 by Dijkstra.
2. Maintain a tree, T, and a set of T's vertices, C. Initially, add a random vertex to C.
3. Repeat the following operation until all vertices are in C.
   3.1 Always pick the least weight edge that connects a vertex in C and a vertex in $\{V - C\}$.

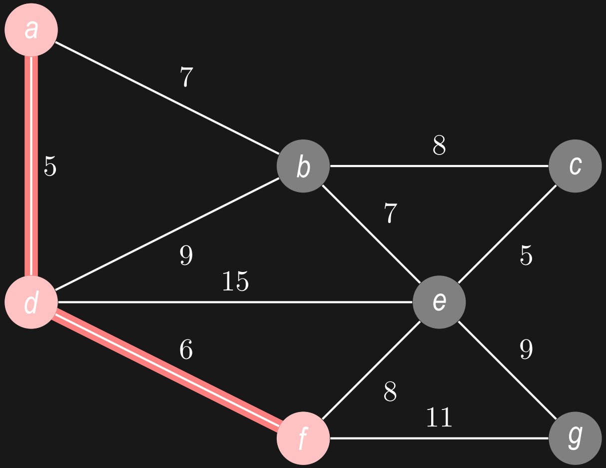
An Example of Prim's Algorithm



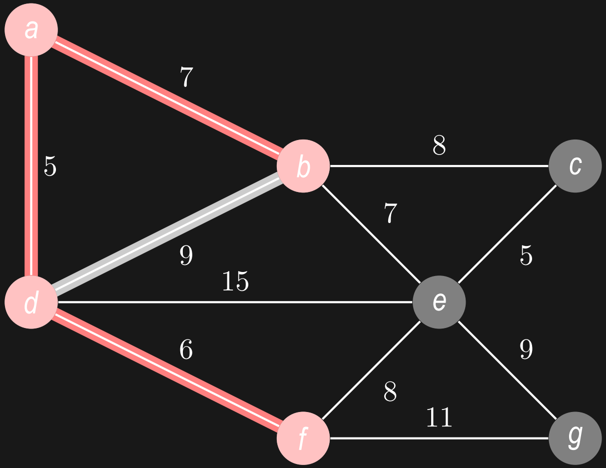
An Example of Prim's Algorithm



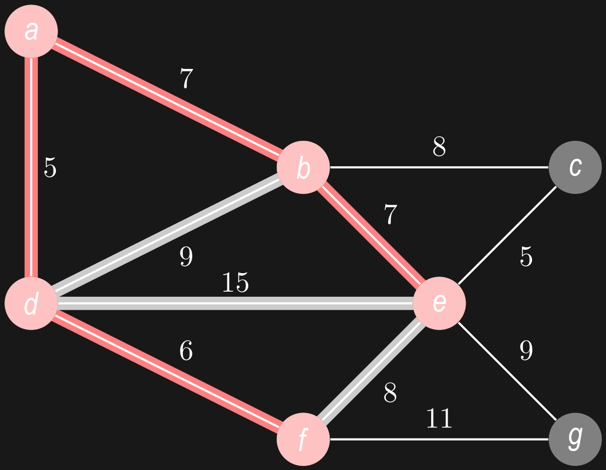
An Example of Prim's Algorithm



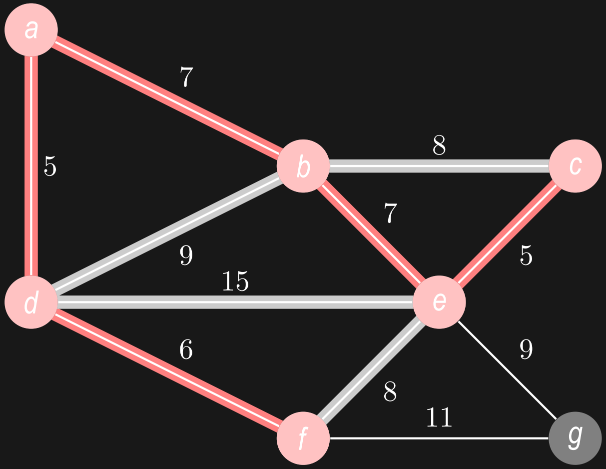
An Example of Prim's Algorithm



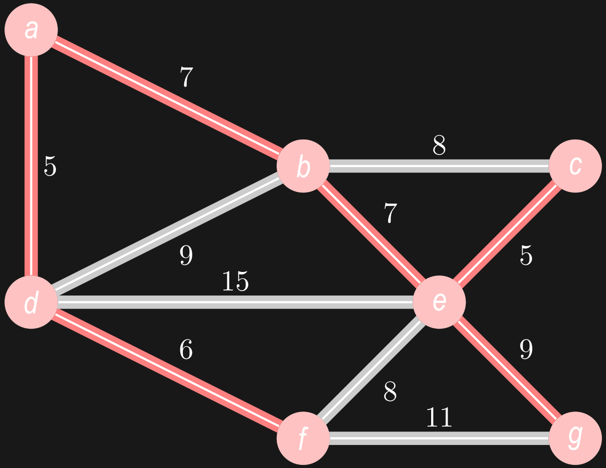
An Example of Prim's Algorithm



An Example of Prim's Algorithm



An Example of Prim's Algorithm



Implementing Prim's Algorithm

- The difficult part of Prim's algorithm is to find the pair of vertices in C and $\{V - C\}$ with least weight edge.
- With a brute-force implementation, we may need to examine $|C| \times (|V| - |C|) = O(V^2)$ edges for each added new edge.
- To reduce this cost, Prim's algorithm is usually implemented with a priority queue to manage the vertices in $\{V - C\}$.
  - The vertices in $\{V - C\}$ are sorted based on the weights of the edges that connect them to the vertices in C.
  - The priority queue Q is used to maintains this sorted order of vertices of $\{V - C\}$. The priority queue allows fast extracting-min and update (reorder) operations.

Pseudo-code of Prim's Algorithm

Algorithm 2: Prim's algorithm with a priority queue.

```
Function MST_Prim(graph G(V, E))
    C = empty set; T = empty set;
    Q = V; // initially all vertices are in {V − C}, which is managed by Q;
    for each u ∈ Q do
        u.key = ∞; // key is weight of the least-weight edge that
          connects u to a vertices in C;
        u.π = NIL; // π is the least-weight edge that connects u to a
          vertices in C;
    while Q is not empty do
        // Add the vertex (of {V − C}) with the least-weight edge to C to
          MST;
        u = Q.Extract_Min();
        C.add(u);T.add(u.π);
        // Update the vertices (of {V − C}) with respect to the new
          vertex (u) of C;
        for each v connected to u do
            if v ∈ Q and wegith(u, v) < v.key then
                v.π=(u, v);
                v.key=weight(u, v);
```

Run-time of Prim's Algorithm

- Assuming a search tree is used as the priority queue.
- Building the search tree takes $O(|V|)$ time.
- There are $|V|$ calls to Extract_Min, while each call costs $O(\lg |V|)$ in a search tree. Total costs of all Extract_Min is $O(|V| \lg |V|)$.
- The inner loop at line 11 examines the edges and updates the tree. Given the are $|E|$ edges and each update costs $O(\lg |V|)$, the total cost of the inner loop is than $O(|E| \lg |V|)$.
- The total cost of Prim's Algorithm is then $O(|V| \lg |V| + |E| \lg |V|) = O(|E| \log |V|)$ (a fully-connected graph has more edges than vertices, $|E| \geqslant |V|$).