Algorithm Design and Analysis
Assignment 2

Wu Jiabao 523030910241

1. Firstly we construct G's reverse graph G', and employ DFS on it with finish time. DFS G by descending order of the finish time. Then we've got SCCs of G. Let each SCC in G be a super vertex, and these vertices form a new graph G^*. DFS G^* and if it finds a super vertex u such that it is not reachable to the super vertex containing t, the required path from s to t doesn't exist. Otherwise the path can be found.

   Prove. The SCCs of G form a partition. for each SCC, we can explore all the vertices in it if we can reach one of these points. Thus we can seem the SCC as a super vertex. During DFS, if we find a super vertex u such that it has no not-explored-yet neighbors and $t \notin u$, we can conclude that after exploring u, we cannot explore the SCC containing t.

   Time complexity: Employing DFS for three times, constructing new graph and checking reachability. The final complexity is:

$$O(|V| + |E|) + O(|E|) + O(|V| + |E|) = O(|V| + |E|)$$

2. 1. If G is a tree:
      Suppose there are two paths from s to a point u in G: $(s, x_1), (x_1, x_2) \ldots, (x_n, u)$ and $(s, y_1), (y_1, y_2) \ldots, (y_n, u)$. Then obviouly there exists a path: $(s, x_1), (x_1, x_2) \ldots,$ $(x_n, u), (u, y_n), \ldots, (y_1, s)$, which is a cycle. Thus $\forall v \in V$, there is only a unique path from s to v.
      For either DFS or BFS, it explores v along the path from s to v. Therefore, the parent-child relationship established during DFS and BFS are the same, leading to identical results.
   2. If DFS and BFS trees are identical:
      Assume G is not a tree, then there is a cycle in G, then there exists at least one point in the cycle, there are at least two paths from s to it. For DFS, it will trace deep into the path; For BFS, it will spread its exploring range. The difference in exploring and multiple paths to a same point lead to different results. Contradiction!

   Thus the DFS and BFS trees rooted at s are identical if and only if G is a tree.

3. We use Dijkstra Algorithm.

    1. Let $T = \{s\}$, $tdist[s] \leftarrow 0,\ tdist[v] \leftarrow \infty$ for all v other than s. $tdist[v] \leftarrow w(s, v)$ for all $(s, v) \in E$.
    2. $B[u] \leftarrow false$ for all $u \in V$.
    3. Find $v \notin T$ with smallest $tdist[v]$, $T \leftarrow T \cup \{v\}$.
    4. $tdis[u] \leftarrow min\{tdist[u],\ tdist[v] + w(v, u)\}$ for all $(v, u) \in E$.
    5. If $tdist[u] = tdist[v] + w(u, v)$, $B[u] = true$.

    Prove of correctness.

    1. If we employ the algorithm above but find $B[u] = true$ and the shortest path from s to u is unique. Then there exists v such that $tdist[u] = tdist[v] + w(u, v)$. Thus we can go from s to v than u, or go from s to u by another path. Otherwise when judging whether $tdist[u] = tdist[v] + w(u, v)$, we will find that $tdist[u] = \infty \neq tdist[v] + w(u, v)$. Contradiction!
    2. If we find $B[u] = false$ but the path is not unique, then $\forall (u, v) \in E$, $tdist[u] \neq tdist[v] + w(u, v)$. Thus the shortest path is always unique.

    The time complexity of Dijkstra Algorithm: $O((|E| + |V|) \log |V|)$.

4. (a)
    1. We use DFS to find the topological order of G.
    2. Then let $dist[s] \leftarrow 0, dist[v] \leftarrow \infty$ for all v other than s.
    3. Iterate $u \in V$ by the topological order:
       For all $(u, v) \in E, dist[v] = min(dist[v], dist[u] + w(u, v))$.
    4. output dist.

    Prove of correctness. Since we iterate V by topological order, there will not exist a shorter path to u after iterating u. Since we are using Bellman-Ford Algorithm to culculate the distance to each point, the culculation result is correct. The time complexity is: Finding topological order $O(|V| + |E|)$; Iterating edges and vertices: $O(|V| + |E|)$. So it is: $O(|V| + |E|)$.

    (b) Yes, we can. Prove. Suppose we have a path $P = \Sigma w_i$ from s to u in G, then we negate every weight to form G , we have P $= -\Sigma$iwi. Thus if we find the longest path in G , we find the shortest path in G. We can use the algorithm in (a) to solve it