1) Bridge Crossing
    a.

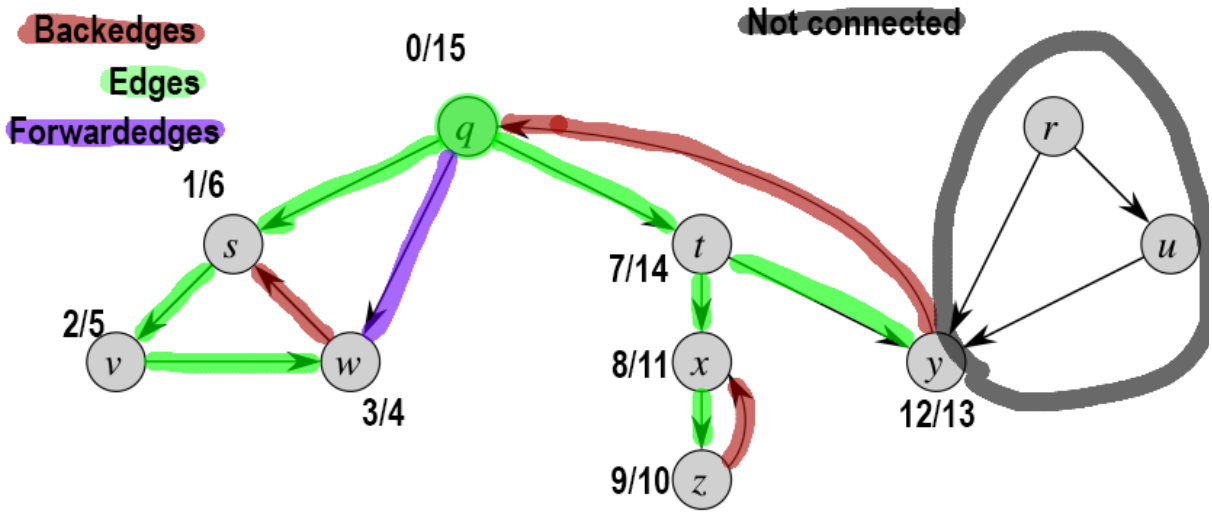
```
int findTime(int[] times, int n) {
        int time = 0;
        int[] startSide = {1..n}
        int[] endSide;
        while (startSide != 0) {
                startToFinish(times, time, startSide, endSide, n);
                finishToStart(times, time, startSide, endSide, n);
        }
        return time;
}

void startToFinish(times, time, startSide, endSide, n) {
        int smallest = 0, second = 0;
        for i in startSide {
                if times[i] < times[smallest] {
                        second = smallest;
                        smallest = i;
                }
        }
        time += times[second];
        startSide.delete(smallest);
        startSide.delete(second);
        endSide.append(smallest);
        endSide.append(second);
}

void finishToStart(times, time, startSide, endSide, n) {
        int smallest = 0;
        for i in endSide {
                if times[i]  < times[smallest] {
                        smallest = i;
                }
        }
        time += times[smallest];
        startSide.append(smallest)
        endSide.delete(smallest)
}
```

2) Rumor Spreading
    a.

```
void rumors(rumors, people, n) {
        for (int i = 2; i < n; i++) { //after this loop, person[0] knows all rumors
                people[i].send(people[0]);
        }
        for (int i = 2; i < n; i++) { //then he spreads them to everyone
                people[0].send(people[i]);
        }
}
```

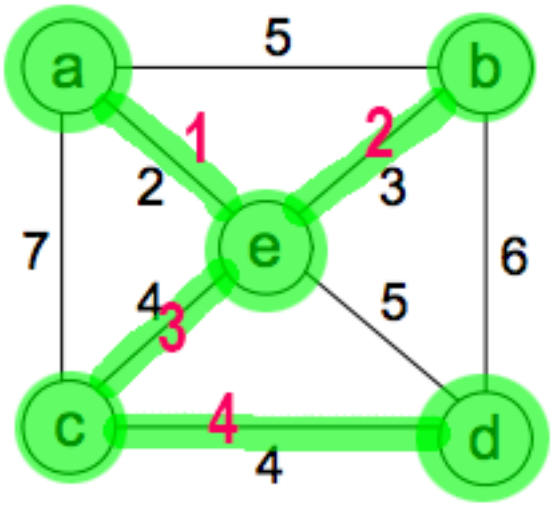
    b. This takes 2n – 2 emails to spread all rumors.

3) DFS and BFS of a Graph



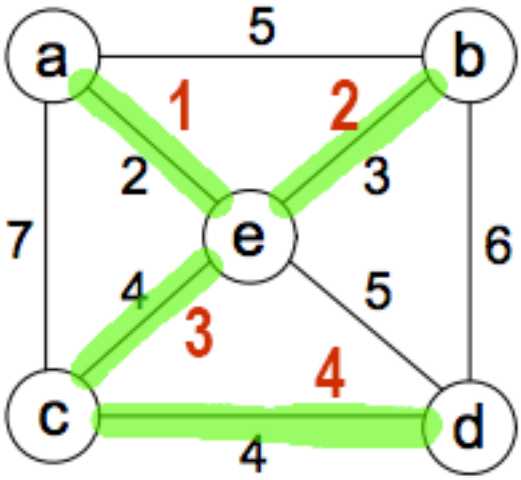
    a.
    b.
        i. Any edges which could become a back or forwards edge would be explored before moving on to any ancestors. In BFS, those are just edges.
        ii. If (u, v) is a tree edge, v's distance is u's distance added to (u, v).length. Assuming this graph is unweighted, (u,v).length = 1 and v.d = (r, u) + (u, v) = u.d + (u, v).length

4) MST
    a.



b. You could assign all edges weight 1. It's not particularly suited to the task because all edges would be equal so it'd just take vertices in alphabetic order, but it should still work.

c.

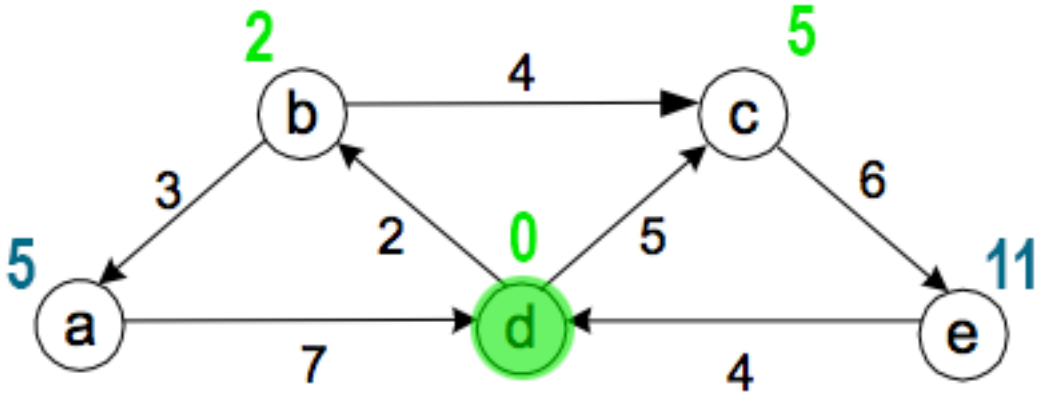


d.

i. True. Kruskal's algorithm would pick it up.
ii. False. It is possible that the edges needed to involve that edge equal another spanning tree that doesn't include it.
iii. False. You could have a tree with lots of small edges instead of few large edges.
iv. I'm not sure.

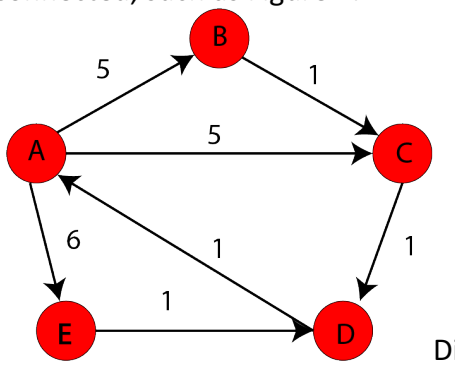
5)

a.



b.

i. True. Dijkstra only fails to provide a spanning tree on graphs that are not truly connected, such as Figure 1.



ii. False. Dikjstra with root A gives a tree of weight 17, kruskal with root A gives a tree of weight 14.

c.