Spanning Trees

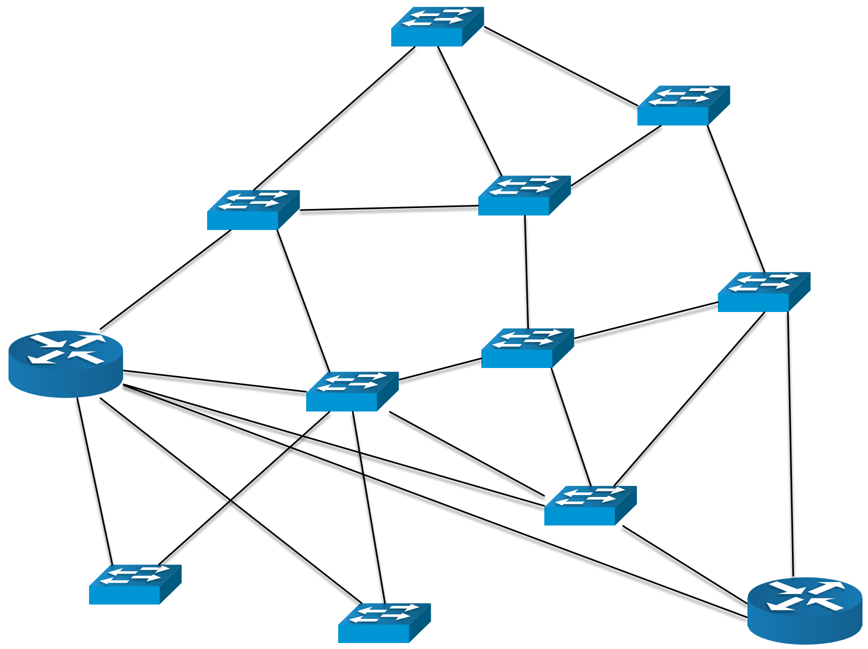
Thomas Schwarz, SJ

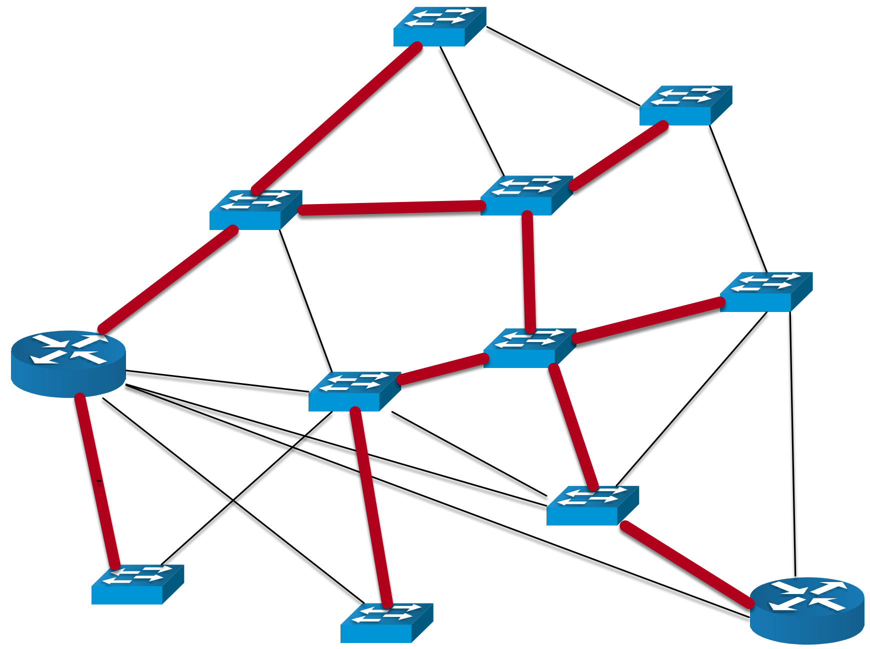
Problem

- Networking: LAN
  - Switches are connected by links
    - Cycles can create problems:
      - Broadcast radiation
        - A broadcast or multicast message is repeatedly received by the same switch and resend and resend and resend and resend …

Problem

- Solution:
  - Use an acyclic subgraph that contains all switches for broadcasting, multicasting, and in general for addressing purposes



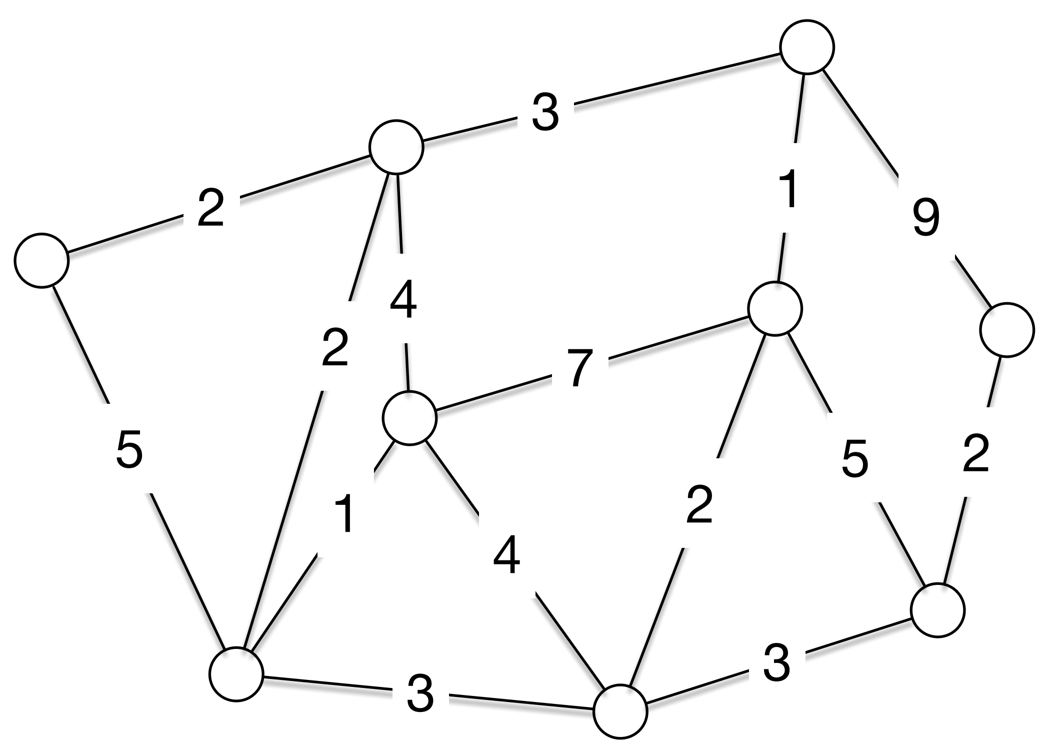


Trees

- A tree is a graph that is:
  - acyclic
  - connects all vertices

Weighted Graphs

- We look at graphs where each edge has a weight
  - Depending on application, some weights can be negative or all weights have to be positive



Weighted Graphs

- Other example:
-

Minimum Spanning Trees

- Given a weighted graph:
  - Find a subset T of edges such that
    - connects all vertices
    - is acyclic
    - Total weight is minimal

$$w(T) = \sum_{(u,v)\in T} w(u,v) \longrightarrow \infty$$

    - Called a minimum weight spanning tree, but "weight" is usually omitted

Minimum Spanning Trees

- Two greedy algorithms, Kruskal's and Prim's
- Use a loop invariant:
  - Let A be the set of edges currently selected
  - Invariant: A is a subset of some minimum spanning tree
  - At each step of the algorithm: only add an edge (u, v) if the invariant remains true after inserting the edge
  - Such an edge is a *safe* edge

Minimum Spanning Trees

- Generic MST algorithm

  1. $A = \varnothing$

  2. While A is not a spanning tree

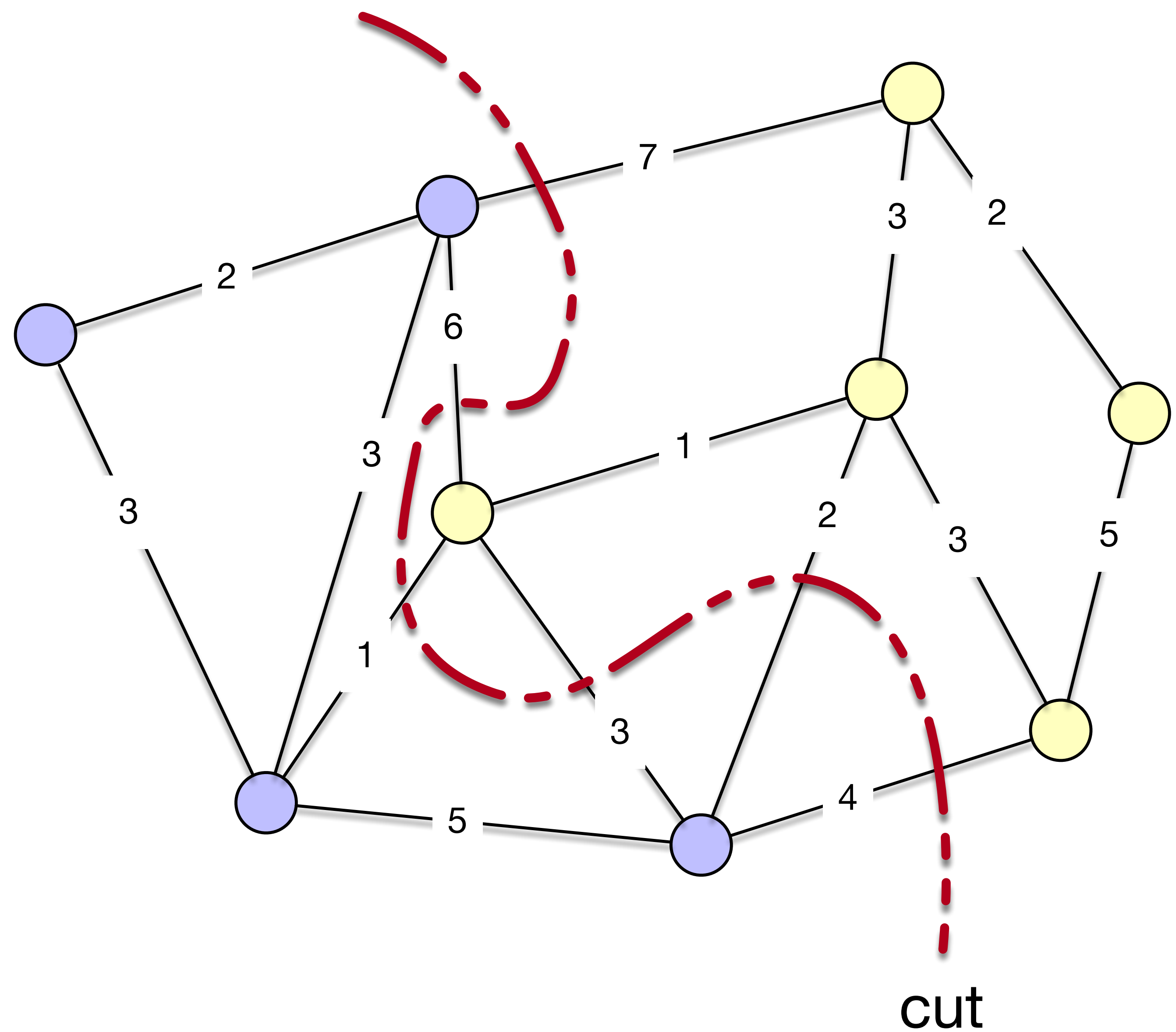
     1. Find a safe edge

     2. Add the safe edge to A

  3. Return A

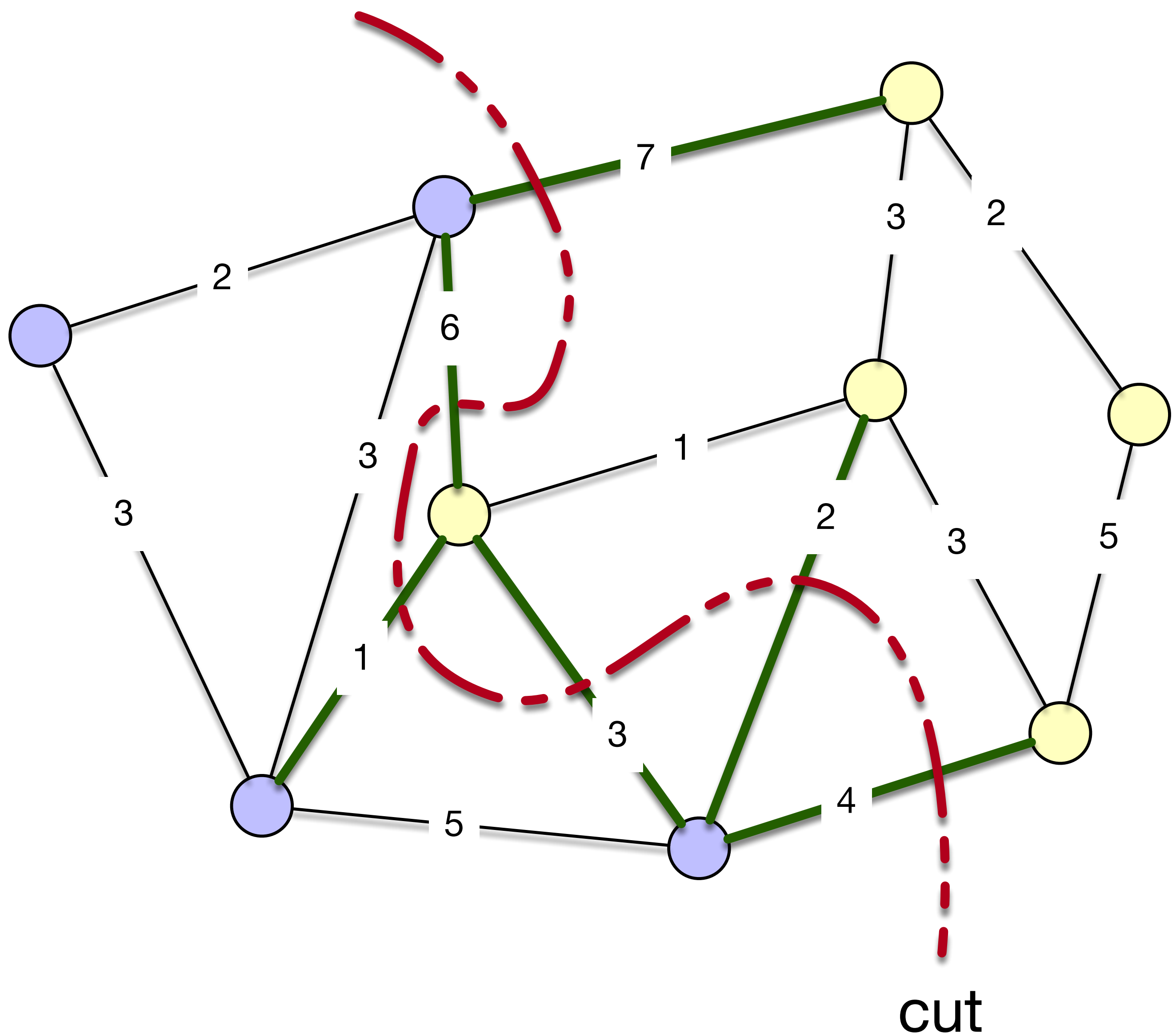
Minimum Spanning Trees

- A *cut* is a partition of the vertices of the graph



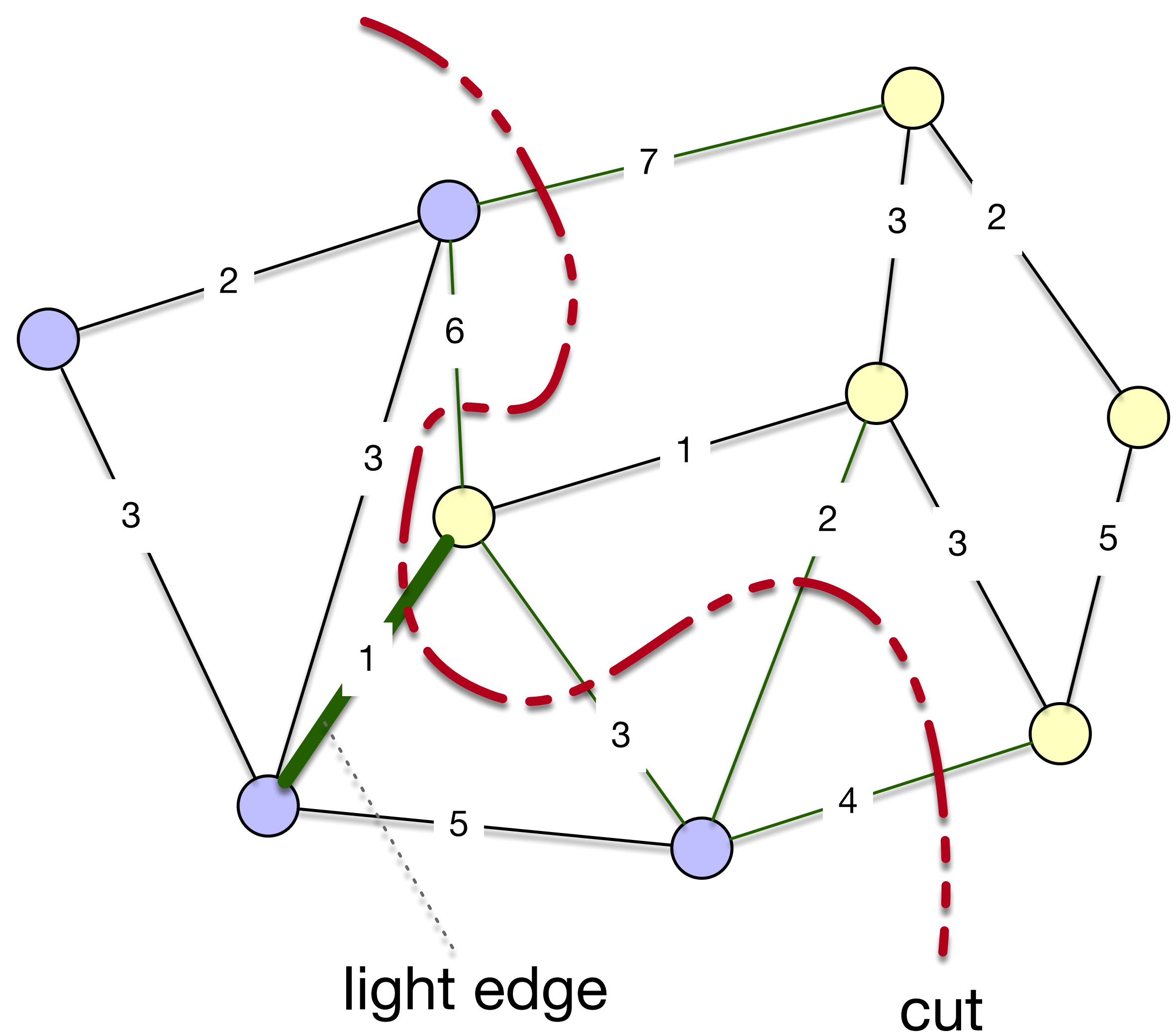
Minimum Spanning Trees

- Edges can "cross the cut"



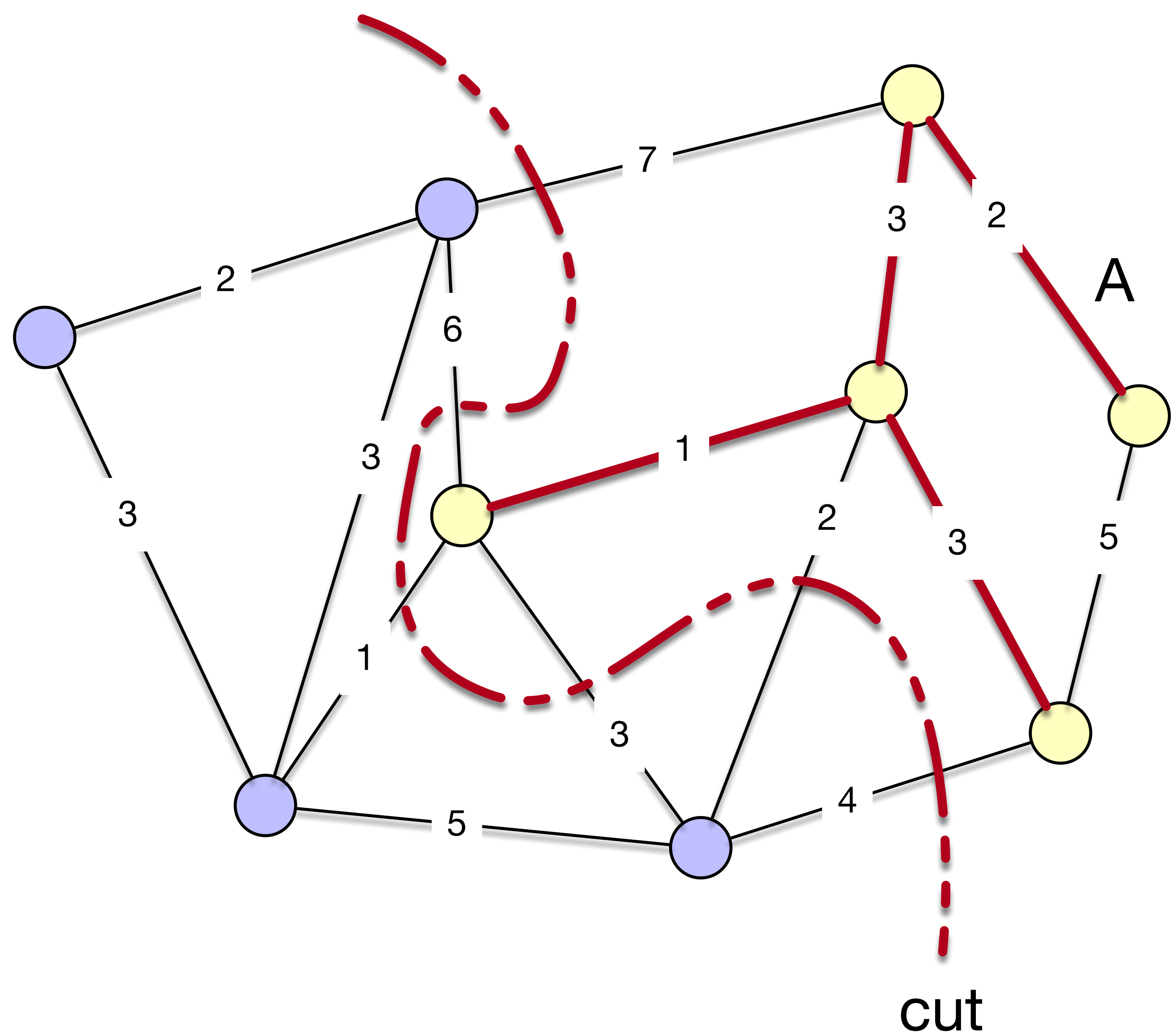
Minimum Spanning Trees

- Edges are "light" if the cross the cut and no other edge crossing the cut has a smaller weight



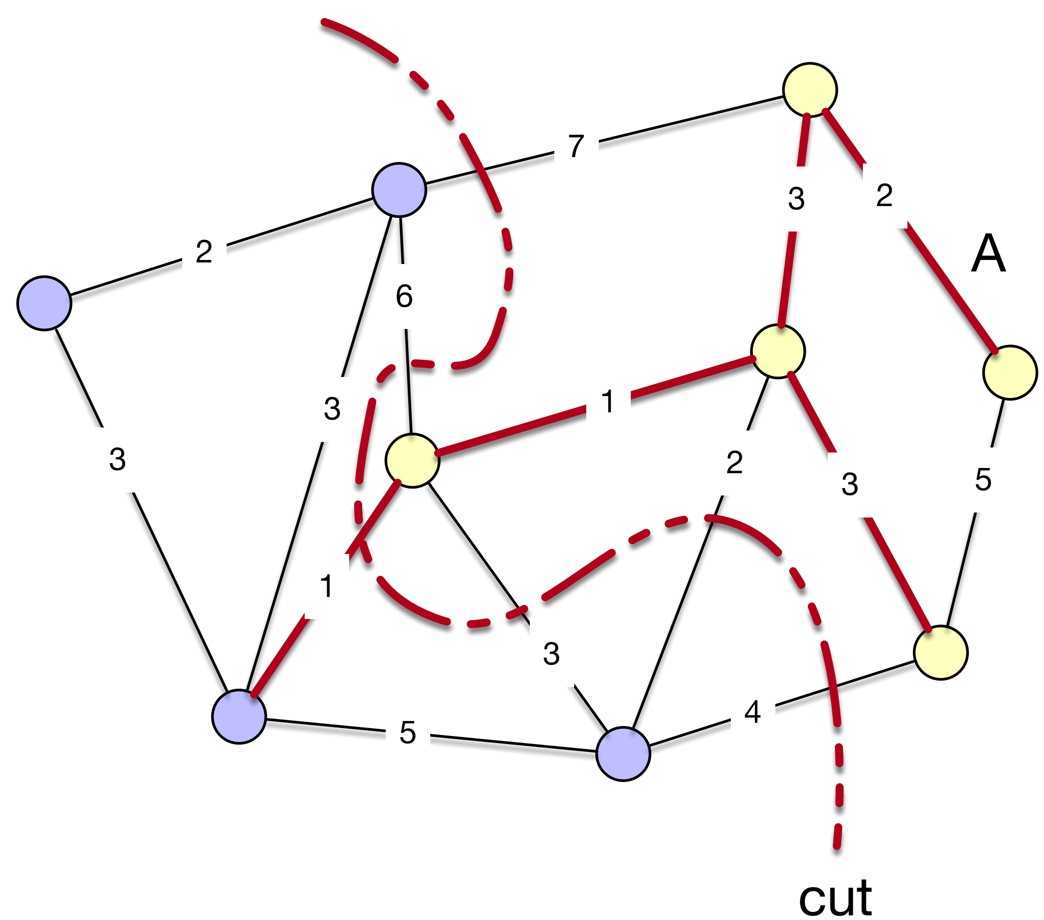
Minimum Spanning Trees

- A cut respects A if no edge of A crosses the cut



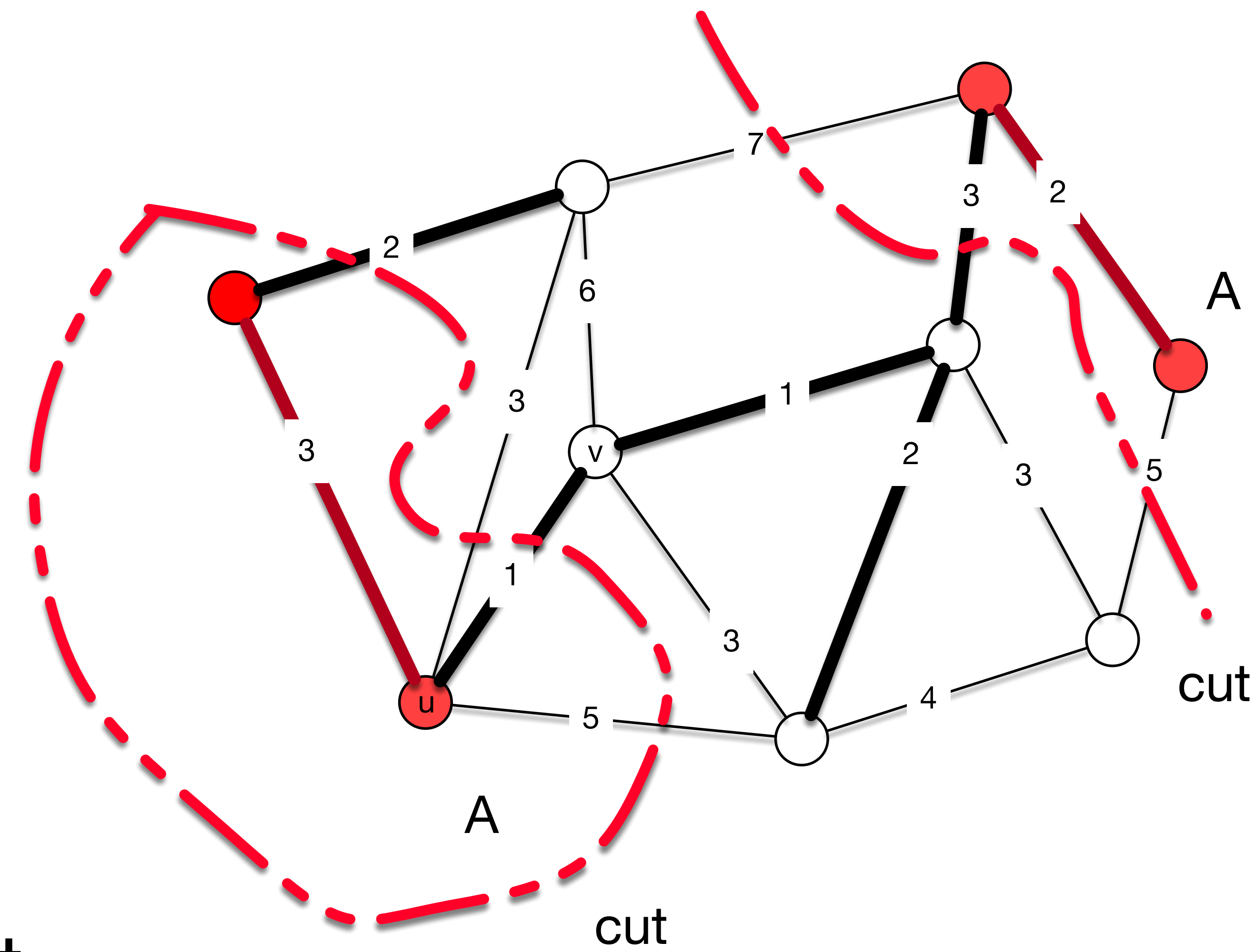
Minimum Spanning Trees

- Theorem: Let A be a subset of E included in some minimum spanning tree, let $(S, E - S)$ be a cut respecting A and let (u, v) be a light edge crossing the cut. Then this edge is safe



Minimum Spanning Trees

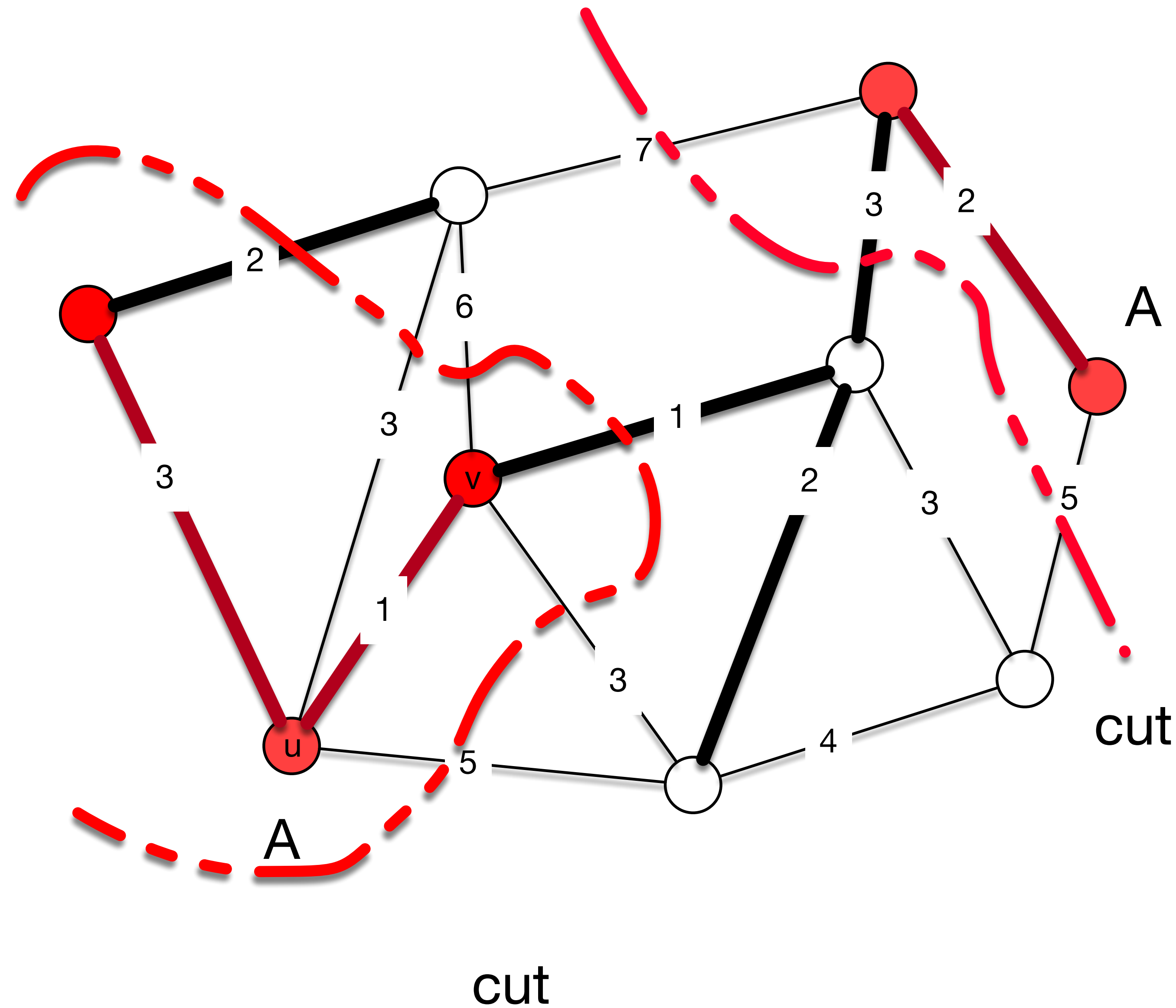
- Proof:
  - We have a subgraph A that is part of a minimum spanning tree T
  - We have a minimum weight crossing edge (u, v) crossing the cut that separates A from the rest of the graph



This set A has two different connected components and consists of red vertices
T is given by the fatter edges

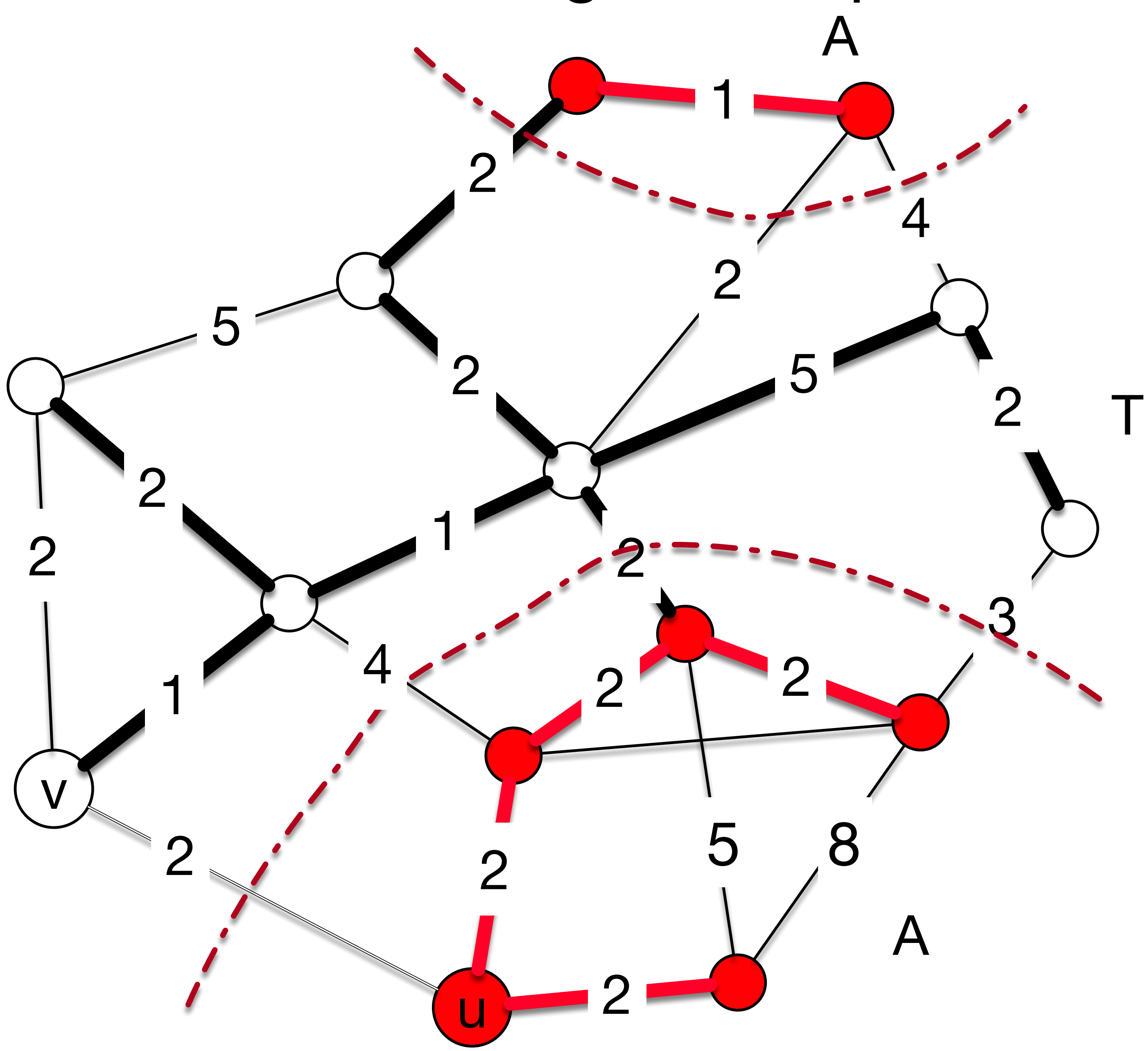
Minimum Spanning Trees

- Case 1: The edge is part of T
- Then there is nothing to show since adding the edge still gives us a subgraph that is part of a minimum spanning tree



Minimum Spanning Trees

- Case 2: The edge is not part of T



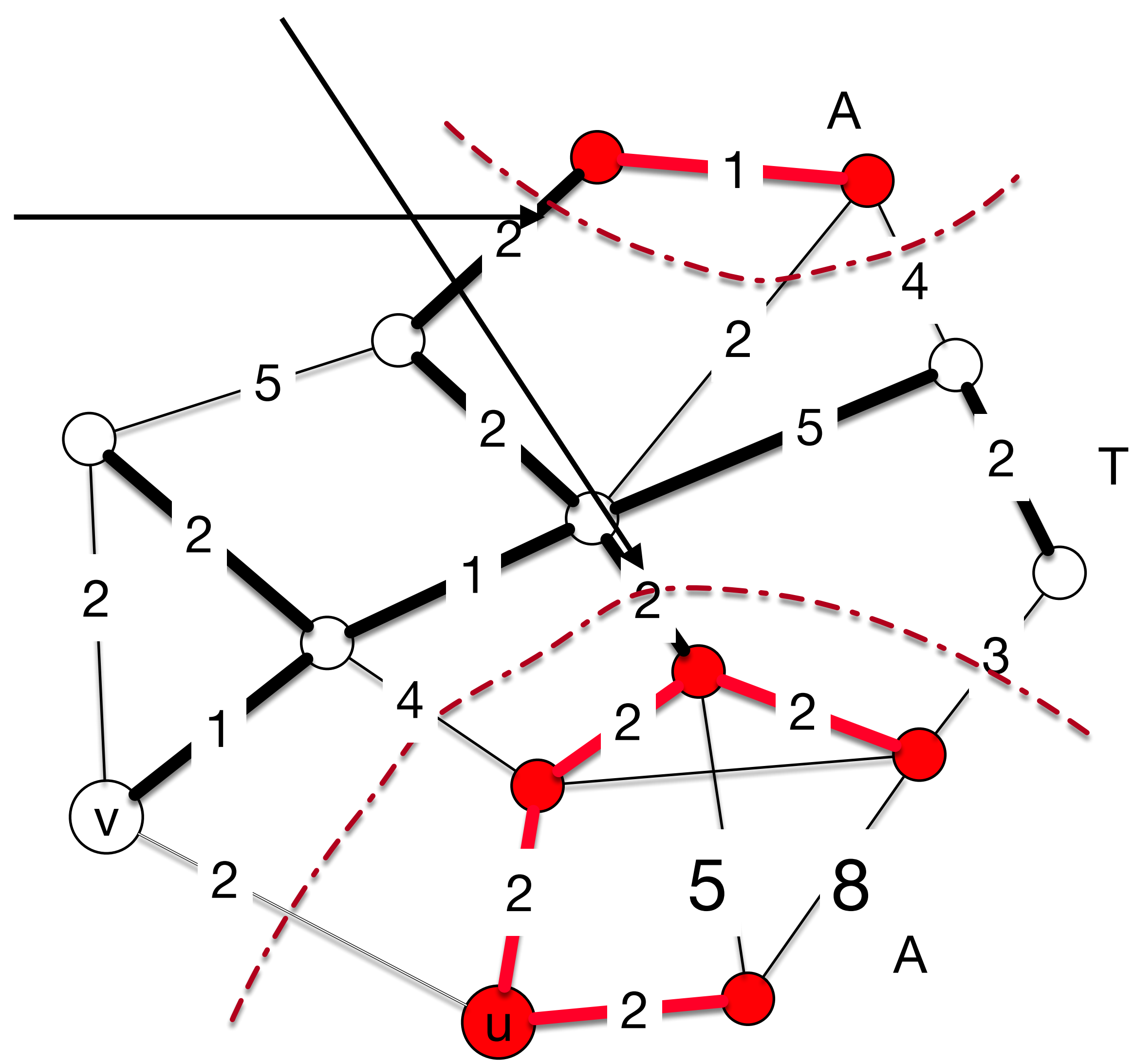
Edges and vertices in A are red
Edges in T are fat
This includes the red edges
u and v are at the lower left

Minimum Spanning Trees

- In this case, we need to construct a new minimum weight spanning tree
- Observe that there has to be an edge of T that crosses the cut
  - Because we can travel from every node to every node in T and not all nodes are in A

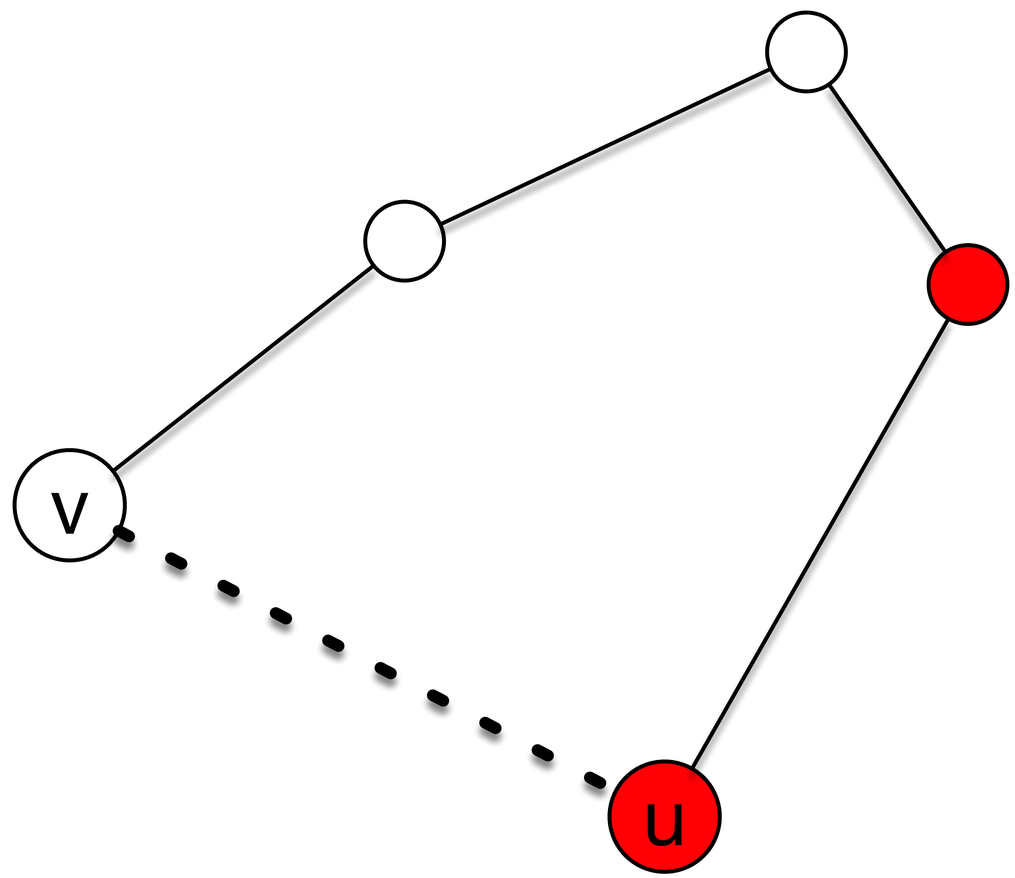
Minimum Spanning Trees

- This edge in T that crosses the cut also has weight 2 in our example, but for sure, it has weight ≥ the weight of (u, v)
- There is another edge



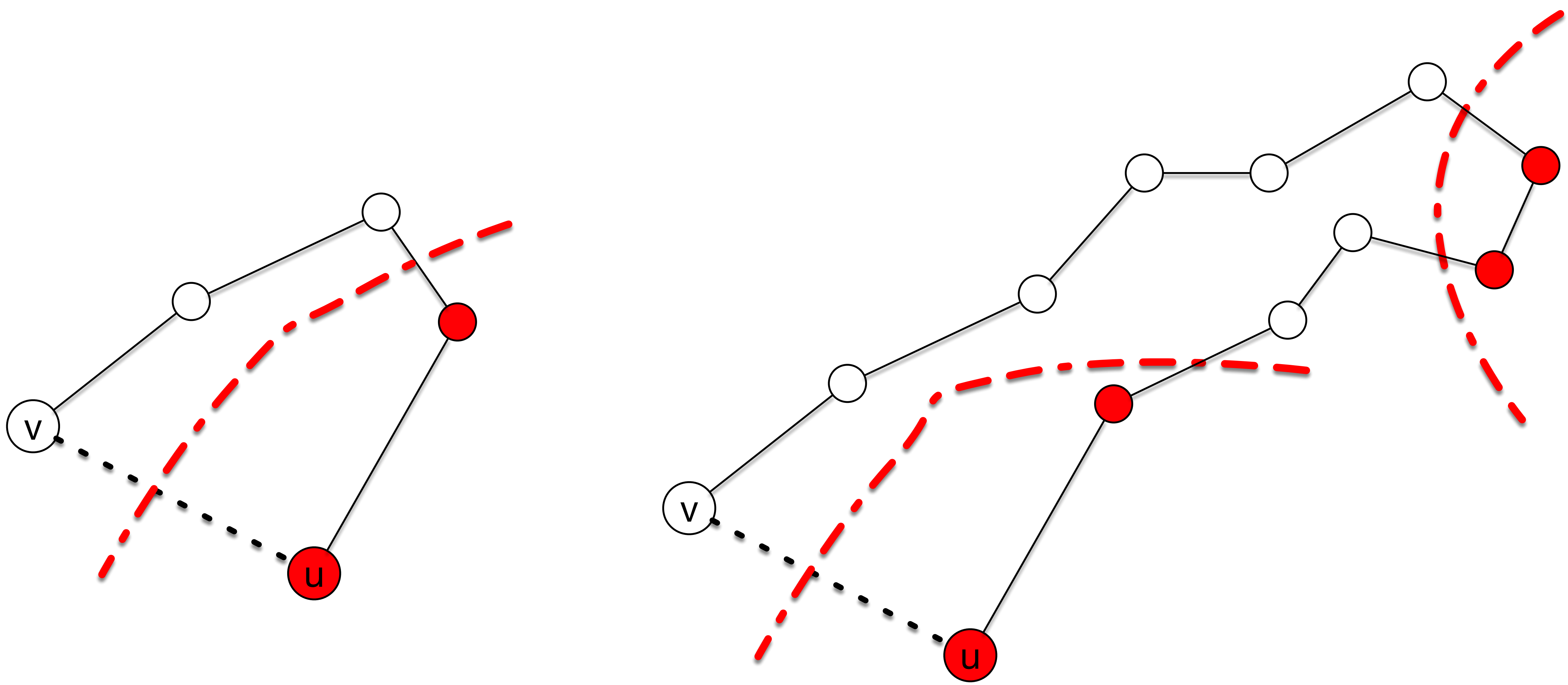
Minimum Spanning Trees

- There has to be a path from u to v in T because T is a spanning tree



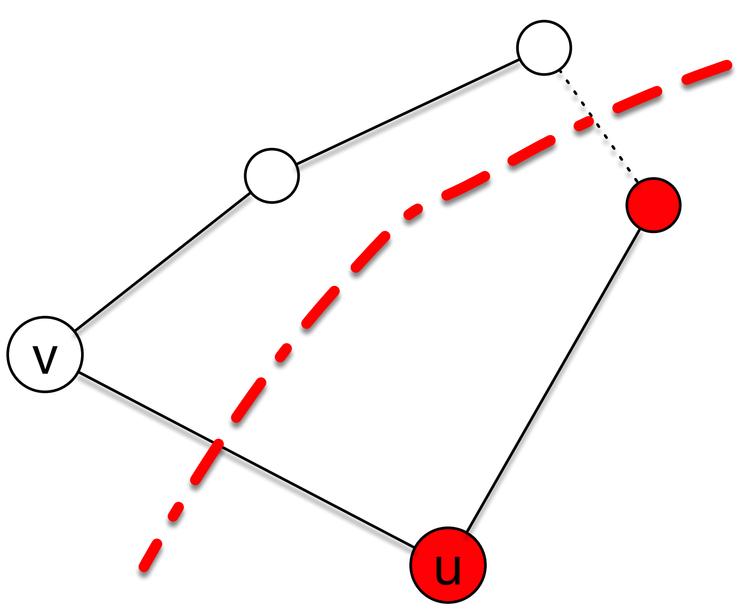
Minimum Spanning Trees

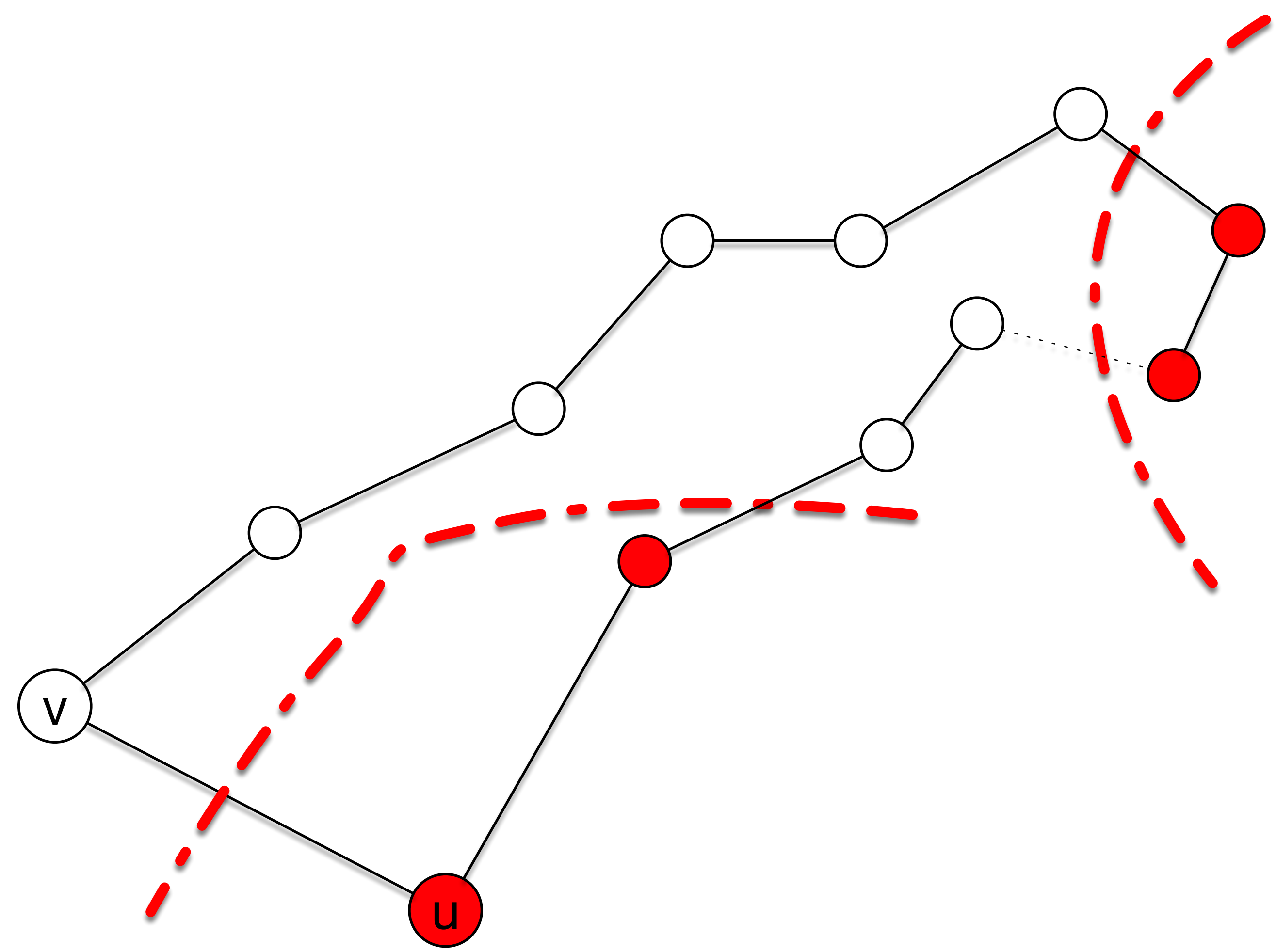
- This path has to have at least one edge that crosses the cut



Minimum Spanning Trees

- Take one of these edges and replace it with (u, v) in T
- Call the result T'





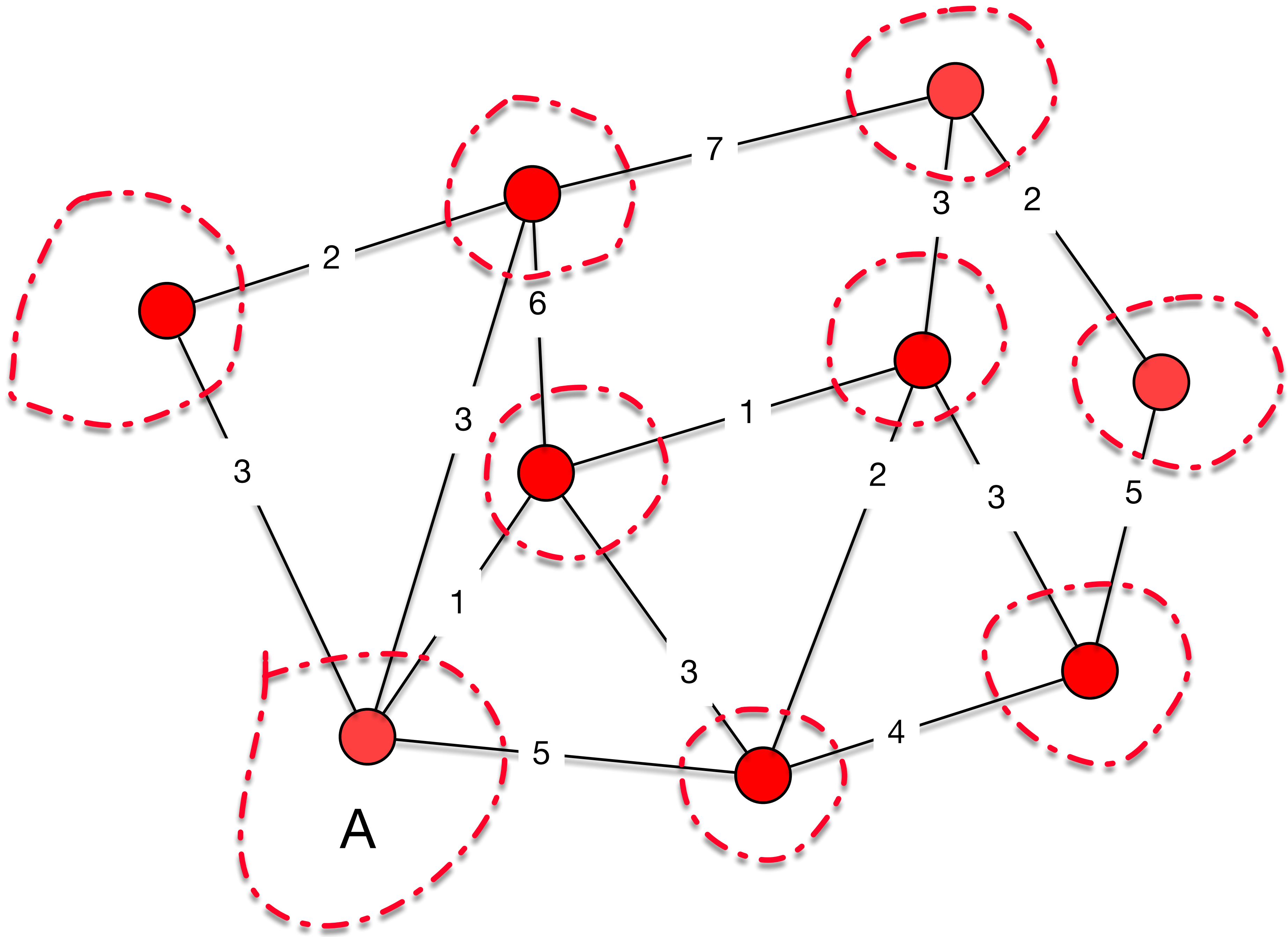
Minimum Spanning Trees

- T' still connects all of the vertices
    - If a, b are two vertices that are connected in T by the deleted edge:
        - Can reroute through the edge (u, v)
    - T' has a weight changed by replacing the weight of (u, v) with the weight of the deleted edge
        - But because the weight of (u, v) is minimal among all edges crossing the cut and the deleted edge also crossed the cut, T' weight can only be lower
- Thus, A after adding the edge (u, v) fulfills still the invariant. qed

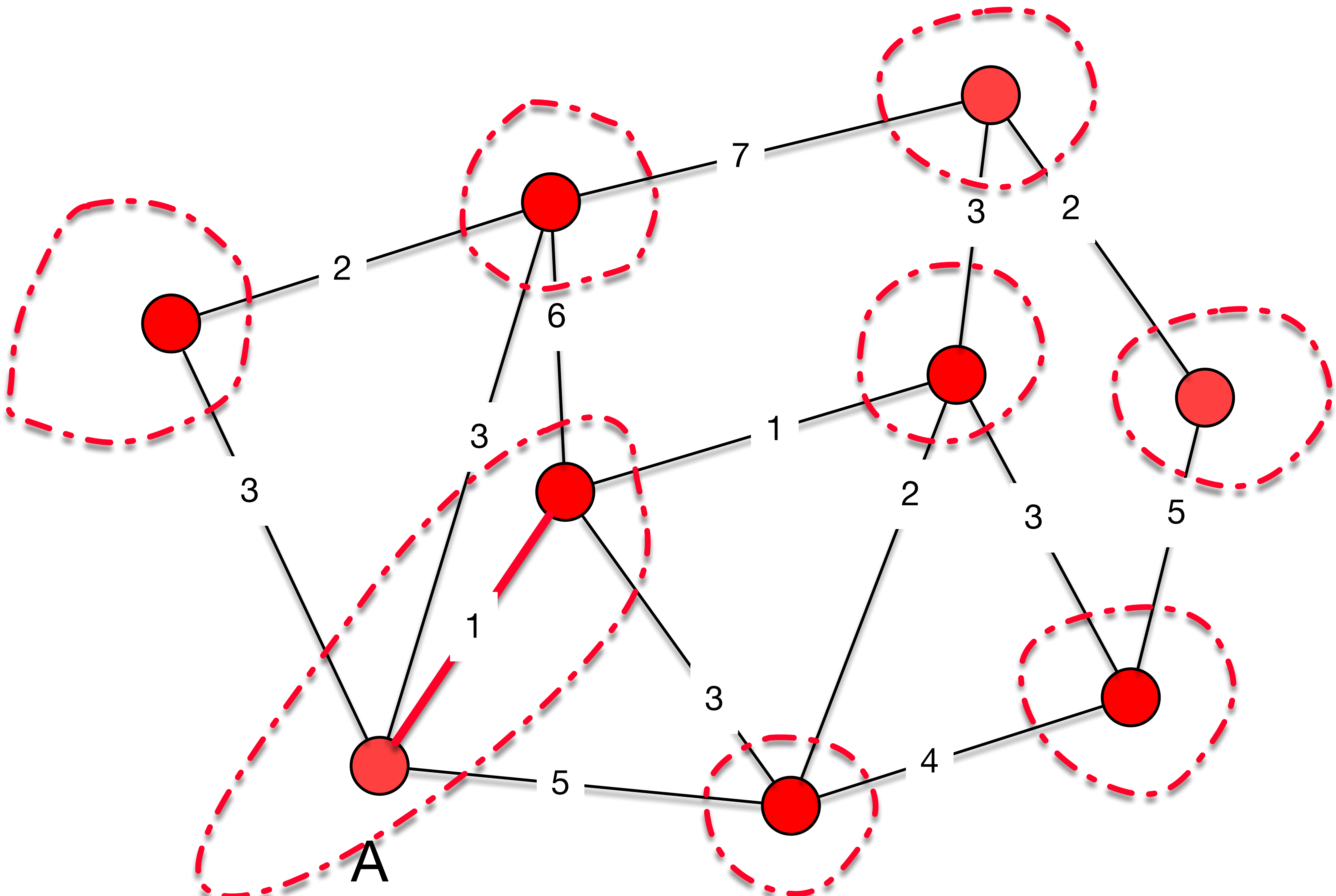
Kruskal's Algorithm

- Kruskal's algorithm works by joining subtrees
  - Start out with all vertices being their own subtrees
    - Thus, the cut is around all of the vertices
  - While we have more than one subtree:
    - We select a cutting edge (i.e. between different subtrees) with minimum weight
    - This combines two subtrees

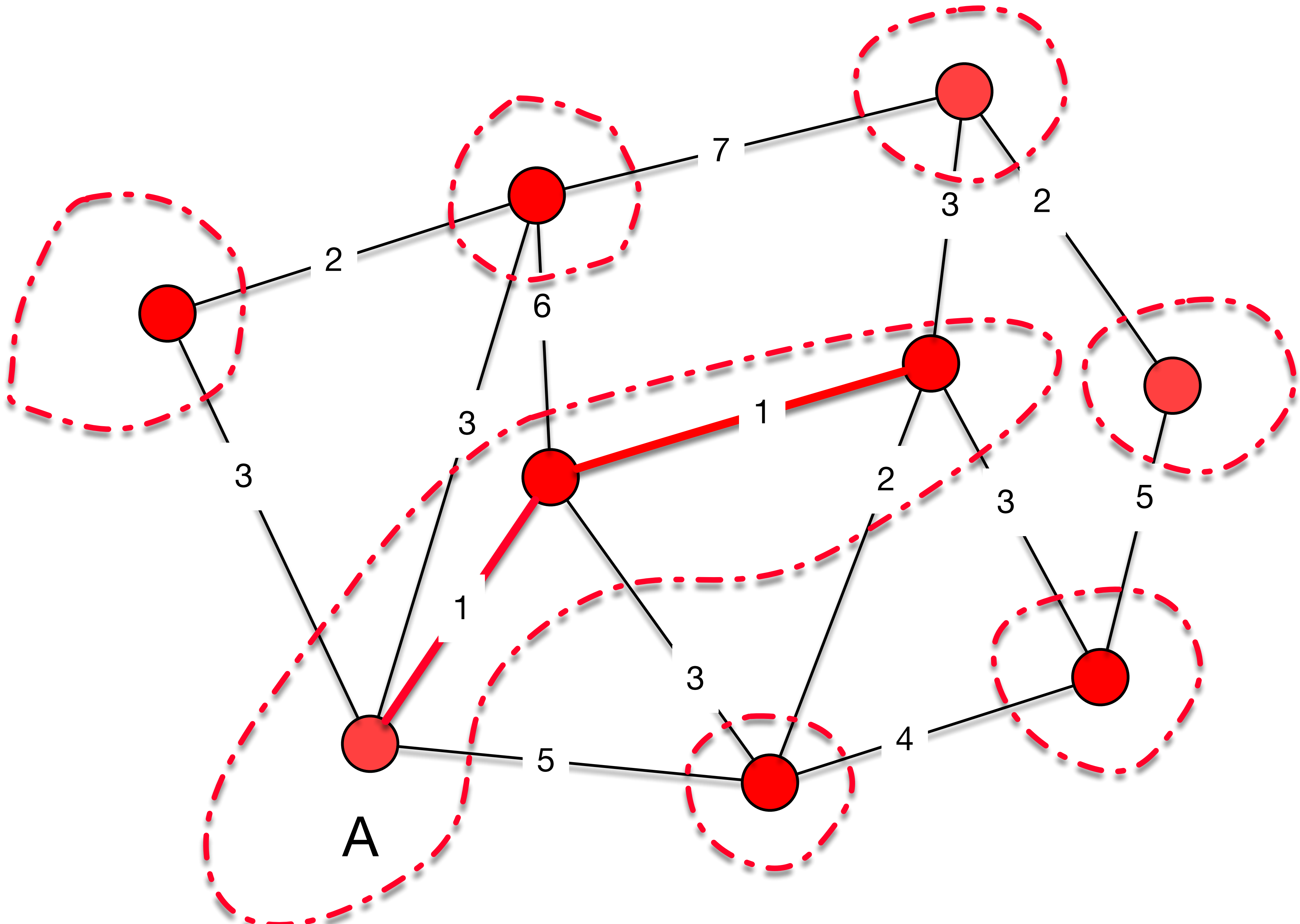
Kruskal's Algorithm



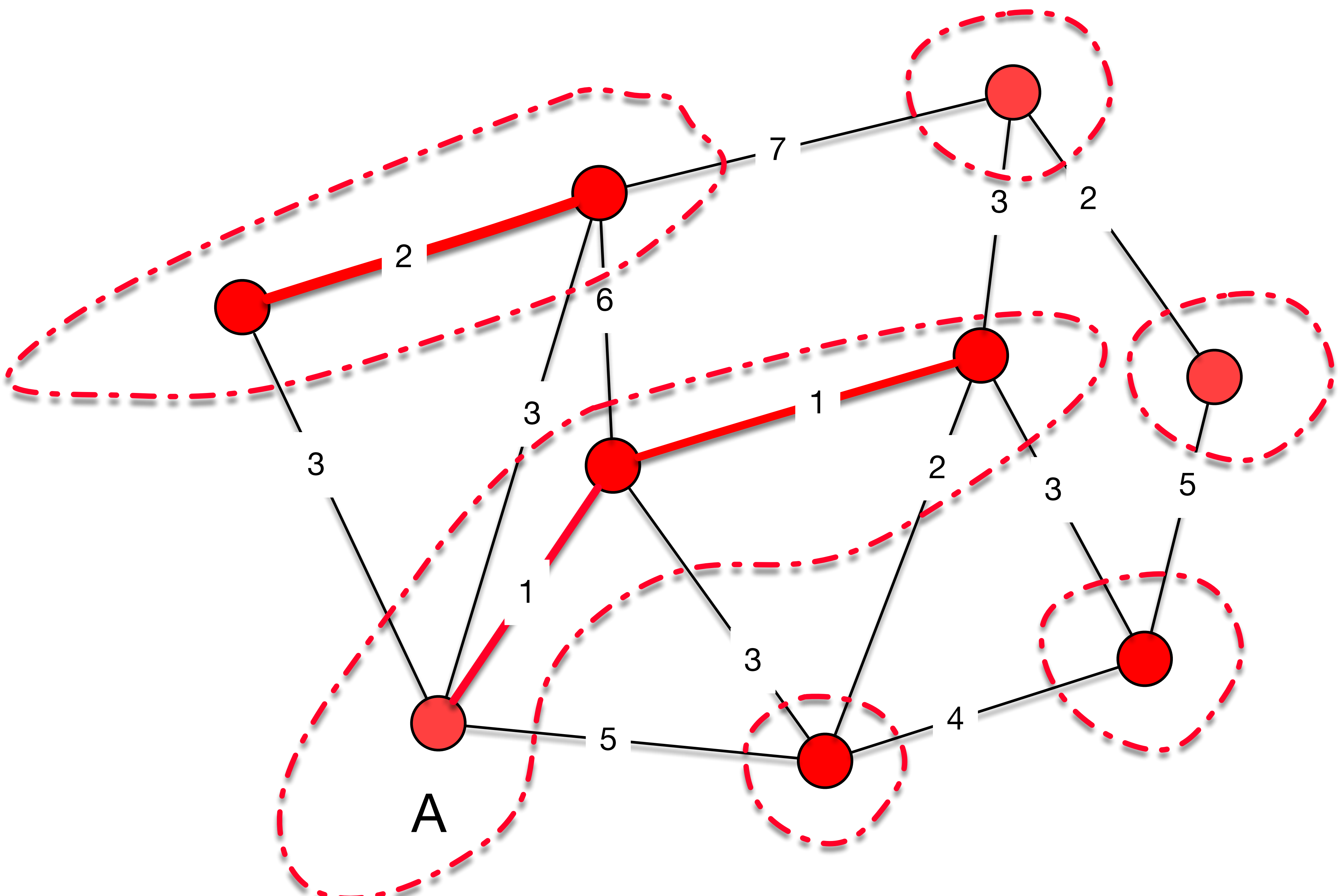
Kruskal's Algorithm



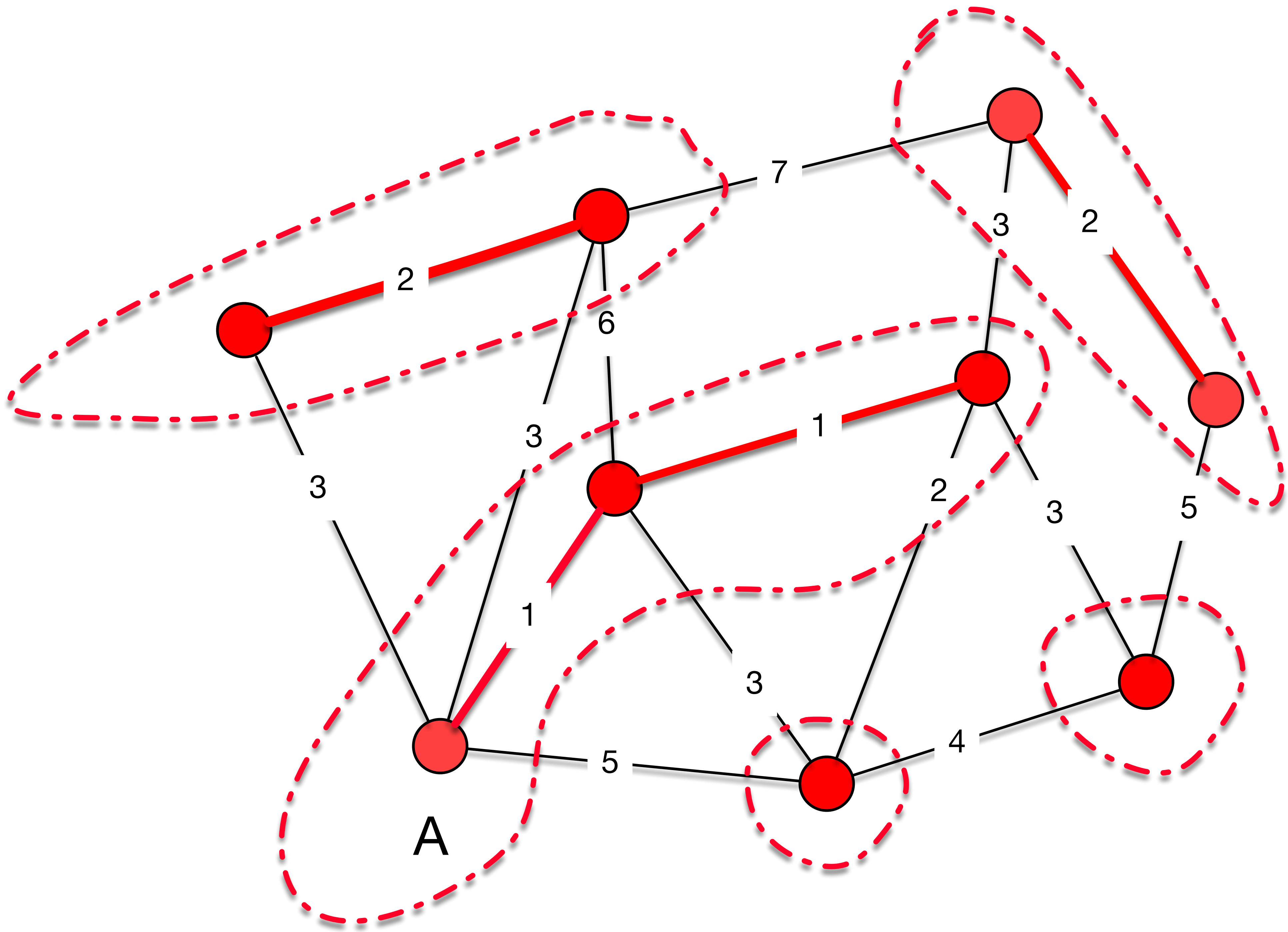
Kruskal's Algorithm



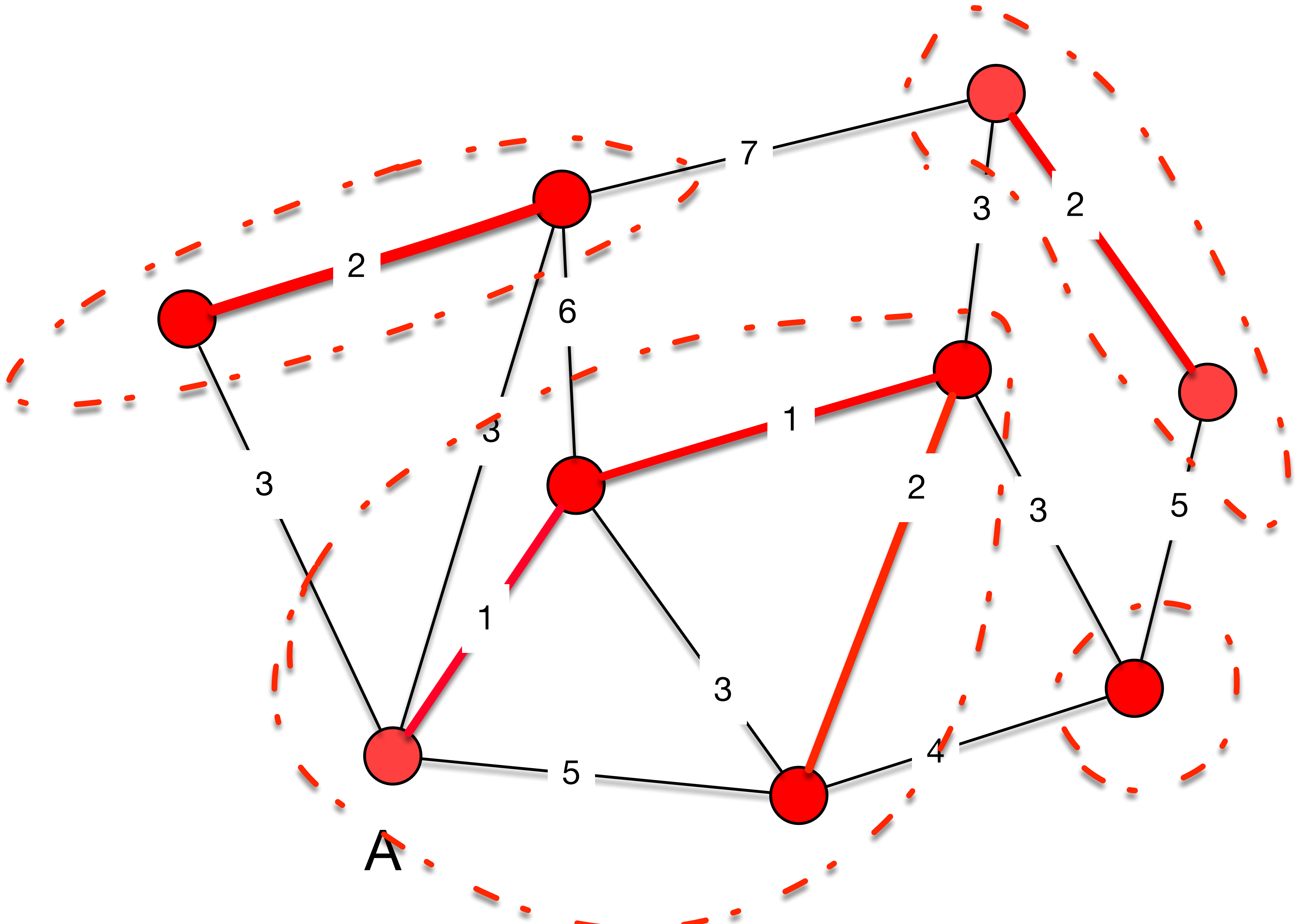
Kruskal's Algorithm



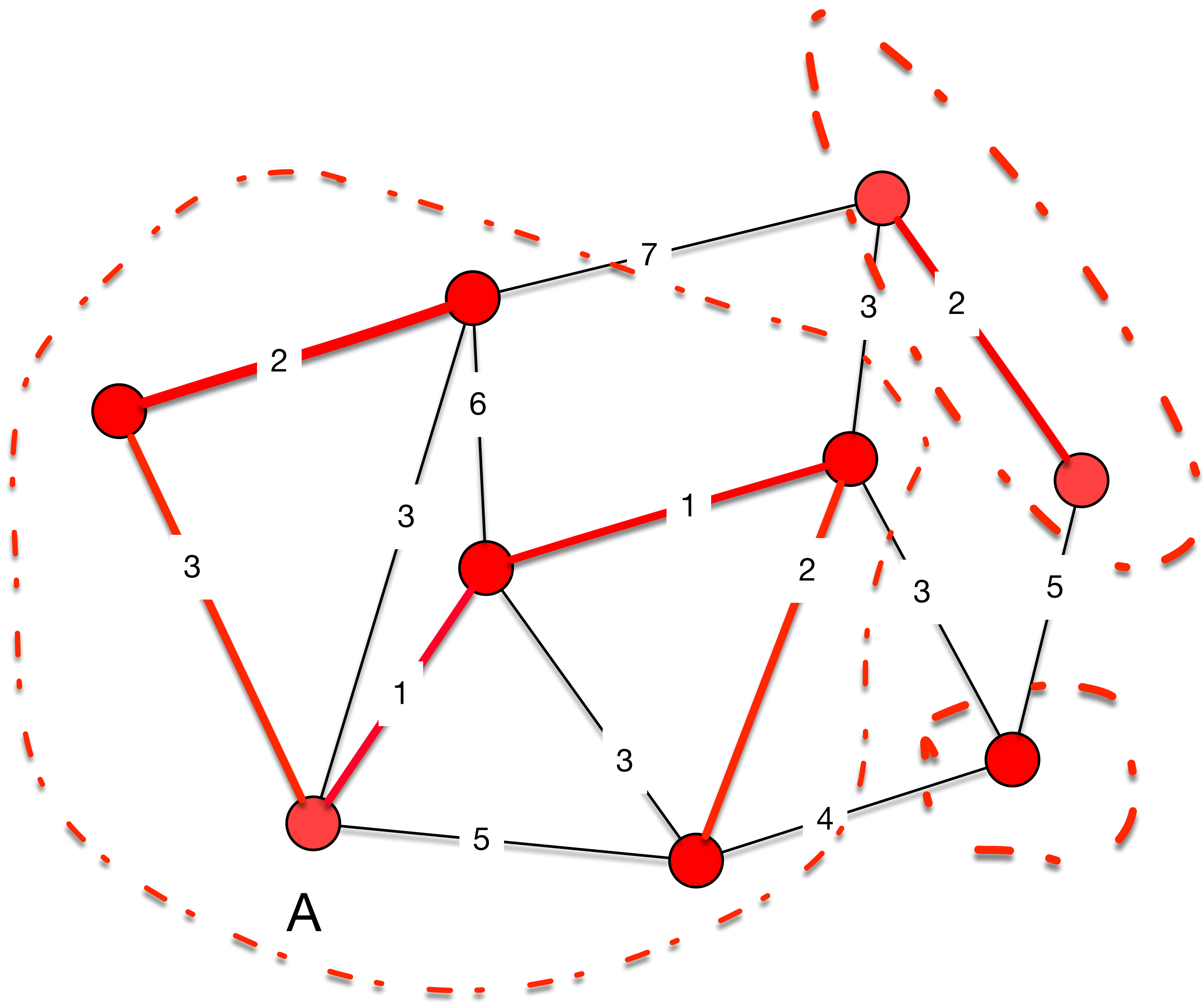
Kruskal's Algorithm



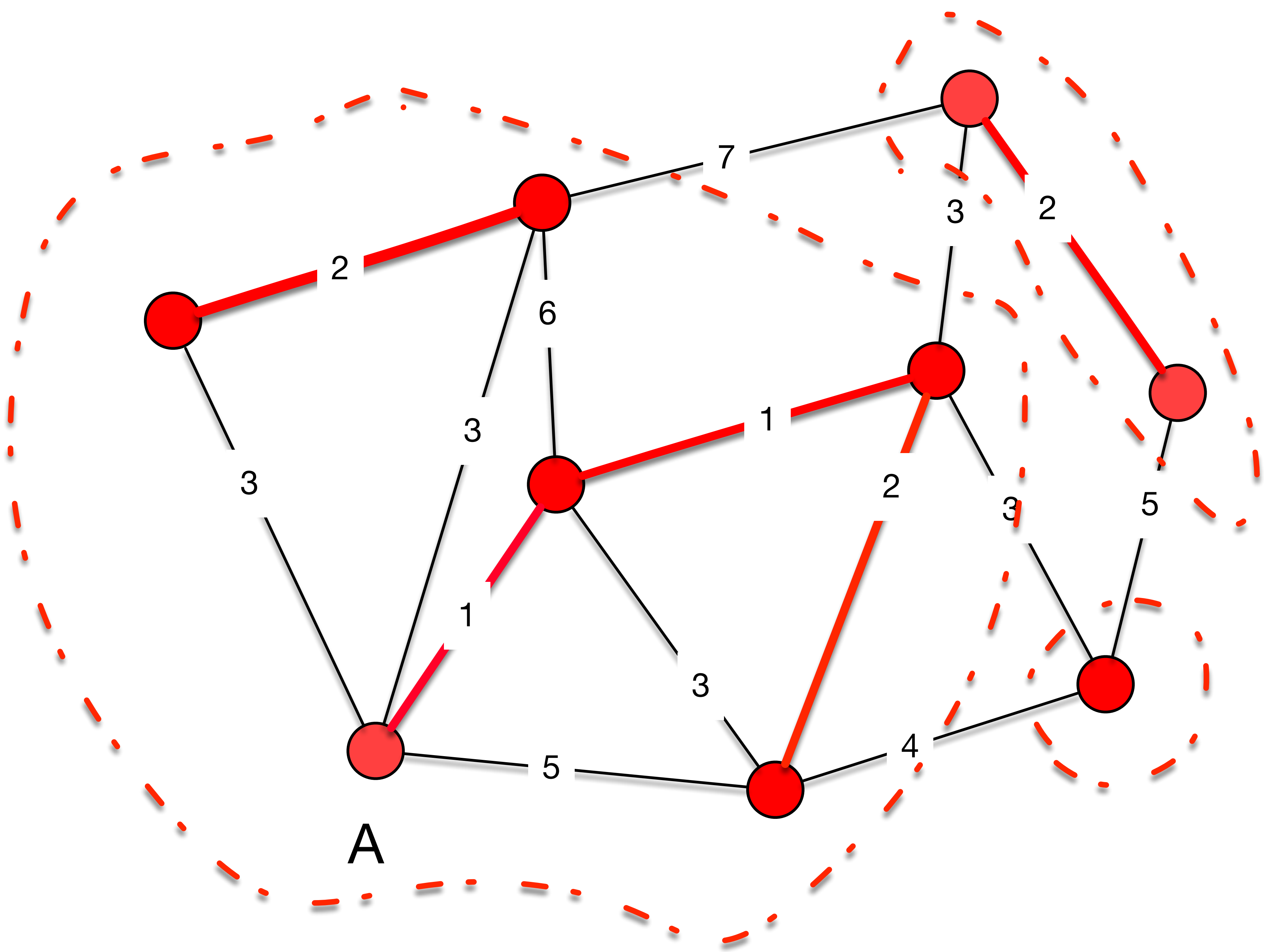
Kruskal's Algorithm



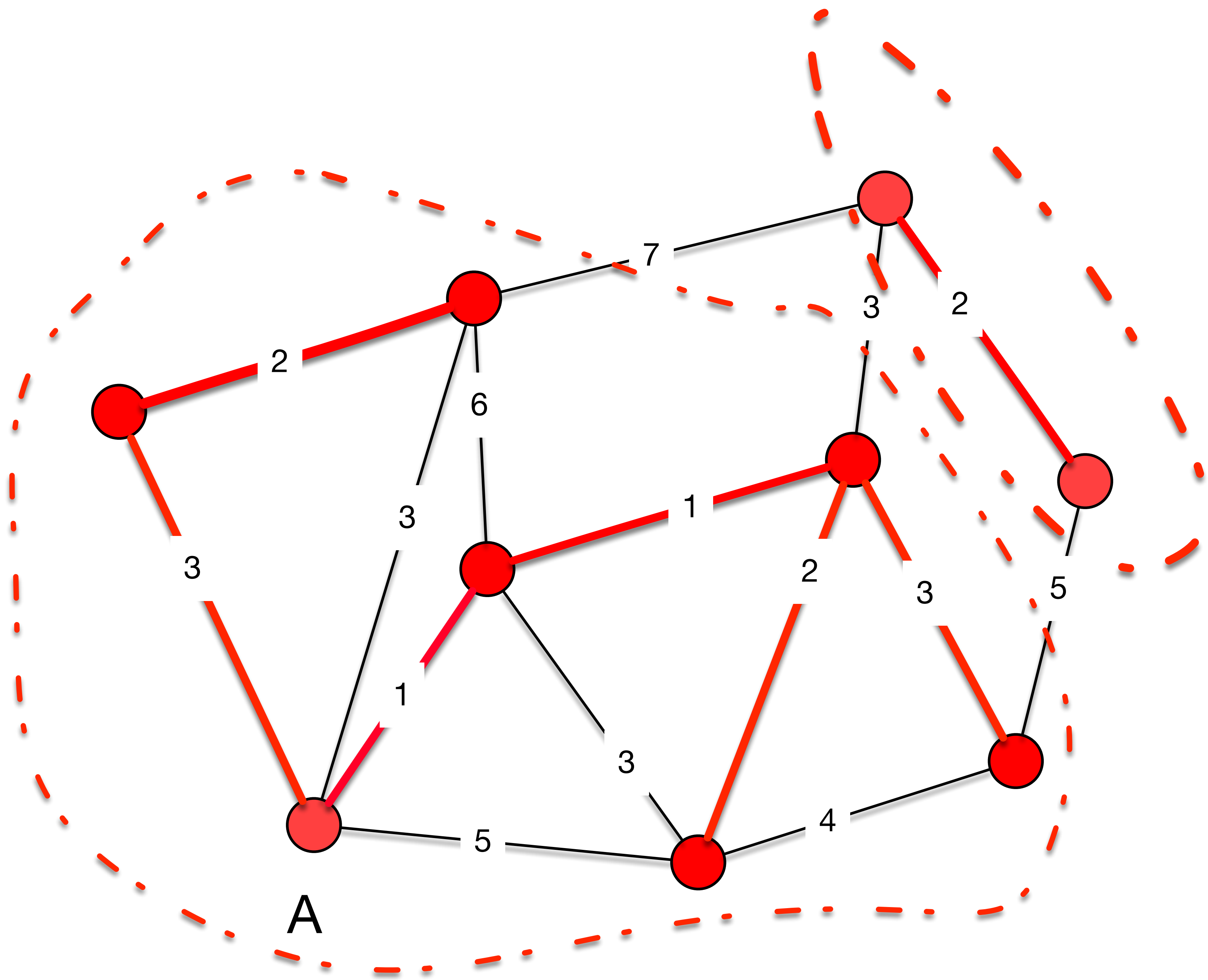
Kruskal's Algorithm



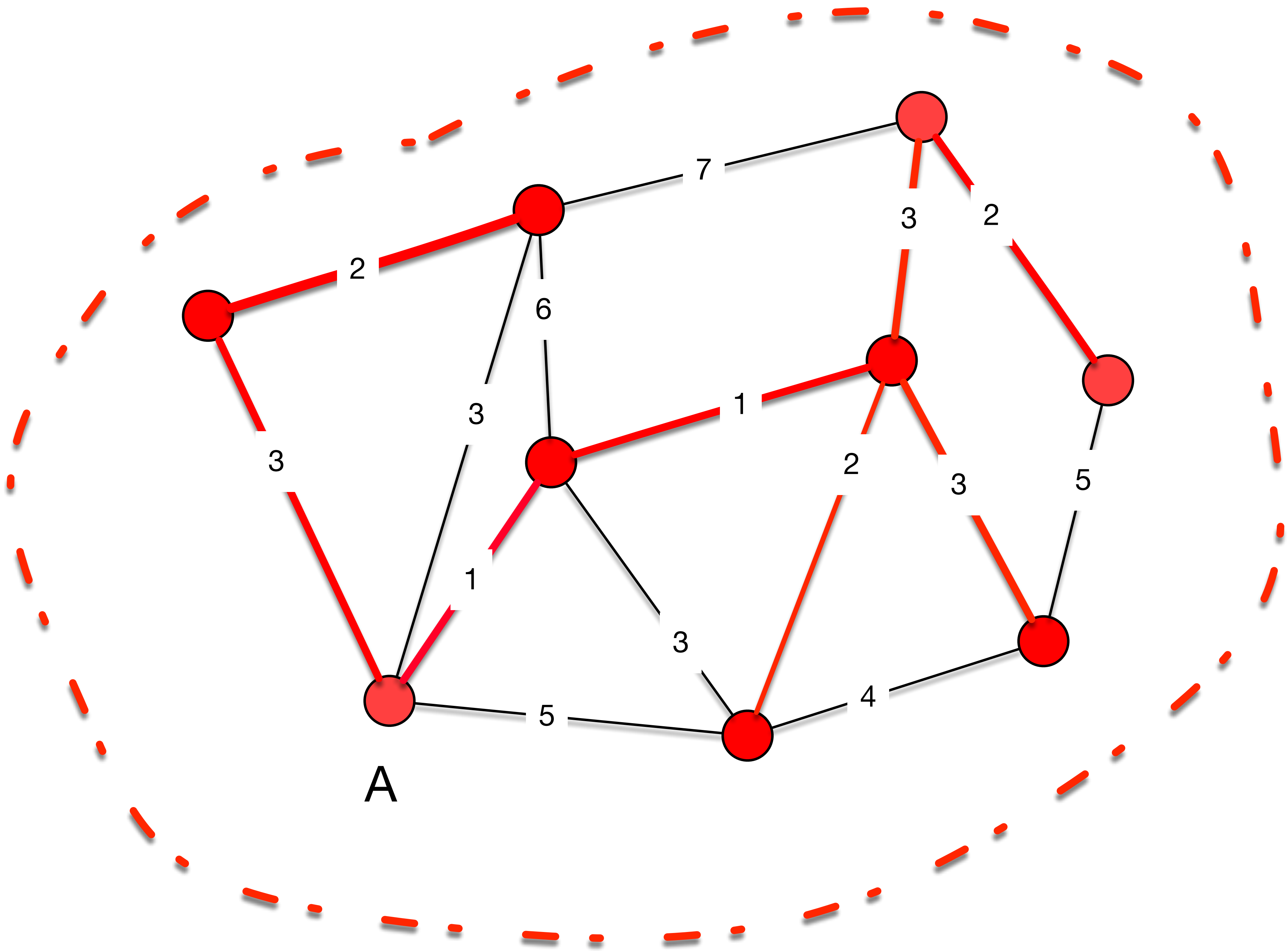
Kruskal's Algorithm



Kruskal's Algorithm



Kruskal's Algorithm

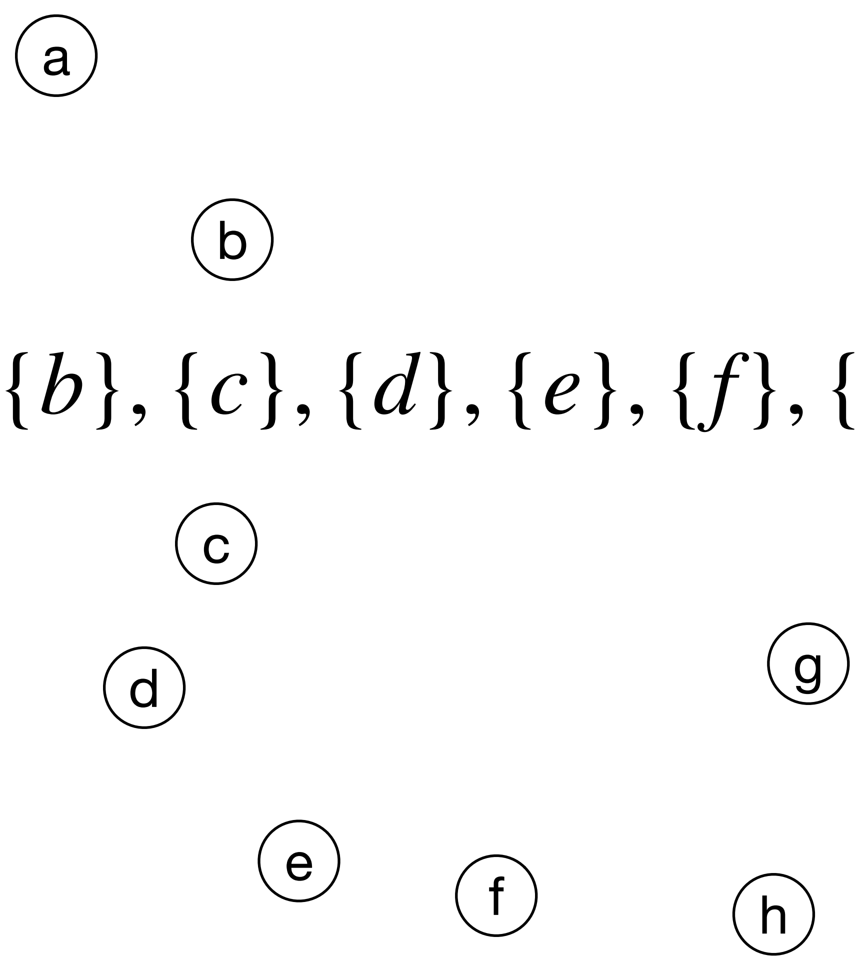


Kruskal's Algorithm

- Because Kruskal's algorithm only adds safe edges, it generates a minimum weight spanning tree
- How to organize it?
  - We can order all of the edges by weight
  - And then remove edges if they no longer are cutting edges
  - Best way:
    - Maintain vertices in the same subtree in a set
    - Determine quickly whether something is in a set

Kruskal's Algorithm

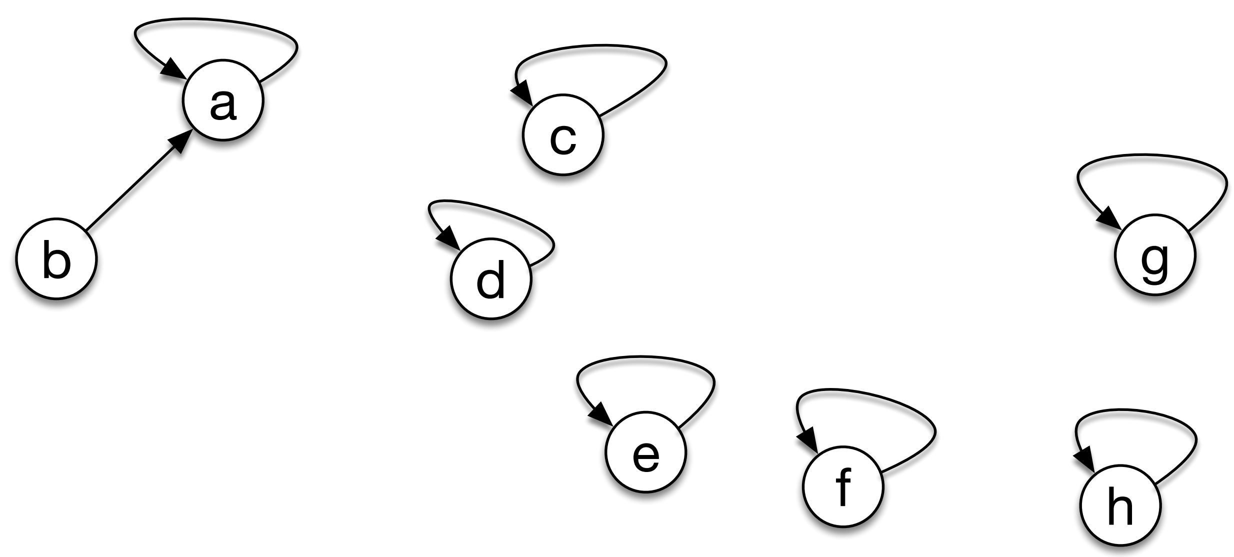
- Best solution known to humanity for the disjoint set problem:
  - have vertices organized by a directed edge to the "set leader"



$$\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}$$

Kruskal's Algorithm

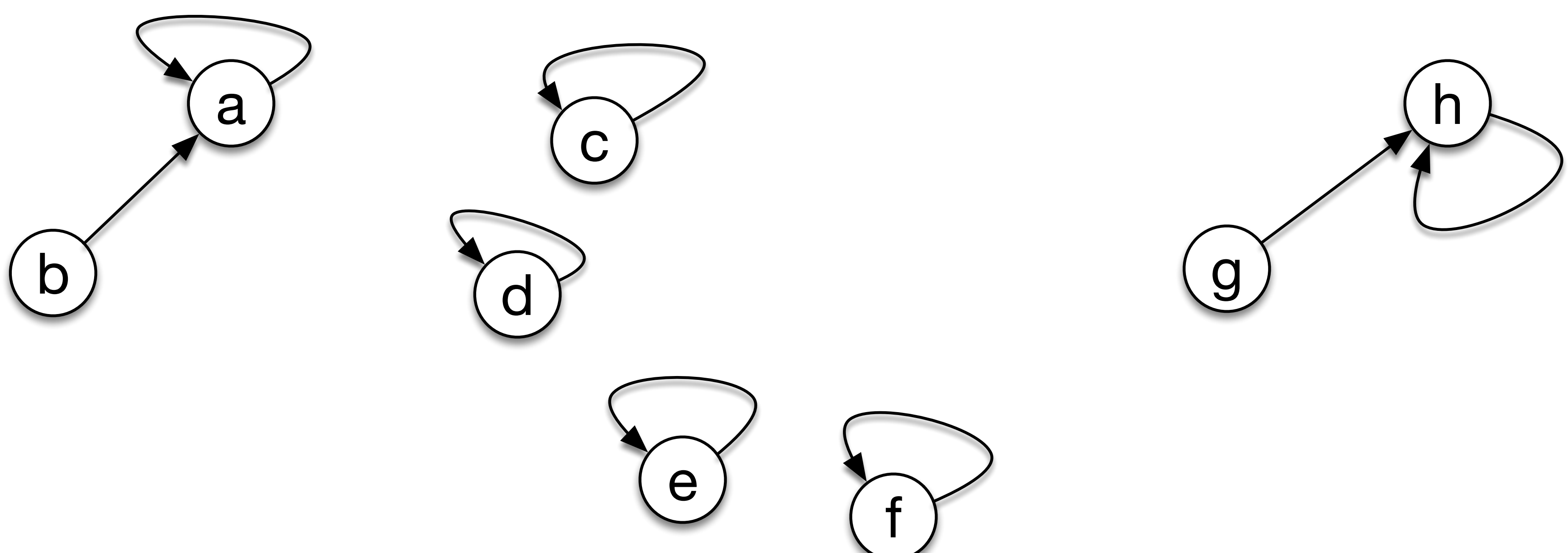
- If we unite $\{a\}$ and $\{b\}$, we have one point to the other



$\{a, b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}$

Kruskal's Algorithm

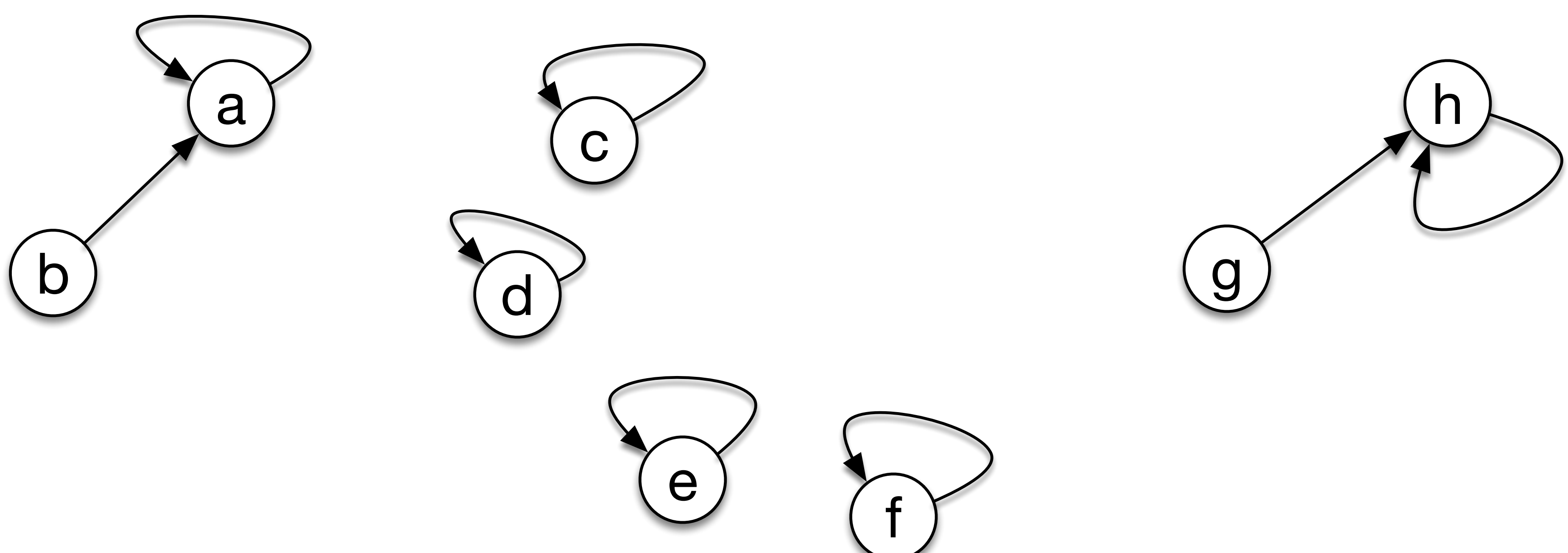
- Same if we unite $\{g\}$ and $\{h\}$



$$\{a, b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g, h\},$$

Kruskal's Algorithm

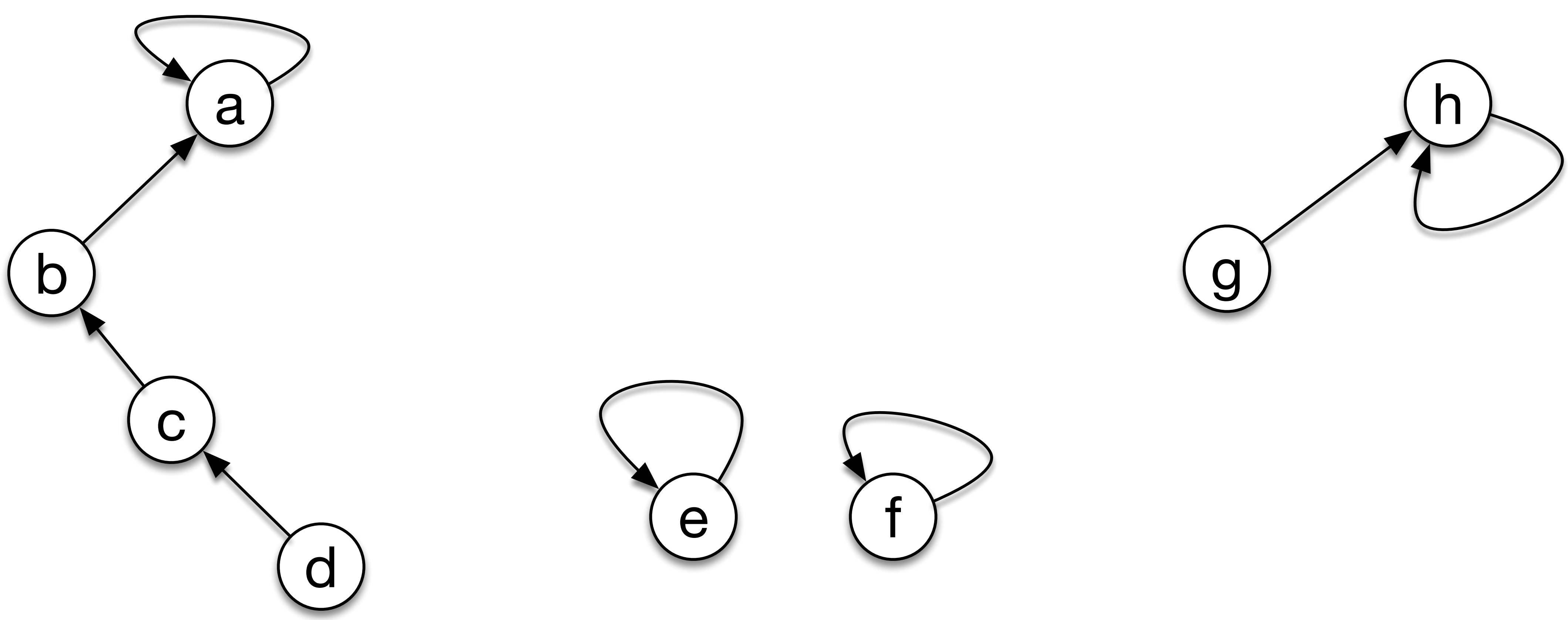
- If we ask whether b and g are in two different components, we follow the arrow and see whether the leaders are the same or not.



$$\{a,b\},\{c\},\{d\},\{e\},\{f\},\{g,h\},$$

Kruskal's Algorithm

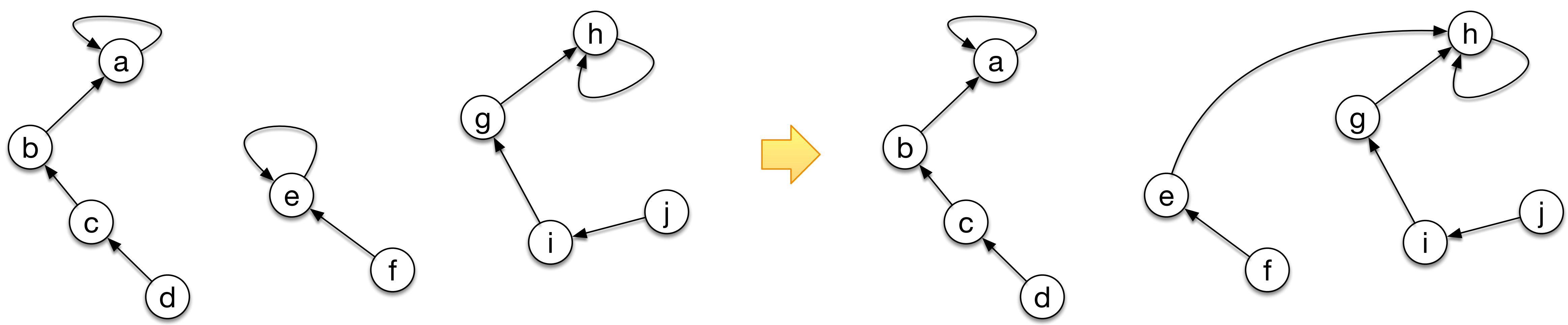
- This can be optimized:
  - There is the possibilities of having long chains



$$\{a, b, c\, d\}, \{e\}, \{f\}, \{g, h\},$$

Kruskal's Algorithm

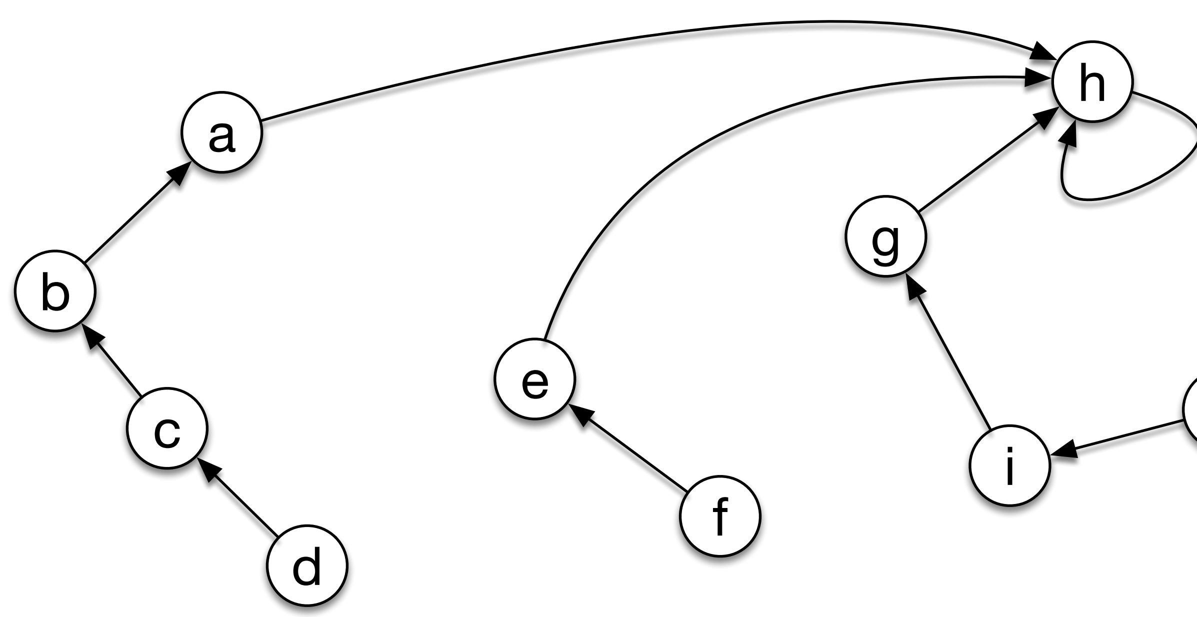
- When we join, we connect one leader to the other leader
  - Always make the larger set the head



$$\{a, b, c\ d\}, \{e, f\} \cup \{g, h, i, j\},$$

Kruskal's Algorithm

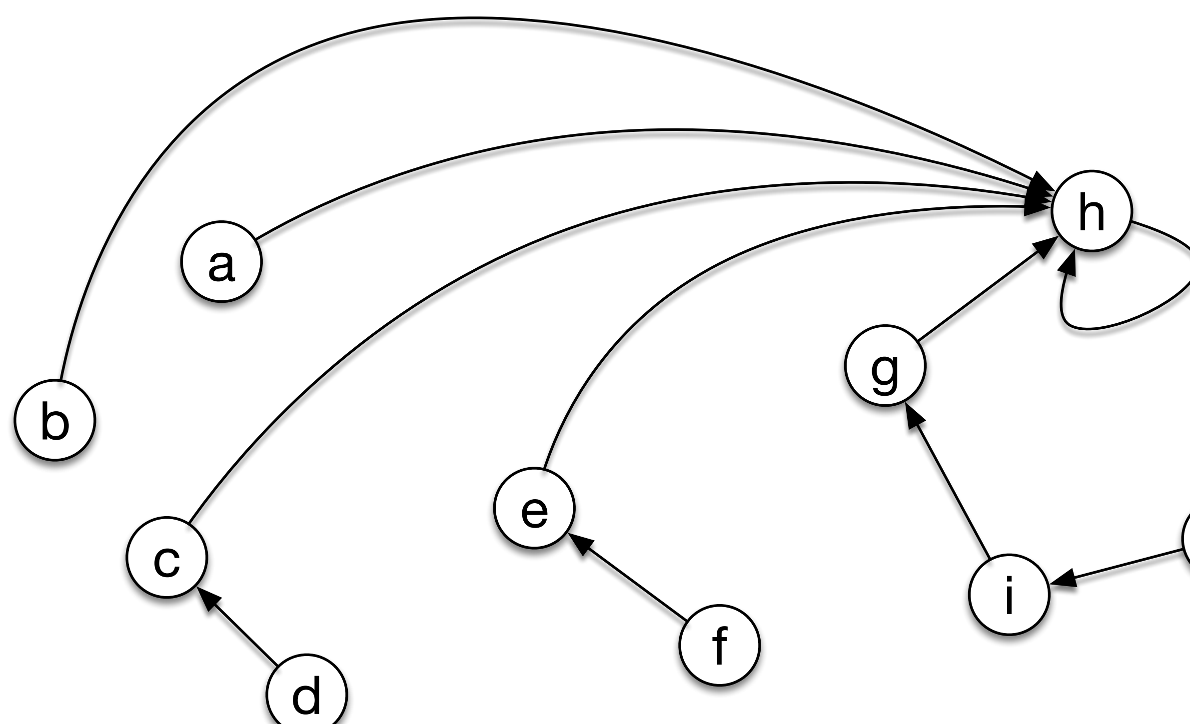
- When we do a look up:
  - What is the head of c?



  - Follow three links to get to 'h'

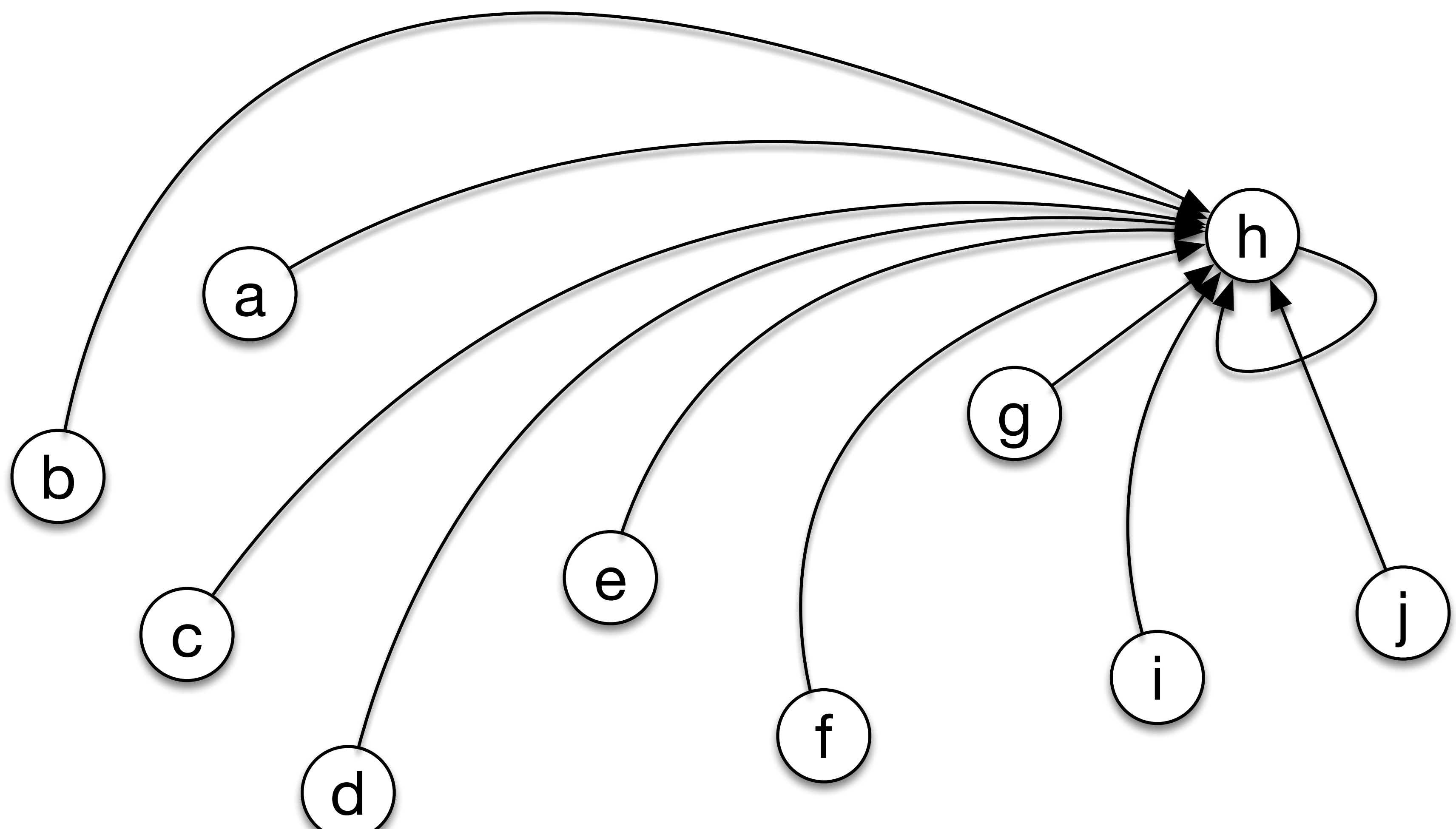
Kruskal's Algorithm

- When we do a look up:
  - Reconnect the node and all we travel to directly to the head



Kruskal's Algorithm

- Best possible case: Every node points directly to the head



Kruskal's Algorithm

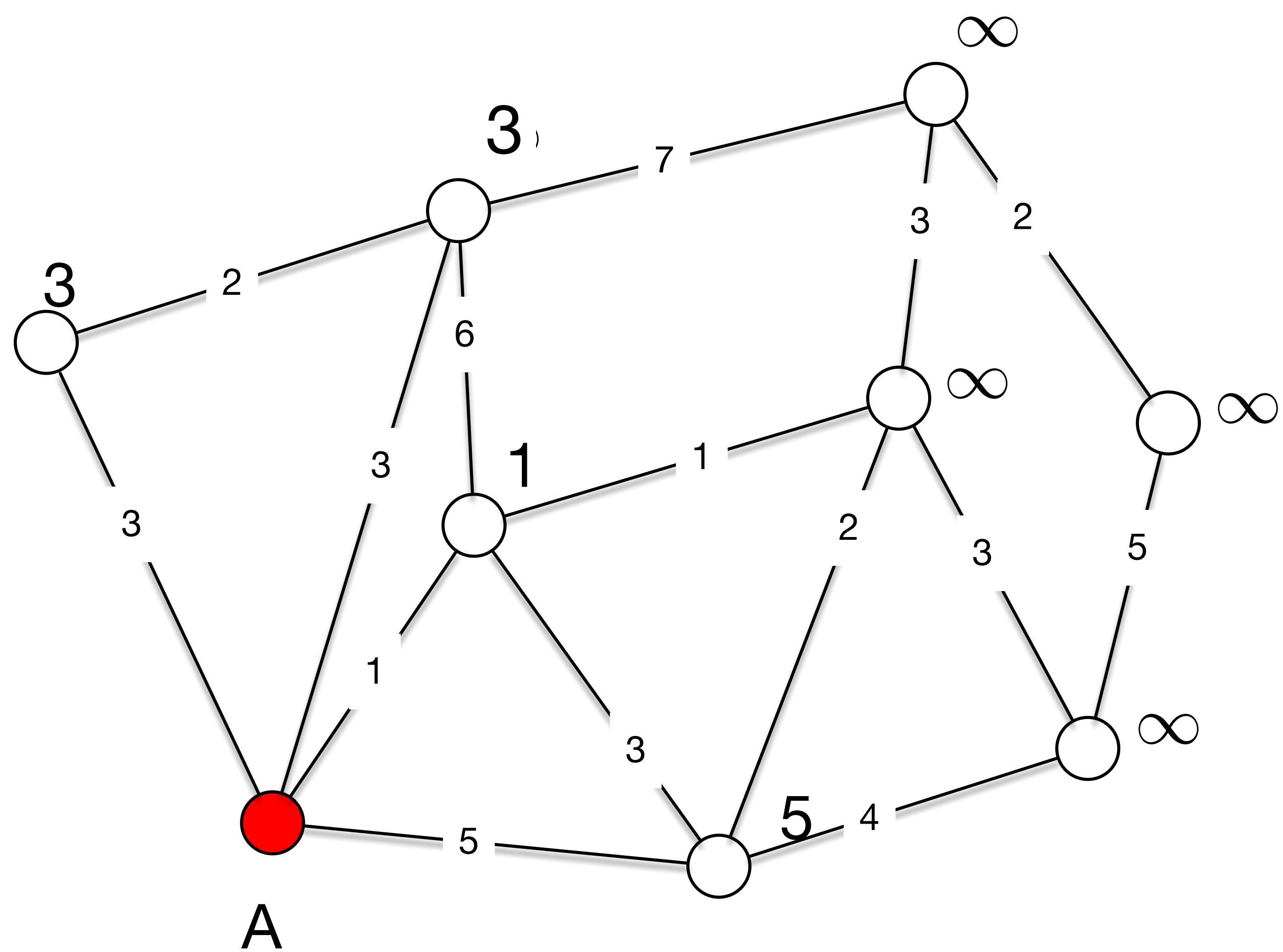
- With this "disjoint union data structure":
  - Maintaining the disjoint set data structure costs $\alpha(|V|)$ per operation where α is a function that grows very slowly
  - Kruskal's algorithm then runs in time $O(|E|\log(|E|))$

Prim's Algorithm

- Prim's algorithm starts A at a single node and then adds edges to it.
- Thus, the intermediate results are always connected
  - Maintain a priority queue of all other vertices
    - The vertices are ordered by distance to A

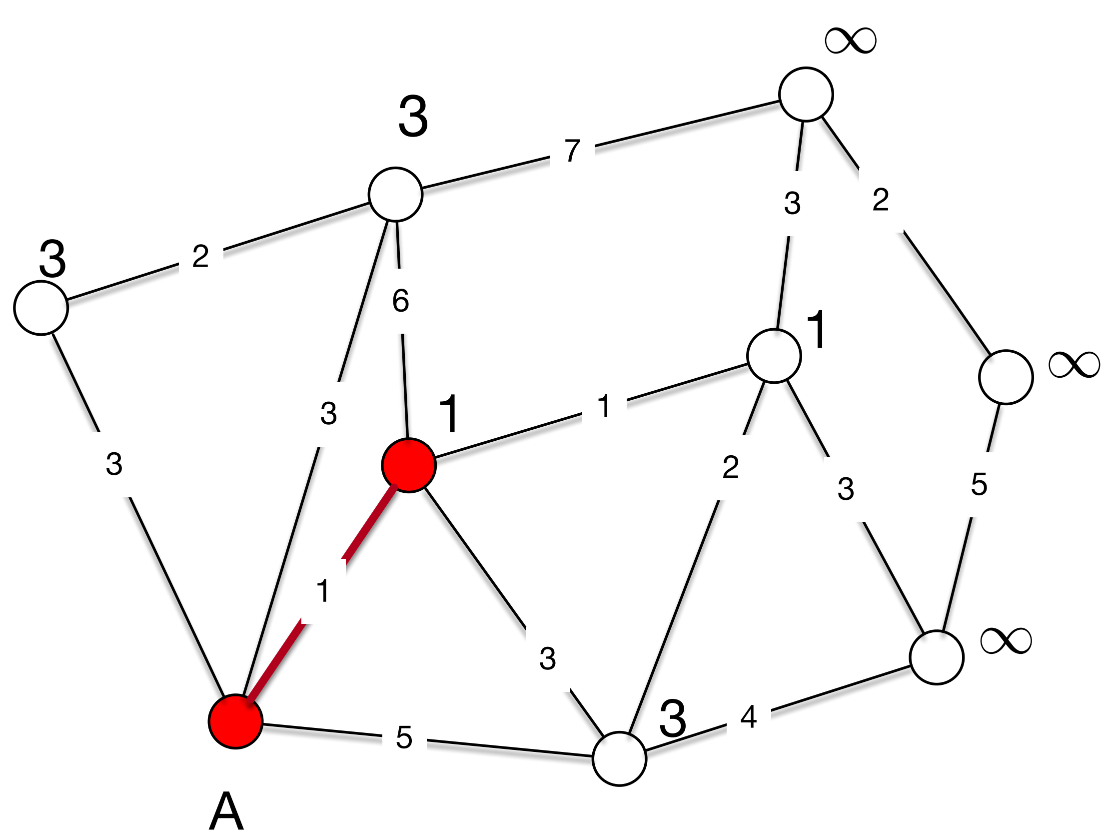
Prim's Algorithm

- We use the same example as before
- We can start at any node

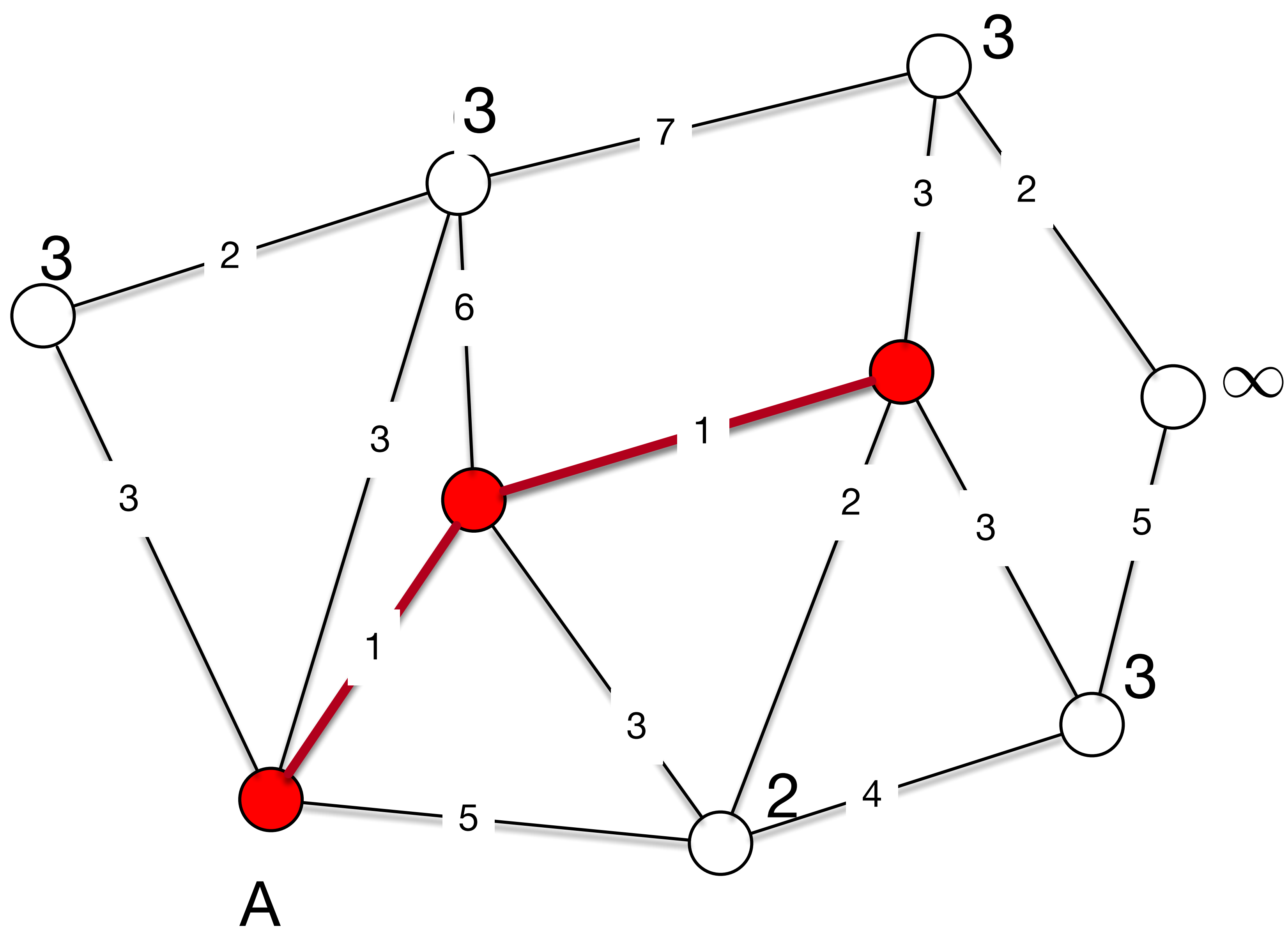


Prim's Algorithm

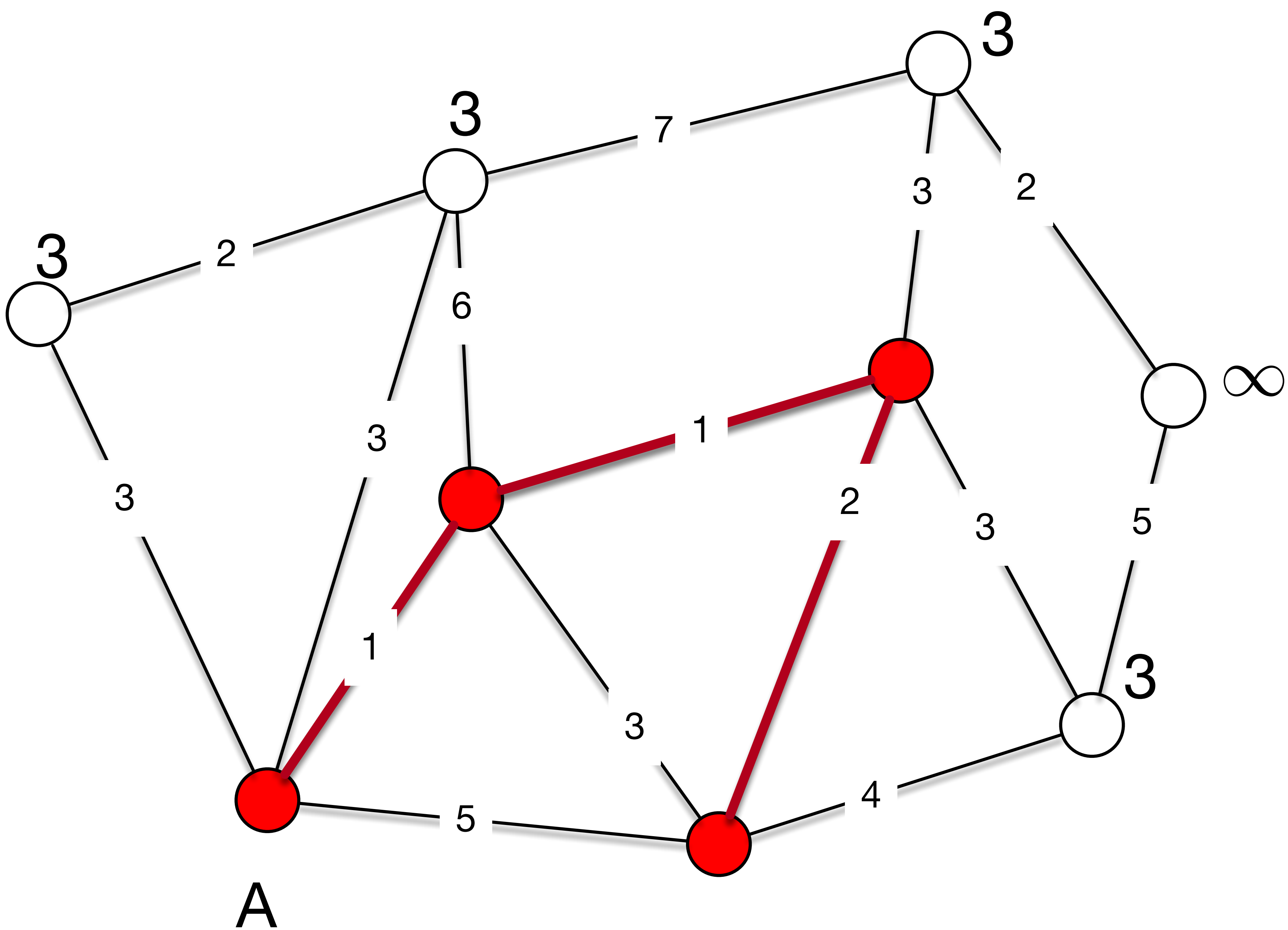
- The priority queue tells us which node to select
- After selecting edge and node, we need to update some nodes
- Namely those in the adjacency list of the new node



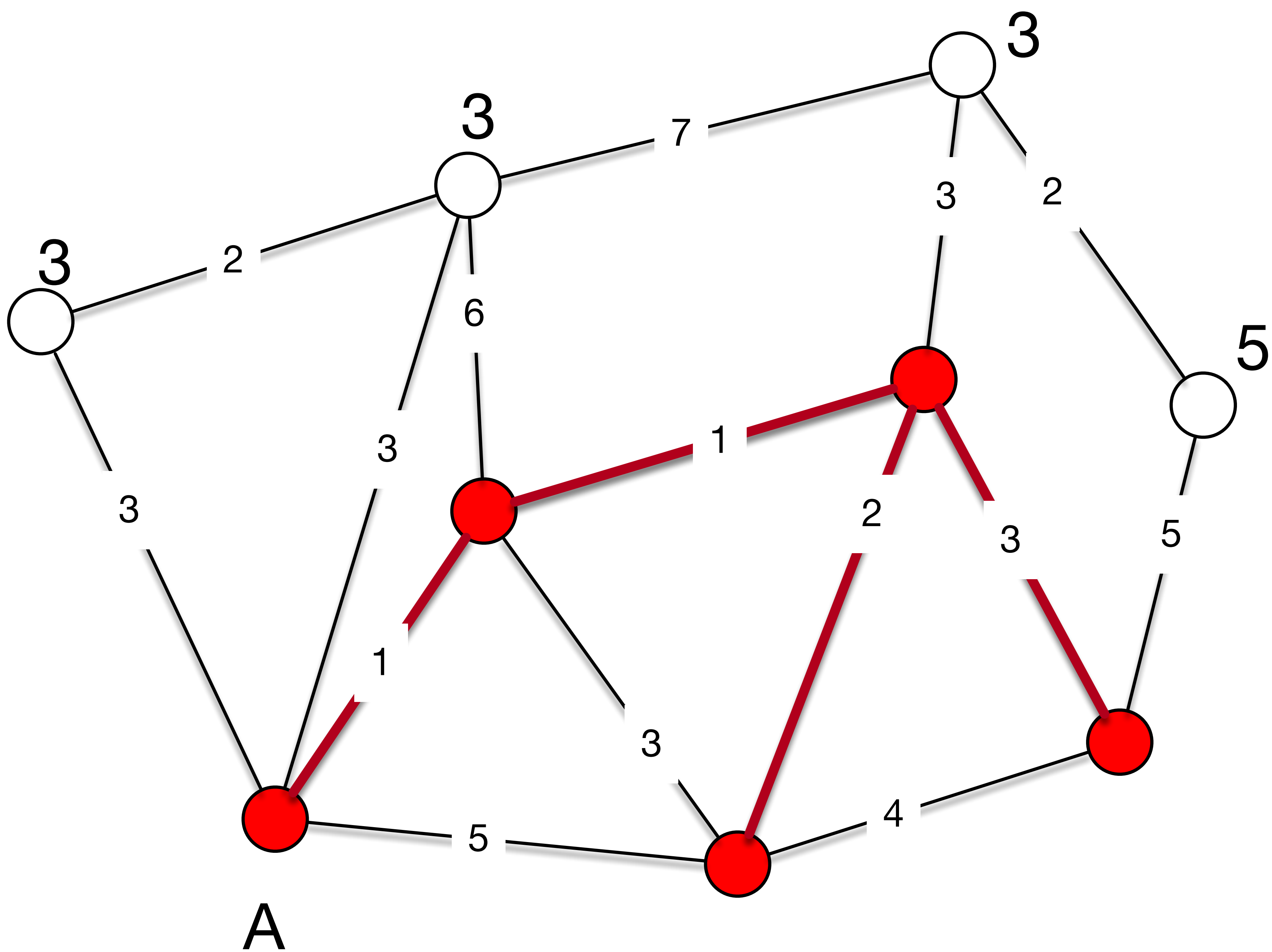
Prim's Algorithm



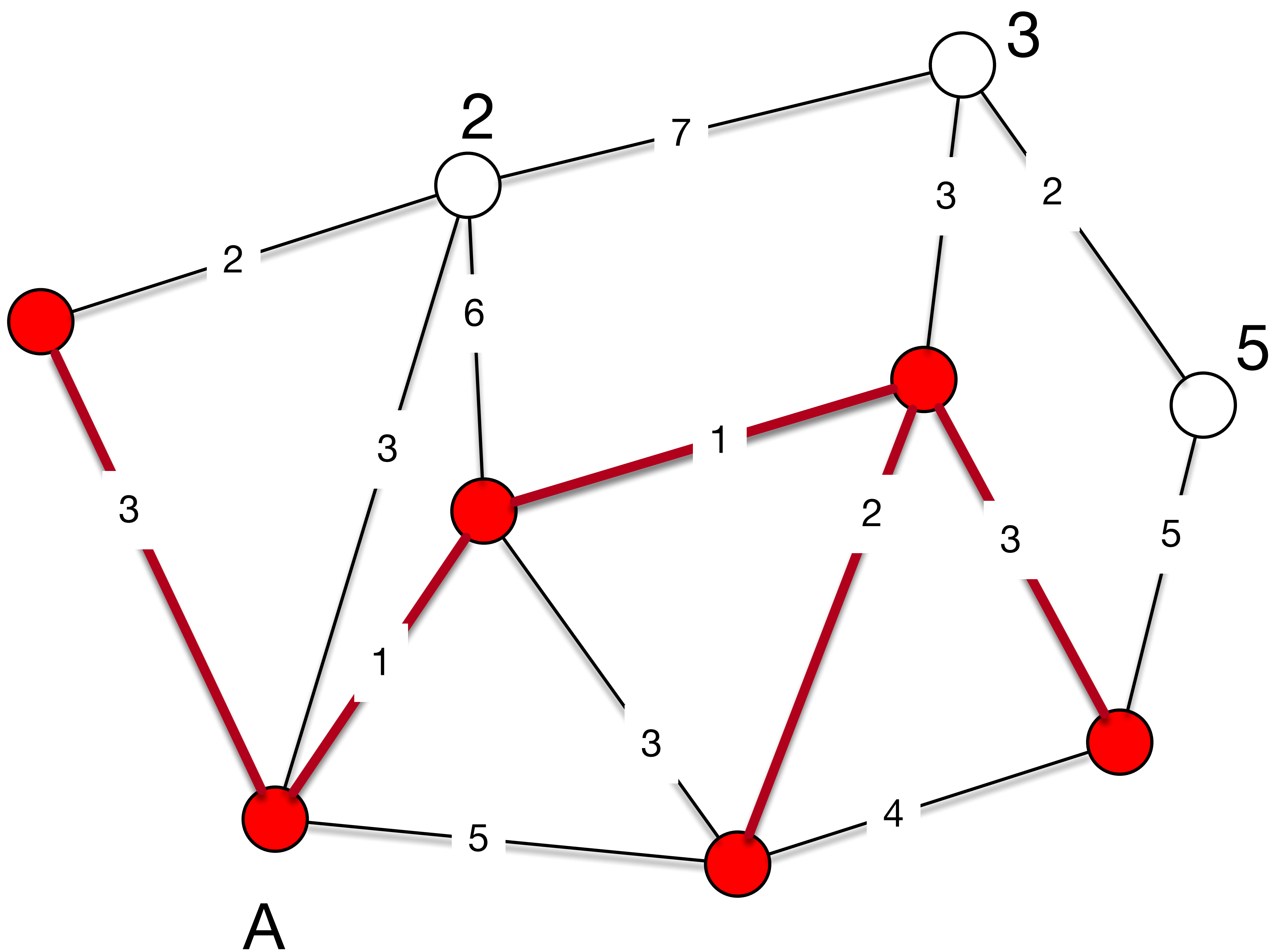
Prim's Algorithm



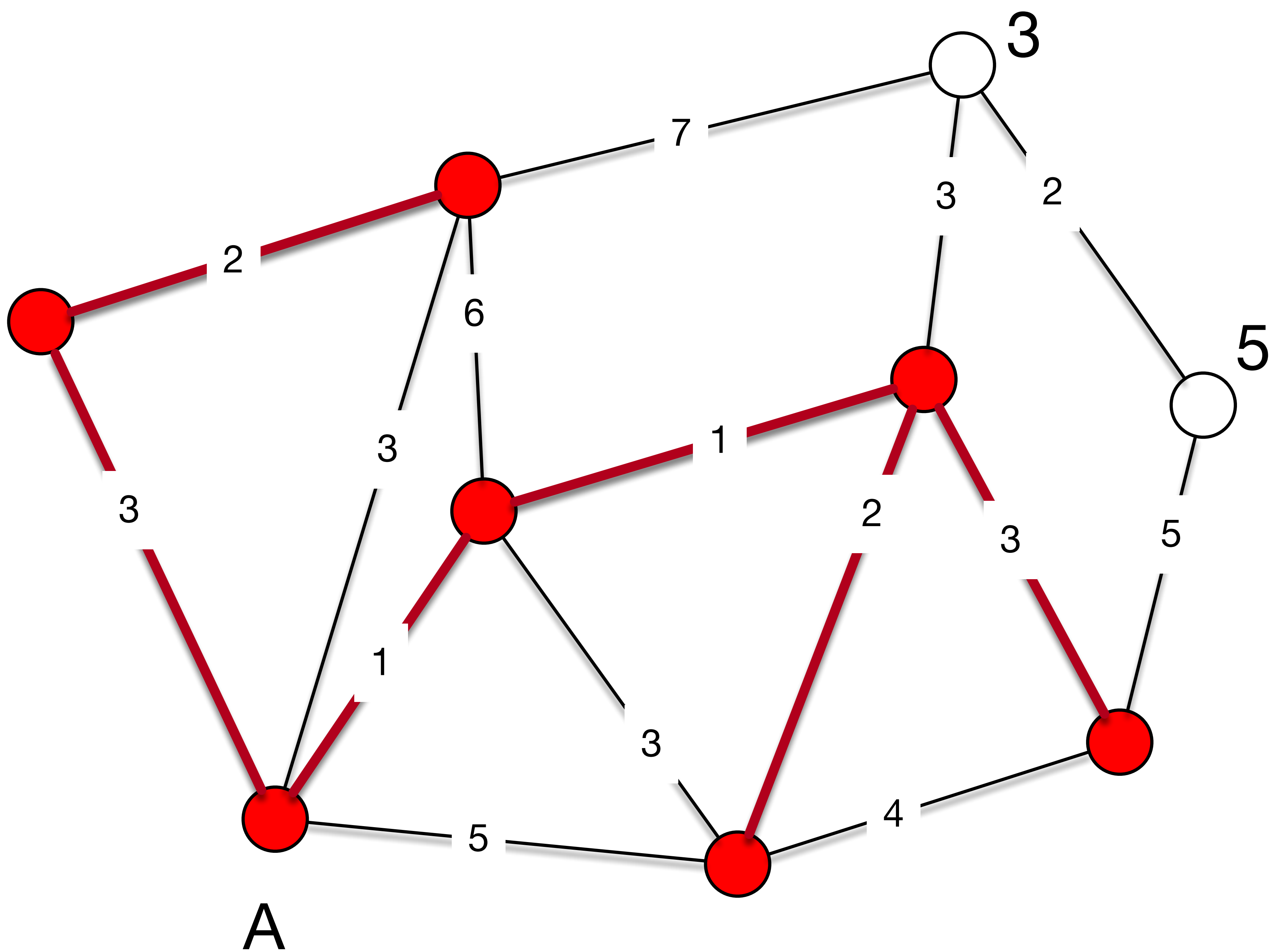
Prim's Algorithm



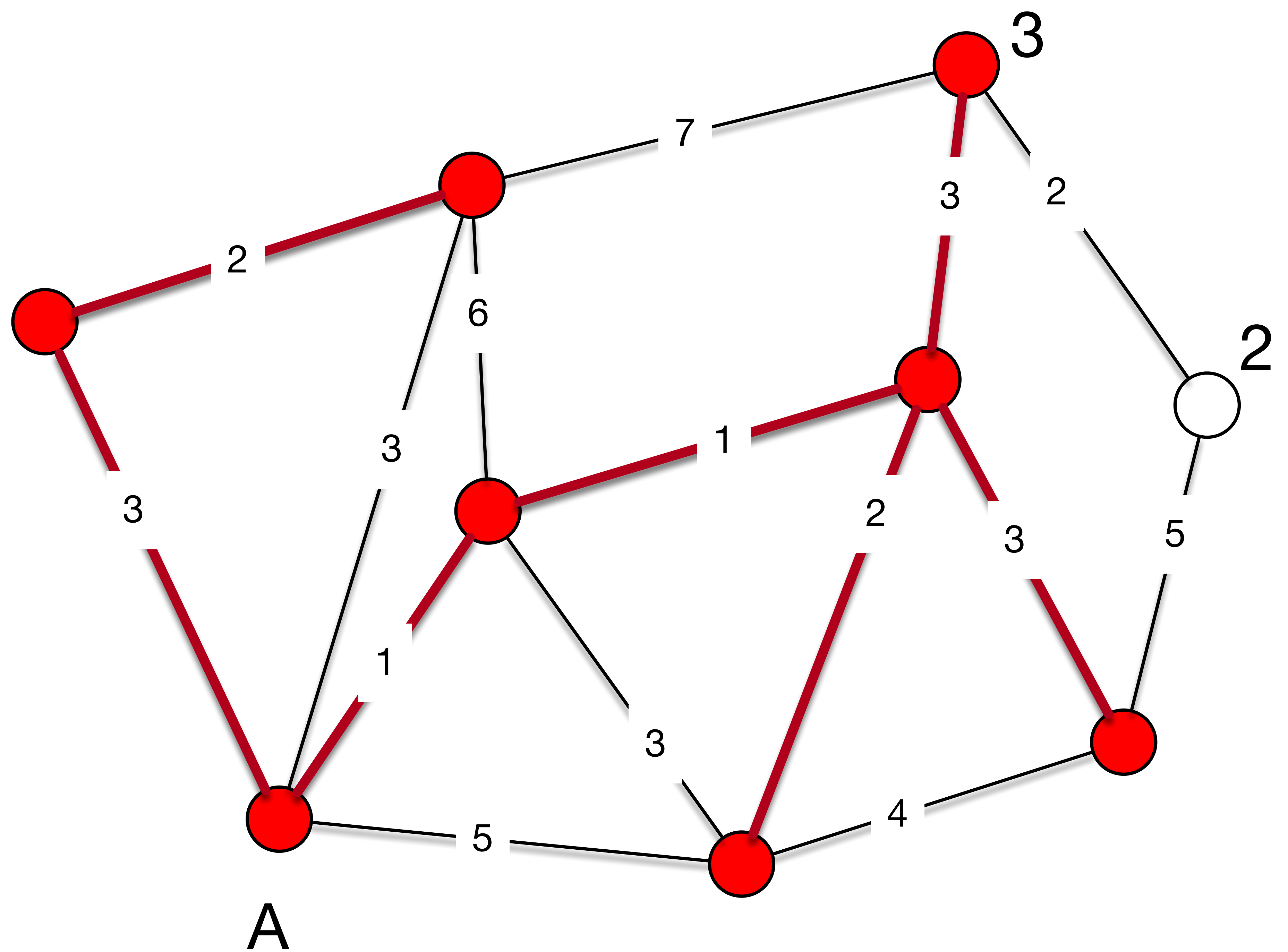
Prim's Algorithm



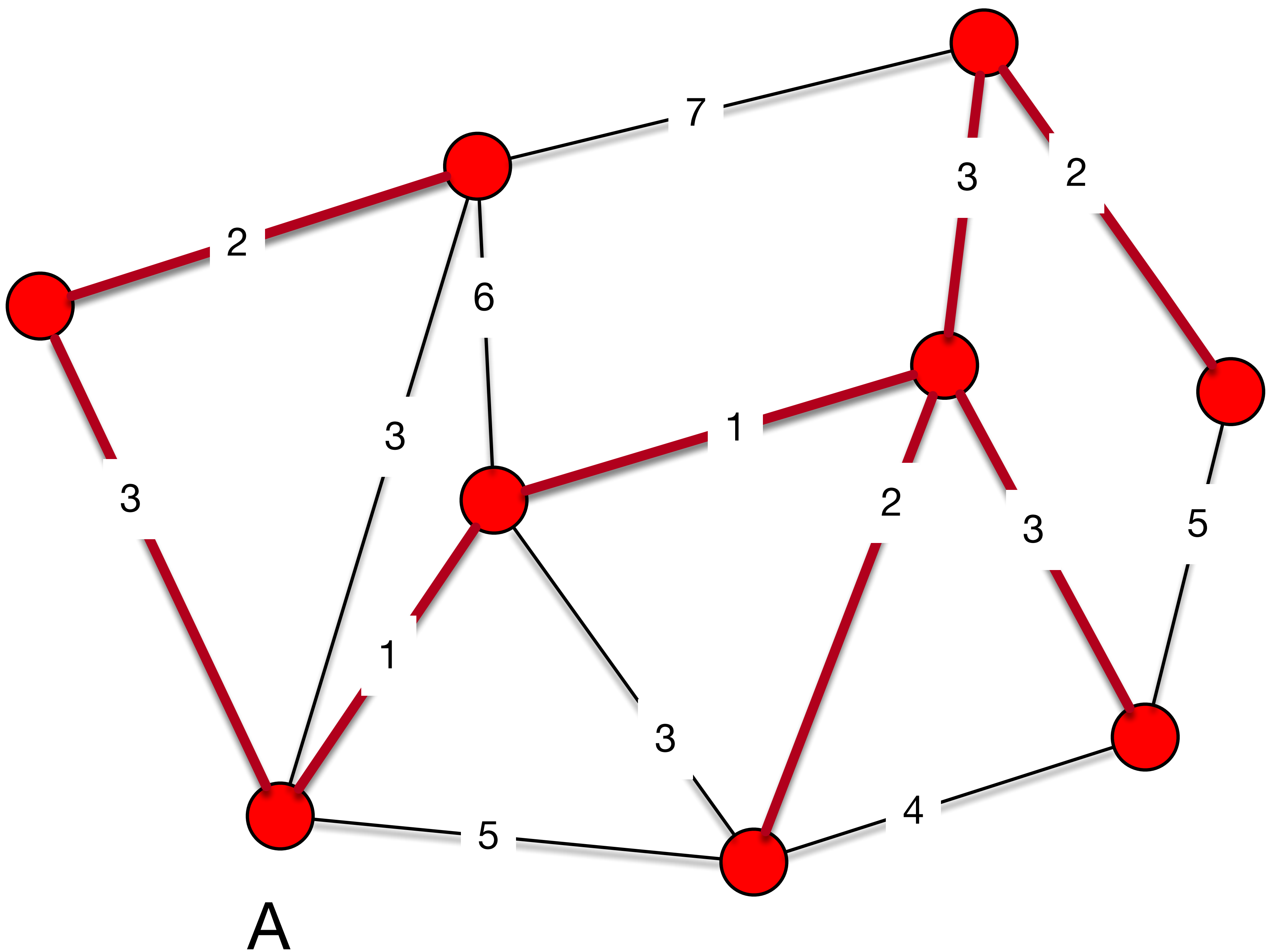
Prim's Algorithm



Prim's Algorithm



Prim's Algorithm



Prim's Algorithm

- Because Prim's algorithm only selects safe edges, it correctly calculates a minimum spanning tree
- The run-time of Prim's algorithm depends on the implementation of the priority heap
  - The best type is a Fibonacci heap
    - In which case the run time is $O(|E| + V\log(|V|))$
  - Or we can use a normal priority heap which gives us
    - $O(E\log(V))$