Analysis of Algorithms HW4

Rajan bhargava
Instructor: Prof. Eleni Drinea

April 14, 2016

1 Problem 1

This algorithm is a modified BFS, starting at node *v*. For each node *u* processed by BFS, we record two values: the number of shortest *v*-*u* paths *u.num*, and the length of the shortest paths *u.len*. Finally, return *w.num*. See Algorithm 4.1 for details.

Algorithm 1 Algorithm 4.1

```
procedure NumShortestPaths(G, v, w)
    while $Q \neq 0$ do
        ... ;
        $u \leftarrow Dequque(Q)$;
        for each $r \in G.Adj[u]$ do
            if $r$ is not visited then
                ... ;
                $r.len \leftarrow u.len + 1$;
            else
                if $r.len = u.len + 1$ then
                    $r.num \leftarrow u.num + r.num$
                end if
            end if
        end for
    end while
    finish BFS;
    return $w.num$
end procedure
```

Note that if G is not connected, and there are no paths between v and w, the algorithm will return 1.

Since this algorithm is a variation of BFS, and the additional operations cost constant time, the total running time is the same as BFS, i.e. O(m + n).

2 Problem 2

a)

Assume that such a node doesn't exist. Then, we have 2 paths from s to t without any common nodes, and both are of $length > n/2$. This would mean that there are more than n nodes in the graph, which is a contradiction.

b)

There are at least $n/2$ layers in the BFS tree. This means that at least 1 layer has only 1 node in it. Removing this nodes deletes all $s - t$ paths because of the nature of BFS. We do a BFS, and if we find a layer with only 1 node in it, we return that node.

3 Problem 3

a)

Forward direction:
Notice that an Euler tour is composed of several edge-disjoint cycles. For each simple cycle, every vertex has one edge come in and another edge goes out. So $in-degree(v) = out - degree(v)$ is true for every simple cycles. Now we observe the entire graph, and note that since an Euler tour exists, each simple cycle must be connected together, where each cycle has an edge coming in and an edge going out. We can therefore say that for each vertex v in the graph, $in - degree(v) = out - degree(v)$.

Backward direction:
Suppose we start from a particular vertex v, we leave this vertice and get to another vertice other than v, keep going through the vertex until we encounter vertice v again, mark all the edges we visited. Then starting from another unvisited edge. We start from a visited node u connected to that edge, follow the procedure above and back to u. Because the $in - degree = out - degree$ for all vertex, we can always do this until every edge is visited. And the Euler-tour is constructed.

b)

Algorithm:
The basic procedure is the same as described in the backward direction part. We can start from one vertex and build a simple cycle from it. Then pick another unvisited edge, find another cycle and it to the previous one until all the edges are visited.

Proof of Correctness:
The proof is similar to the proof for the backward direction in part (a).

Running Time Analysis:
Since every edge is visited once and all additional work done is constant, the algorithm runs in O(m) time.

4 Problem 4

We have

$$\prod W_i > 1 \Rightarrow \sum -W_i < 0.$$

We change the weights of edges to the negative of their logarithm. Then, the problem transforms to finding whether a negative cycle exists in the graph. We can detect negative cycles using the Floyd-Warshall algorithm.
Running time = $O(n^3)$.

5 Problem 5

We modify the update function of Dijkstra's algorithm. Refer Algotirhm 4.2.

Algorithm 2 Algorithm 4.2

```
procedure UPDATE(u,v)
    if dist(v) ¿ max(dist(u), weight(u,v)) then
        dist(v) = max(dist(u), weight(u,v));
        prev(v) = u;
    end if
end procedure
```