CS330 – Minimum Cost Spanning Trees

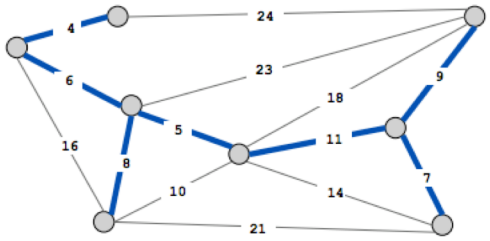
Purpose: There are many applications such as network design where we want the minimum cost set of edges that connect everything in the graph. We call that set of edges the "minimum cost spanning tree". We will examine two algorithms for determining that spanning tree.

Knowledge: This activity will help you become familiar with the following content knowledge:

- the "cut" and "cycle" properties of minimum cost spanning trees.
- Kruskal's algorithm
- Prim's algorithm
- implementation costs for both algorithms

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 60 minutes.

1. Consider the problem of taking a connected graph with weighted edges and finding a tree from those edges that connects all the vertices such that the total weight is minimized.

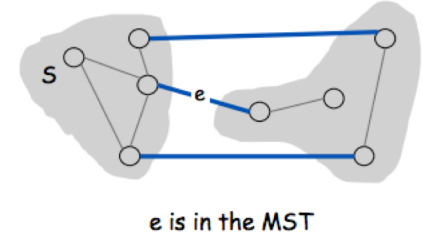


cost(T) = 50

Is this a problem we can solve by brute force? Brute force would require looking at all possible trees. Approximately how many trees could there be with a graph with V vertices?

2. There is a useful property called the "cut property":

   Let S be any subset of vertices and let e be the minimum cost edge with exactly one endpoint in S. Then the minimum cost spanning tree T^* contains e.



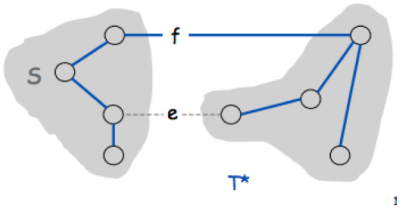
e is in the MST

   We can prove the cut property by using a proof by contradiction. Suppose that e does not belong to T^* and we will examine what will happen. Adding e to T^* creates a unique cycle C in T^*. Some other edge in C, say f, has exactly one endpoint in S. Look at the tree T in which we delete f from T^* and add e instead. This new tree T is also a spanning tree.

   What do we know about the weight of f compared to the weight of e (remember how we chose e)? Why does this give us a contradiction?

3. Another property called the "cycle property":

   Let C be any cycle in the graph and let f be the maximum cost edge belonging to C. Then the minimum cost spanning tree T^* does not contain f.

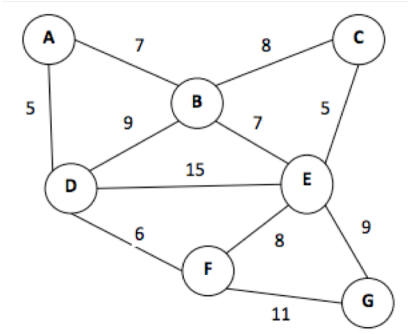


   We can prove the cycle property by using a proof by contradiction. Suppose that f does belong to T^* and we will examine what will happen. Deleting f from T^* disconnects T^*. Let S be one side of the cut. Some other edge in C, say e, has exactly one endpoint in S. Look at the tree in which we delete f and add e. This new tree T is also a spanning tree.

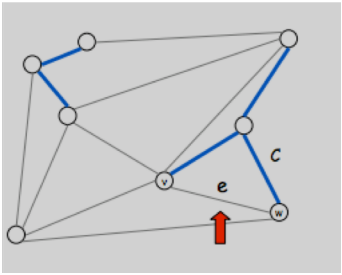
   Why does this give us a contradiction?

4. One algorithm for finding a minimum cost spanning tree is Kruskal's algorithm: Consider edges in ascending order of cost. Add the next edge to T unless doing so would create a cycle.

   What tree would Kruskal give us with this graph?

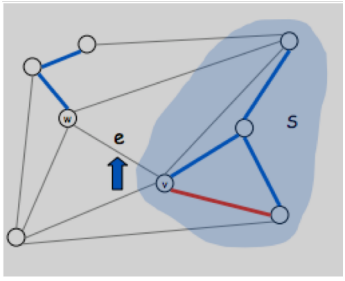


5. To prove that Kruskal will always produce the minimum cost spanning tree, consider the case where adding edge e to T creates a cycle.



   Why must e be the max weight edge in the cycle C?
   Why does the cycle property show that e is not in the minimum cost spanning tree?

   Now consider the case where adding edge $e = (v, w)$ to T does not create a cycle.



   Why must e be the minimum weight edge with exactly one endpoint in S?
   Why does the cut property show that e must be in the minimum cost spanning tree?

6. In the implementation of Kruskal, we can use DFS to check if adding an edge creates a cycle. What would be the cost per cycle check? What would be the overall cost for Kruskal's algorithm with a graph with V vertices and E edges?

7. Perhaps we can do better in the efficiency for Kruskal. Suppose we maintain sets for each connected component of the edges in T.

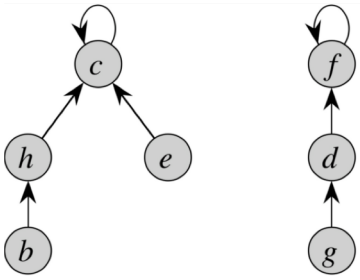
   If we have edge $e = (v, w)$ and we see that v and w are in the same connected component, what does this tell us regarding the two cases we examined in the algorithm? What does it tell us if v and w are in different components?

8. Union-Find algorithms implement the representation of sets in which we can perform the union operation to join sets, and we can perform the find operation to determine the set containing a particular value:

   `MakeSet(x)` creates a new set containing just x
   `Union(x,y)` results in the union of the set containing x with the set containing y
   `FindSet(x)` returns the set containing x

   We could store the set name of element i in array `X[i]`. Give the order of growth for each of the operations.

   Alternatively, we could represent the set with a tree where the children point back to the root and use the root value as the name of the set. When we perform a union, we would make the root of one of the sets, become of child of the root of the other set.

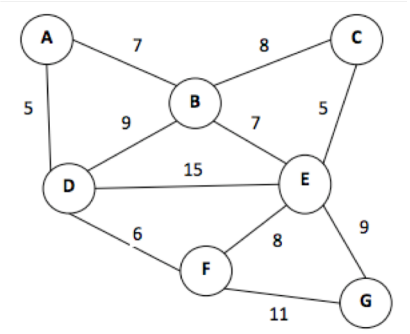


   Give the order of growth for each of the operations with this representation.

   Using this last representation for sets, what would be the order of growth for Kruskal's algorithm?

9. Another algorithm for finding a minimum cost spanning tree is Prim's algorithm: Start with a chosen root vertex and greedily grow tree T. That means that at each step, add the cheapest edge that has exactly one endpoint in T

   What tree would Prim give us with this graph if we started with vertex A?



10. To prove that Prim's algorithm always gives the minimum cost spanning tree, let S be the subset of vertices in the current tree T. Prim adds the cheapest edge e with exactly one endpoint in S.

    Which of the properties shows that e must also be in the minimum cost spanning tree?

11. In Prim, to find the cheapest edge with exactly one endpoint in S we can maintain the edges with (at least) one endpoint in S in a heap (priority queue). Then we would get and delete the minimum edge from the heap and add it to T if it has only one endpoint in the tree. Upon adding the edge to T we would add new edges to the heap that are incident to an endpoint of this new tree edge.

    What is the order of growth for deleting the minimum value from a heap?
    What then is the overall order of growth for Prim's algorithm?