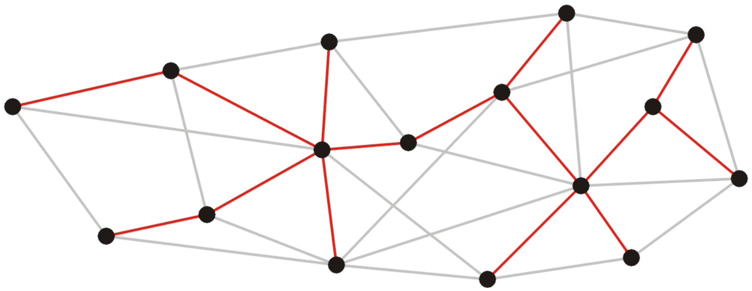
Minimum spanning trees



Carlos Moreno
cmoreno@uwaterloo.ca
EIT-4103

https://ece.uwaterloo.ca/~cmoreno/ece250

These slides, the course material, and course web site are based on work by Douglas W. Harder

Minimum spanning trees

Standard reminder to set phones to silent/vibrate mode, please!

Minimum spanning trees

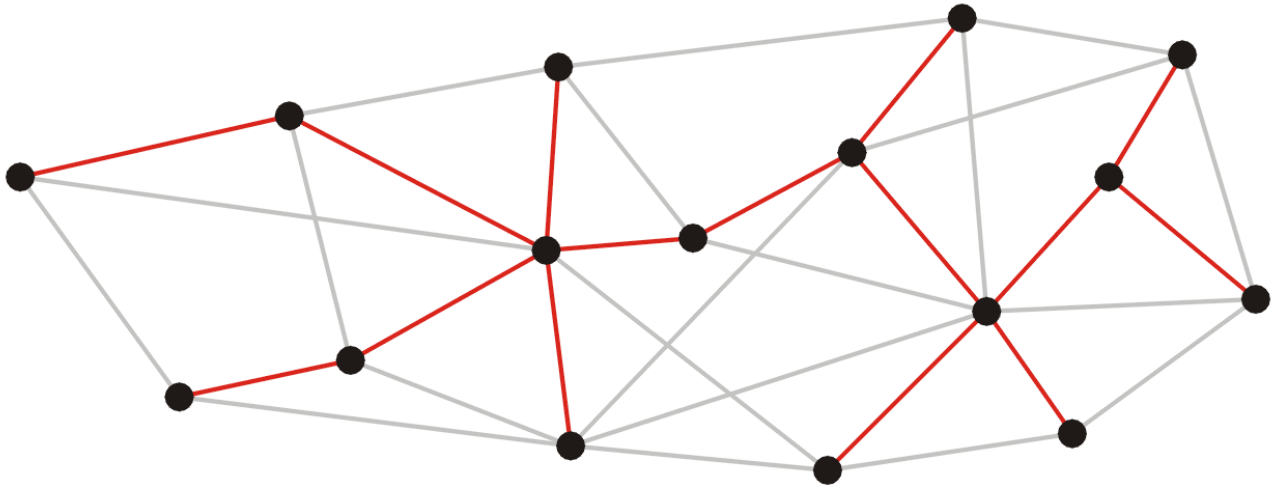
- During today's lesson:
  - Introduce the notion of spanning tree for a connected graph
  - Discuss the notion of minimum spanning trees
  - Look into two algorithms to find a minimum spanning tree:
    - Prim's alforithm
    - Kruskal's algorithm

Minimum spanning trees

- Given a connected graph with n vertices, a spanning tree is a collection of $n-1$ edges that connect all n vertices.
  - $n-1$ is the minimum number of edges required to connect n vertices, resulting in a tree structure.
    - If we take any vertex to be the root, we form a tree by treating adjacent vertices as children.
- We observe that a spanning tree is not necessarily unique.

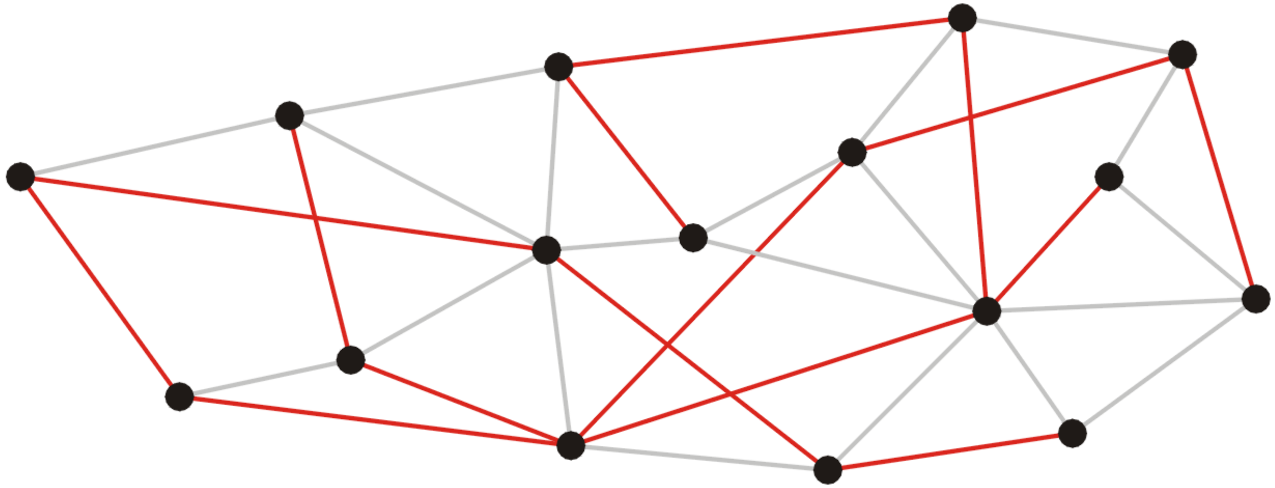
Minimum spanning trees

- This is an example of a spanning tree:



Minimum spanning trees

- For the same graph, this is also a spanning tree:



Minimum spanning trees

- If the graph is weighted, then a spanning tree has a weight, given by the sum of the edges that constitute the spanning tree.

Minimum spanning trees

- If the graph is weighted, then a spanning tree has a weight, given by the sum of the edges that constitute the spanning tree.

- A minimum spanning tree is a spanning tree with minimum weight.
  - A minimum spanning tree is not necessarily unique!
  - That is, there may be several different spanning trees with the same weight — a weight such that no spanning tree has a weight lower than this.

Minimum spanning trees

- We'll look at some examples of applications in class.
- We'll also discuss two algorithms to obtain a minimum spanning tree.

Minimum spanning trees

- Prim's algorithm has certain aspects in common with Dijkstra's algorithm.
  - At each iteration, the spanning tree is expanded by choosing the vertex with smallest distance to the “current” spanning tree.
    - Similar idea, and in fact, as we'll see, the reason why it works (and the argument to prove that this step works) is almost identical to Dijkstra's algorithm.
    - A key difference is that in Dijkstra's algorithm we select the vertex with lowest distance (the “total” distance from the starting vertex) — with Prim's algorithm, we select the lowest distance given by the edge that connects to the current spanning tree.

Minimum spanning trees

- The algorithm is quite simple:
- Initialization:
  - Select a root node and set its distance as 0
  - Set the distance to all other vertices as ∞
  - Set all vertices to being unvisited
  - Set the parent pointer of all vertices to NULL

Minimum spanning trees

- Then, Iterate while there are unvisited vertices with distance < ∞
  - Select the unvisited vertex with minimum distance
  - Mark that vertex as visited
  - For each adjacent vertex, if the weight of the connecting edge is less than the current distance associated to that vertex:
    - Update the distance to equal the weight of the edge
    - Set the current vertex as the parent of the adjacent vertex

Minimum spanning trees

- Kruskal's algorithm takes a different — but also interesting — approach:
- Put the edges in order by weight, and add the lowest weight edge to the spanning tree if it does not create a cycle.

Minimum spanning trees

- Kruskal's algorithm takes a different — but also interesting — approach:
- Put the edges in order by weight, and add the lowest weight edge to the spanning tree if it does not create a cycle.
  - How do we (efficiently!) determine whether adding an edge will create a cycle? (we'll discuss this detail in class)