Practice For Quiz 2

1.

(a) Suppose that a bridge not in a BFS tree, so if we remove it, we know the edge in BFS tree edges must connect the whole graph nodes, however, as the defintion of the bridge, removing it would break the graph into two pieces. We meet the confilect so the assuming must wrong, we know every bridge must appear in every BFS tree of the graph.

(b) Let's say that e is (u,v), we choose u as the start node of running BFS, we explore all edges expect (u,v), if we reach the node v, then we know that e is a bridge, otherwise, it doesn't. Since the graph is connected, we know $|E| \geq |V|$,so the $O(E+V) = O(E)$.

(c) Run BFS on a random node R, We know that the possible bridge must in the tree, so there are $|V|-1$ edge at most. We know the part(b) algothrim on each edge, then we know which edges are bridges.

2.

We need to find the total number of paths in G from s to t. We first doing a modified topologicy sort and then get the answer.

ALGOTHRIM-P5$(G=(V,E),s,v)$

```
1  INITALIZE()
2  for v in V
3        count[v] = 0
4  count[s] = 1
5
```

3.

We run a modified version of BFS, which also memory the number of shorest paths to each node on the path.

PATHCOUNT-BFS(G, s, t)

```
1  for v in V
2      count[v] = 0
3      level[v] = -1
4  count[s] = 1
5  level[s] = 0
6  i = 1
7  frontiers = [s]
8
9  while frontiers ≠ [ ]
10     next = [ ]
11     for v in frontiers
12         for (v, u) in v.adj
13             if level[v] == -1
14                 next = next + [v]
15                 level[v] = i
16             if level[v] == i
17                 count[v] = count[v] + count[u]
18
19     i = i + 1
20     frontier = next
21 return count[t]
```

4.

DEGREEZEROSORT$(G = \{V, E\})$

```
1  for u in V
2      inDegrees = 0
3  for (u, v) in E
4      inDegrees[v] = inDegrees[v] + 1
5  frontier = [ ]
6  for u in V
7      if inDegrees[v] == 0
8          frontier = frontier + [v]
9  result = COPY(frontier)
10 while frontier ≠ [ ]
11     next = [ ]
12     for u in frontier
13         for (b, v) in u.adj
14             inDegrees[v] = inDegrees[v] − 1
15             if inDegrees[v] == 0
16                 next = next + [v]
17     frontier = next
18     result = result + next
19 return result
```

If G has cycles, this algothrim will terminate and leave the cycly nodes which $inDegress$ aren't zero, returning the topological ordering on the vertices that it has found.

5.

Use BFS to find three shortest paths in the undirected graph: the path from s to t, the path from s to u, and the path from u to t. Then compare d[s, t] to d[s, u] + d[u, t] to figure out whether its worth it to take a detour through u.

6.

Modify the Relax operation, and then run the normal Dijkstra's.

NEW-RELAX(u, v, w)

```
1  value = MAX(d[u], w)
2  if value < d[v]
3      d[v] = value
```

7.
Miss.
8.

We generate two graphs that one contains only red edges and the other contains only blue edges from the original graph. For each vertex in red graph, connect a edge from it to its corresponding vertex in blue graph with weight five. We should be careful that s_r or s_b may not be connected to any edge, in that case, we only need run Dijkstra on the other start point, and return MIN$(d[t_r], d[t_b])$. OtherWise, run Dijkstra on both s_r and s_b, then return the min $d[t]$.
This algothrim First select out the red and blue edges cost O(E), then cost O(V+E) for copying vertices and rebuild adj matrixes. After that, check the start point and it cost at most O(1), and we run Dijkstra

9.
Miss..

10.

We model this problem as a directed graph with weighted edges as following:
First, we creat n nodes that represent n days as a sequence, then link the front one to the behind one with a directed edge that weights zero. For each bid$(s_i, e_i, \$_i)$, draw a directed edge from v_{si} to v_{ei} with weight $-\$_i$ and spocal mark to indiect that . We could figure out that the graph is DAG, so first do topological sort then relax each edge. We finally follow the parent pointers and store each edge marked then we get the answer.

11.

We first modify the relax operation for Dijkstra to find all shorest paths bewteen two nodes.

RELAX(u, v, w)

1 $value = d[u] + w(u, v)$
2 **if** $value == d[v]$
3 $\quad p[v] = p[v] + [u]$
4 **else if** $value < d[v]$
5 $\quad p[v] = [u]$

Now, Dijkstra can find out all shorest pahts for us. Let's run Dirkstra start from s, then run BFS on t follows the parent pointers and mark all vertices on the way 'touched'. We then run Dirkstra start from u, and run BFS on v, if we meet any nodes that are 'touched' we know there exists nodes which is part of some shortest path from u to v and also a part of some shortest path from s to t. We run twice BFS cost $O(V + E)$ and twice Dijkstra cost $O(E + VlogV)$, so the total running time is $O(E + VlogV)$.