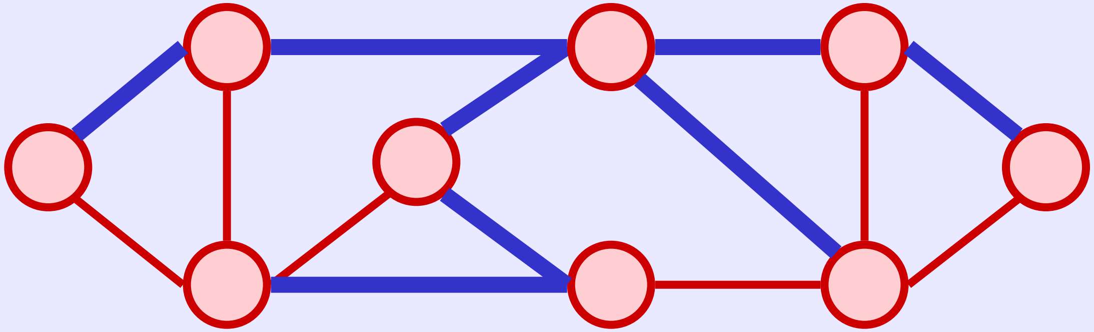
Chapter 21: Minimum Spanning Tree

About this lecture

- What is a Minimum Spanning Tree?
- The Greedy Choice Lemma
  - Kruskal's Algorithm O(E log V)
  - Prim's Algorithm O(E log V)

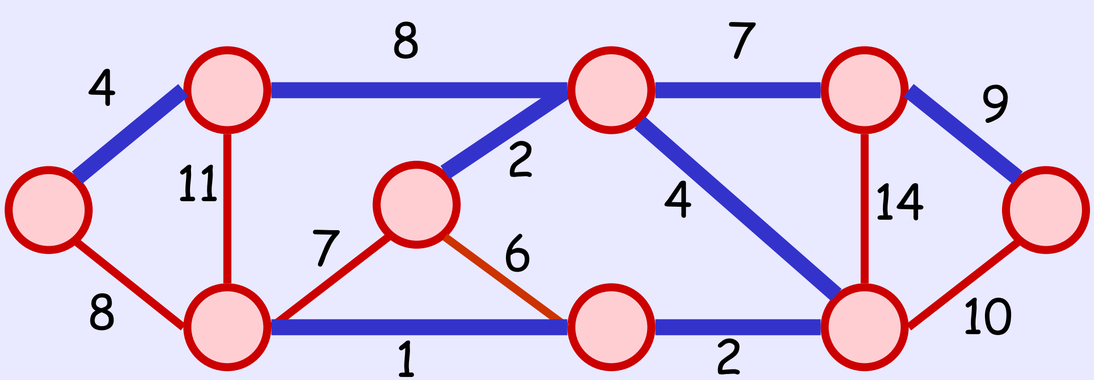
Minimum Spanning Tree

- Let $G = (V, E)$ be an undirected, connected graph
- A spanning tree of G is a tree, using only edges in E, that connects all vertices of G



Minimum Spanning Tree

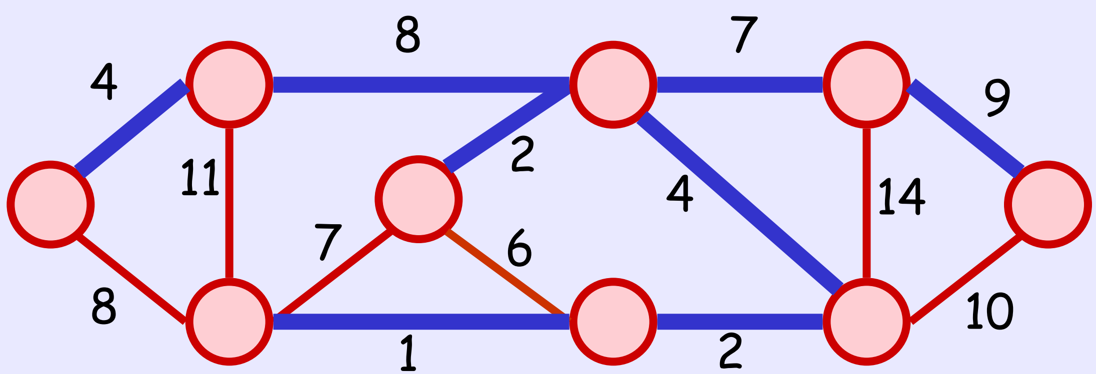
- Sometimes, the edges in G have weights
  - weight ⇔ cost of using the edge
- A minimum spanning tree (MST) of a weighted G is a spanning tree such that the sum of edge weights is minimized

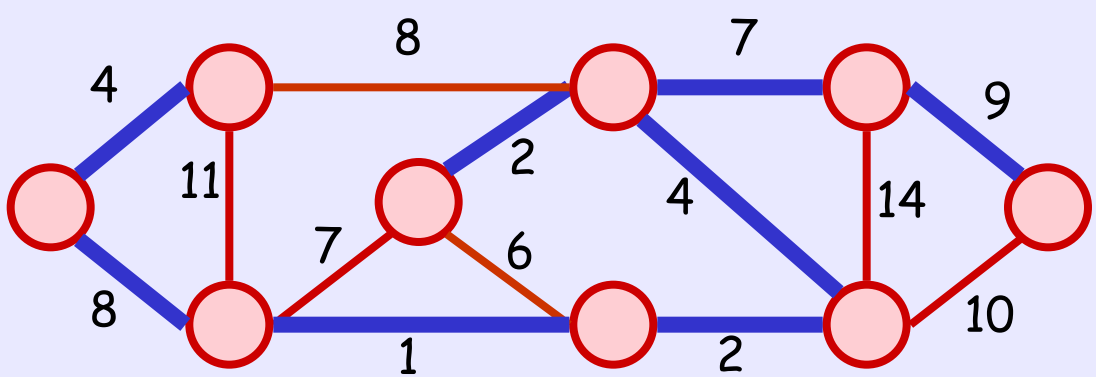


Total cost = 4 + 8 + 7 + 9 + 2 + 4 + 1 + 2 = 37

Minimum Spanning Tree

- MST of a graph may not be unique





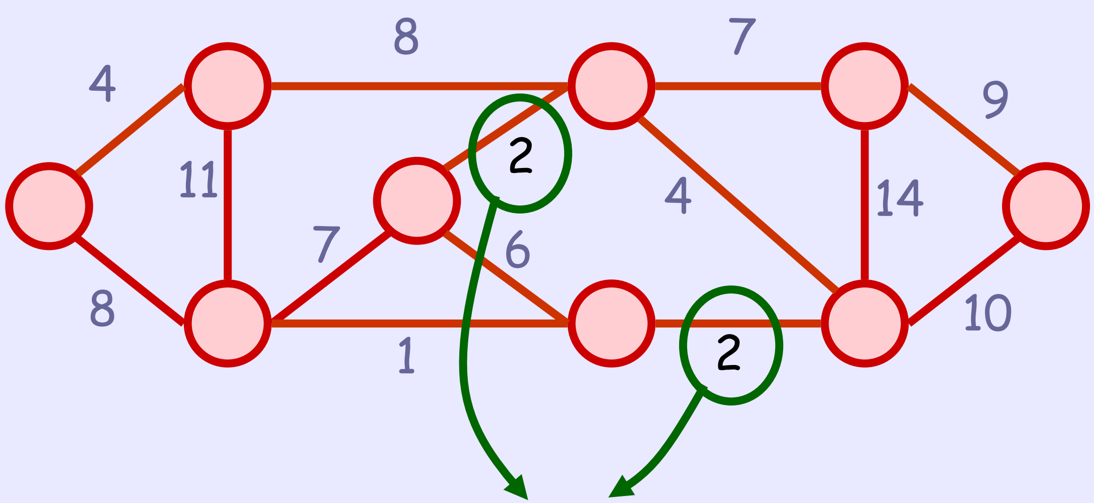
Designing a greedy algorithm

- **Greedy-choice property:** A global optimal solution can be achieved by making a local optimal (optimal) choice.

- **Optimal substructure:** An optimal solution to the problem contains its optimal solution to subproblem.

Greedy Choice Lemma

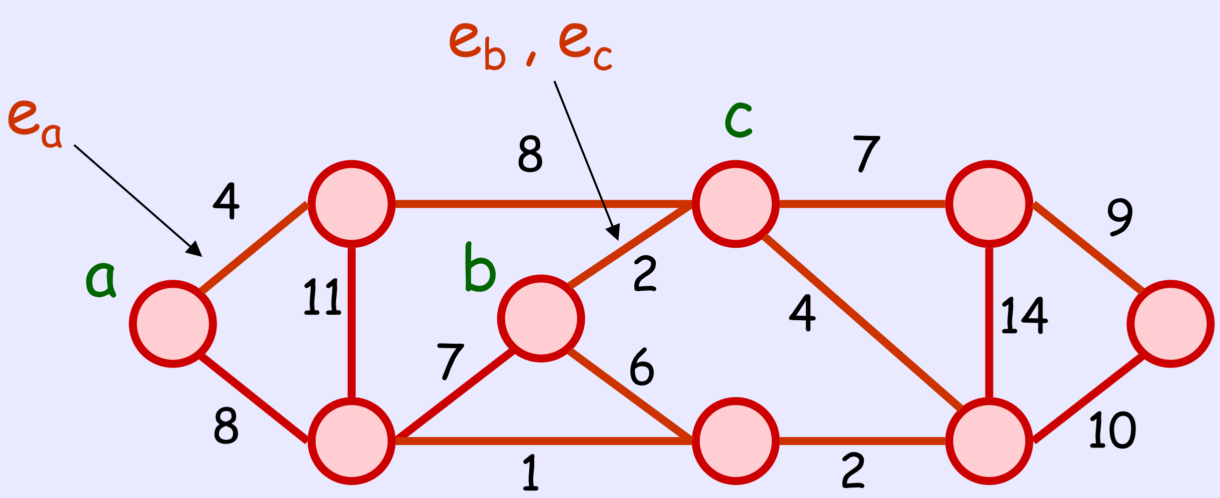
- Suppose all edge weights are distinct
  - If not, we give an arbitrary ordering among equal-weight edges
- E.g.,



Give an arbitrary ordering among these two edges, so that one costs "fewer" than the other

Greedy Choice Lemma

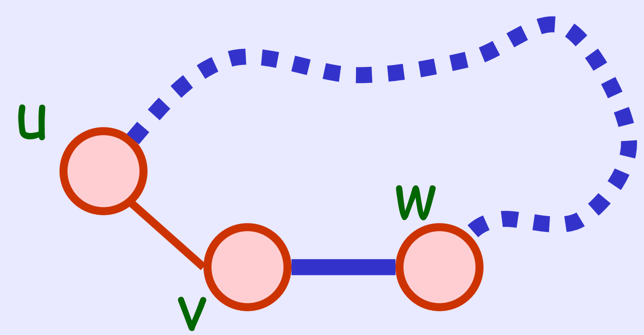
- Let e_v to be the cheapest edge adjacent to v, for each vertex v



Theorem: The minimum spanning tree of G contains every e_v

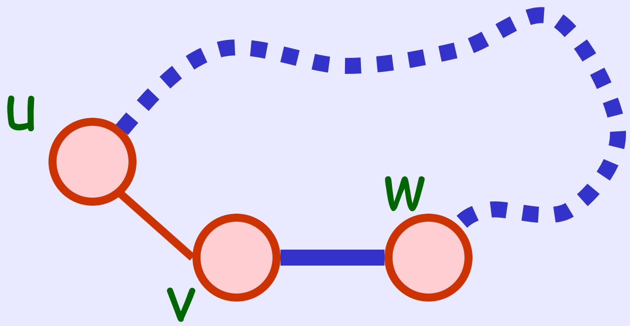
Proof

- Recall that all edge weights are distinct
- Suppose on the contrary that MST of G does not contain some edge $e_v = (u,v)$
- Let T = optimal MST of G
- By adding $e_v = (u,v)$ to T, we obtain a cycle $u, v, w, \ldots, u$ [why??]

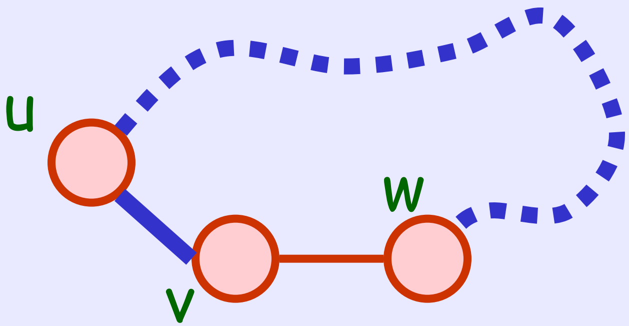


Proof

- By our choice of e_v , we must have weight of (u,v) cheaper than weight of (v,w) to T



- If we delete (v,w) and include e_v, we obtain a spanning tree cheaper than T



➔ contradiction !!

Optimal Substructure

Let E' = a set of edges which are known to be in an MST of $G = (V, E)$

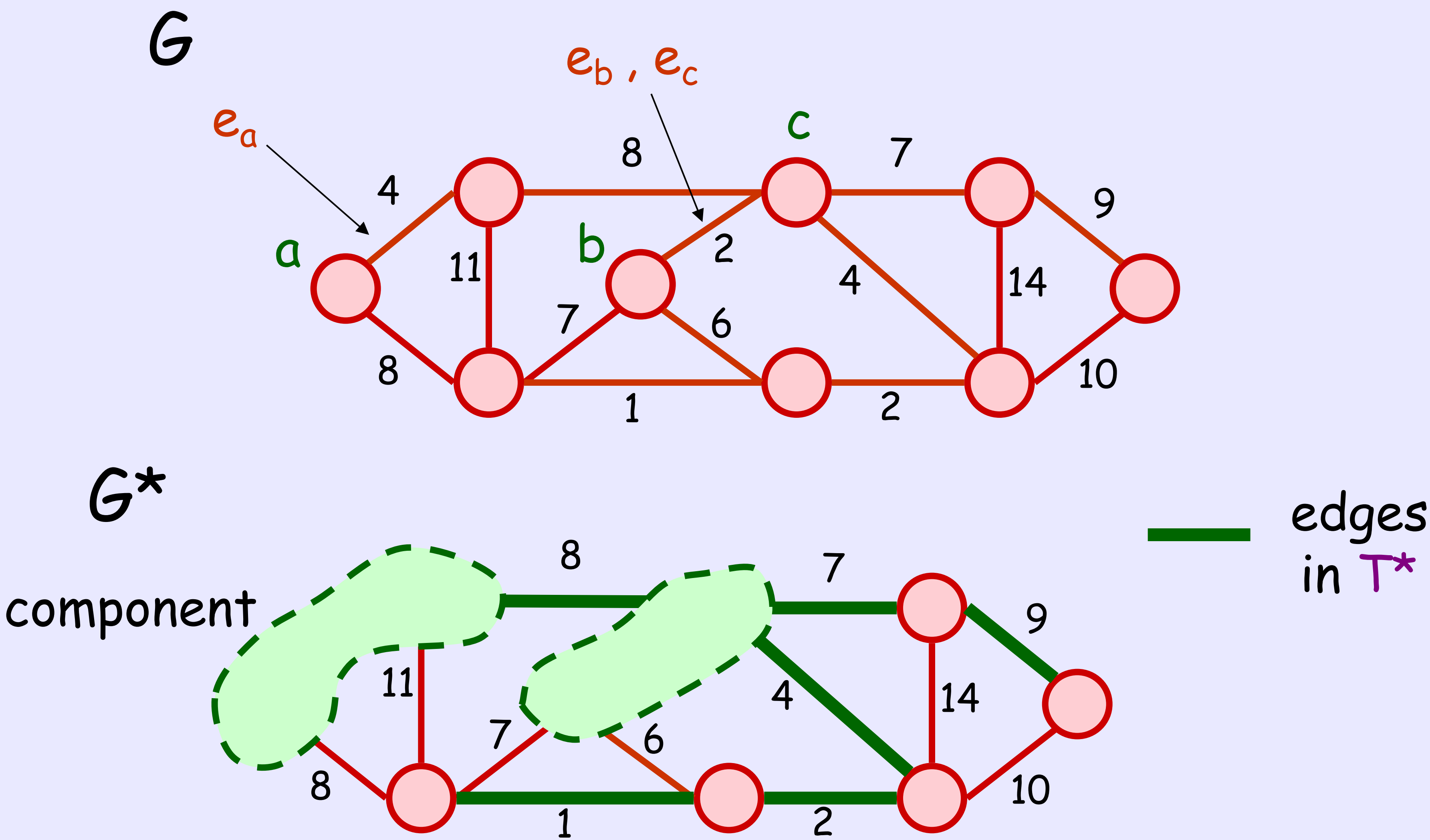
Let G^* = the graph obtained by contracting each component of $G' = (V, E')$ into a single vertex

Let T^* be (the edges of) an MST of G^*

Theorem: $T^* \cup E'$ is an MST of G

Proof: (By contradiction)

Example


G
e_b , e_c
e_a
c
a
b
8
7
4
9
11
2
4
14
7
6
8
10
1
2
G*
edges in T*
8
7
component
9
11
4
14
7
6
8
10
1
2

Proof

- Let T and W(T) denote the MST of G and its corresponding cost, respectively
- If $T^* \cup E'$ is not an MST of G, then $W(T) < W(T^*) + W(E')$
- Since E' is a set of edges in MST of G, it implies that $W(T) - W(E') = W(T^*)$ because T^* be the edges of an MST of G^*

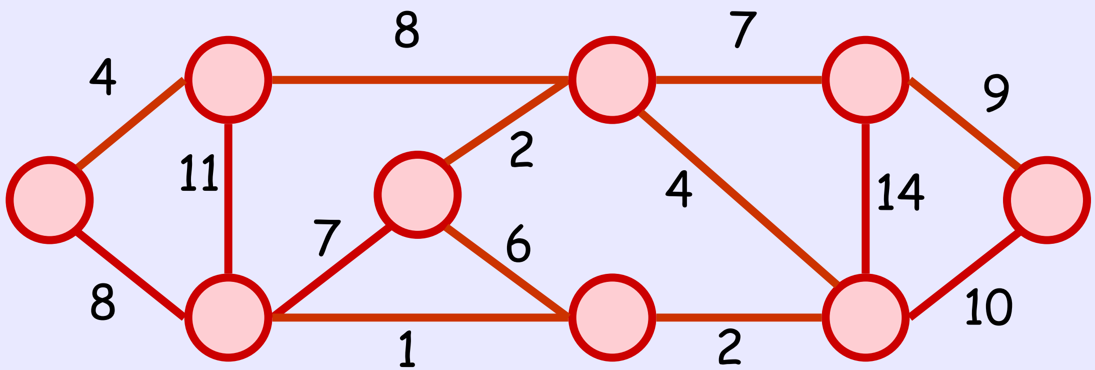
➔ Contradiction

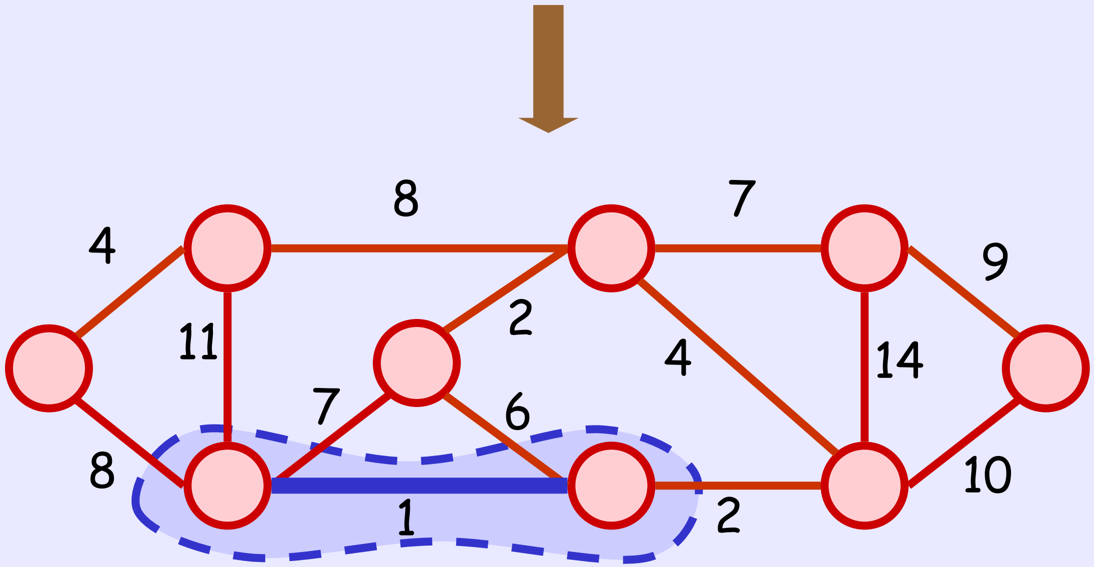
Kruskal's Algorithm

Kruskal-MST(G)

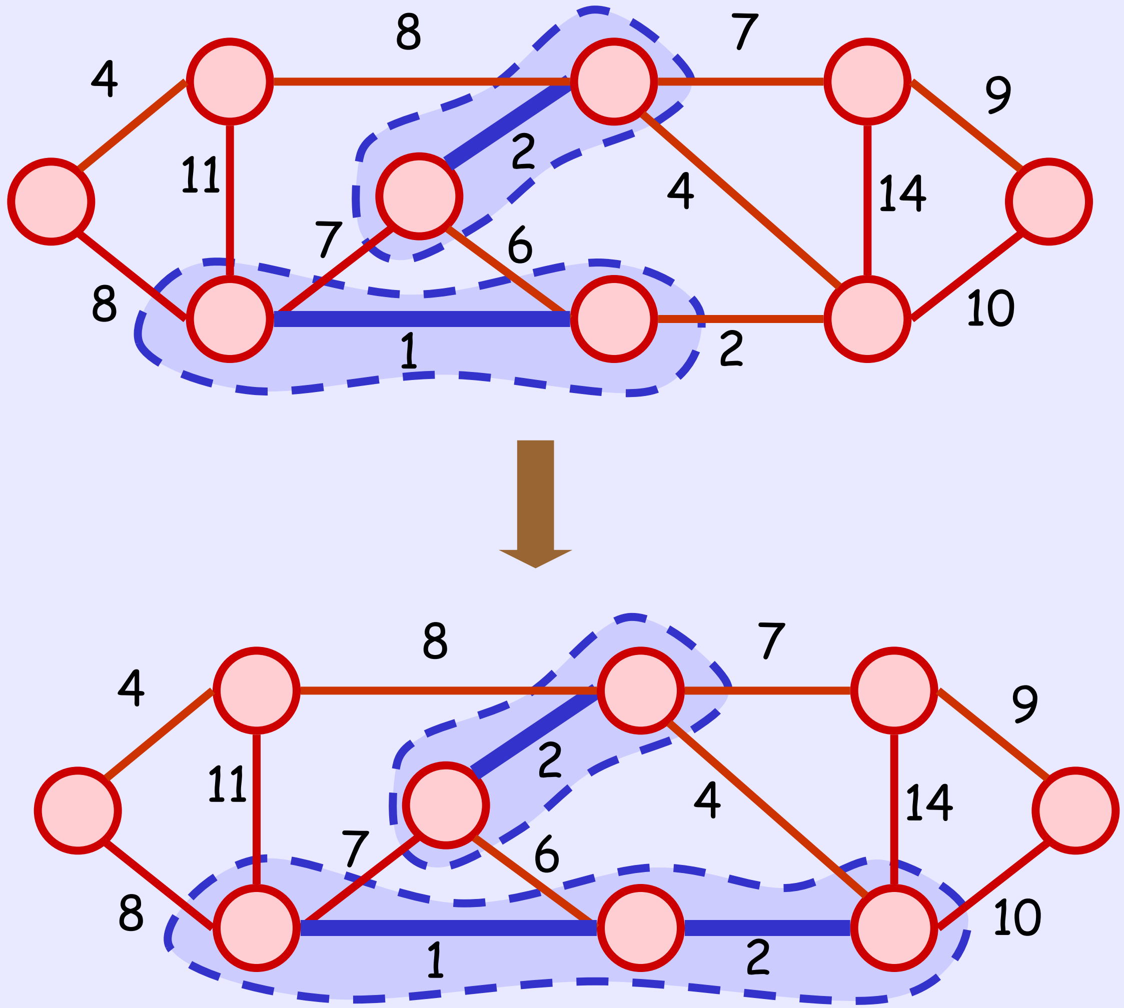
- Find the cheapest (non-self-loop) edge (u,v) in G
- Contract (u,v) to obtain G^*
- Kruskal-MST(G^*)

Example

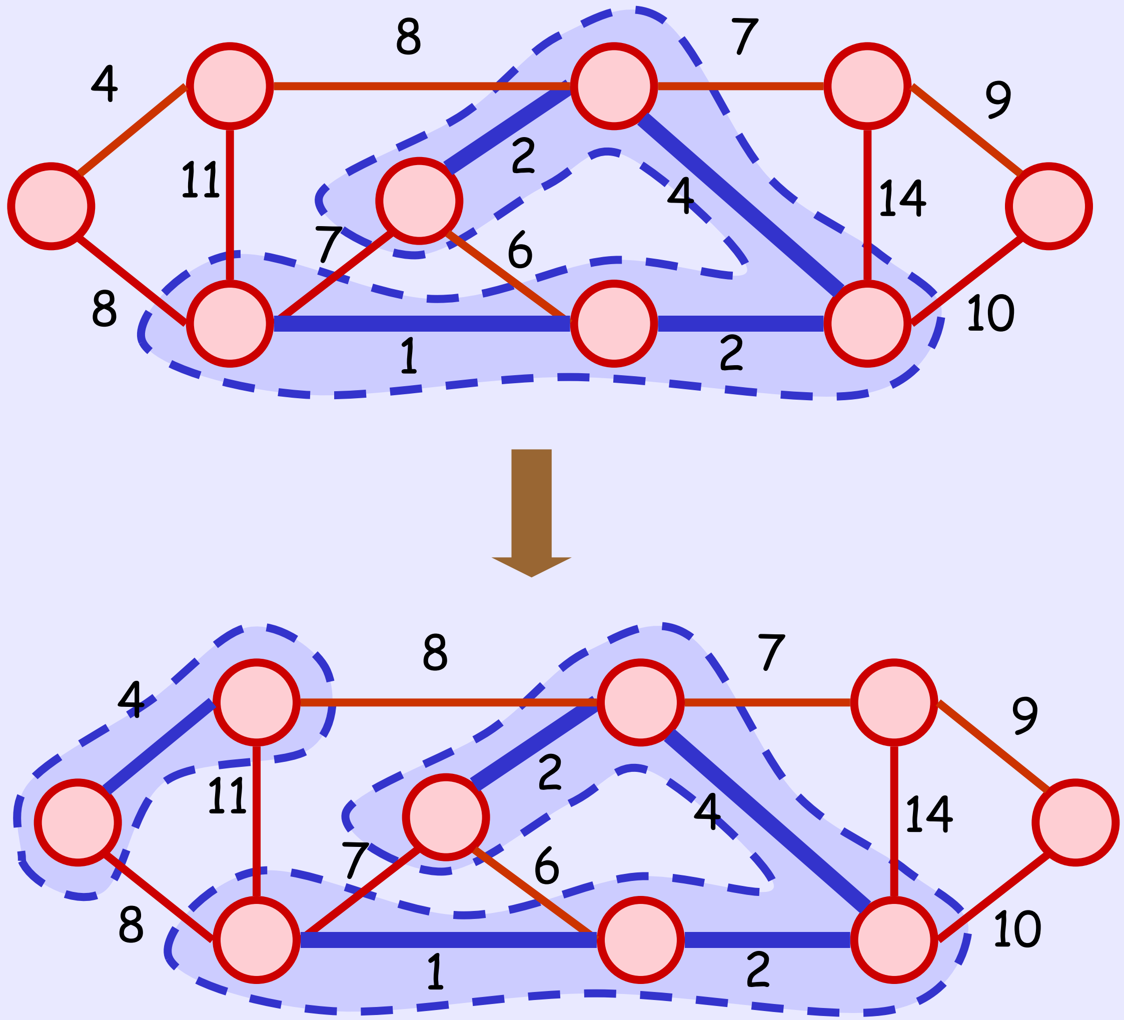




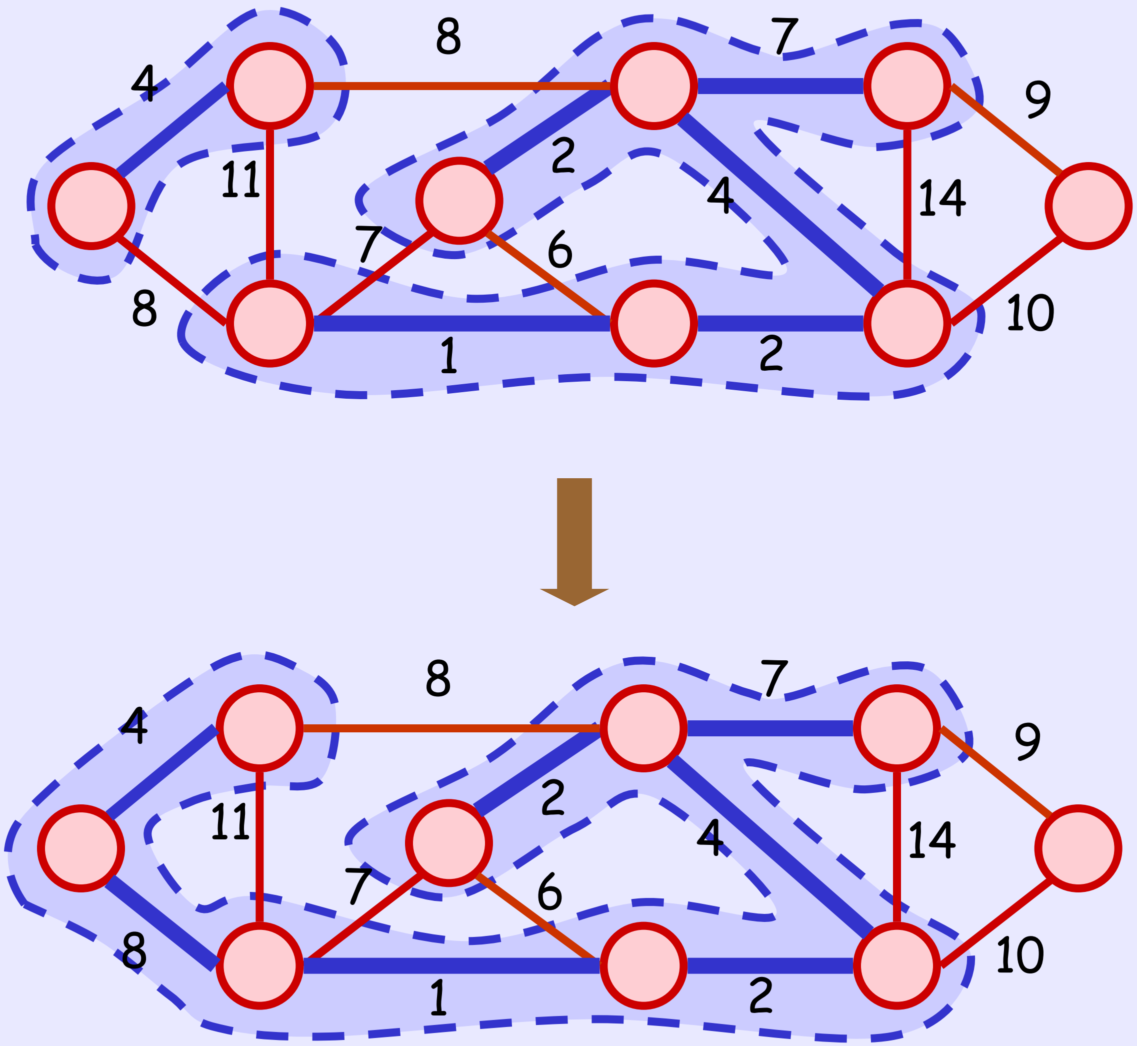
Example



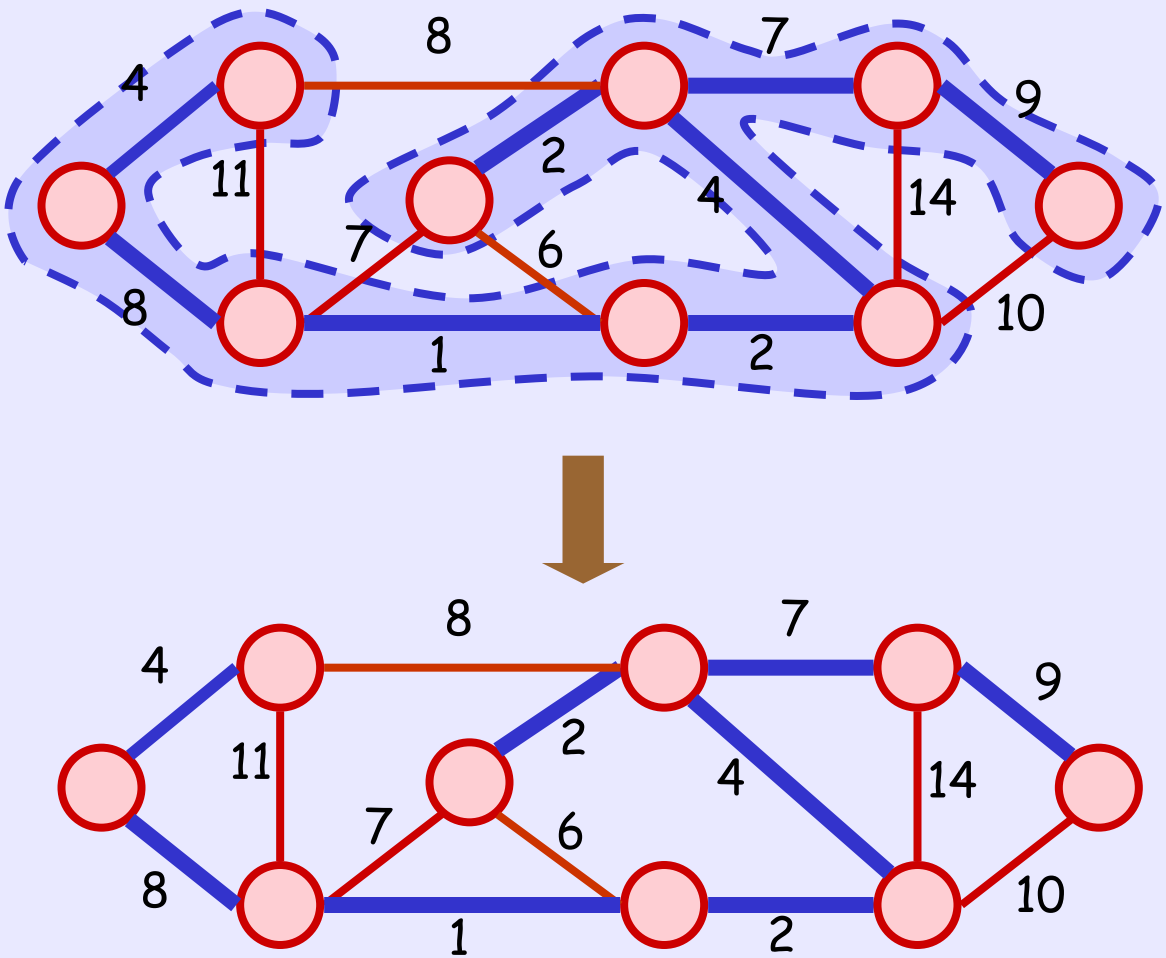
Example



Example



Example



Performance

- Kruskal's algorithm can be implemented efficiently using Union-Find (Chapter 19)
- First, sort edges according to the weights
- At each step, pick the cheapest edge
  - If end-points are from different component, we perform Union (and include this edge to the MST)

  → Time for Union-Find = $O(E\alpha(E))$

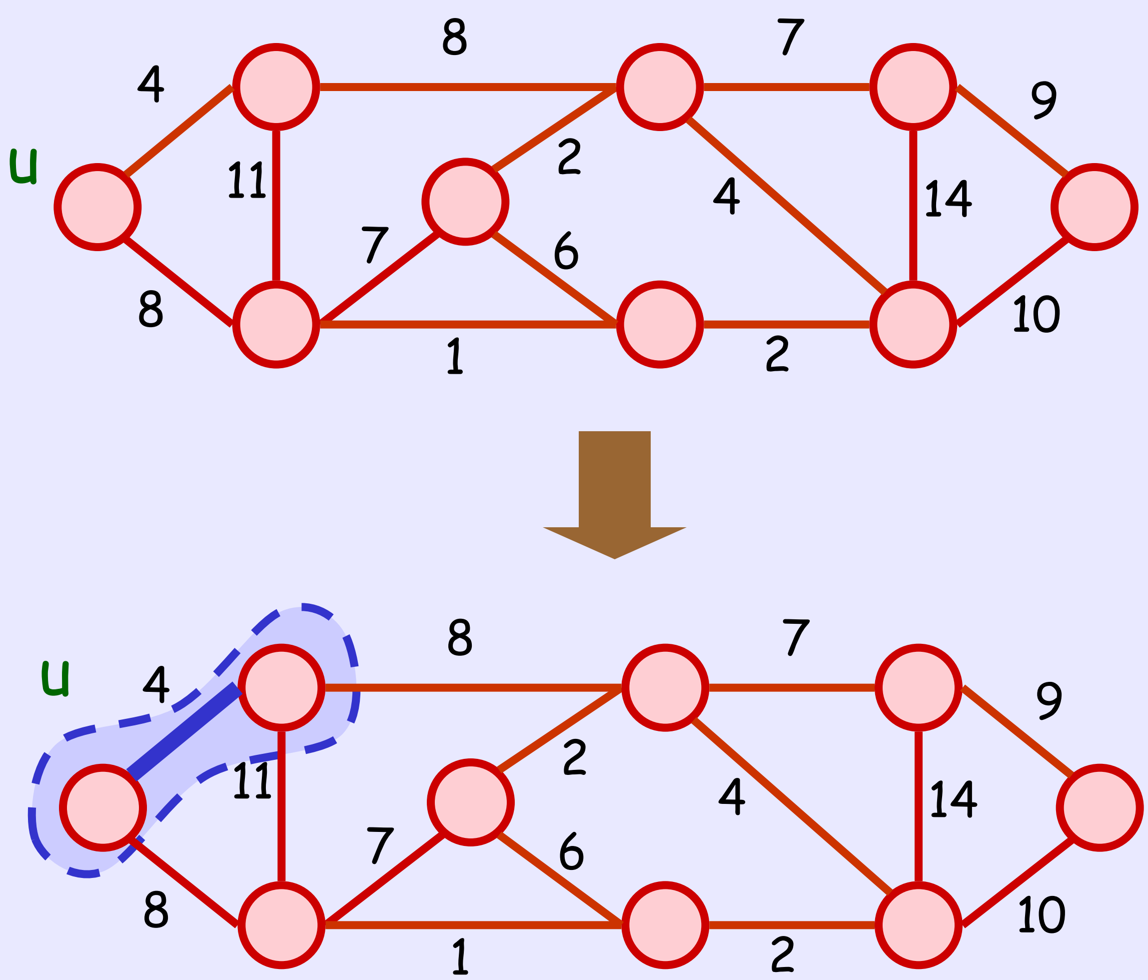
Total Time: $O(E \log E + E\,\alpha(E)) = O(E \log V)$

Prim's Algorithm

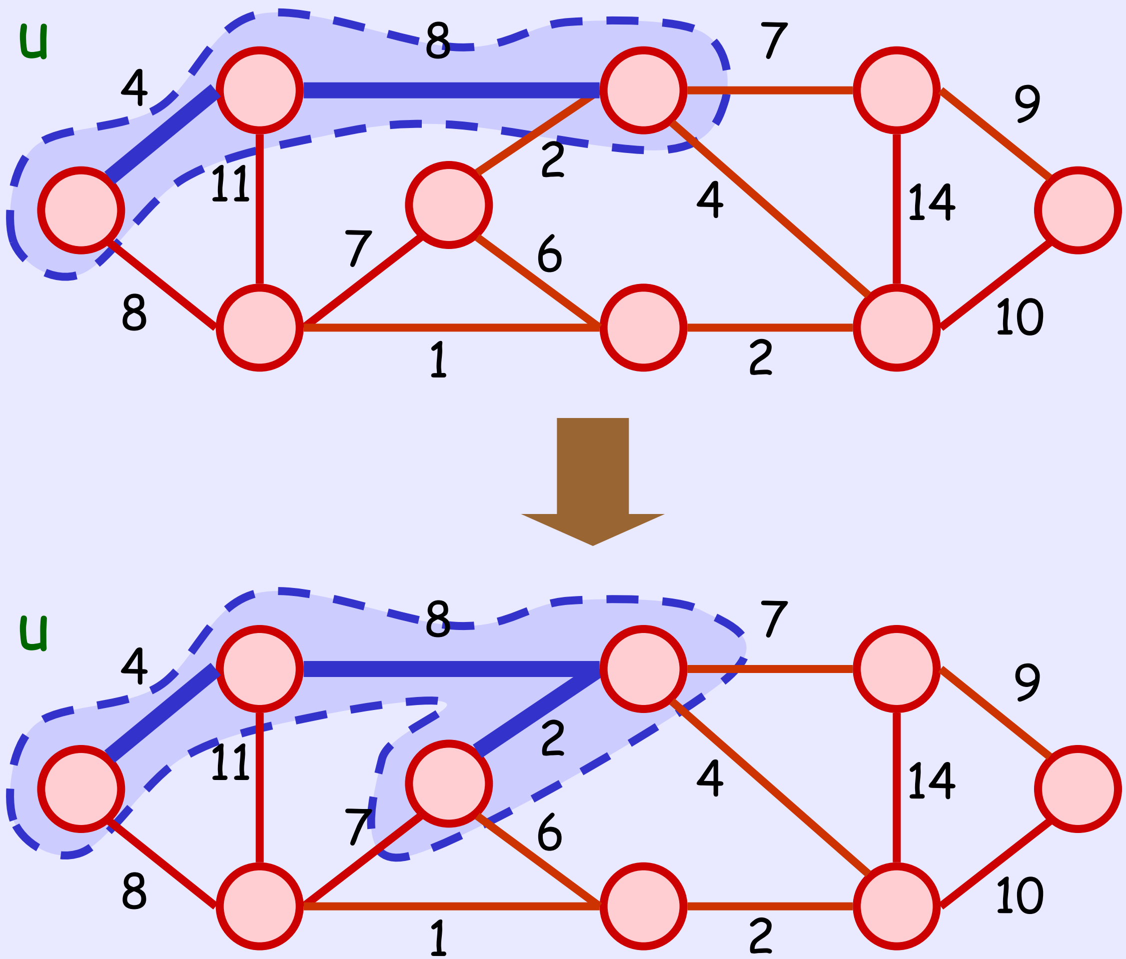
Prim-MST(G, u)

- Set u as the source vertex
- Find the cheapest (non-self-loop) edge from u, say, (u,v)
- Merge v into u to obtain G^*
- Prim-MST(G^*, u)

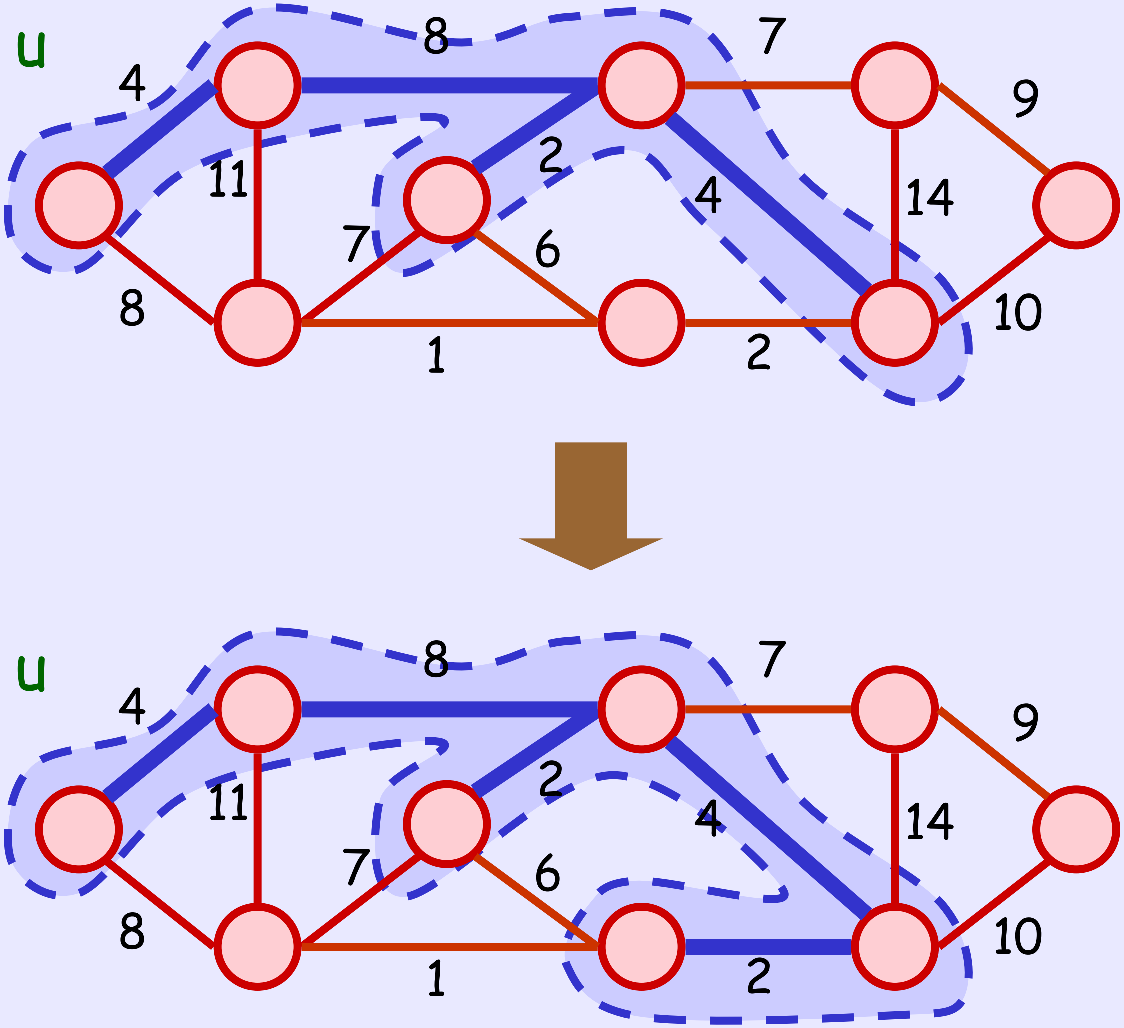
Example



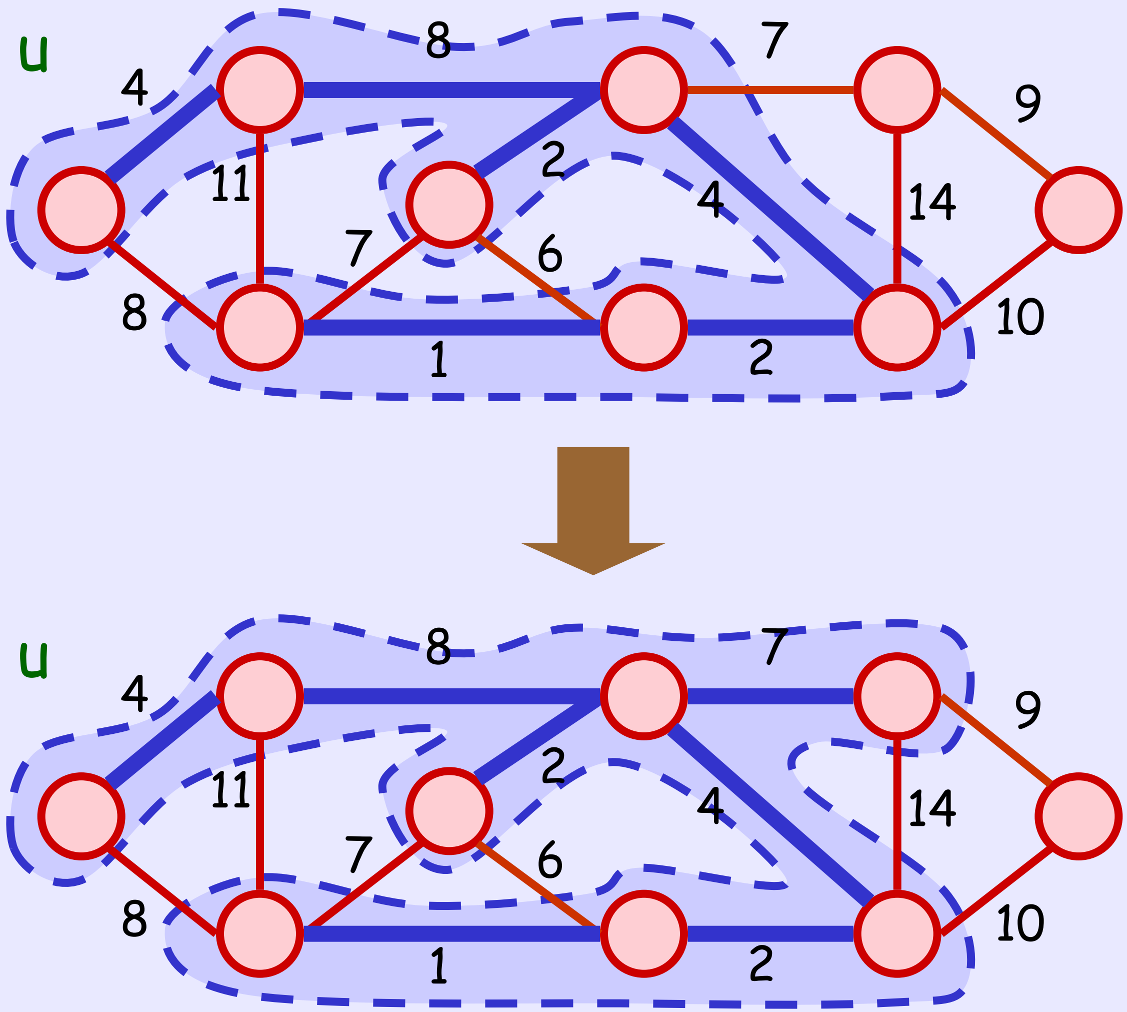
Example



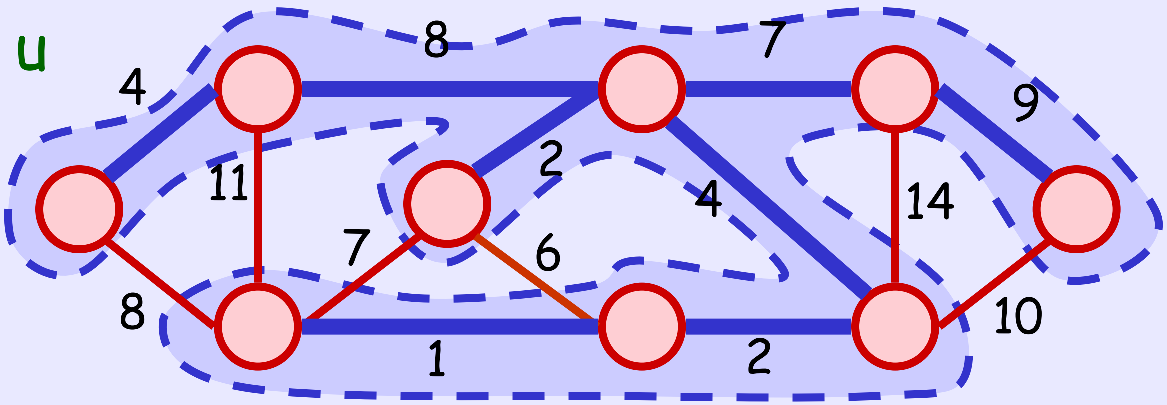
Example

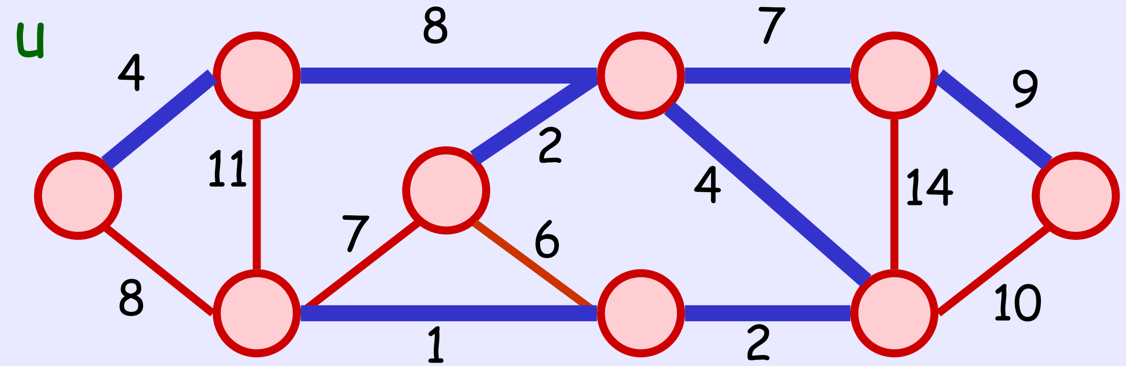


Example



Example





Performance

- Prim's algorithm can be implemented efficiently using Binary Heap H:
- First, insert all edges adjacent to u into H
- At each step, extract the cheapest edge
    - If an end-point, say v, is not in MST, include this edge and v to MST
        - Insert all edges adjacent to v into H
- At most O(E) Insert/Extract-Min

➔ Total Time: $O(E \log E) = O(E \log V)$

Practice at home

- Exercises: 21.1-1, 21.1-4, 21.1-5, 21.1-6, 21.1-7, 21.1-9, 21.1-10
- Exercises: 21.2-1, 21.2-2, 21.2-4, 21.2-7