ANALYSIS AND DESIGN OF ALGORITHMS

Sudheer Sahu

1. Since there is atmost one simple path from a vertex to another, so we can perform DFS starting from the source s and when we visit a vertex we should update its distance. Since there is only one simple path from the source s to any vertex u, we can fix its distance then itself because it won’t be reached from there from any other simple path. Only other way it might come to the same vertex is through a loop and in that case the new distance will be more than the earlier tentative distance on that vertex.

And in case even after going through the cycle that is reaching to a vertex again the new distance is lesser, it implies there is a negative cycle.

Algorithm:
Initially the array visited[1..n]’s all entries are initialised to 0. and initialise distance[s]=0;

dfs_shortest_path(u)
{
    visited[u]=1;
    for all outgoing edges (u,v) from u
    {
        if (visited[v]==0)
        {
            distance[v]=distance[u]+length(u,v);
            dfs_shortest_path(v);
        }
        else if (visited[v]==1 and distance[v] > distance[u]+length(u,v) )
            print (“NEGATIVE CYCLE EXISTS”);
    }
}

Running Time: Since in the dfs all nodes are visited only once and also every edge is visited only once. So the complexity is linear.

2. Actually a graph might have many MSTs and the spanning tree which has the minimum maximum-edge is always a MST.

And if we find out MST using the Kruskal’s algorithm then we always end up finding minimum max-edge.
Proof by contradiction:
Suppose the minimum max-wt edge spanning tree $T$ is not the MST. Then there is some other spanning tree $T'$ which is minimum spanning tree and whose maximum edge $e'$ is greater than maximum edge $e$ of our minimum max-wt spanning tree $T$. But still its total weight is less than that of $T$.
If I remove $e'$ from $T'$ then we will get two connected components. There must be some edge $e_2$ in $T$ which is connecting those two components. Now $e_2 \leq e < e'$. So if we now include the edge $e_2$ in $T'$-{$e'$} we will again get an spanning tree and whose cost is lower than the cost of $T$. It proves that $T'$ can’t be minimum spanning tree, which is a contradiction. Hence our assumption was wrong. Hence $T$ is a MST.
So it will be done using the simple Kruskal’s algorithm and hence the time complexity is $O(E \log V)$.

3
a) False.
Counter example.

Edge of length 50 will have to be included in the minimum spanning tree.

b) True
Because if it contains longest edge in the cycle you can always include any other edge (from the cycle) and remove this longest edge. and the resulting
spanning tree will have lower cost than the original tree, which means that the MST can’t have the longest edge in a cycle.

c) False
Counter example.

If we make vertex ‘a’ as root, then though the directed spanning tree \{(a,b),(a,c) and (a,d)\} with ‘a’ as root exists, but by running Prim’s algo on the above graph and starting with ‘a’ as root we won’t be able to find the minimum spanning tree.

4
Negate all the edges and then solve it to find minimum spanning tree in that. Then use Kruskal’s algorithm to solve the problem.
Another approach can be to modify Kruskal so that instead of picking up the minimum at every step it choses the maximum edge to include into the forest. It will also give the maximum spanning tree.
Running time: \(O(E \log V)\).

5.
Dijkstra’s algo can’t be modified to solve it because it doesn’t have optimal substructure property.
Explanation:
Consider two paths from $s$ leading to a vertex $v$. One is $p_1$ with maxima=9 and minima=2 so its variance is 7. While another path $p_2$ with maxima=15 and minima=10 so its variance=5. Now if there is an edge $(v,w)$ of length 3. So for the path $p_1 + (v,w)$ the maxima=9 minima=2 and variance=7 while for path $p_2 + (v,w)$ the maxima=15 minima=3 and variance=12.

Dijkstra’s is a greedy algorithm in which to find the shortest path $p$ from $s$ to any vertex $v$, we have to first find out the shortest path for all the vertices that comes on the path from $s$ till $v$. And for the shortest path problem it works because to reach to a vertex $v$ from $s$ in shortest distance all the vertices that comes in between should also be reached by shortest distance. While here as we have seen in the example constructed by me that in the shortest variance path from $s$ to $v$, it is not necessary that the vertices which comes on the path are also be reached by shortest variance path.

In the example given by me: To reach vertex $w$ by shortest variance path the vertex $v$ has to be reached by a non-shortest variance path.

It means that the greedy strategy won’t work for this problem. So Dijkstra’s can’t be modified to solve this problem.

6. At each vertex keep one more field no_of_paths. Initialise no_of_paths in source vertex to 1. And modify the relaxation step of Dijkstra’s algorithm as follows:

```plaintext
relax(u)
{
  for all outgoing edges(u,v) from u
    if (distance[v] > distance[u]+length(u,v) )
      { 
          distance[v]=distance[u]+length(u,v);
          no_of_paths[v]=no_of_paths[u];
      }
}
```

It will have the same time complexity as Dijkstra’s. So it is polynomial time. Its complexity is $O(n \log n + m \log n)$ and if we use a fibonacci heap then complexity is $O(n \log n + m)$. 

4