CS3933 Fall 2007 – Group Assignment 3
Due Wednesday October 17, 2007, at 5pm.

Note that although desired running times are not specified, asymptotically faster algorithms are better and are thus worth more marks.

1. (a) Design and write an algorithm that, given a simple undirected graph $G = (V, E)$ and vertices s and t, will find the first vertex (after s) on the shortest path from s to t (where "shortest" means "smallest number of edges").

```
mark all vertices as unvisited
let Q be an empty queue
mark s as visited
enqueue s on Q
start[s] ← s
start[t] ← −1 (default; used if t not found)
while Q is not empty
    u ← dequeue(Q)
    for all v such that (u, v) ∈ E
        if v is unvisited
            if u = s
                start[v] ← v
            else
                start[v] ← start[u]
            enqueue v on Q
            mark v as visited
return start[t]
```

   (b) Asymptotically analyse the running time of your algorithm from (a).

   Marking the vertices as unvisited will take $c \cdot n$ time for some constant c. Each vertex is visited (and thus enqueued and dequeued) at most once. For each visited vertex u, the for loop will iterate $deg(u)$ times, so the total number of iterations of the for loop is at most $\sum_{u \in V} deg(u) = 2m$. In the worst case, all the edges are part of the same connected component as s, so the for loop will iterate $2m$ times. The rest of the algorithm contributes a constant factor, so the algorithm runs in time $\in \Theta(m + n)$.

2. (a) Design and write an algorithm that, given a simple undirected graph $G = (V, E)$ and vertices s and t, determines if there is more than one simple path in G that

goes from s to t. Note that for two paths to be different, there only needs to be one edge that is in one path that is not in the other path.

Idea: we mark vertices as either (potentially) above t, or below t. A backedge to a vertex that is above t will produce a second simple path from s to t. Information about whether t is found and whether a backedge is found (on the path from s to t) needs to be both fed forward and fed back.

```
calling algorithm:
initialize all vertices to no type
initialize all edges to unused
if DFalt(G, s, t, false, false) = 3
  return true

DFalt(G, u, t, found, back): returns integer (0, 1, 2, or 3)
ret ← 0
if u = t
  found ← true
  ret ← 2
if found
  mark u as type b
else
  mark u as type a
if back
  ret ← ret + 1
for all v such that (u, v) ∈ E
  if v has no type
    mark (u, v) as used
    ret = max(ret, DFalt(G, v, t, found, back))
  else
    if (u, v) is unused
      mark (u, v) as used
      if ret = 2 or (found and v is type a)
        ret ← 3
      if v is type a
        back ← true
return ret
```

(b) Asymptotically analyse the running time of your algorithm from (a).

Initializing the vertices takes $\Theta(n)$ time.
Since each vertex is marked with a type when it is visited, and we only recurse when the vertex has no type, then there is at most one recursive call per vertex. The for loop for a vertex u iterates $deg(u)$ times; over all recursive calls, it will iterate at most a total of $\sum_{u \in V} deg(u) = 2m$ times. The rest of the algorithm contributes a constant factor. In the worst case, the component containing s includes all edges, so every edge will be examined, giving a lower bound of $c \cdot m$ for the search; thus we have a running time $\in \Theta(m + n)$.