Distributed Systems

Minimum spanning trees

Rik Sarkar

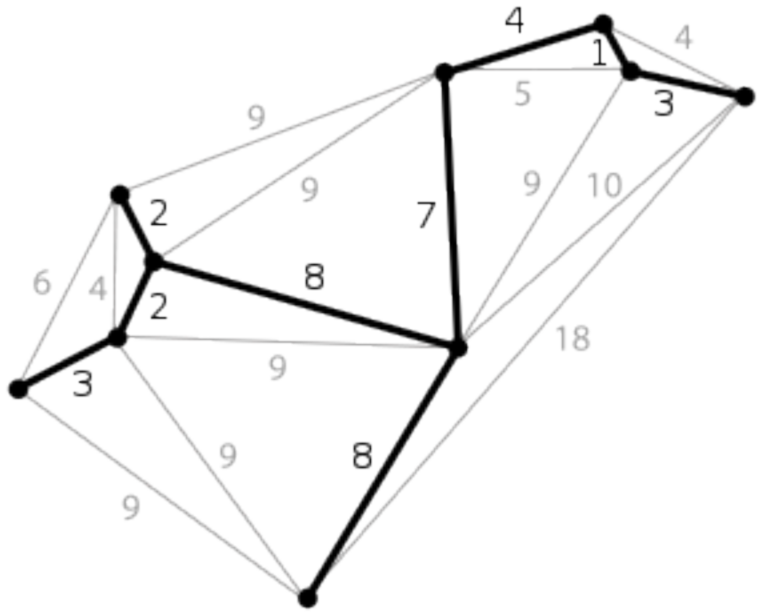
University of Edinburgh

Fall 2018

Minimum spanning trees

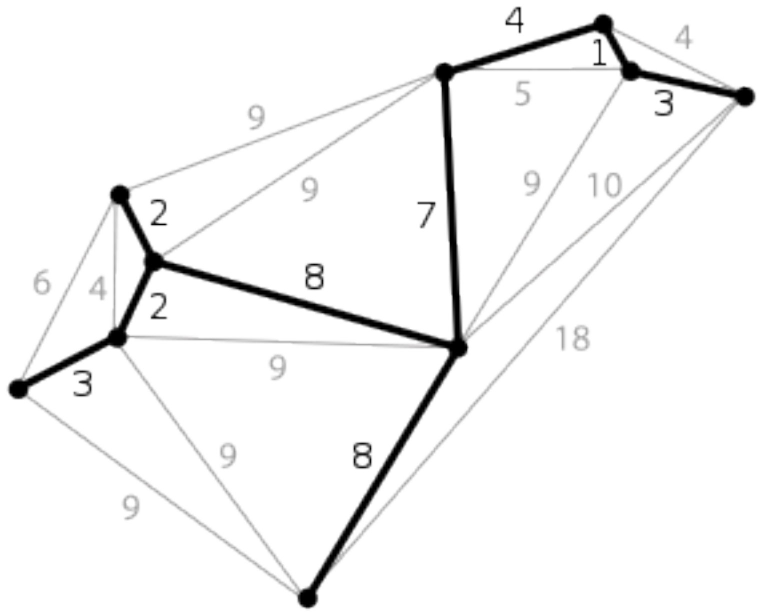
Ref: Wiki

- Definition (in an undirected graph):
  – A spanning tree that has the smallest possible total weight of edges



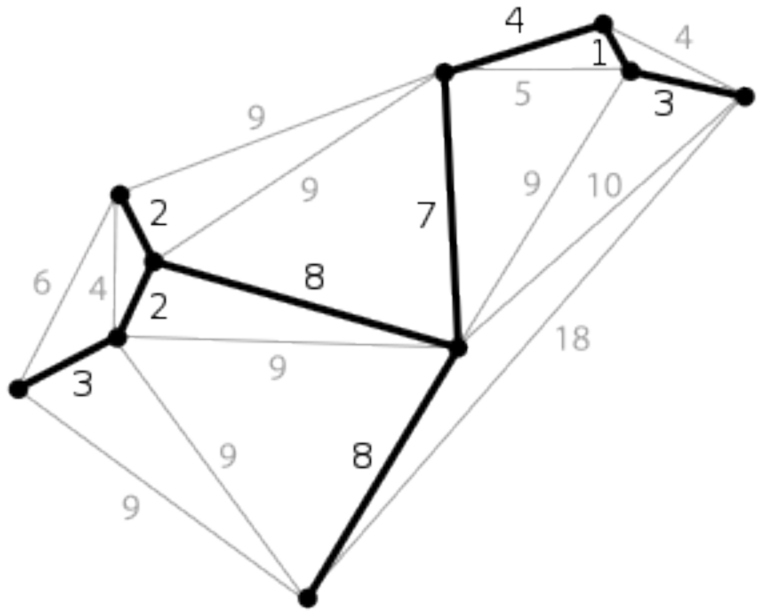
Minimum spanning trees

- Useful in broadcast:
  - Using a flood on the MST has the smallest possible cost on the network



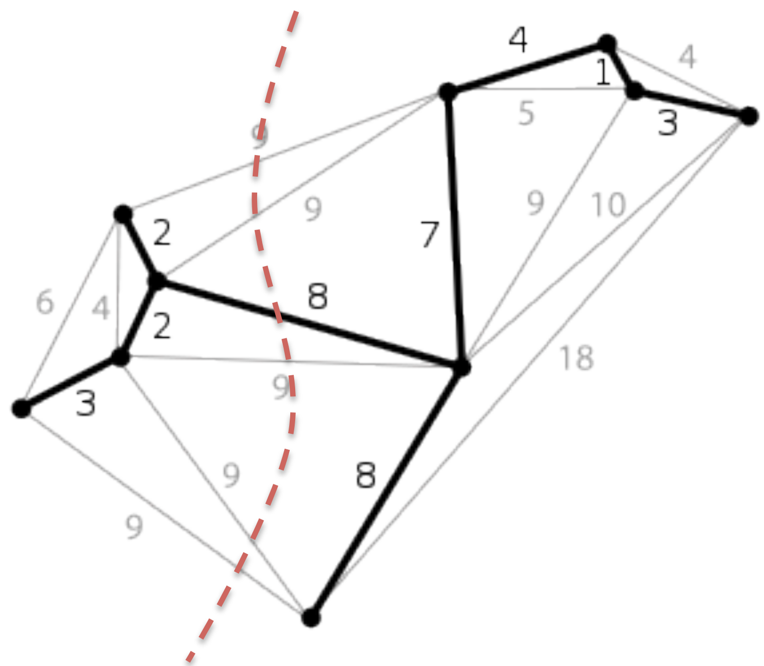
Minimum spanning trees

- Useful in point to point routing:
  - Minimizes the max weight on the path between any two nodes



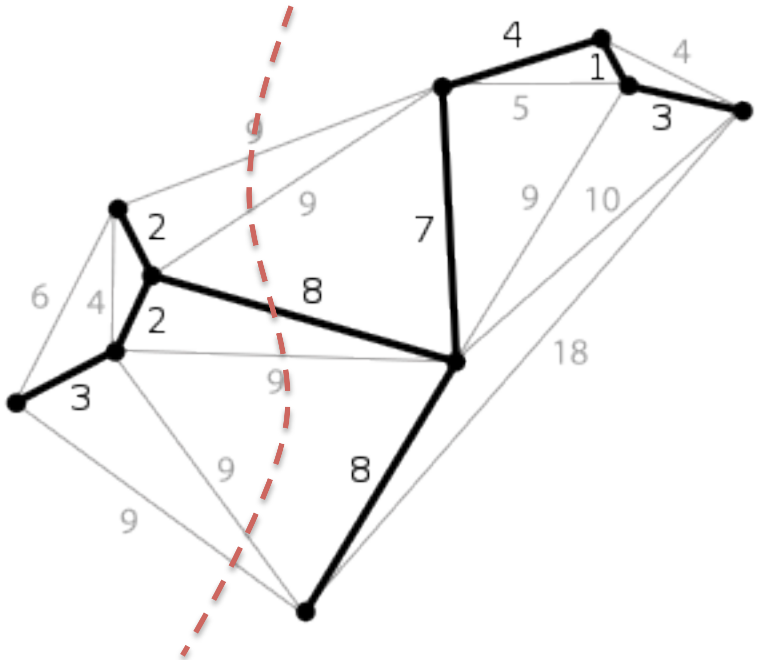
Property: Cut optimality

- Every edge of the MST partitions the graph into two disjoint sets (creates a *cut*)
  - Each set is individually connected by MST edges



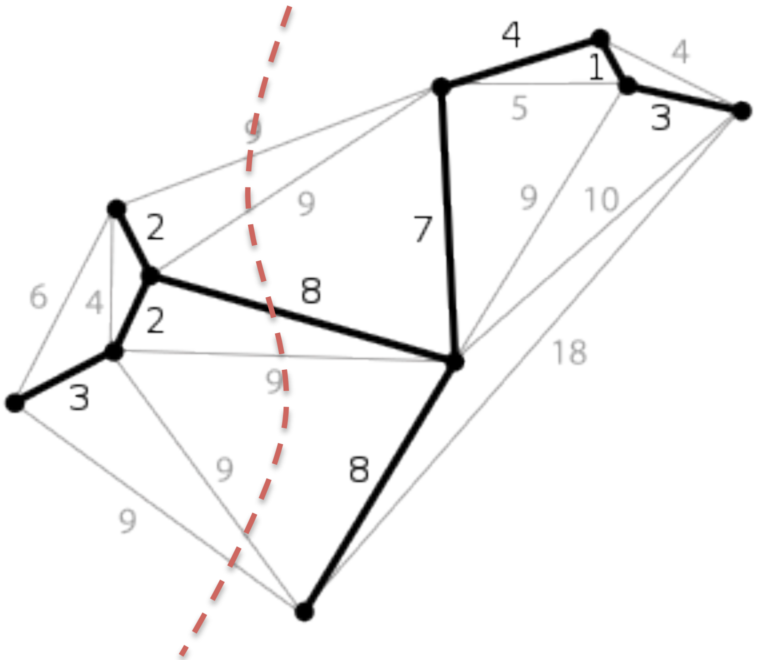
Property: Cut optimality

- Every edge of the MST partitions the graph into two disjoint sets (creates a *cut*)
  - Each set is individually connected by MST edges
- No edge across the cut can have a smaller weight than the MST edge



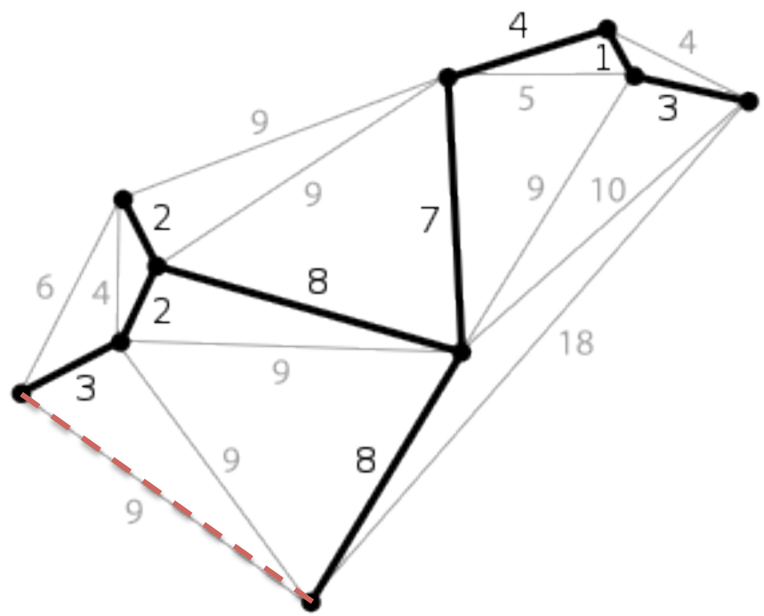
Property: Cut optimality

- Every edge of the MST partitions the graph into two disjoint sets (creates a *cut*)
  - Each set is individually connected by MST edges
- No edge across the cut can have a smaller weight than the MST edge
- Proof: If there was such an edge, then we can swap it for the current edge and get a tree of smaller total weight



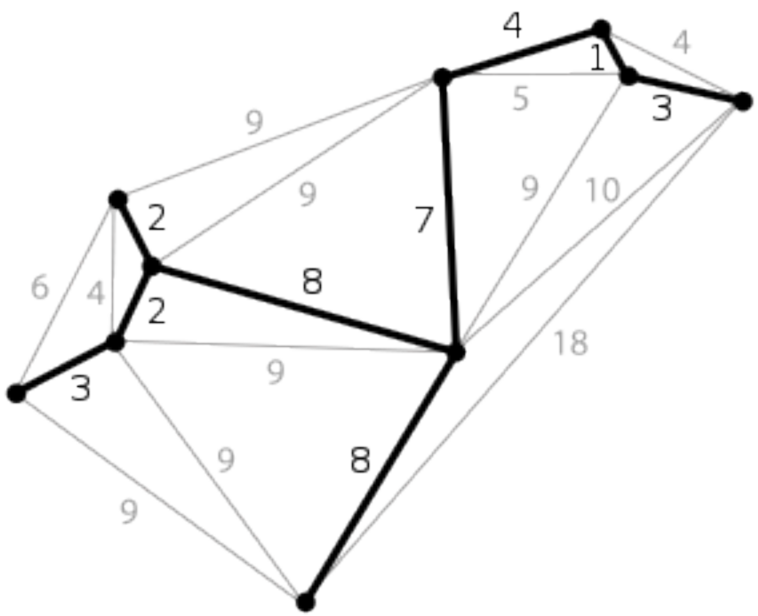
Property: Cycle optimality

- Every non-MST edge when added to MST set creates a cycle
- It must have max weight in the cycle



MST: Not necessarily unique

- Why?



MST: Not necessarily unique

- Assume:
  - All edge weights are unique

Prim's Algorithm

- Initialize $P = \{x\}$; $Q = E$
  - (x is any vertex in V)
- While $P \neq V$
  - Select edge (u,v) in the cut $(P, V \setminus P)$
    - (at the boundary of P)
    - With smallest weight
  - Add v to P

Prim’s Algorithm

- If we search for the min weight edge each time: $O(n^2)$

Prim's Algorithm

- If we use *heaps*:
  - O(m log n) [binary heap]
  - O(m + n log n) [Fibonacci heap]

Prim's Algorithm

- Can we have an efficient distributed implementation?

Prim's Algorithm

- In every round, we need to find the lowest weight boundary edge.
- Use a convergecast (aggregation tree based)
  - In every round
  - For n rounds

Prim's Algorithm

- What is the running time?
- What is communication complexity?

Prim’s Algorithm

- The weakness:
- Does not use the distributed computation
- Tree spreads from one point, rest of network is idle

Kruskal's algorithm

- Works with a *forest:* A collection of trees
- Initially : each node is its own tree
- Sort all edges by weight
- For each tree,
  - Find the least weight boundary edge
  - Add it to the set of edges: merges two trees into one
  - Repeat until only 1 tree left

Kruskal's algorithm

- The problem step:
  - "Find the least weight boundary edge"
- How do you know which is the boundary edge?
- Maintain id for each tree (store this at every node)
- Easy to check if end-point belong to different trees
- When merging trees, update the id of one of the trees
  - Expensive, since all nodes in the tree have to be updated

Kruskal's algorithm

- When merging trees, update the id of one of the trees
  - Expensive, since all nodes in the tree have to be updated
- Solution: always update the id of the smaller tree (the one with fewer nodes)
- The cost for all id updates is O(n log n)

Kruskal's algorithm

- Claim: The cost for all id updates is O(n log n)
- Proof: (by induction on levels)
  - Suppose the final list of n elements was obtained by merging two lists of h elements and n-h elements in the previous level
  - And $h \le n/2$
  - Then cost of creating final list is (for some const p):
    - Cost for creating two lists $\le ph \lg h + p(n-h)\lg (n-h)$
    - Cost for updating labels $\le ph$
    - Total $\le ph \lg h + p(n-h)\lg (n-h) + ph$
    - Total $\le ph (\lg (n/2) + 1) + p(n-h)\lg (n-h)$
    - $\le pn \lg n$
- Note: Kruskal also needs time to sort the edges initially

GHS Distributed MST Algorithm

Ref: NL

- By Gallagher, Humblet and Spira
- Each node knows its own edges and weights

GHS Distributed MST Algorithm

- Works in levels
- In level 0 each node is its own tree
- Each tree has a leader (leader id == tree id)
- At each level k:
  - All Leaders execute a convergecast to find the min weight boundary edge in its tree
  - It then broadcasts this in its tree so that the node that has the edge knows
  - This node informs the node on the other side, which informs its own leader

GHS Distributed MST Algorithm

- Observation 1:
  - We are possibly merging more than two trees at the same time
  - Problem: who is the leader of the new tree?
- Observation 2:
  - The merged tree is a tree of trees: it cannot have a cycle
  - We can assign a direction to each edge and each node (tree) has an outgoing edge
  - There must be a pair of nodes (trees) that select each-other (otherwise the merged tree is infinite)
  - We select the edge used to merge these two trees
    - Select the node with higher ID to be leader
  - The leader then broadcasts a message updating leader id at all nodes.

GHS Distributed MST Algorithm

- Complexity:
- The number of nodes at each level k tree is at least 2^k
- Since starting at size 1, the number of nodes in the smallest tree at least doubles every level
- Therefore, there are at most O(log n) levels

GHS Distributed MST Algorithm

- Complexity:
- At each level, at each tree, we use constant number of broadcasts and convergecasts
- Each level costs O(n) time
- Total costs : O(n log n) time

GHS Distributed MST Algorithm

- Complexity:
- At each level, at each tree, we use constant number of broadcasts and convergecasts
- Each level costs O(n) messages
- Total costs : O(n log n + |E|) messages

Distributed MST Algorithm

- Non-unique edge weights
- If edges have duplicate weights
- We make them unique:
  - By ensuring that for any two edges e and e’
  - Either wt(e) < wt(e’) or wt(e’)<wt(e)
  - By using node ids
  - Eg. If (u,v) and (u’,v’) have same weight, we define
    - If u<u’ then wt(u,v) < wt(u’v’)
    - Else if u==u’, and if v<v’ then wt(u,v) < wt(u’v’)