1. Here is the final configuration after linear probing:

<table>
<thead>
<tr>
<th>Key</th>
<th>Hash value</th>
<th>final position (linear, quad)</th>
<th># collisions (linear, quad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A29</td>
<td>7</td>
<td>7, 7</td>
<td>0, 0</td>
</tr>
<tr>
<td>C42</td>
<td>0</td>
<td>0, 0</td>
<td>0, 0</td>
</tr>
<tr>
<td>E12</td>
<td>5</td>
<td>5, 5</td>
<td>0, 0</td>
</tr>
<tr>
<td>D31</td>
<td>1</td>
<td>1, 1</td>
<td>0, 0</td>
</tr>
<tr>
<td>F08</td>
<td>2</td>
<td>2, 2</td>
<td>0, 0</td>
</tr>
<tr>
<td>G34</td>
<td>7</td>
<td>8, 8</td>
<td>1, 1</td>
</tr>
<tr>
<td>B10</td>
<td>0</td>
<td>3, 4</td>
<td>3, 2</td>
</tr>
</tbody>
</table>

And after quadratic probing:

For your convenience (this is not a part of the question), here is the detailed info about the keys:

2. The claim is true (that is, such a graph has to be connected). We can prove it by contradiction. Suppose such a graph $G = (V, E)$ exists. It has $n$ (even) nodes where each node as a degree of at least $n/2$, but it is not connected. This means it has at least two separate components. Simple math shows that there must be at least one component in the graph with a size $\leq n/2$ (equality happens if the graph has two equal size components). If we look at the smallest component, whose size is at most $n/2$, none of its vertices is connected to any other part of the graph except the other nodes in the component. This means that every node is connected to at most $n/2 - 1$ other nodes, contradicting the assumption that the degree of each node is at least $n/2$.

3. You need to modify the BFS algorithm to also maintain, for every node $w$, the number of shortest paths from $v$ to it. Let’s call it $\text{pathsTo}[w]$. Notice that for the algorithm to work, $\text{pathTo}[v] == 1$ and not 0 (and indeed there is one path of size 0 from $v$ to itself). During the original run of BFS (see slides) we only take action if a node $u$ is yet unmarked when it is discovered by another node $t$: We put it in the queue, mark it as marked and set its dist and edgeTo. We still do it if the node $u$ is unmarked, but we also set $\text{pathsTo}[u] = \text{pathsTo}[t]$ (since all the paths leading to $t$ also lead to $u +$ the $t-u$ edge). We also have to add the following, for all of $t$’s marked neighbors as an else to the if statement:

```plaintext
else
{
    if (distTo[t] == distTo[u]+1)
        pathsTo[t] += pathsTo[u];
}
```

Explanation: If $\text{distTo}[t] == \text{distTo}[u]+1$, it means we already discovered a path (or several paths) to $t$ by the time we reach $u$. However, $u$ adds more paths with the same distance so we need to add them up. Notice that by BFS properties, all of $u$’s neighbors have a distance of at most $\text{distTo}[u] \pm 1$, and the first time we assign a distance to a node, it’s its final distance.
4. The DFS order of visitation is as follows (only edges that lead to newly discovered vertices are depicted):

\[ v_1 \]
edge \( v_1 - v_2 \)

\[ v_2 \]
edge \( v_2 - v_4 \)

\[ v_4 \]
edge \( v_1 - v_5 \)

\[ v_5 \]
edge \( v_5 - v_3 \)

\[ v_3 \]
edge \( v_3 - v_6 \)

The graph with tree edges marked looks like this:

![Graph with tree edges marked]

5. Here is a simple example. Vertices B and C are both a distance of 1 from A, but there is a path also from B to C, so a topological order must be A, B, C. However, B and C are equidistant from A, so a topological sorting based on the distance (or BFS) may put C before B.

![Graph with B and C equidistant from A]

6. We run the same algorithm described in the question. The algorithm stops when there are no nodes with an in-degree of 0. This can only happen if either there are no more nodes (in case of a DAG) or there is a cycle (otherwise). To implement the algorithm we can store all the nodes in a priority queue, for example, where the priority is the in-degree. At every step we delete the minimum node from the queue (with in-degree 0) and decrease the priority of all of its neighbors by 1. This is akin to deleting the node and all of its outgoing edges. If there is a cycle, we will end up with the nodes in the cycle still in the priority queue, since none of their in-degrees are 0. So, if by the end of the algorithm the queue is not empty, the remaining nodes in the queue are in a cycle, so we can pick one and cycle through it. The run time is the same as topological sorting, since we either finish the topological sorting algorithm or stop it before we go through all the nodes.