CPSC 320 Tutorial 2

Spanning Trees

Algorithm Spanning($G = (V, E)$, wt())

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
        E-new $= E$-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

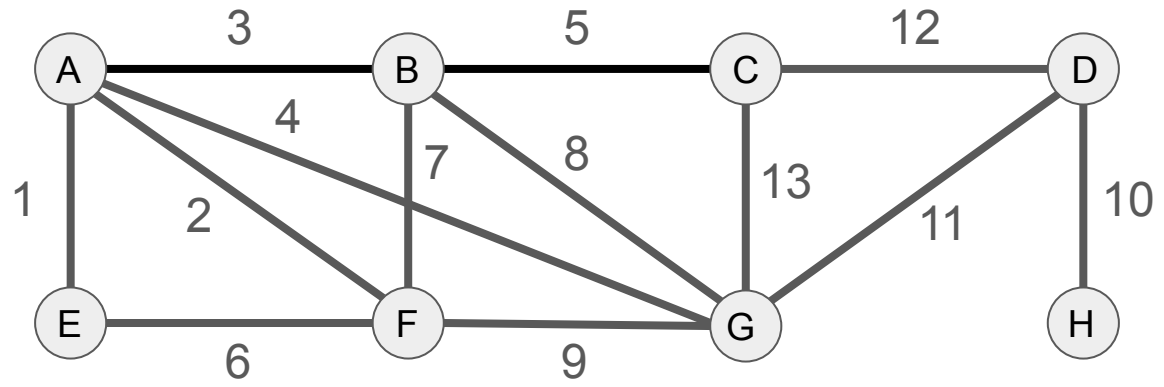
// thus, we can break ties however we like

Loop 1:

Each of A, B, C, D, E, F, G, H are connected components (CCs) are G'.

For each of these, find their minimum weight edge, and proceed.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
  E-new $= \emptyset$
  For each connected component C of $G' = (V, E')$
    Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
    E-new $= E$-new $\cup \{e\}$
  $E' = E' \cup E$-new
Return G'

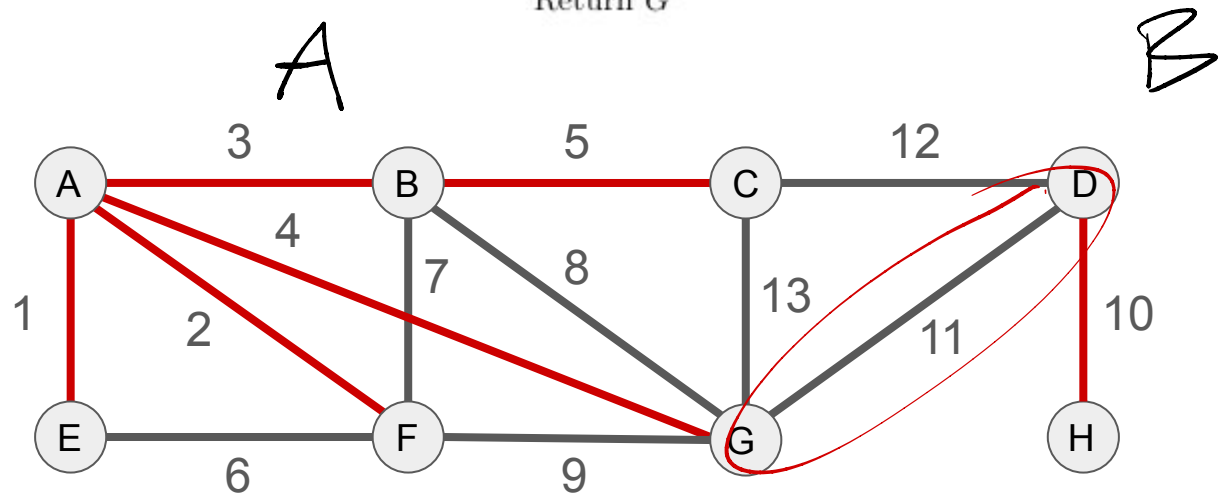


(E' is **red**, E - E' is **black**)

Loop 2:

Now, we only have two connected components. They will both add the same edge, GD.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
  E-new $= \emptyset$
  For each connected component C of $G' = (V, E')$
    Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
    E-new $= E$-new $\cup \{e\}$
  $E' = E' \cup E$-new
Return G'

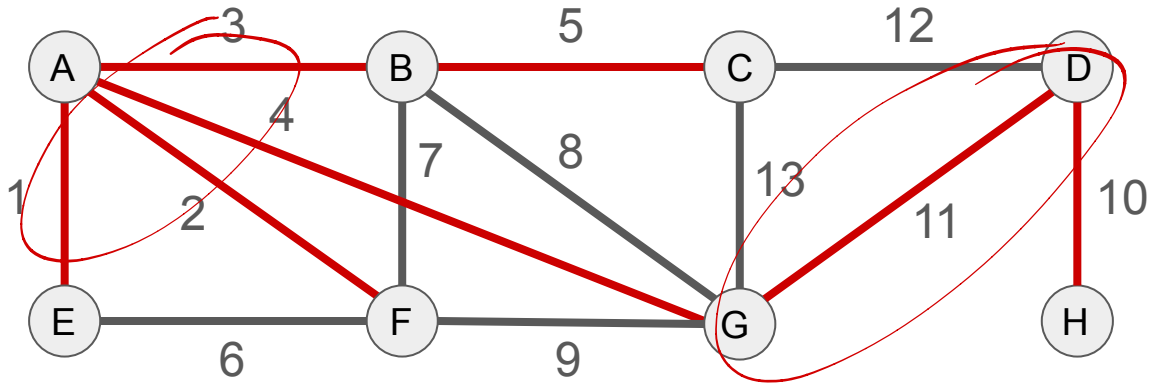


(E' is **red**, E - E' is **black**)

Finished, G' is connected!

Our tree has weight ~~43~~ 36.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
  E-new $= \emptyset$
  For each connected component C of $G' = (V, E')$
    Find an edge $e = (u, v) \in E$ of minimum weight $\text{wt}(e)$ that connects a node u in C to a node v that is *not* in C
    E-new $= E$-new $\cup \{e\}$
  $E' = E' \cup E$-new
Return G'



(E' is **red**, E - E' is **black**)

Question 1, 2:

Find the complexity of the maximum outer iterations executed, and describe the input choices that would lead to this.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
        E-new $= E$-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

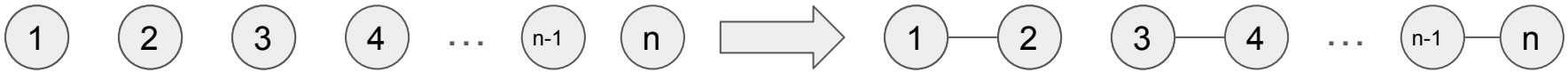
Hint: What is the maximum amount of connected components can we be left with after the first iteration if we started with n connected components?

Question 1, 2:

Find the complexity of the maximum outer iterations executed, and describe the input choices that would lead to this.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
        E-new $= E$-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

If we start with n vertices, we will select n edges in total.
However, each edge may be selected twice (by the two CCs it connects, so we end up with a **maximum of n/2** CCs.



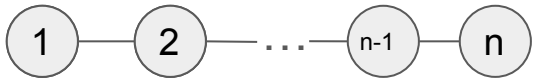
Thus, if this is always possible, the loop can be executed Θ(log n) times in the worst case. Make an example; remember if edge weights are equal, we get to choose.

Question 1, 2:

Find the complexity of the maximum outer iterations executed, and describe the input choices that would lead to this.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight $\text{wt}(e)$ that connects a node u in C to a node v that is *not* in C
        E-new $= E$-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

Solution:



Make the graph a n-path, and set each edge to have the same weight (say 1). We will make the edge choices (inside each iteration) that will connect the first two CCs, and the next two CCs, and so on. This leads to Θ(log n) repetitions in the worst case.

Side note: we could also set the edge weights to be 1, 2, 1, 3, 1, 2, 1, 4, … to model the same pattern.

Question 4, 5 (3 will come later):

Find the complexity of the minimum outer iterations executed, and describe the input choices that would lead to this.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
        E-new $= E$-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

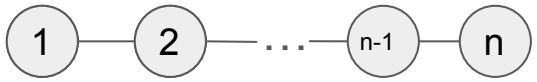
Hint: use the same graph, but make different edge choices.

Question 4, 5 (3 will come later):

Find the complexity of the minimum outer iterations executed, and describe the input choices that would lead to this.

```
Let G' = (V, E') where E' = ∅
While G' is not connected
    E-new = ∅
    For each connected component C of G' = (V, E')
        Find an edge e = (u, v) ∈ E of minimum weight wt(e) that
        connects a node u in C to a node v that is not in C
        E-new = E-new ∪{e}
    E' = E' ∪ E-new
Return G'
```

Solution:



Make the graph a n-path, and set each edge to have the same weight (say 1). Note 1 only has 1 choice for their edge, {1, 2}. For every vertex from 2 to n-1, choose the edge not chosen yet (2 chooses {2, 3}, 3 chooses {3, 4}, and so on). Then n chooses {n - 1, n}, and we have already finished with a tree in 1 step!

Side note: we could also set the edge weights to be n-1, n-2, …, 1 to model the same pattern.

Question 3:

Choose: Spanning(G) is (always, sometimes, never) a tree.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
        E-new $= E$-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

If you're stuck on this type of problem, it's helpful to work through some examples. It will help provide intuition for a proof (if needed).

Question 3:

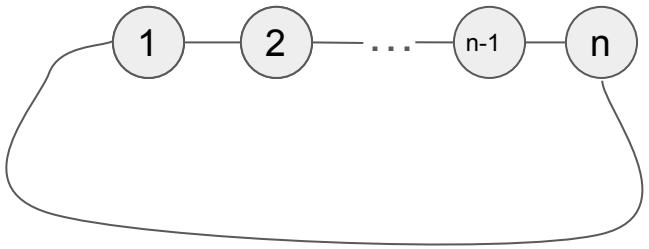
Choose: Spanning(G) is (always, sometimes, never) a tree.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
        E-new $=$ E-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

Solution:

Never is not true, given that we've already gone through an example where a tree is returned.

However, consider a case of the cycle C_n, and all edge weights are 1 (we are adding an extra edge {n, 1}). Use the same edge choices as before, except now vertex n will choose the edge {1, n}. We thus end with a cycle, which is not a tree.



Question 6:

Prove: Spanning(G) is a tree when G has edges of unique weights.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
        E-new $=$ E-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

Hint: a tree is a graph that is connected and acyclic. Which could be false for our final G'? If our final graph is not a tree, when would that condition be introduced?

Question 6:

Prove: Spanning(G) is a tree when G has edges of unique weights.

Let $G' = (V, E')$ where $E' = \emptyset$
While G' is not connected
    E-new $= \emptyset$
    For each connected component C of $G' = (V, E')$
        Find an edge $e = (u, v) \in E$ of minimum weight wt(e) that connects a node u in C to a node v that is *not* in C
        E-new $= E$-new $\cup \{e\}$
    $E' = E' \cup E$-new
Return G'

Solution:

Our final graph must be connected, as otherwise we would not finish the outer while loop. Thus it must contain a cycle.

This cycle must be introduced in one of the loops, and so assume each CC is a tree, and after adding E-new, we create a cycle. Note a ‘cycle’ of the CCs introduces a cycle in the graph and vice versa (as we can take a unique path through any two points in each CC, as it is a tree). Try to use our unique weight condition to try and find a contradiction by assuming a CC cycle exists.

Question 6:

Prove: Spanning(G) is a tree when G has edges of unique weights.

```
Let G' = (V, E') where E' = ∅
While G' is not connected
    E-new = ∅
    For each connected component C of G' = (V, E')
        Find an edge e = (u, v) ∈ E of minimum weight wt(e) that
        connects a node u in C to a node v that is not in C
        E-new = E-new ∪{e}
    E' = E' ∪ E-new
Return G'
```

Solution (continued):

Thus assume we now have a n-cycle of CCs by including n edges. Thus, each CC must include a unique edge, otherwise we don't have enough edges. However, we must have an edge of strictly minimal weight among our n edges. The two CCs it connects must both choose to include it, because it has strictly minimal weight, leading to a contradiction.

Question 7:

Prove: Spanning(G) is a minimum spanning tree when G has edges of unique weights.

```
Let G' = (V, E') where E' = ∅
While G' is not connected
    E-new = ∅
    For each connected component C of G' = (V, E')
        Find an edge e = (u, v) ∈ E of minimum weight wt(e) that
        connects a node u in C to a node v that is not in C
        E-new = E-new ∪{e}
    E' = E' ∪ E-new
Return G'
```

Our intended solution uses the Cut Property of MSTs from KT, 4.17. It states:

Assume that all edge costs are distinct.
Let S be a nonempty strict subset of V, and let edge e = {v, w} be the minimum-cost edge with one end in S and the other in V − S.
Then every minimum spanning tree contains the edge e.

Question 7:

Prove: Spanning(G) is a minimum spanning tree when G has edges of unique weights.

```
Let G' = (V, E') where E' = ∅
While G' is not connected
    E-new = ∅
    For each connected component C of G' = (V, E')
        Find an edge e = (u, v) ∈ E of minimum weight wt(e) that
        connects a node u in C to a node v that is not in C
        E-new = E-new ∪{e}
    E' = E' ∪ E-new
Return G'
```

Solution:

Apply the cut property to S = C in each iteration.

The distinct weight condition is met, our C is nonempty and is a strict subset of V (as otherwise we would have terminated the outer loop).

Thus, the minimum weight edge connecting a node in C to a node not in C, which is the edge we add, must be in every valid MST. This applies to every one of the edges we add, and as we finish with a tree, every MST must contain our tree. Adding any edges to a tree creates a cycle, and so our MST is unique.