Illinois Institute of Technology
Department of Computer Science

Homework 6 Solutions

CS 430 Introduction to Algorithms
Fall Semester, 2010

1. **Problem 22.2-4 on page 602**.

   Solution: If the input graph is represented by an adjacency matrix, then line 12 of the BFS algorithm (page 595) will check each vertex v to see if it is connected to u. This takes $\Theta(V)$ time and is repeated for each vertex u. Hence the runtime is now $\Theta(V^2)$.

2. **Problem 22.2-9 on page 602**.

   Solution: To compute a path in G that traverses each edge once in each direction, we first apply BFS to compute a BFS tree G_π of the graph (see pages 600-601). Then we call OUTPUT-PATH(s) which performs a walk of this tree starting at the root s, and outputs a path that traverses each edge in G exactly once in each direction. Even though the algorithm traverses a BFS tree of the graph, it outputs a path that includes edges that are not part of the tree. It does this in Lines 7-10 by doing a short out-and-back traversal of these edges as it explores nodes s. To avoid traversing one of these edges more than twice, it only explores these edges from the node with the larger lexicographical value.

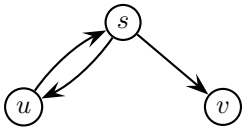
Algorithm 1 OUTPUT-PATH(G, s)

```
1:  for each vertex u ∈ G.adj[s] do
2:    output s
3:    if u.π == s then
4:      This edge is part of the BFS tree. Follow it and continue to output the path.
5:      OUTPUT-PATH(G, u)
6:    else
7:      if s.π ≠ u and s > u then
8:        This edge is not part of the BFS tree, but we still need to include it in our path.
9:        output u
10:     end if
11:   end if
12: end for
13: We have finished visiting descendants of s. Now we return to s.
14: output s
```

To find your way out of a maze, place a penny in every hallway that you travel. Put a penny down once when you enter the hallway and once when you leave. Never travel down a hallway with more than two pennies. By exploring hallways without any pennies before returning down hallways with one penny, you are performing a depth first search of the maze.

3. **Problem 22.3-9 on page 612**.

   Solution: Consider the following graph which contains a path from u to v. If we perform a DFS starting at node s and we visit u first, then we will have $u.f = 3 < 4 = v.d$.



4. **Problem 22.4-5 on page 615**.

   Solution: The algorithm TOPOLOGICAL-SORT implements topological sort in the manner described by the problem.

Algorithm 2 TOPOLOGICAL-SORT(G)

```
1: Calculate the in-degree of each vertex. Takes Θ(V + E) time.
2: Initialize the in-degree of each vertex to 0.
3: for each vertex u ∈ G.V do
4:    for each vertex v ∈ G.adj[u] do
5:       v.inDegree = v.inDegree + 1
6:    end for
7: end for
8: Initialize queue with vertices with in-degree 0. Takes Θ(V) time.
9: Q = ∅
10: for each vertex u ∈ G.V do
11:    if u.inDegree == 0 then
12:       ENQUEUE(Q, u)
13:    end if
14: end for
15: Remove and output vertices as their in-degree drops to 0. Takes O(V + E) time.
16: while Q ≠ ∅ do
17:    u = DEQUEUE(Q)
18:    output u
19:    for each vertex v ∈ G.Adj[u] do
20:       v.inDegree = v.inDegree − 1
21:       if v.inDegree == 0 then
22:          ENQUEUE(Q, v)
23:       end if
24:    end for
25: end while
26: Check for loops. Takes O(V) time.
27: for each vertex u ∈ G.V do
28:    if u.inDegree ≠ 0 then
29:       There is a cycle.
30:    end if
31: end for
```

The runtime of this algorithm is dominated by the amount of time it takes to calculate the in-degree of each vertex, which is $\Theta(V + E)$. Hence, this algorithm takes $O(V + E)$. If there are cycles, then this algorithm will topologically sort the vertices that are not part of the cycle. Those vertices that are part of the cycle will always have a positive in-degree as each one always has an edge leading to it. This algorithm will detect and report if a cycle exists by looking for vertices with a positive in-degree.