NPTEL MOOC,JAN-FEB 2015
Week 4, Module 6

DESIGN AND ANALYSIS OF ALGORITHMS

Spanning trees: Prim's algorithm

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Spanning tree

- Weighted undirected graph, G = (V,E,w)
  - Assume G is connected
- Identify a spanning tree with minimum weight
  - Tree connecting all vertices in V
- **Strategy 1:**
  - Start with minimum cost edge
  - Keep extending the tree with smallest edge

Prim's algorithm

```
algorithm Prim_V1

Let e = (i,j) be minimum cost edge in E

TE = [e]       //List of edges in tree
TV = [i,j]     //List of vertices connected by tree

for i = 3 to n
   choose edge f = (u,v) of minimum cost
      such that u in TV and v not in TV
   TE.append(f)
   TV.append(v)

return(TE)
```

Correctness

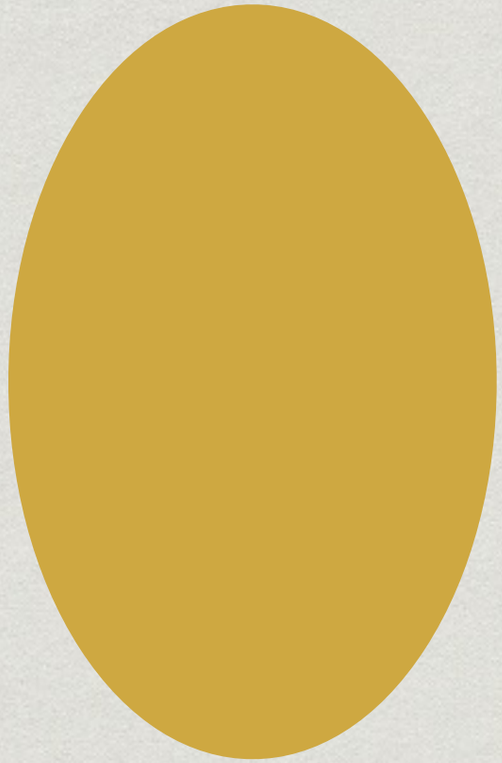
- Prim's algorithm is a greedy algorithm
  - Like Dijkstra's single source shortest path
- A local heuristic is used to decide which edge to add next to the tree
- Choices made are never reconsidered
- Why does this sequence of local choices achieve a global optimum?

Minimum separator lemma

- Let V be partitioned into two non-empty sets U and W = V - U
- Let e = (u,w) be minimum cost edge with u in U and w in W
  - Assume all edges have different weights (relax this condition later)
- Then every minimum cost spanning tree must include e

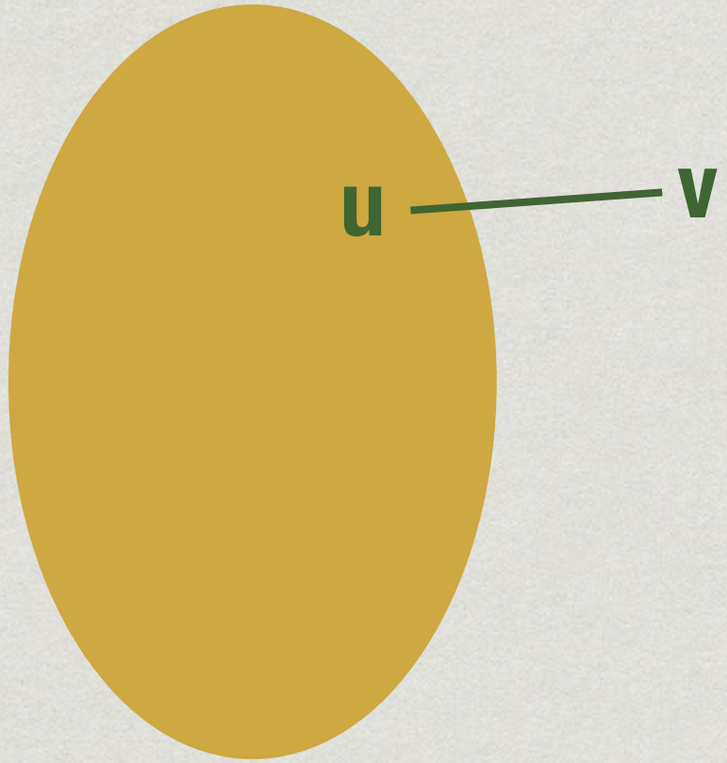
Minimum separator lemma

- Let T be a minimum cost spanning tree, e = (u,w) not in T
- u in U and w in W are connected by a path p in T
- p starts in U and ends in W
- Let f = (u’,w’) be the first edge on p such that u’ in U and w’ in W
- Drop f and add e to get a smaller spanning tree



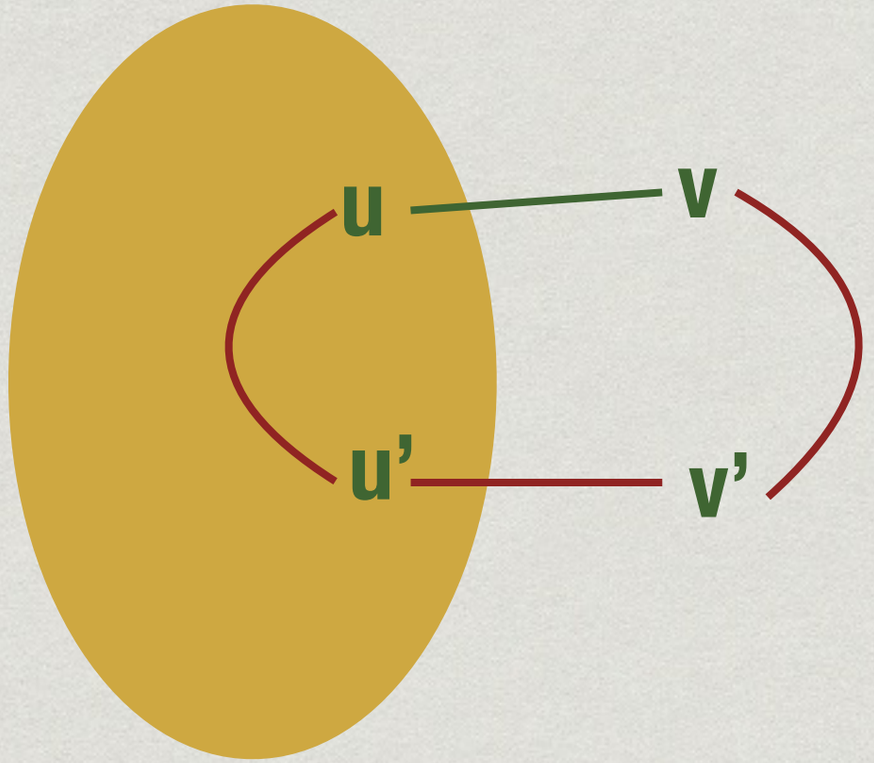
Minimum separator lemma

- Let T be a minimum cost spanning tree, e = (u,w) not in T
- u in U and w in W are connected by a path p in T
- p starts in U and ends in W
- Let f = (u’,w’) be the first edge on p such that u’ in U and w’ in W
- Drop f and add e to get a smaller spanning tree



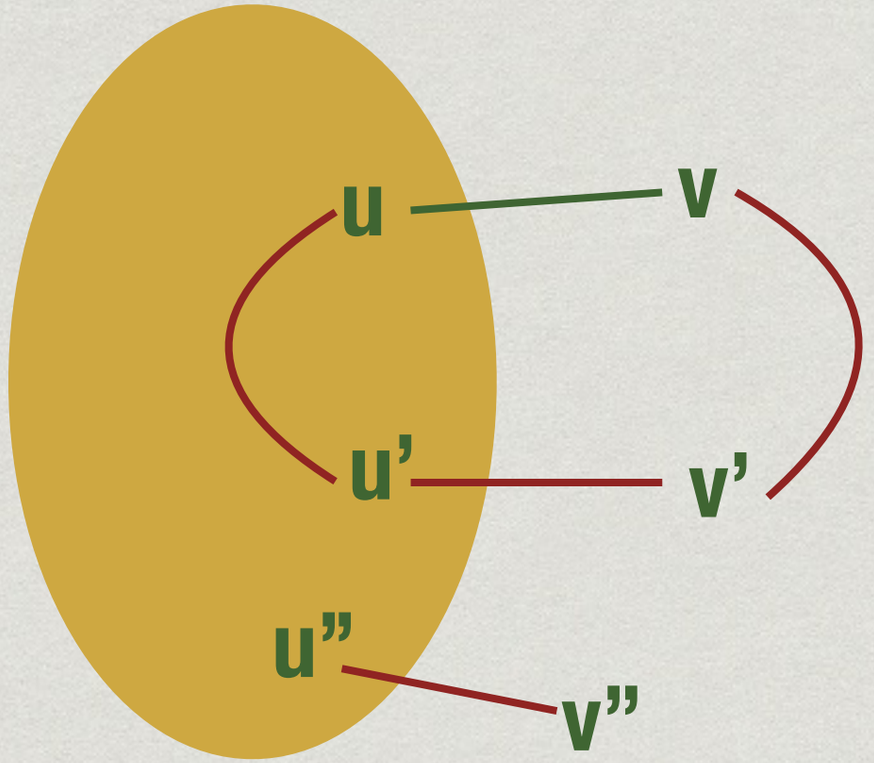
Minimum separator lemma

- Let T be a minimum cost spanning tree, e = (u,w) not in T
- u in U and w in W are connected by a path p in T
- p starts in U and ends in W
- Let f = (u’,w’) be the first edge on p such that u’ in U and w’ in W
- Drop f and add e to get a smaller spanning tree



Minimum separator lemma

- Proof of the lemma is slightly subtle
- Not enough to replace any edge from U to W by e = (u,v)
- Need to identify such an edge on the path from u to v



Correctness of Prim's algorithm

- Correctness follows directly from minimum separator lemma
- At each stage, TV and (V-TV) form a non-trivial partition of V
- The smallest edge connecting TV to (V-TV) must belong to every minimum cost spanning tree
  - This is the edge that the algorithm picks

Further observations

- Need not start with smallest edge overall
  - For any vertex v, smallest edge attached to v must be in the minimum cost spanning tree
    - Consider the partition {v}, V-{v}
- Can start with any such edge

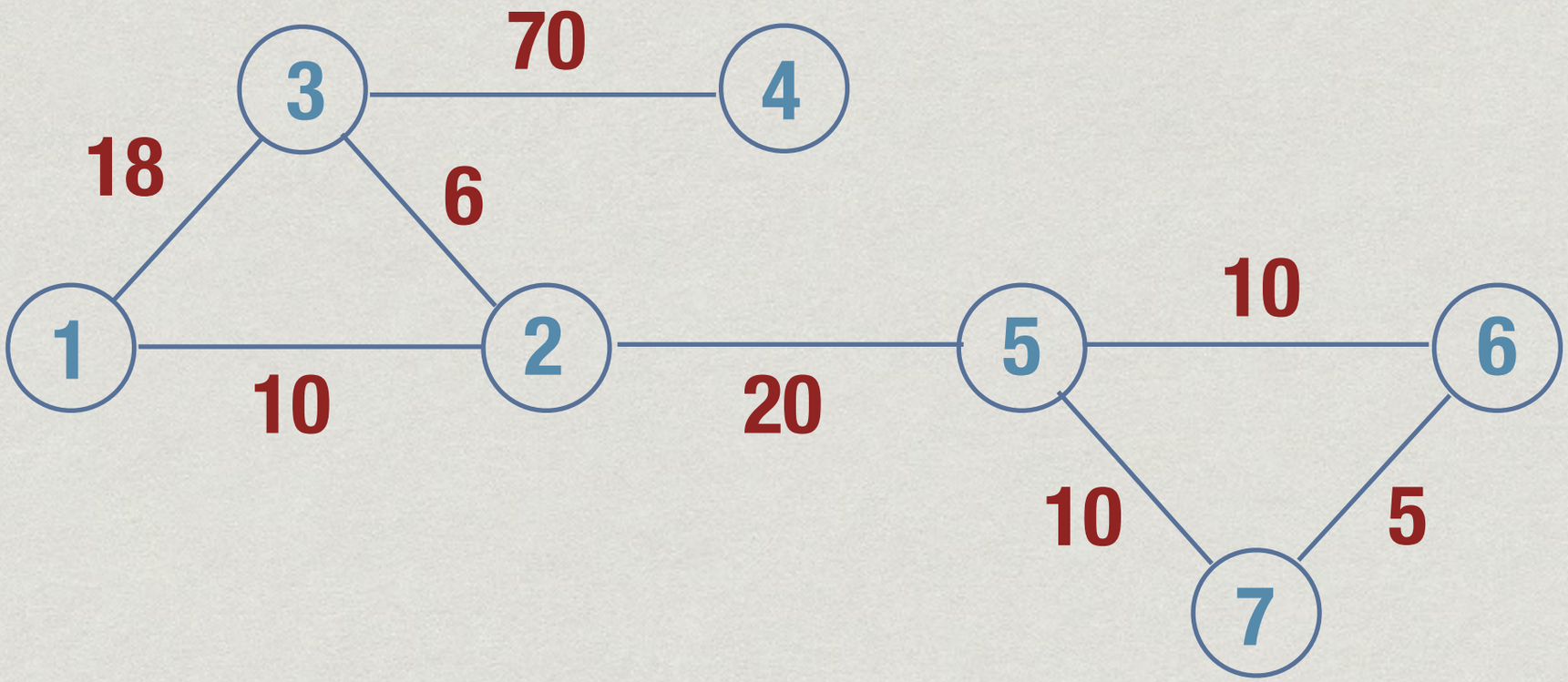
Prim's algorithm revisited

- Start with TV = {s} for any vertex s
- For each vertex v outside TV, maintain
  - Distance_TV(v), smallest edge weight from v to TV
  - Neighbour_TV(v), nearest neighbour of v in TV
- At each stage, add to TV ("burn") vertex u with smallest Distance_TV(u)
  - Update Distance_TV(v), Neighbour_TV(v) for each neighbour of u
- Very similar to Dijkstra's algorithm!

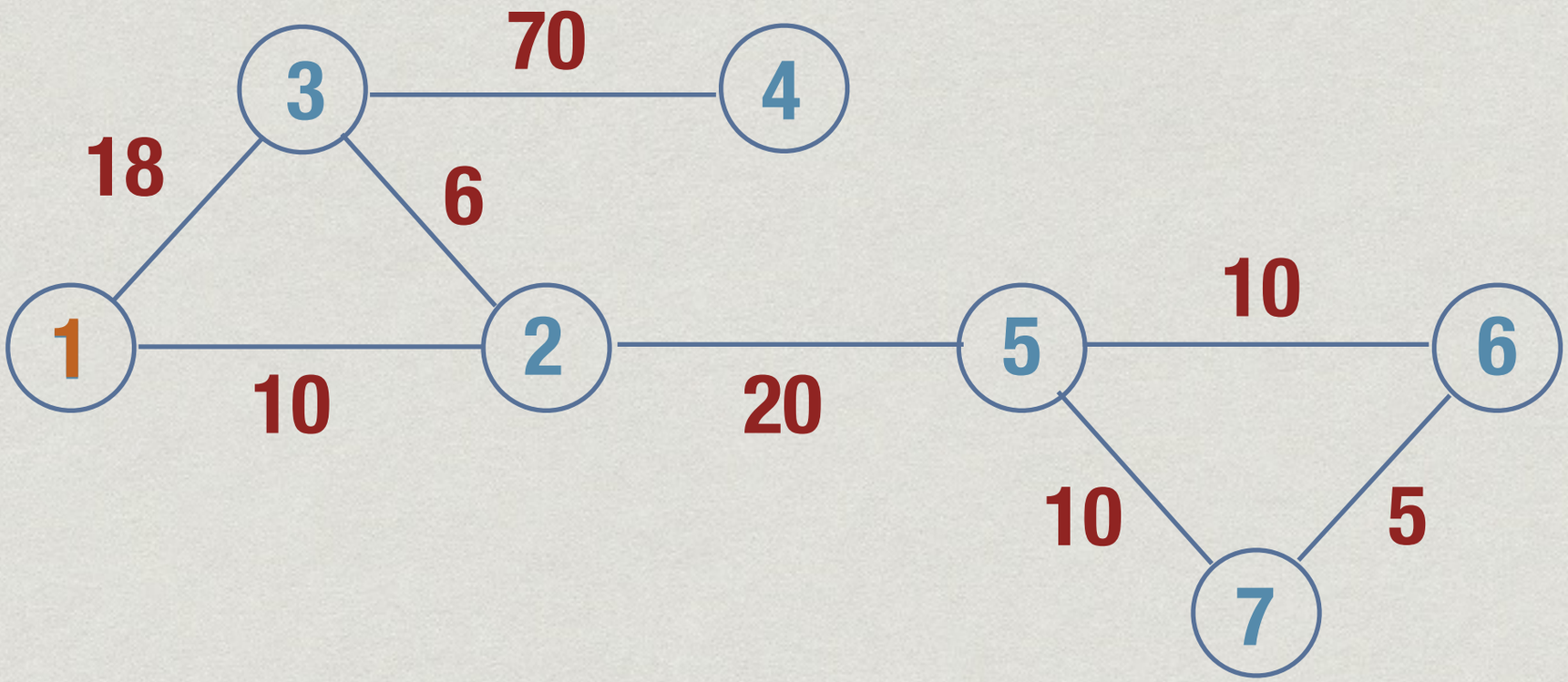
Prim's algorithm, refined

```
function Prim
  for i = 1 to n
    visited[i] = False; Nbr_TV[i] = -1; Dist_TV[i] = infinity

  TE = []   //List of spanning tree edges
  visited[1] = True
  for each edge (1,j)
    Nbr_TV[j] = 1; Dist_TV[j] = weight(1,j)

  for i = 2 to n
    Choose u such that Visited[u] == False
                           and Dist_TV[u] is minimum
    Visited[u] = True
    TE.append{(u,Nbr_TV[u])}
    for each edge (u,v) with Visited[v] == False
      if Dist_TV[v] > weight(u,v)
        Dist_TV[v] = weight(u,v); Nbr_TV[i] = u
```

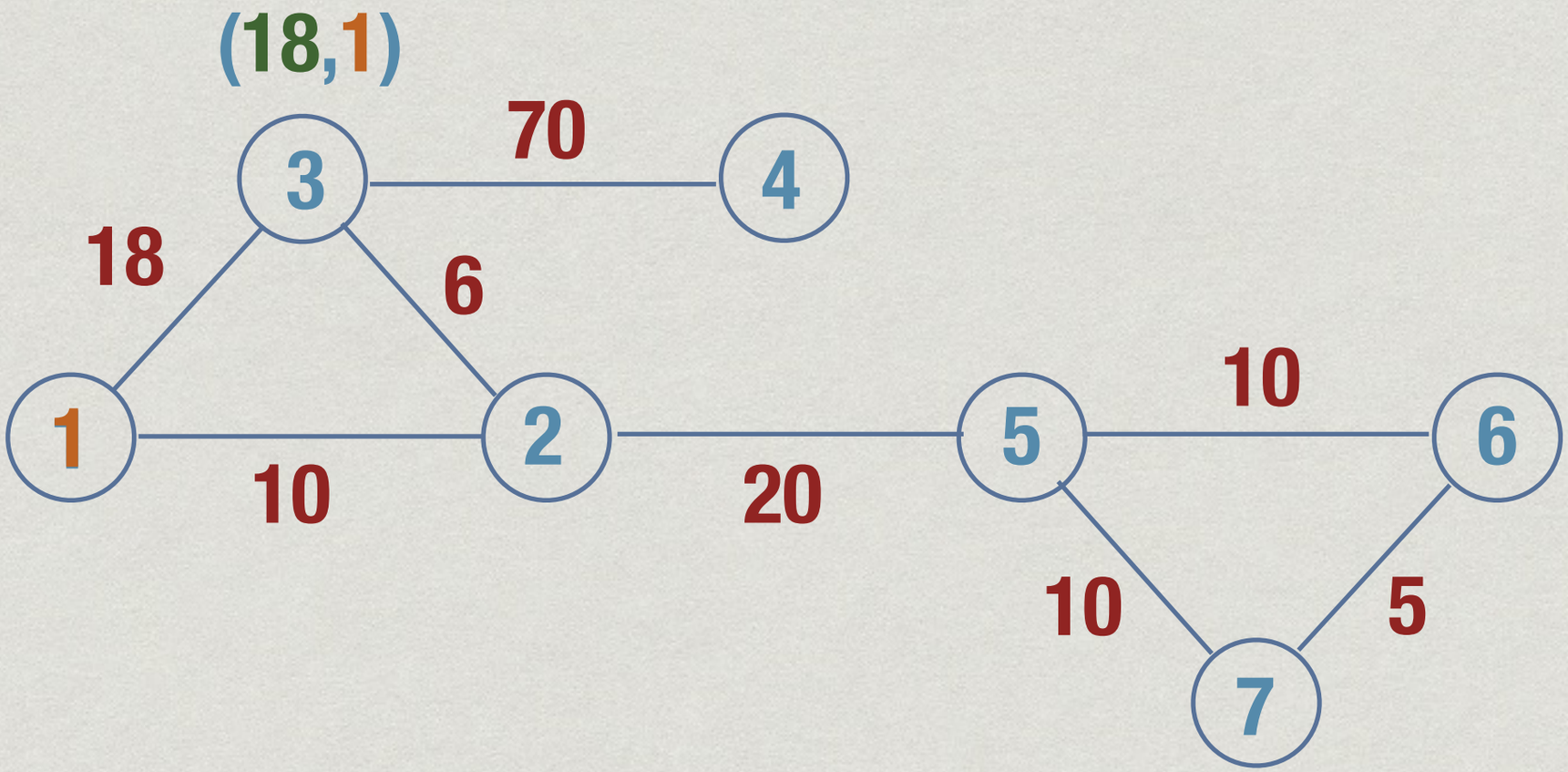
Prim's algorithm



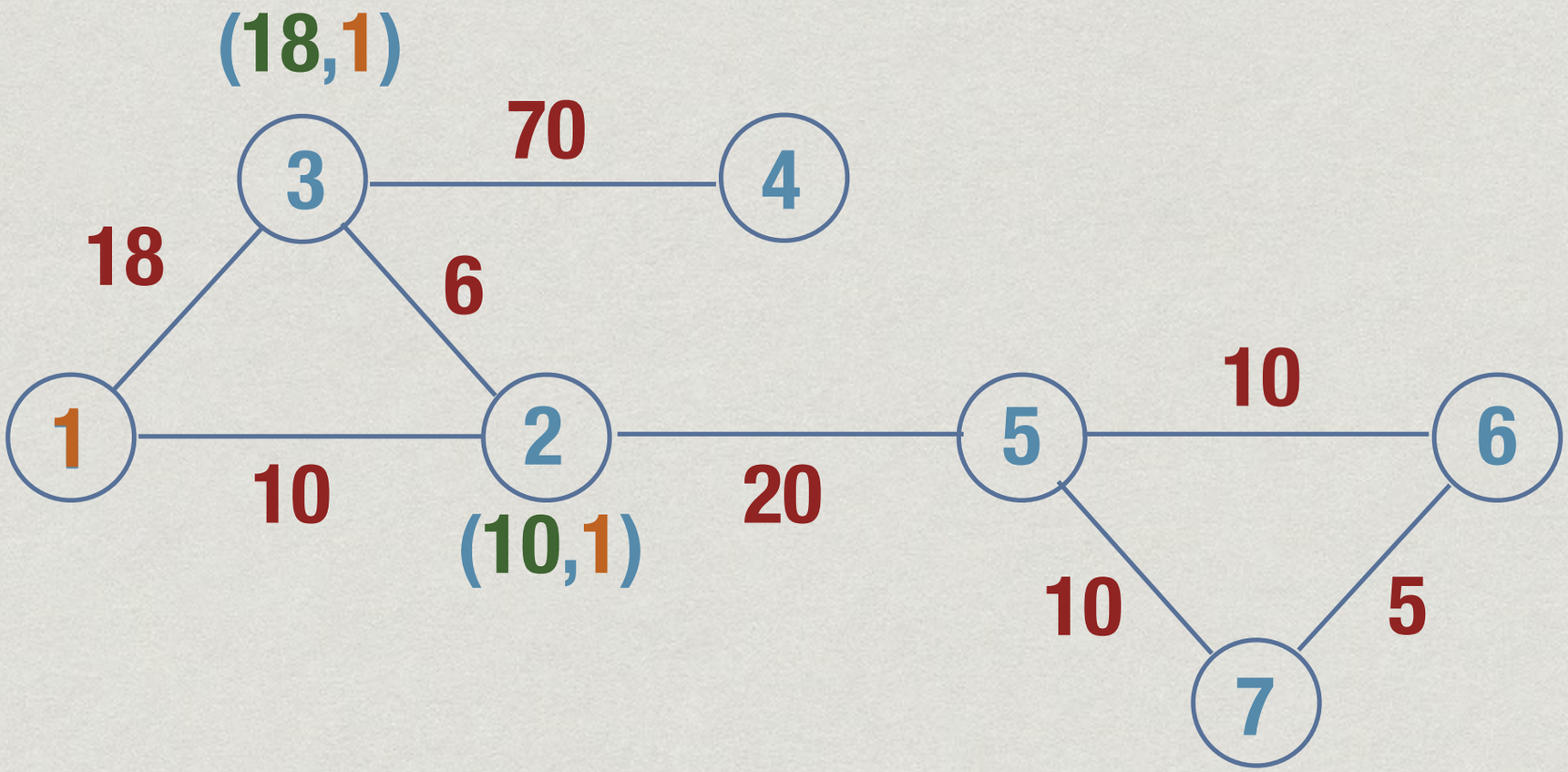
Prim's algorithm



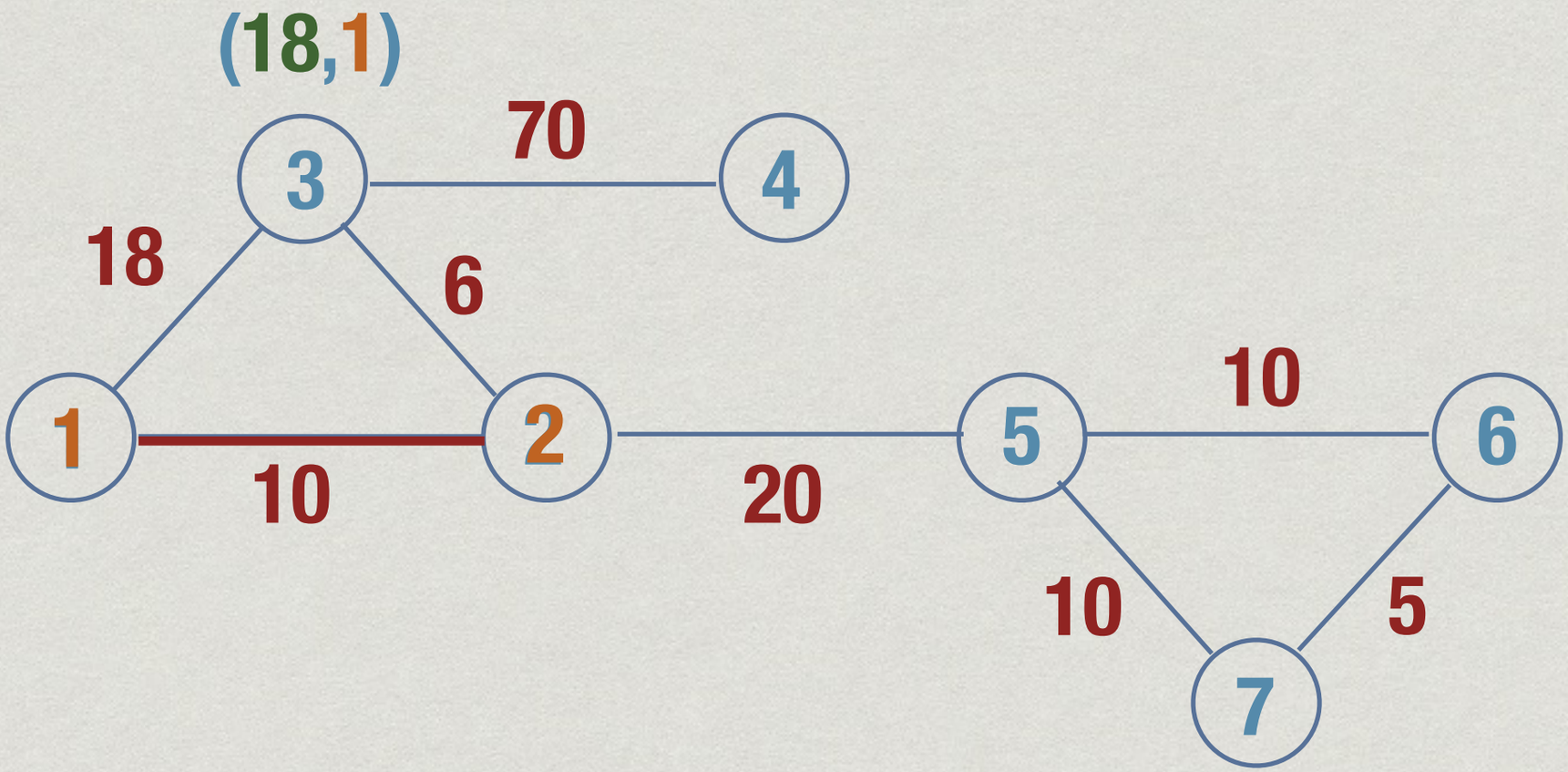
Prim’s algorithm



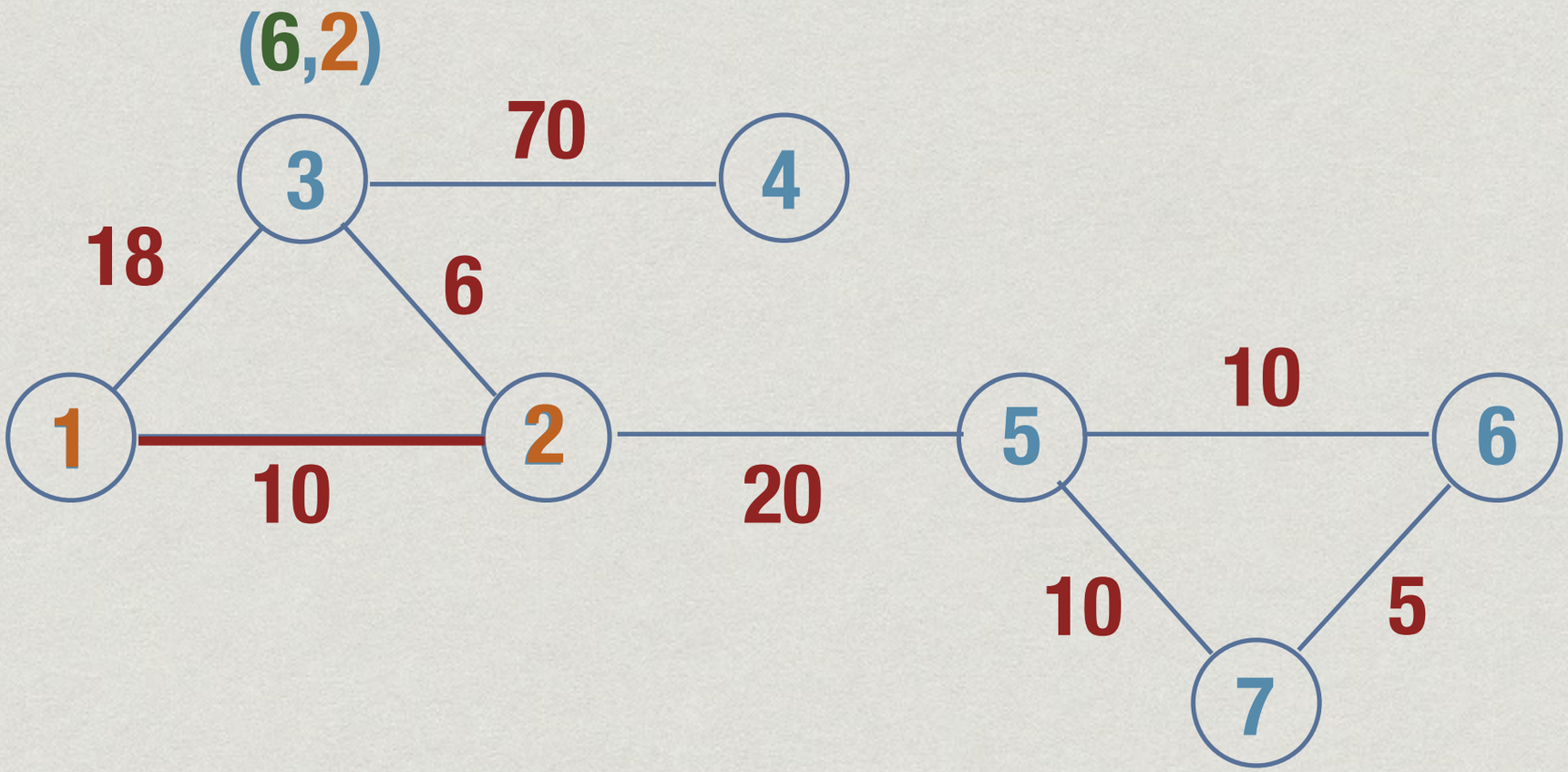
Prim’s algorithm



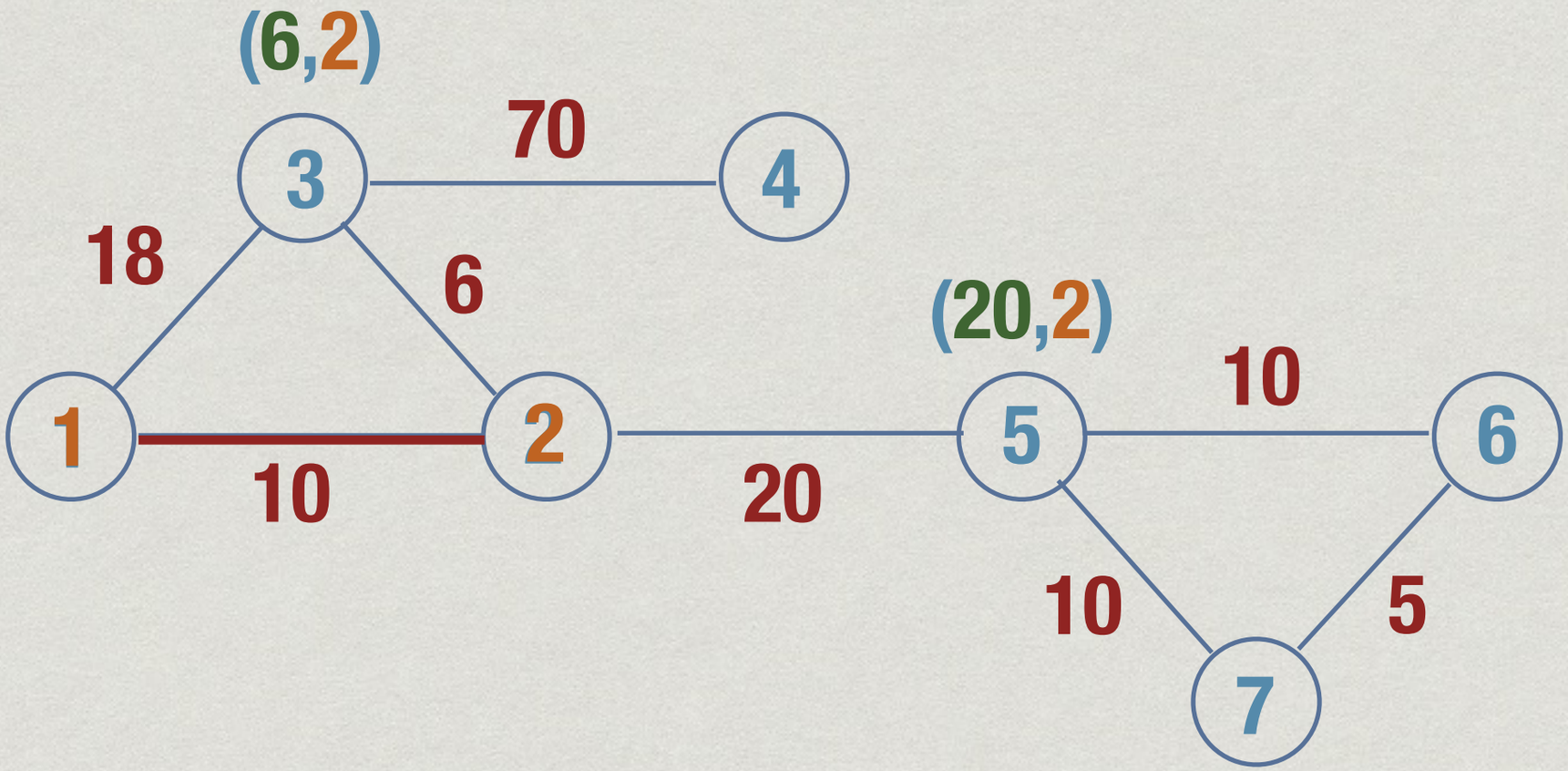
Prim's algorithm



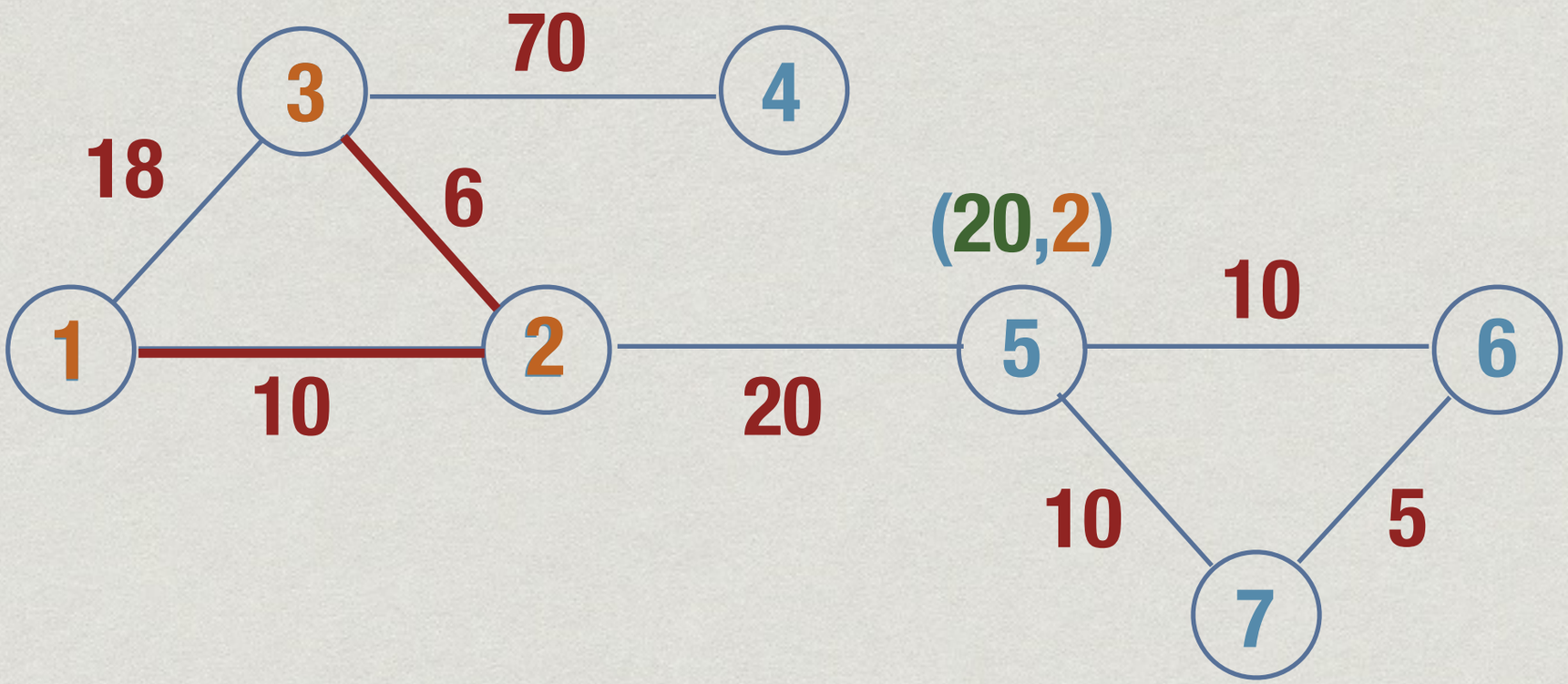
Prim’s algorithm



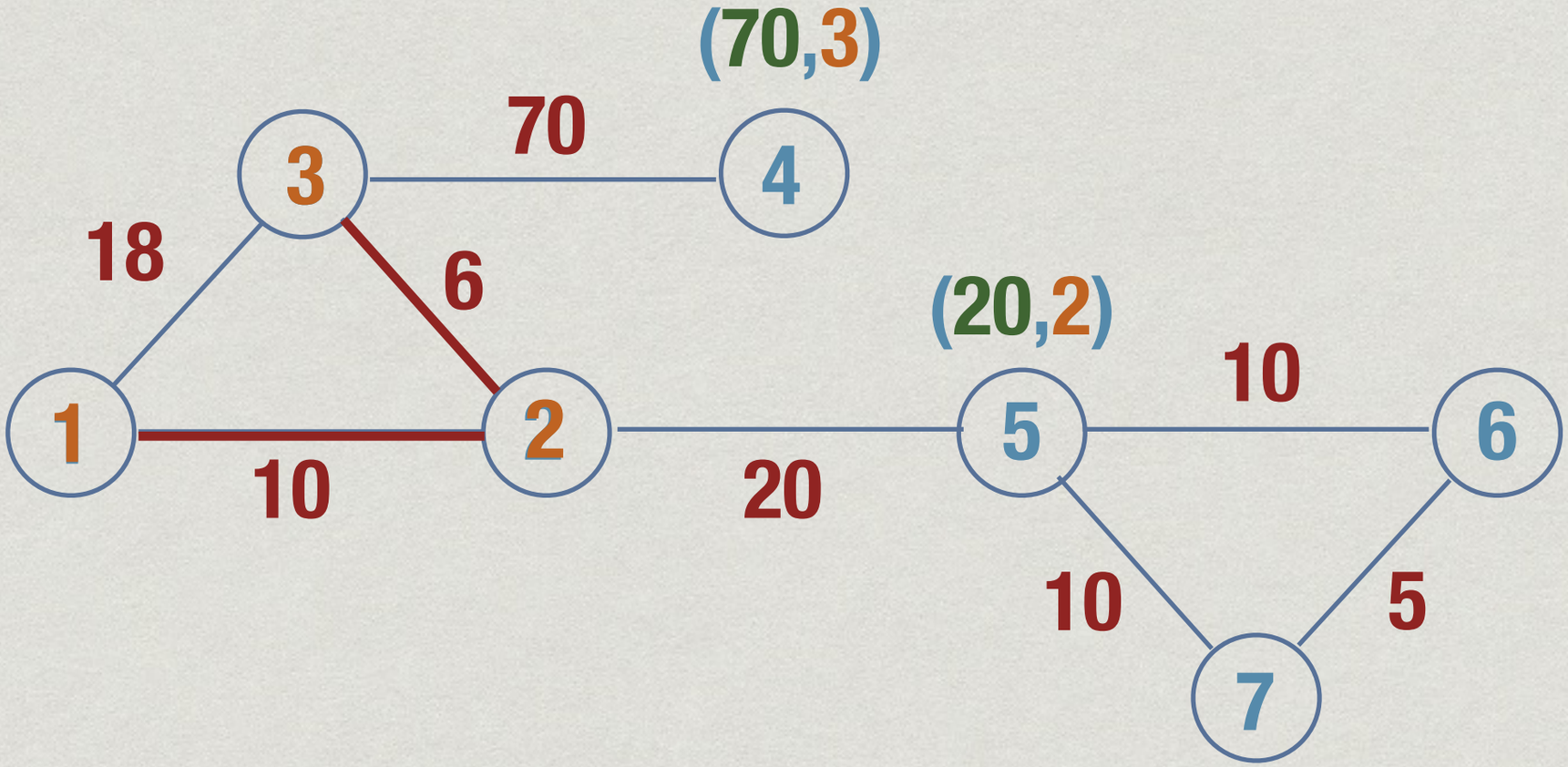
Prim’s algorithm



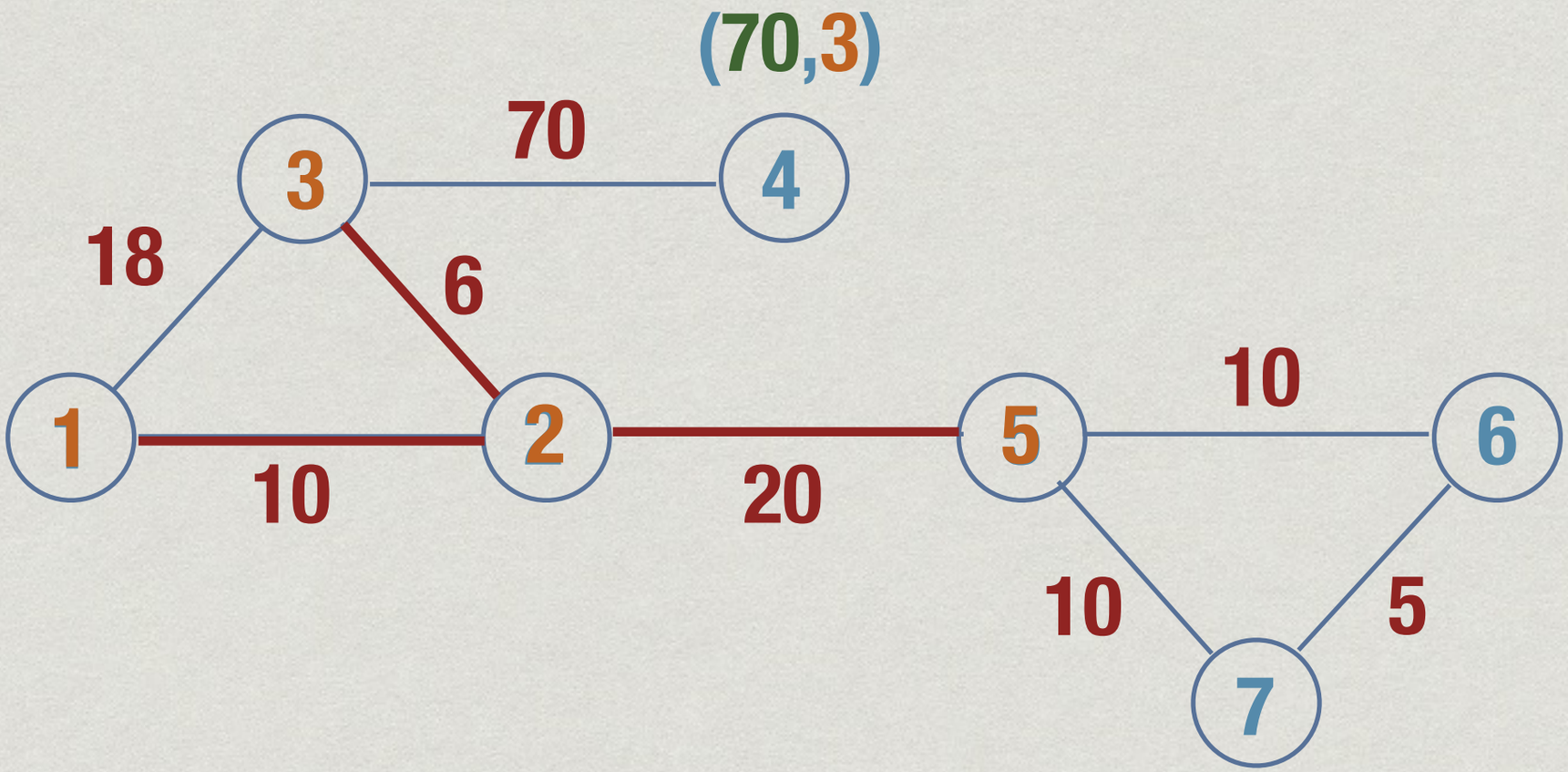
Prim’s algorithm



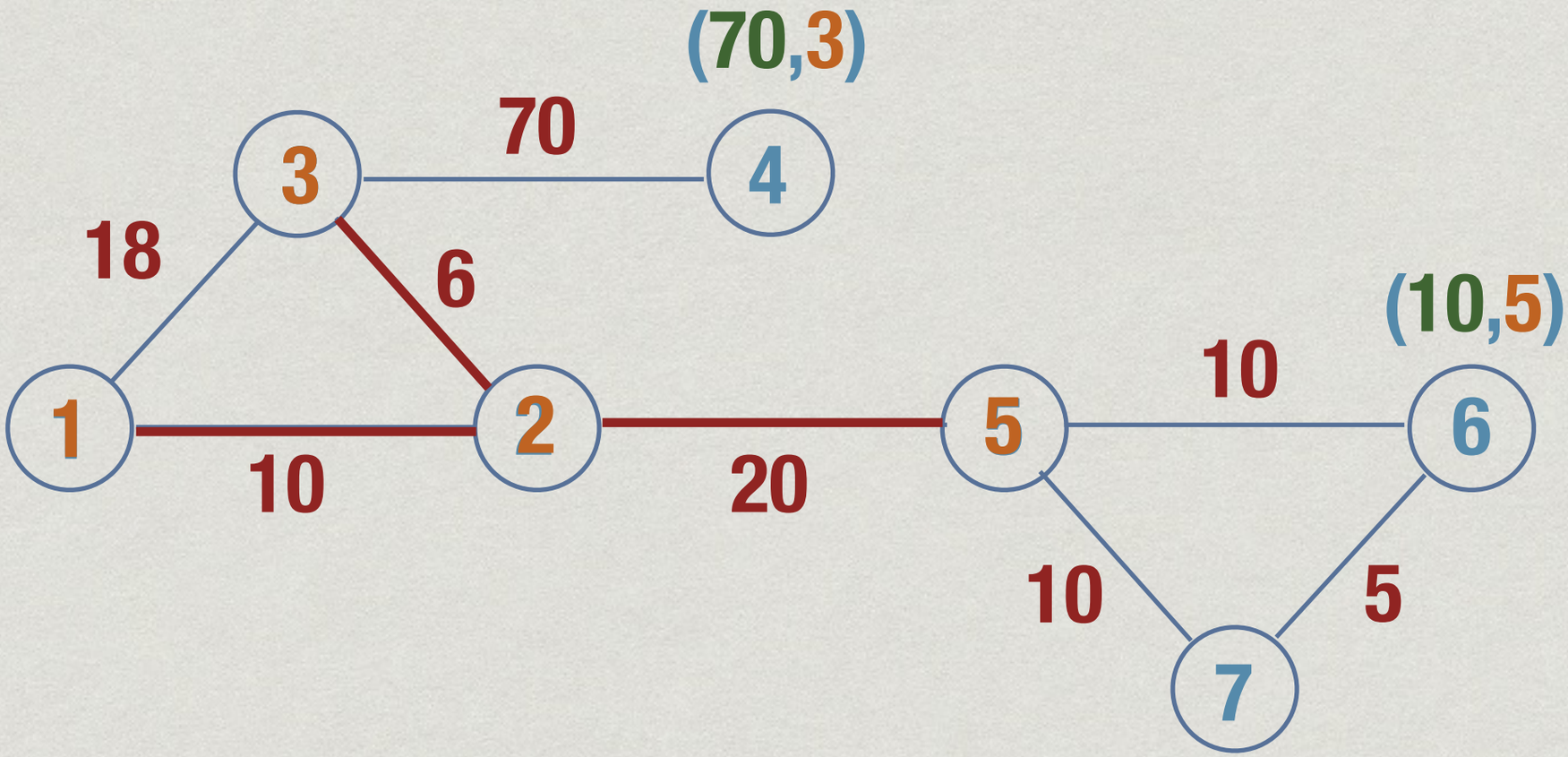
Prim’s algorithm



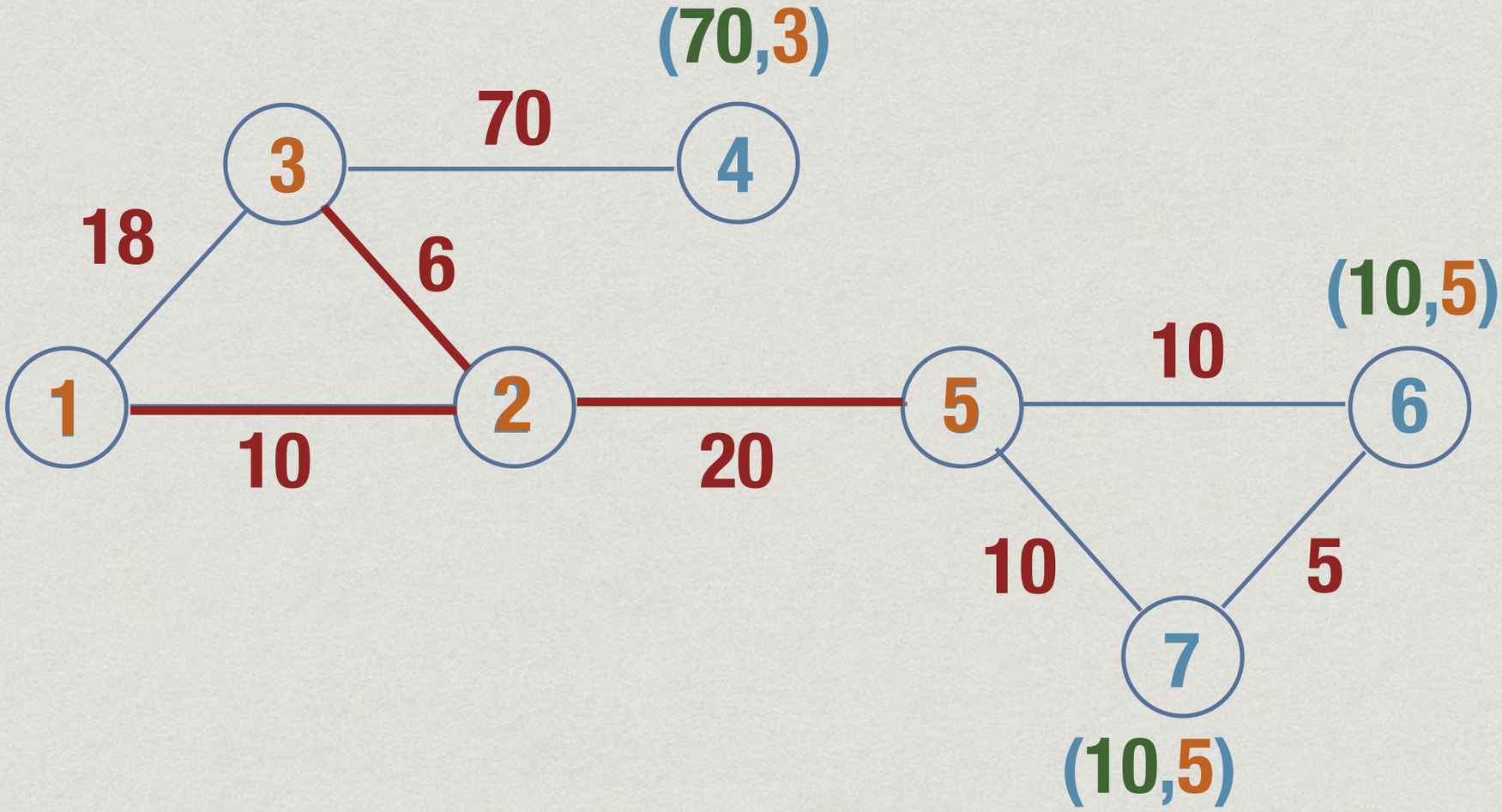
Prim's algorithm



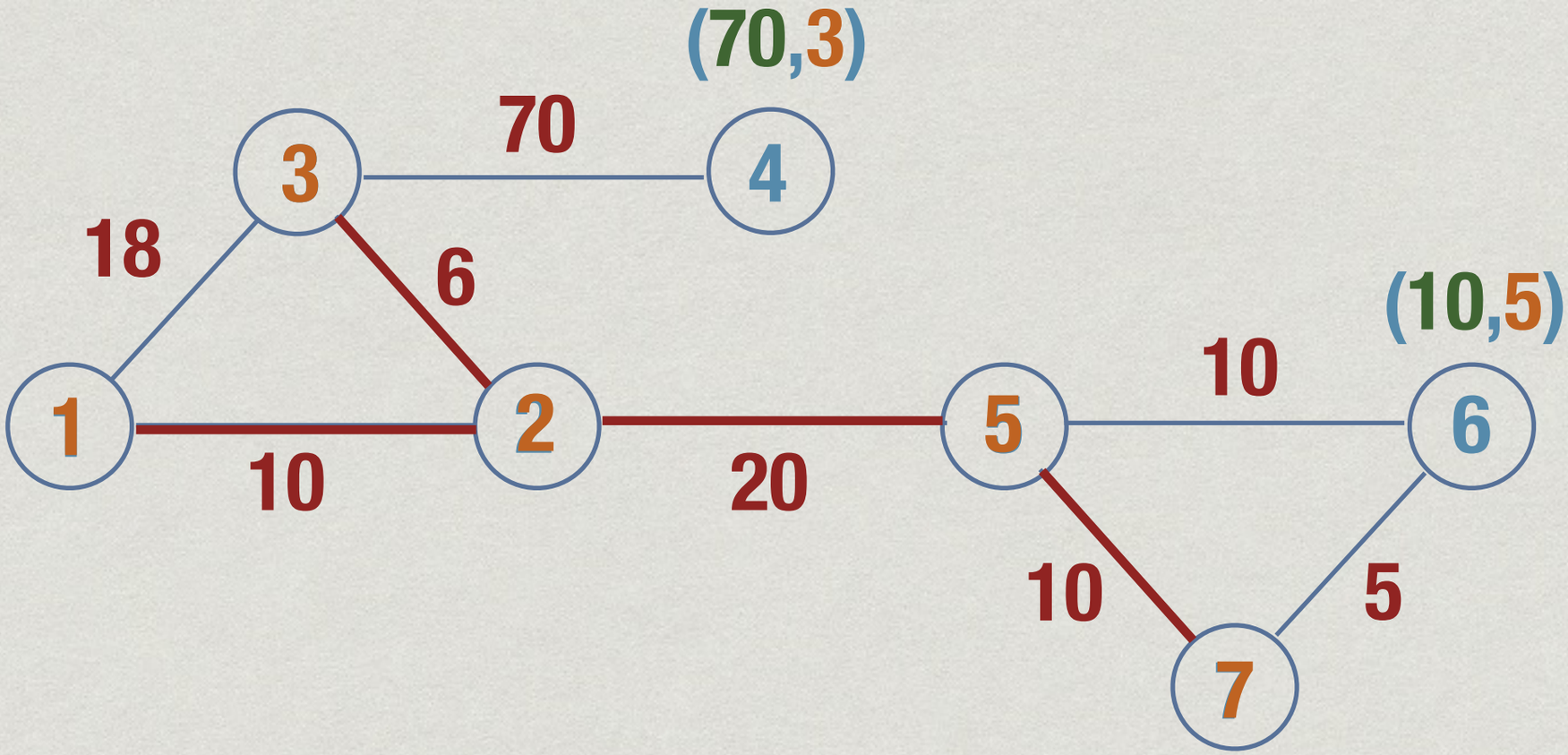
Prim’s algorithm



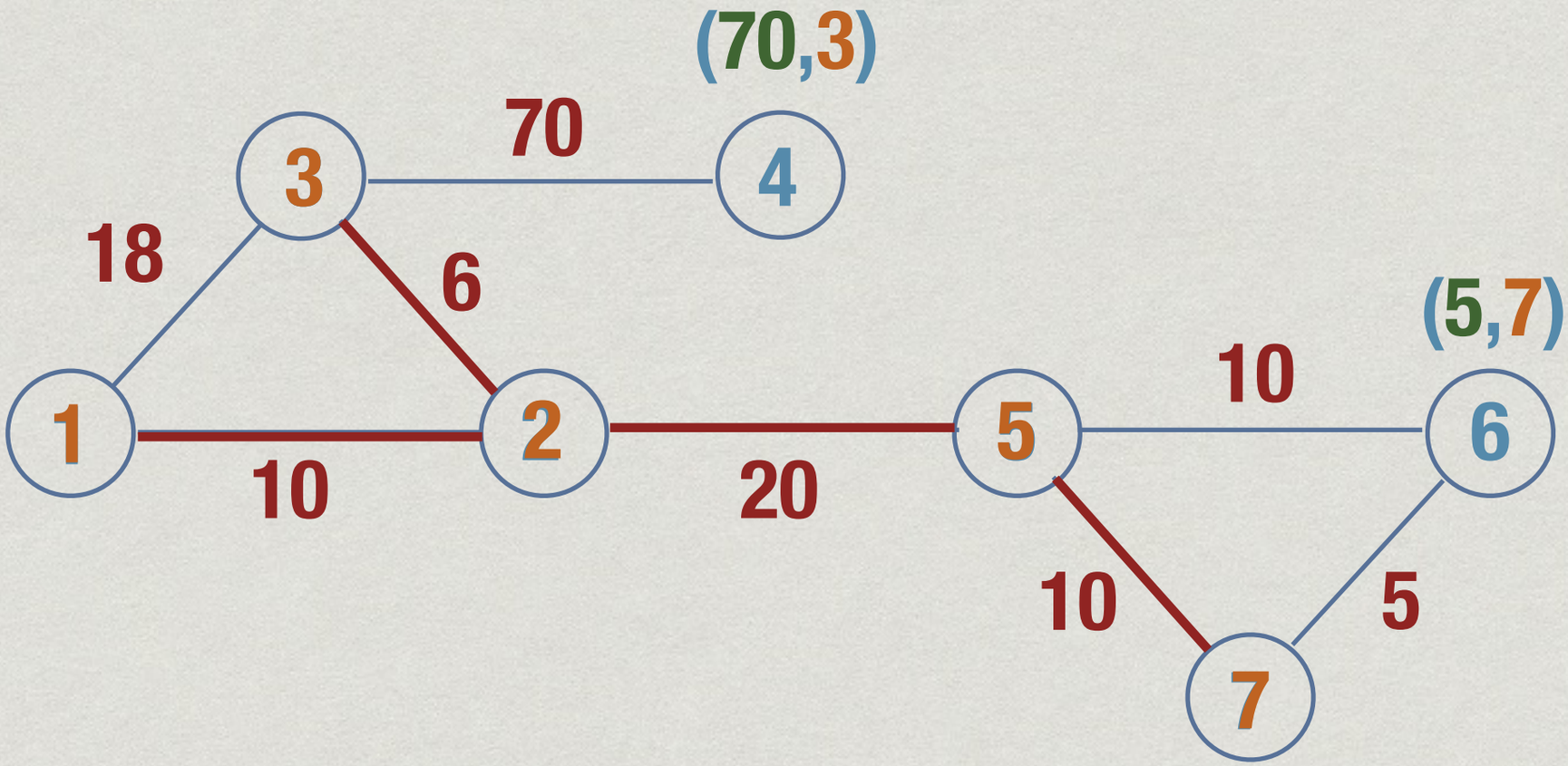
Prim’s algorithm



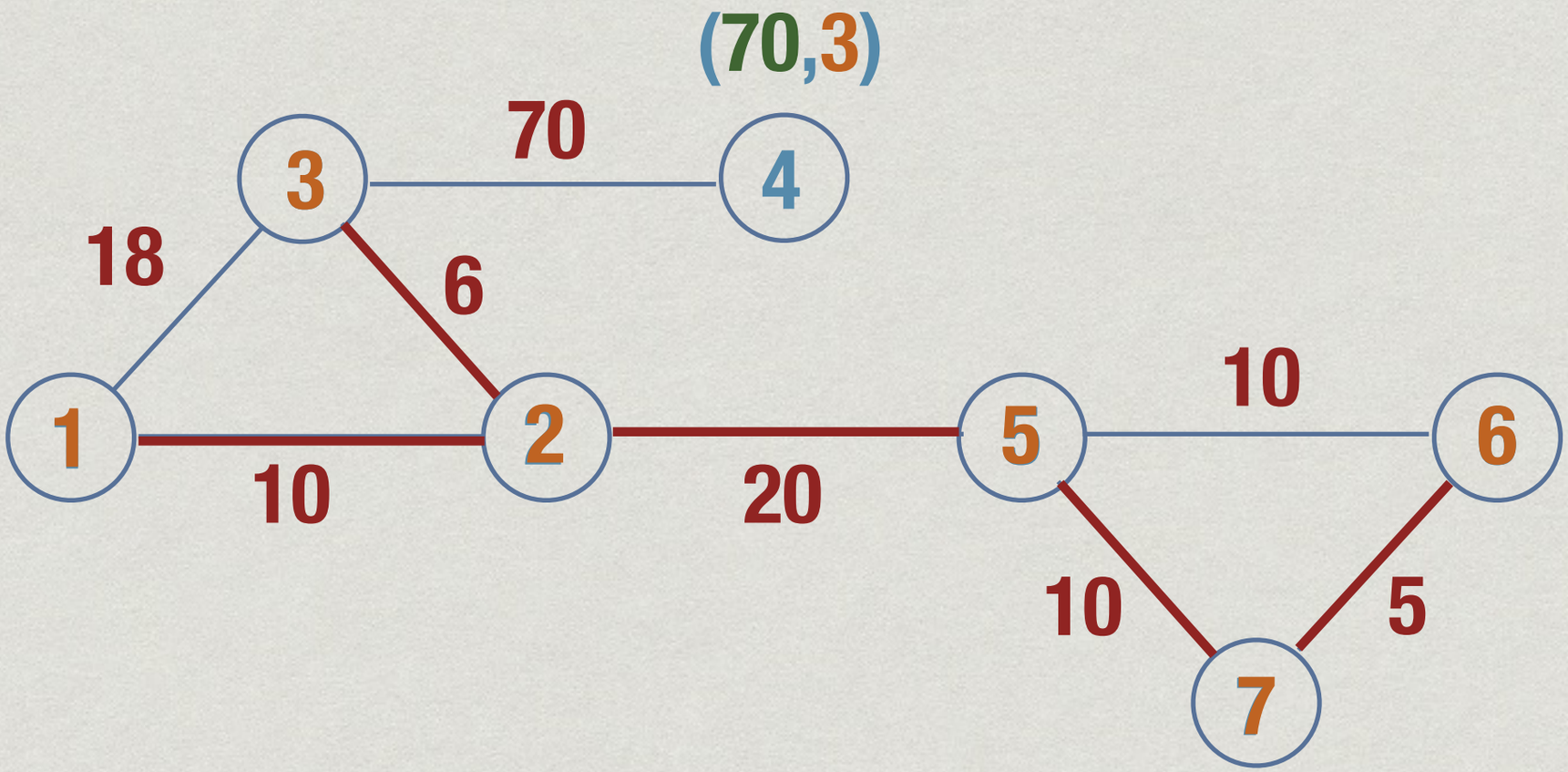
Prim’s algorithm



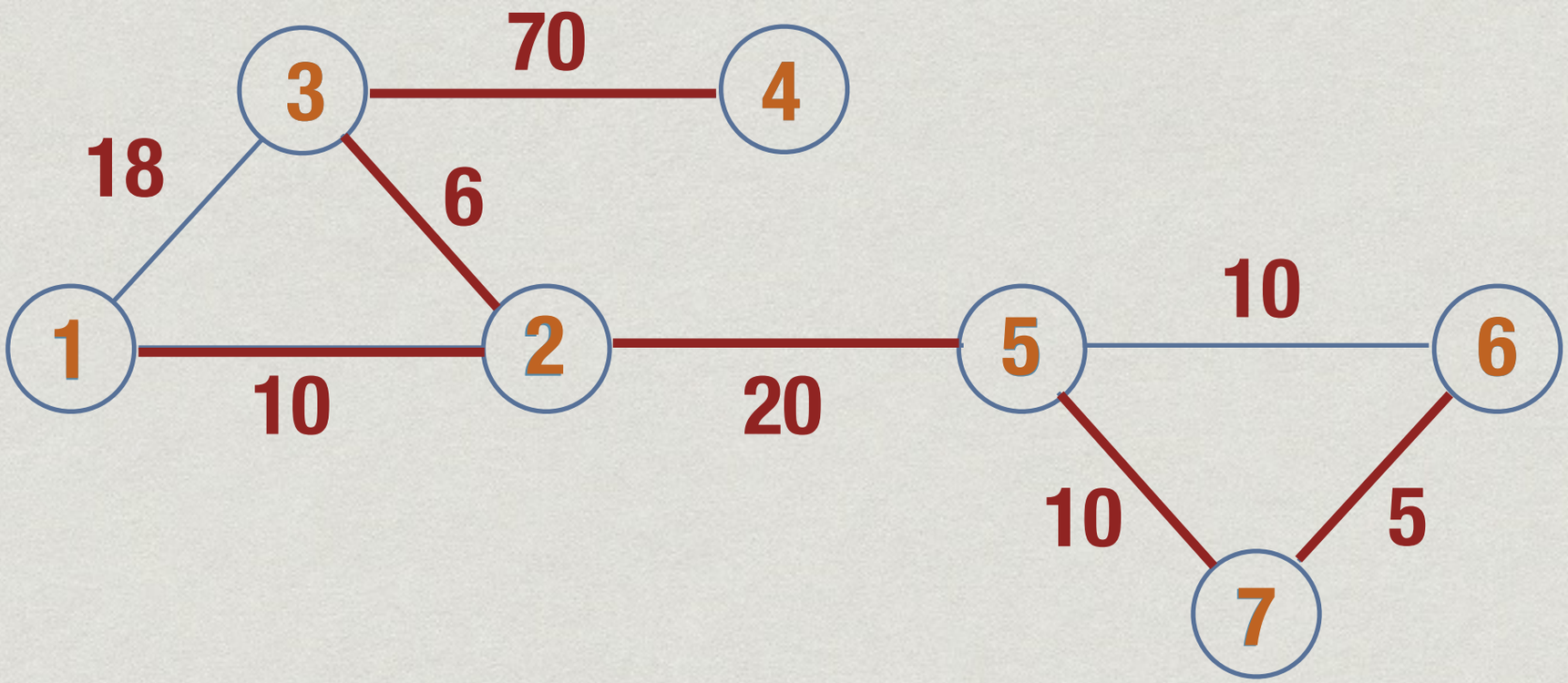
Prim’s algorithm



Prim’s algorithm



Prim's algorithm



Complexity

- Similar to Dijkstra's algorithm
- Outer loop runs n times
  - In each iteration, we add one vertex to the tree
    - O(n) scan to find nearest vertex to add
- Each time we add a vertex v, we have to scan all its neighbours to update distances
  - O(n) scan of adjacency matrix to find all neighbours
- Overall $O(n^2)$

Complexity

- Moving from adjacency matrix to adjacency list
  - Across n iterations, O(m) to update neighbours
- Maintain distance information in a heap
  - Finding minimum and updating is O(log n)
- Overall O(n log n + m log n) = O((m+n) log n)

Minimum separator lemma

- We assumed edge weights are distinct
- Duplicate edge weights?
  - Fix an overall ordering {1,2,…,m} of edges
  - Edge e = ((u,v),i) is smaller than f = ((u’,v’),j) if
    - weight(e) < weight(f)
    - weight(e) = weight(f) and i < j

Multiple spanning trees

- If edge weights repeat, the minimum cost spanning tree is not unique
  - "Choose u such that Dist_TV(u) is minimum"
- Different choices generate different trees
  - Different ways of ordering edges {1,2,…,m}
- In general, number of possible minimum cost spanning trees is exponential
  - Greedy algorithm efficiently picks out one of them