Q1: Prove or disprove: You are given a connected undirected graph \( G = (V, E) \) with a weight function \( w \) defined over its edges. Let \( s \in V \) be an arbitrary vertex in \( G \). Starting at vertex \( s \), if you do a depth-first search (DFS) in \( G \) such that the edges going out of any vertex are always explored in increasing order of weight, then the resulting DFS tree is also a minimum spanning tree.

Solution. The assertion is false. Consider the graph \( G = (V, E) \) where \( V = \{a, b, c\} \) and \( E = \{(a, b), (b, c), (a, c)\} \). The weight function is \( w(a, b) = 1, w(a, c) = 2 \) and \( w(b, c) = 3 \). If we perform a DFS from the vertex \( a \) using the above rule, the unique DFS tree is given by the edges \( \{(a, b), (b, c)\} \), and its weight is 4. On the other hand, the MST is formed by edges \( \{(a, b), (a, c)\} \), and has a weight of 3.

Q2: Give an example of a weighted connected undirected graph \( G = (V, E) \) and a vertex \( v \) such that the minimum spanning tree of \( G \) is different than the shortest path tree rooted at \( v \). Can the two trees be completely disjoint?

Solution. They can’t be completely disjoint as the smallest edge incident on \( v \) will be the same in both trees (assuming that the smallest weight edge incident on \( v \) is unique). The two trees can be different though, as the following example shows: consider a graph \( G \) that is a cycle on \( n \) vertices \( \{v_0, v_1, v_2, ..., v_{n-1}\} \). Let the edge \( e = (v_0, v_{n-1}) \) have a weight of \( n - 2 \) and all other edges in \( \{(v_0, v_1), (v_1, v_2), ..., (v_{n-2}, v_{n-1})\} \) have a weight of 1. The shortest path rooted at \( v_0 \) will contain the edge \( e \), whereas the minimum spanning tree will not contain \( e \).

Q3: Given a directed graph with \( n \) vertices and \( m \) edges, design an \( O(mn) \) algorithm to find the length of the directed cycle with the minimum number of edges (or report that the graph is acyclic). Assume \( n \leq m \leq n^2 \).

Solution. The critical observation is that the shortest directed cycle is a shortest path (number of edges) from \( s \) to \( v \), plus a single edge \( v \rightarrow s \).

For each vertex \( s \):
- Use BFS to compute shortest path from \( s \) to each other vertex.
- For each edge \( v \rightarrow s \) entering \( s \), consider cycle formed by shortest path from \( s \) to \( v \) (if the path exists), plus the edge \( v \rightarrow s \).

Return the shortest overall cycle.

The running time is \( O(mn) \). The single-source shortest path computation from \( s \) takes \( O(m + n) \) time per \( s \) using BFS. Finding all edges entering \( s \) takes \( O(m + n) \) time by scanning all edges (though a better way is to compute the reverse graph at once and access the adjacency lists). We must do this for each vertex \( s \). Thus the overall running time is \( O(n(m + n)) \), which resolves to \( O(mn) \) since \( n \leq m \leq n^2 \).
Q4: Consider the following directed graph.

A. Run recursive depth-first search, starting at vertex A. Assume the adjacency lists are in lexicographic order, e.g. when exploring vertex E, consider E − D before E − G or E − H. Complete the list of vertices in preorder (visit-time).

Solution. A B C E D G H F I

B. Run breadth-first search, starting at vertex A. Assume the adjacency lists are in lexicographic order. Complete the list of vertices in the order in which they are enqueued.

Solution. A B D E C F G H I

Q5: Suppose you know the MST of a weighted graph $G = (V, E)$. Now, a new edge $(v, w)$ of weight $c$ is inserted into $G$ to form a weighted graph $G'$. Design an $O(V)$ time algorithm to determine whether the MST in $G$ is also an MST in $G'$. You may assume all edge weights are distinct.

Solution. Find the unique path between $v$ and $w$ in the MST of $G$. This takes $O(V)$ time using BFS or DFS because there are only $V − 1$ edges in the MST subgraph. We claim that the MST in $G$ is the same as the MST in $G'$ if and only if every edge on the path has length less than $c$ (recall that we assume all edge weights are distinct).

- If any edge on the path has weight greater than $c$, we can decrease the weight of the MST by swapping the largest weight edge on the new path with $(v, w)$. Hence weight of the MST for $G'$ is strictly less than the weight of the MST for $G$.

- If the weight of $(v, w)$ is larger than any edge on the path between $v$ and $w$, then the cycle property asserts that $(v, w)$ is not in the MST for $G'$ (because it is the largest weight edge on the cycle consisting of the path from $v$ to $w$ plus the edge $(v, w)$). Thus, the MST for $G$ is also the MST for $G'$.

Q6: You are given an array of $n$ integers and a number $k$. Determine whether there is a pair of elements in the array that sums to exactly $k$. For example, given the array [1, 3, 7] and $k = 8$, the answer is “yes” since $1 + 7 = 8$, but given $k = 6$ the answer is “no”. Your algorithm should run in expected $O(n)$ time.

Solution. This is a famous problem called the two-sum problem. Here’s the optimal $O(n)$ space and expected $O(n)$ time solution (in code). On the exam, please don’t use code unless otherwise specified :)

public boolean sumsToTarget(int[] arr, int k) {
    Set<Integer> values = new HashSet<>();
    for (int i = 0; i < arr.length; i++) {
        if (values.contains(k - arr[i])) {
            return true;
        }
        values.add(arr[i]);
    }
    return false;
}

Q7:
A. Suppose that the following keys are inserted in the order: A B C D E F G into an initially empty linear-probing hash table of size 7, using the following hash function:

<table>
<thead>
<tr>
<th>key</th>
<th>hash(key, 7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
</tr>
</tbody>
</table>

What is the result of the linear-probing array? Assume that the array size is fixed.

Solution.

B. Suppose we use a hash function $h$ to hash $n$ distinct keys into an array $T$ of length $m$. Assuming simple uniform hashing and hashing with chaining, what is the expected number of collisions? Provide a short explanation.

Solution. There are $\frac{n(n-1)}{2m}$ expected collisions.

We use $x_{ij}$ to signify the event that keys $k_i$ and $k_j$ hash to the same value. For each key pair $(k_i, k_j)$, where $1 \leq i < j \leq n$, the likelihood of a collision is $P(x_{ij}) = 1/m$. There are a total of $n(n-1)/2$ possible (unordered) key pairs. So the total number of expected collisions is $\frac{n(n-1)}{2m}$.

Q8: True/False:
A. If $G$ has a unique heaviest edge $e$, then $e$ cannot be a part of any MST.
B. If $G$ has a unique lightest edge $e$, then $e$ must be a part of some MST.
C. If $G$ has a unique lightest edge $e$, then $e$ must be a part of every MST.
Solution.

A. False (e could be a cut edge)

B. True (it is the first edge added by Kruskal)

C. True: given an MST $T$ that doesn’t contain $e$, we can add $e$ to $T$ to get a cycle, then remove the heaviest edge on that cycle. This gives a spanning tree of strictly lower weight, so $T$ wasn’t actually an MST.

Q9: Given a DAG $G$ and two vertices $s$ and $t$, count the total number of paths from $s$ to $t$.

Solution. Topologically sort the graph. Then use the following recurrence to compute the number of paths:

$$\text{Paths}(u) = \sum_{v \in \text{Adj}[u]} \text{Paths}(v)$$

where $\text{Paths}(t) = 1$. Note that by storing $\text{Paths}$ as an array, you can implement this in $O(n + m)$ time (if you recursively re-compute $\text{Paths}(v)$ each time, the runtime could be exponential—saving the result in an array “caches” it so you only compute $\text{Paths}(v)$ once for each vertex $v$).

Q10: Recall the fibonacci numbers $f_1 = 1, f_2 = 1, f_k = f_{k-1} + f_{k-2}$ for $k > 2$. Suppose you are given an alphabet of size $n$ with characters $c_1, ..., c_n$. Suppose that the frequency of $c_i$ is $f_i$. What is the length of the huffman code for $c_i$?

Solution Draw the tree! Should be $n + 1 - i$ for $i > 1$ and $n - 1$ for $i = 1$. Note, you can prove that $\sum_{i=1}^{t} f_i < f_{t+2}$ using induction, which will rigorously show why the tree looks that way.
Some Useful Facts

These will be stapled to the back of your exam for reference (if needed).

1. $\lg n = \log_2 n$
   $\ln n = \log_e n$

2. Below are some formulas that may come handy.
   - $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
   - $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
   - $(a + b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}$

3. Log rules:
   - $a \log_a b = b$
   - $\log ab = \log a + \log b$
   - $\log \frac{a}{b} = \log a - \log b$
   - $\log a^b = b \log a$
   - $\log_a a = 1$
   - $\log 1 = 0$

4. Expected value:
   - $E[X] = \sum x P(X = x)$
   - Linearity of Expectation: for any finite collections of random variables $X_1, X_2, ..., X_n$, $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$