**Question 1** (Pre and Post Orderings, 30 points). *Give the pre- and post- orders obtained for all vertices when DFS is run on the directed graph below. Assume that whenever the algorithm has a choice of which vertex to explore next, it always picks the alphabetically first one.*

The ordering is as follows:

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Pre</th>
<th>Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>C</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>G</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>H</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>I</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>J</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>
Question 2 (Closest Fire Station, 35 points). The layout of Tindertown is given by an undirected, unweighted graph $G = (V,E)$ (all of the roads in the city are of exactly the same length). A subset $F \subseteq V$ of these intersections have fire stations. The town council wants to better understand fire response times, and thus wants to know for every vertex $v \in V$ what the distance to the closest fire station is. Provide an algorithm that given $G$ and $F$ computes for every vertex $v \in V$ the distance to the closest element of $F$. For full credit, your algorithm should run in time $O(|V| + |E|)$ or better.

We can solve this with a slight modification of BFS. The idea is that all vertices in $F$ (rather than just a single start vertex) need to be assigned to distance 0. The algorithm will then work as usual to correctly find the vertices at distances 1, 2, 3, . . . from any of these start vertices. The algorithm will run in time $O(|V| + |E|)$ for the same reasons that BFS does. The pseudocode is given below:

```python
Distances(G,F)
Create queue Q
For v in V set dist(v)<- infinity
For v in F set dist(v)<-0, and enqueue(v)
While Q not empty
    u <- eject(Q)
    For (u,w) in E
        If dist(w) = infinity
            Set dist(w)<-dist(u)+1
            enqueue(w)
```

Alternate Solution: Alternatively, we can simply create a new “central hub” node that connects to each of the fire stations. Then the minimum distance from $v$ to the closest fire station will be one less than the distance from $v$ to the hub, so we can simply run BFS on the new graph. The algorithm is as follows:

```python
Distances(G,F)
Create a new vertex h
Add edges from h to v for each v in F
Run BFS(G,h)
For each v in V decrease dist(v) by 1
```
Question 3 (Air Travel, 35 points). The nation of Graphania has cities given by a set $V$. These cities are connected by two kinds of (undirected) edges (forming a set $E$), roads and air routes. Give an algorithm that given $V$ and $E$, with edges labelled as either roads or air routes, determines whether or not it is the case that every pair of cities have a path between them with at most one air route (and any number of roads). For example, your algorithm should say ‘yes’ if the graph is connected even without the air routes, and should say ‘no’ if there is any pair of cities that requires at least 2 air routes to get between. For full credit, your algorithm should run in time $O(|V| + |E|)$ or better.

There is such a path between $v$ and $w$ if and only if $v$ and $w$ are connected by roads, or if $v$ is connected by roads to some $v'$ and $w$ to some $w'$ so that there is an air route between $v'$ and $w'$. This means that if we look at the connected components just formed by roads, we need to know whether $v$ and $w$ are either in the same connected component or if there is some air route between their connected components. Thus, the final algorithm really only wants us to determine whether or not there is an air route between every pair of these connected components. This gives us the following algorithm:

Connectivity($G$)
Create a graph $G'$ which is a copy of $G$ but using only the roads and not the air routes
Run DFS($G'$) to label the vertices by connected component
Create a table $T$ of all pairs of connected components, initially marking all pairs as unconnected
For each air route $(v, w)$ in $E$
    Mark $(CC(v), CC(w))$ as connected in $T$
If all pairs in $T$ are marked as connected
    Return 'yes'
Else
    Return 'no'

We can construct $G'$ in $O(|V| + |E|)$ time by copying everything in $G$ except the air routes. The DFS also runs in $O(|V| + |E|)$ time. The loop over all edges runs in $O(|E|)$ time. Therefore the final runtime is $O(|V| + |E|)$.