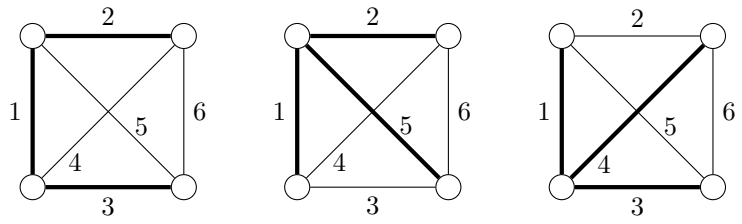
CS 521: Data Structures and Algorithms I
Extra Credit 2

Dustin Ingram

November 30, 2011

1. **(23-1) Solution:**

   (a) Since every edge in the undirected, connected graph $G(V, E)$ is unique, this means that when creating the MST, there must be a unique light edge across every cut of the graph, regardless of how the cut is made, which results in a unique MST. The second-best spanning tree, however, will not necessarily be unique, as can be shown by the following example:



(a) (b) (c)

   Here, (a) is the minimum spanning tree of total weight $W = 6$, while (b) and (c) are the second-best MSTs, each with weight $W = 8$. We see that with this configuration, the second-best MST is not unique.

   (b) If T is the MST of G such that $(u, v) \in T$ and $(x, y) \notin T$, let the second-best MST of G be T_2 such that $T_2 = \{T - (u, v)\} \cup \{(x, y)\}$. If we consider the graph of $T \cup T_2$, there will be one cycle in the graph, and the edges (u, v) and (x, y) will be on the cycle. By definition, $w(u, v) < w(x, y)$ must be true, otherwise we could replace (u, v) with (x, y) in T to produce an MST with a lower weight. Therefore we see that T is the lowest-weight MST, and that T and T_2 differ by only one edge.

   (c) For every pair of vertices (u, v) in the graph, the path $u \rightsquigarrow v$ is a unique simple path. This means that we can simply do a DFS (or,

BFS) search from every vertex $u \in G.V$, storing the maximum traversed edge for each pair of vertices (u, v). Since we are determining the maximum for all pairs (u, v) in the graph, we discover V^2 total edges for a runtime of $O(V^2)$.

(d) As previously established, finding the second-best MST T^2 requires replacing one edge (x, y) in the MST T with one edge $(u, v) \notin T$. This means that for every path $\{u \rightsquigarrow v\} \cup (u, v)$, we produce a cycle, and must remove the maximum edge that is not (u, v). Since we can produce an MST in at least $O(V^2)$ time, and use the algorithm described in (c) to compute the maximum-edge for every path in $O(V^2)$ time, the remaining problem becomes finding a path $u \rightsquigarrow v$ in T and an edge $(u, v) \notin T$ such that

$$w(u, v) - w(\textsc{Max}(u \rightsquigarrow v))$$

is minimized; once (u, v) is found, the second best MST is

$$T^2 = \{T - \textsc{Max}(u \rightsquigarrow v)\} \cup (u, v)$$

The minimization simply requires examining every edge $\notin T$, therefore the overall runtime is guaranteed to be $O(V^2)$.

2. **(15-1) Solution:** Given a DAG $G(V, E)$ with edge weights $w \in \mathbb{R}$ and to vertices s and $t \in G.V$, we must find the longest weighted simple path $s \rightsquigarrow t$.

LONGEST-DAG-PATH(G, w, s, t, u)

```
if u == s then
  s.π = NIL
  for each vertex v ∈ G.V do
    v.w = 0
else if u == t then
  r = [u]
  while u ≠ NIL do
    r.PUSH(u.π)
    u = u.π
  return r
for each vertex v ∈ G.Adj[u] do
  if v.w < u.w + w(u, v) then
    v.w = u.w + w(u, v)
    v.π = u
  LONGEST-DAG-PATH(G, w, s, t, v)
```

Since the graph is a DAG, we can consider each vertex topologically, therefore the subgraph is simply one vertex and it's adjacencies. To start from vertex s, we make an initial call LONGEST-DAG-PATH(G, w, s, t, s), which

sets the parent of s to NIL and sets the weight to each node in the graph to an initial 0. We then consider every vertex $v \in G.Adj[s]$, storing the weight to that vertex from it's parent. At this point in the operation, we must consider if there has already been a path to v of heavier total weight–if there hasn't, we set $s.\pi$ and $s.w$, if there has, we leave them as is. Then we make a recursive call to LONGEST-DAG-PATH for each vertex v.

When we reach t, the algorithm is nearly finished (since the graph is a DAG we do not need to consider vertices adjacent to t). At this point, the algorithm forms a stack and traverses through the parent hierarchy until we return to s, eventually returning the list of the longest path from $s \rightsquigarrow t$.

Since this algorithm will traverse at most V vertices in the worst-case scenario (both to find t and to trace back to s) the running time is $O(V)$.