Name: Suhail Basalama

Homework 9

23.1-1

We can prove that by contradiction.

Let's suppose (u, v) does not belong to any minimum spanning tree of G.

1. For some minimum spanning tree T there is a path from v to u because it is connected and undirected graph.
2. For any edge (x, y) (that edge must have larger weight than (u, v) as given in the problem) on the path from v to u take the edge (x, y) out of the tree and add the edge (u, v).
3. There is still a path from x to y in the new tree T' through the edge (u, v).
4. Since the weight of (u, v) is less than or equal to the weight of all edges including the edge (x, y), the new tree T' has weight less than or equal to T.
5. So, T' is a minimum spanning tree, which is a contradiction.

Therefore, (u, v) is a member of some minimum spanning tree.

23.1-5

We can prove that by contradiction.

Let T be a minimum spanning tree of G'. Then T is also a spanning tree of G since G' and G have the same vertices.

Let's suppose T is a minimum spanning tree of G'but not of G.

1. The only difference between G and G' is the edge e.
2. So, if T is not a minimum spanning tree of G then there must be a tree T' in G that is a minimum spanning tree with weight less than T and containing the edge e.
3. But e is a maximum edge on a cycle. So, remove the edge e from T' and add an edge (x, y) from the cycle that is not already in T' to make T''.
4. T'' must be a tree with weight less than T' since the edge (x, y) has weight less than e (it is given e is maximum).
5. But then T' is not a minimum spanning tree which is a contradiction.

Therefore, T is a minimum spanning tree of G and G'.

23.2-2

At each step of the algorithm we will add an edge from a vertex in the tree created so far to a vertex not in the tree, such that this edge has minimum weight. Thus, it will be useful to know, for each vertex not in the tree, the edge from that vertex to some vertex in the tree of minimal weight. We will store this information in an array A, where $A[u] = (v, w)$ if w is the weight of (u, v) and is minimal among the weights of edges from u to some vertex v in the tree built so far. We'll use $A[u].v$ to access v and $A[u].w$ to access w.

```
MST_PRIM_ADJ(G, w, r)
    initialize A with every entry = (NIL, ∞)
    T = {r}
    for i = 1 to |V|
        if Adj[r, i] != 0
            A[i] = (r, w(r, i))
    for each u in V - T
        k = min(A[i].w)
        T = T ∪ {k}
        k.π = A[k].v
        for i = 1 to V
            if Adj[k, i] != 0 and w[k, i] < A[i].w
                A[i] = (k, w[k, i])
```

The first for loop runs |V| times. The second outer for loop runs |V| times, and the internal for loop runs |V| times as well, so the total running time of the algorithm is $O(|V| + |V^2|) = O(V^2)$

Another algorithm that runs in $O(V^2)$ is shown below.

If Graph $G = (V, E)$ is represented as an adjacency matrix, for any vertex u, to find its adjacent vertices, instead of searching the adjacency list, we search the row of u in the adjacency matrix. The Prim's algorithms is modified as follows:

```
MST_PRIM_ADJ(G,w,r)
    for each u in G.V
        u.key = ∞
        u.π = NIL

    r.key = 0
    Q = G.V
    while Q != ∅
        u = EXTRACT_MIN(Q)
        for each v in G.Adj[u,:]
            if G.Adj[u,v] != 0 and v in Q and w(u,v) < v.key
                v.π = u
                v.key = w(u,v)
```

The outer while loop runs $|V|$ times and the inner for loop runs $|V|$ times. Hence the algorithm runs in $O(V^2)$.

24.1-3

If the greatest number of edges on any shortest path from the source is m, then the path-relaxation property tells us that after m iterations of BELLMAN-FORD, every vertex v has achieved its shortest-path weight in $v.d$. By the upper-bound property, after m iterations, no d values will ever change. Therefore, no d values will change in the $(m+1)$st iteration. Because we do not know m in advance, we cannot make the algorithm iterate exactly m times and then terminate. But if we just make the algorithm stop when nothing changes any more, it will stop after $m+1$ iterations. We can make that change to the BELLMAN-FORD and the RELAX functions as follows:

```
changes = true  //global variable

ENHANCED_BELLMAN_FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    changes = true
    while changes == true
        changes = false
        for each edge(u, v) ∈ G.E
            RELAX_M(u, v, w)

RELAX_M(u, v, w)
    if v.d > u.d + w(u, v)
        v.d = u.d + w(u, v)
        v.π = u
        changes = true
```