# ComS 311
# Recitation 3, 2:00 Monday
# Homework 5

Sean Gordon

November 3, 2019

---

**Algorithm 1** Find MST of G'.

---

Add e with its updated cost to the MST $T$
#There will now be a single cycle in the MST
Run BFS recursively beginning at a vertex v in edge e
    While running BFS, make a list *lst* of the edges that have been passed so far
    When vertex v is found again we know we have found the cycle, so we can pass lst back up to the top to be used later
Find the maximum cost edge in lst and remove it from the MST

---

**Runtime of algorithm:**
Adding e to MST: O(1)
Running BFS: O(m+n)
Runtime = O(n+m)
**Proof:**
A valid MST contains no cycles.
If we add the edge $e$ with modified weight to the MST T, there will be a single cycle present.
In order to re-form the MST, we must remove the heaviest edge in this cycle.
We use BFS to build a list of the edges that make up the cycle.
Let A be the heaviest edge in that cycle.
Removing A from T will remove the cycle and re-form the MST.

**Algorithm 2** Find MST of G, with all S as leaf nodes.

```
Create empty array B
for all vertex in G do
    if vertex is not in S then
        Add vertex and cost to arr
    end if
end for
Run Prim's algorithm on B to find MST
for all vertex in S do
    vertex smallestEdge = -∞
    for all edge in vertex do
        if edge does not go to a vertex in S then
            if edge.cost < smallestEdge.cost then
                smallestEdge = edge
            end if
        end if
        if smallestEdge == -∞ then
            return false      #This MST is impossible
        end if
        Add vertex to the MST with smallestEdge
    end for
end for
```

**Runtime of algorithm:**

Loop through G: O(n)

  If inside loop O(1)

Prim's Algorithm: O(mlog(n))

Nested loop for each edge in S: O(m)

  If inside loops O(1)

Runtime = $O(n + m\log(n) + m) \Rightarrow O(m\log(n))$

**Algorithm 3** Does d[v] correctly contain the length of all shortest paths

```
Create new array A of size n
for all vertex in G do
    for all edge in vertex do
        #Turn outgoing edges to incoming edges
        #If edge in u from u → v, make edge in v from v → u
        Add the incoming edge to A
    end for
end for
for all vertex in A do
    if vertex == S then
        continue
    end if
    for all edge in vertex do
        #Assume values in d are correct, and build off of them
        int cost = edge.cost + d[edge.v]
        if cost == d[vertex] then
            goto OK
        end if
    end for
    #If the cost is incorrect, d is incorrect
    return false
    OK:
end for
return true
```

**Runtime of algorithm:**
Nested loop for each edge in G: O(m)
Nested loop for each edge in A: O(m)
Runtime = O(m+m) = O(2m) = O(m)

**Algorithm 4** Schedule cake baking and decorating in minimum time

Assume cake information is given in $C$

Use mergesort to sort C in order of longest decoration time to shortest decoration time

When two decoration times are equal, use longest baking times to break the tie (longest goes first)

**Proof:**

Assume our ordering C is not optimal
Assume that there is an optimal ordering O
This means there must be an inversion.
Then, we can modify O to find O' that is:
- No worse than O
- Closer to A in some measurable way

This process will continue until we reach A
O $\Rightarrow$ O' $\Rightarrow$ O" $\Rightarrow$ A
This gradually transforms O into A without hurting the solution
Therefore A must be optimal