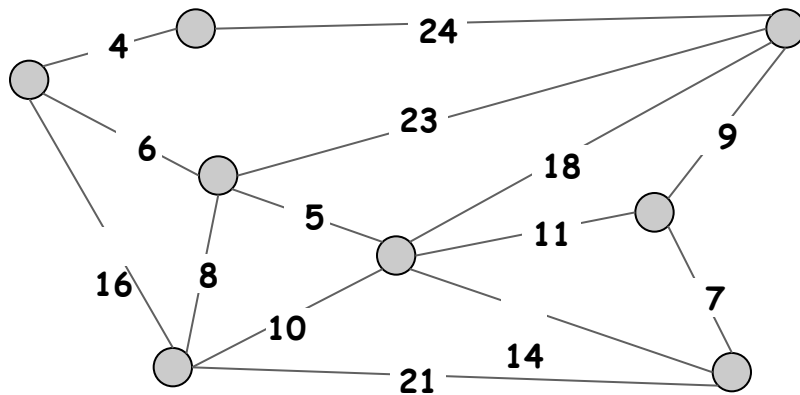


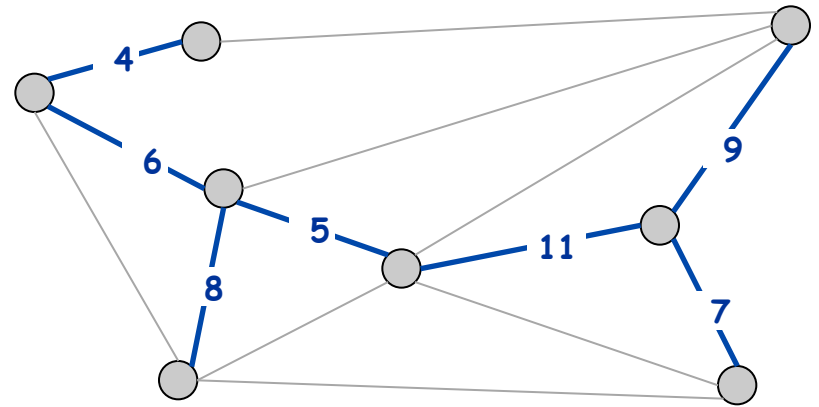
4.5 Minimum Spanning Tree

Minimum Spanning Tree

Minimum spanning tree. Given a connected graph $G = (V, E)$ with real-valued edge weights c_e , an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge weights is minimized.



$G = (V, E)$



$T, \sum_{e \in T} c_e = 50$

Cayley's Theorem. There are n^{n-2} spanning trees of K_n .

↑
can't solve by brute force

Greedy Algorithms

Kruskal's algorithm. Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Reverse-Delete algorithm. Start with $T = E$. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T .

Prim's algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T .

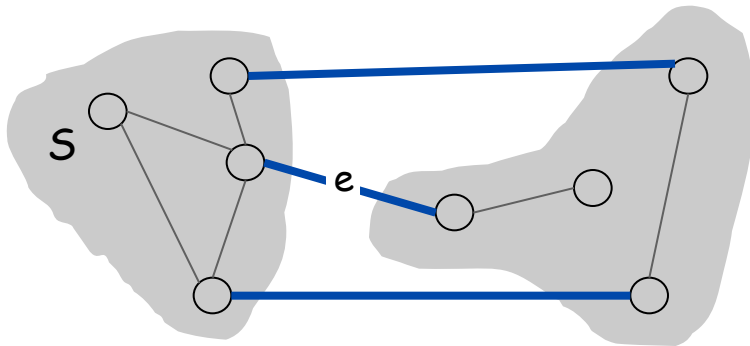
Remark. All three algorithms produce an MST.

Greedy Algorithms

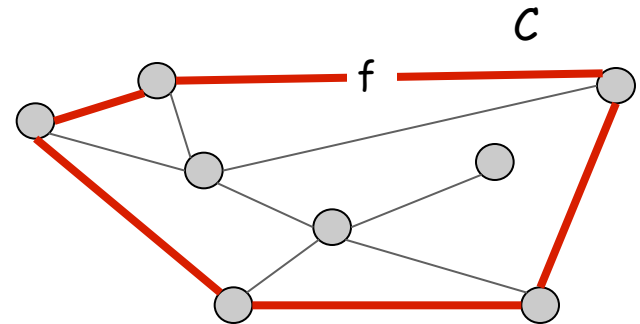
Simplifying assumption. All edge costs c_e are distinct.

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S . Then the MST contains e .

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C . Then the MST does not contain f .



e is in the MST

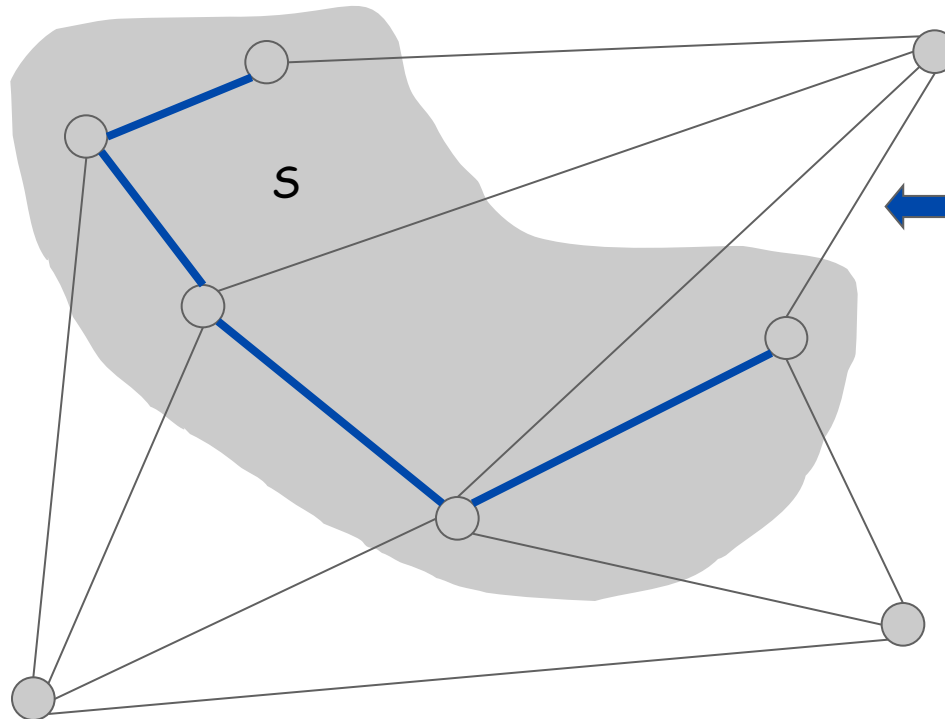


f is not in the MST

Prim's Algorithm: Proof of Correctness

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]

- Initialize S = any node.
- Apply cut property to S .
- Add min cost edge in cutset corresponding to S to T , and add one new explored node u to S .



Implementation: Prim's Algorithm

Implementation. Use a priority queue ala Dijkstra.

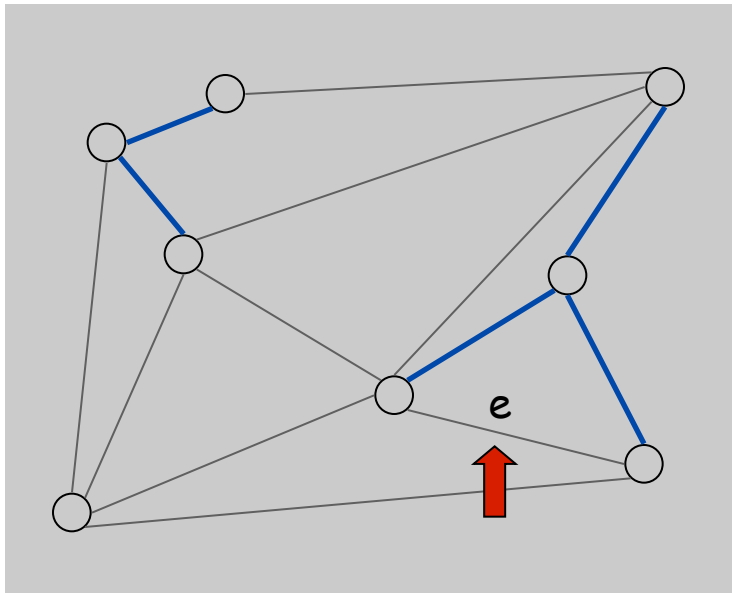
- Maintain set of explored nodes S .
- For each unexplored node v , maintain attachment cost $a[v]$ = cost of cheapest edge v to a node in S .
- $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

```
Prim(G, c) {  
    foreach (v ∈ V) a[v] ← ∞  
    Initialize an empty priority queue Q  
    foreach (v ∈ V) insert v onto Q  
    Initialize set of explored nodes S ← ∅  
  
    while (Q is not empty) {  
        u ← delete min element from Q  
        S ← S ∪ { u }  
        foreach (edge e = (u, v) incident to u)  
            if ((v ∉ S) and (ce < a[v]))  
                decrease priority a[v] to ce  
    }  
}
```

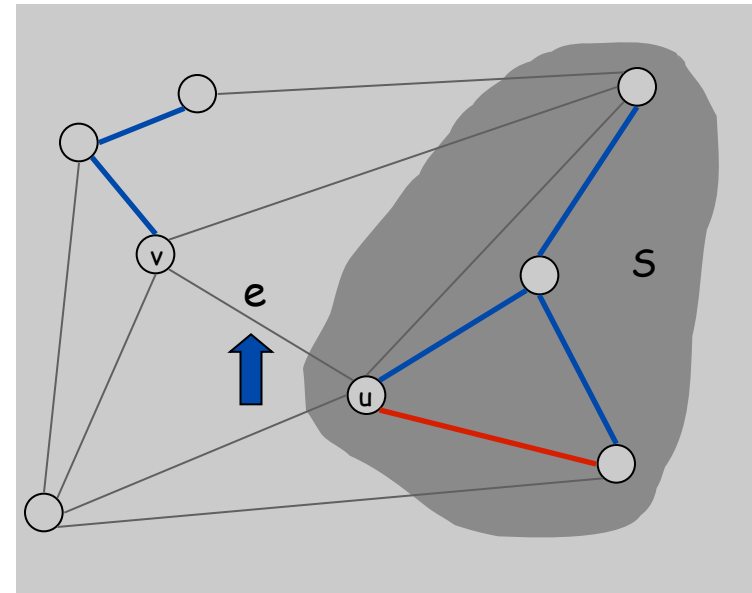
Kruskal's Algorithm: Proof of Correctness

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding e to T creates a cycle, discard e according to cycle property.
- Case 2: Otherwise, insert $e = (u, v)$ into T according to cut property where S = set of nodes in u 's connected component.



Case 1



Case 2