Mahesh Yanasi 11/27/16

U84571744

Homework 7

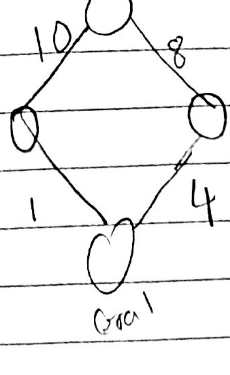
G(V, E)

1\. a) If we have an undirected graph, w/ connected components, we can use a BFS traversal to find the number of vertices and edges. For example, if we use 3 colors for the nodes : white (unvisited), gray (working on), and black (done with), eventually all the nodes will become black and thus we will have the total number of vertices (all the black nodes). And anytime we look at a neighbor and check if it is white, we cross an edge and by doing so, we can count the total number of edges.

b) We can use a BFS algorithm with a runtime of $O(V+E)$. In BFS, all the neighboring nodes are visited using the maximal independent set vector w/ the start node. When a node is visited we check to see if the current node is not a vertex to the MIS. We repeat this for every vertex as the start node so that we can compare it w/ the MIS vector to see which is the largest. It takes $O(E)$ time to access a node's neighbor and the runtime of checking the MIS is $O(V \cdot E)$. This process is repeated w/ every node so the total runtime is $O(V(V+V \cdot E))$

c) This will not always find the shortest path because the shortest path may be within a certain branch that weighs more than the adjacent one.

For examples:



Going left is the shortest but initially going right is shorter.

2. Graph $\rightarrow G(V, E)$ Edge weights $1 \ldots W$ for constant W

a) If we implement Prim's Algorithm using a min heap, the first two steps: key[v] $\leftarrow \infty$ for all vertices in the graph and $Q \leftarrow$ all $v \in V$, take $\Theta(V)$ time. The body of the while loop executes W times and each extract_min operation takes $\Theta(\log V)$ so in total $\Theta(V \log V)$. The for-loop: for each v adjacent to u takes time $\Theta(E)$. We can however make the line key[v] = $w(u, v)$ constant time by keeping a bit for each vertex that tells if it is in Q and updating the bit when the vertex is removed from Queue. If the graph is connected, in the end Prim's Algorithm can run in $\Theta(E \log V)$ time.

b) Assuming a bunch of linked lists w/ a weighted heuristic, Kruskal's algorithm can run G w/ a runtime of $\Theta(V+E(\log V))$ $\rightarrow$ $E \Rightarrow$ # of edges, $V \Rightarrow$ # of vertices.
Kruskal uses a disjoint-set Data Structure. The algorithms makes disjoint sets for each vertex and unions each of the sets based on a sorted list of all the edge weights $\Rightarrow$ runtime $\Theta(E \log E)$. The sets being unioned depends on their weight, if it is a minimum, and when the edges don't form a cycle. This takes $\Theta(V \cdot E)$ since unioning takes $\Theta(V)$ but we go through the whole linked list $\Theta(E)$. With the weighted heuristic, adding the smaller set to the larger set for linked lists takes $\Theta(V \log V)$ time. In the end we get $\Theta((V+E) \log V)$.

3. a) MAYBE-MST-A (G)

```
T = empty
for each edge e, taken in arbitrary order
    if T ∪ {e} has no cycles
        T = T ∪ {e}
return T
```

This is not a Minimum Spanning Tree because the order of unionization matters. The algorithm doesn't take the weight of the edges into account. If the edges and vertices form a cycle with the edge whose weight is greater, the other edge is not added.

b) MAYBE-MST-B (G)

```
sort the edges into non increasing order of edge weights w
T = E
for each edge e, in non-decreasing order by weight
    if T - {e} is a connected graph
        T = T - {e}
return T
```

This is a MST. Since we check if T - {e} is a connected graph, it makes sure we break cycles when we meet and remove an edge w/ a greater weight