1. (10pts) Given a dense directed graph G(V,E) with possibly negative weights (but no negative cycle), for each of the following scenarios very briefly explain how to use one of the fastest algorithms (in the lectures) and write down its time complexity.
   a) Find cycles
   b) All-pairs shortest paths
   c) Single-source shortest paths
   d) Minimum spanning tree

(a) Augment DDFS with a Stack to keep track of the current path. If the path cycles back to a vertex that is on the Stack, it is a cycle. Runtime: O(V + E)
(b) Use the altered Floyd-Warshall Algorithm that loops through all vertex pairs, i and j, as well as an intermediary vertex k to check for the shortest path between i and j. Total runtime: O(V^3)
(c) Use Bellman-Ford since Dijkstra’s does not allow negative weights—systematically relax all edges to find shortest path, starting with all paths of one edge, then two edges, etc. until you check all possible paths—total runtime: O(VE)
(d) MST—multiple answers are possible, but since the graph is described as dense, Prim’s is probably the fastest—start with an arbitrary vertex and choose the smallest of all the edges that will expand the tree to a new vertex. Time: O(ElogV)
2. (10pts) Use Prim’s algorithm to find the minimum spanning tree of the following graph. Start from vertex 8 and list the edges in the order in which they are chosen.

There are more than one answer
Edges:
(7,8)
(4,7)
(6,8)
(5,6)
(3,5)
(1,4)
(2,6)
3. (10pts) Using Huffman tree encoding, find an efficient code for a file of 115 characters and the following frequency table. Draw your tree and label the edges as follows:
   1. For each node put its smaller (value) sub-tree on the left.
   2. Left-edges are numbered 0 and right-edges are numbered 1.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>15</td>
<td>16</td>
<td>11</td>
<td>5</td>
<td>37</td>
<td>7</td>
</tr>
</tbody>
</table>
4. (10pts) Let \((u,v)\) be a minimum-weight edge in a connected graph \(G\). (Exclude the case in which all edges have equal weights). Write an argument that \((u,v)\) belongs to some minimum spanning tree of \(G\). (Hint: Prove by contradiction. Take an arbitrary MST [i.e. \(T\)] and argue that you can find another tree whose weight is less than \(T\).)

Assume the edge \((u,v)\) doesn’t belong to any MST. Let \(T\) be an MST. Adding \((u,v)\) to \(T\), creates a cycle inside the tree \(T\). Since \((u,v)\) is a minimum-weight edge there should be at least one edge belonging to the cycle, whose weight is more than \((u,v)\). Remove that edge from \(T\) and add \((u,v)\), it creates a spanning tree whose weight is less than \(T\), but it’s a contradiction since we assumed \(T\) is an MST.
5. (10pts) In a text of length 1000 with the following pattern
1234123412341234...
How many successful comparisons and how many unsuccessful comparisons are
made by the brute-force string-matching algorithm to search for all occurrences of 341?
We are looking for two answers here. (A successful comparison is when one character
from the pattern matches one character from the text, otherwise it is unsuccessful.)

To find all the occurrences of a pattern of length m in a text of length n, the brute-force
algorithm shifts the pattern (by a single character) in n-m+1 iterations, but in each
iteration the algorithm in some cases might make all the m comparisons before shifting
the pattern. Therefore here the total number of iterations are 1000-3+1 = 998, it means
the last two characters are ignored. To find the successful comparisons, the iterations
should start from character 3. There are 250, characters equal to 3 in the text, but since
one of them is among the last 2 characters so we just consider 249, characters 3. Hence
starting the iterations from each character 3, we have three consecutive successful
comparisons, in total we have 249*3 = 747 successful comparisons.
The number of unsuccessful comparisons is the number of characters equal to 1, 2 and
4, which is 250+250+249 = 749, respectively.

Successful: 747
Unsuccessful :749
6. (10pts) Write the pseudocode of an algorithm to compute Last-Occurrence Function in time $O(m + |\Sigma|)$, where $m$ is the size of pattern $P$ and $\Sigma$ is the set of symbols.

Solution1:

**Algorithm 1 Compute LOF**

1. **Input:** $P$, $\Sigma$
2. **Output:** $L$
3. for character $c \in \Sigma - P$ do
   4. $L(c) \leftarrow -1$
5. end for
6. for $i = m - 1 : 0$ do
7. if $L(P[i]) = -1$ then
8. $L(P[i]) \leftarrow i$;
9. end if
10. end for
11. return $L$

Solution2:

**Algorithm 2 Compute LOF**

1. **Input:** $P$, $\Sigma$
2. **Output:** $L$
3. for each character $c \in \Sigma$ do
   4. $L(c) \leftarrow -1$
5. end for
6. for $i = 0 : m - 1$ do
7. $L(P[i]) \leftarrow i$;
8. end for
9. return $L$
7. (10pts) Use Boyer-Moore’s algorithm to find the substring “puppet” in the text "puppy puppet looks happy".
(a). First compute the last occurrence function. For the alphabet assume it is the letters that compose the text. Ignore blank spaces.
(b). Following the Boyer-Moore algorithm, what is the number of comparisons required for finding the pattern in the given text? Include a trace of the algorithm execution, similar to those shown in class, positioning puppet beneath the beginning of the text and then at all its subsequent positions until found in the text.

(a)

<table>
<thead>
<tr>
<th>c</th>
<th>u</th>
<th>p</th>
<th>e</th>
<th>t</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>L(c)</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>-1</td>
</tr>
</tbody>
</table>

(b) 9
1st comparison
puppypuppetlookshappy
puppet

2nd comparison
puppypuppetlookshappy
puppet

3rd comparison
puppypuppetlookshappy
puppet

4th comparison
puppypuppetlookshappy
puppet

5 more comparisons are needed.
8. (10pts) Suppose that all characters in pattern $P$ are different. Show the pseudocode of an algorithm that accelerates the brute-force algorithm to run in time $O(n)$ on a text of $n$ characters.

Solution 1:

We know that one occurrence of $P$ in $T$ cannot overlap with another, so we don’t need to double-check the way the naive algorithm does. If we find an occurrence of $P_x$ in the text followed by a non-match, we can increment $s$ by $k$ instead of 1.

```
Algorithm 4 Pattern Matching
1: Input: Strings $T$ (text) with $n$ characters and $P$ (pattern) with $m$ characters
2: Output: Starting index of the first substring of $T$ matching $P$, or an indication that $P$ is not a substring of $T$
3: $i \leftarrow 0$;
4: while $i < n$ do \hspace{1cm} $\triangleright$ $i < n - m$ is also correct
5: \hspace{1.2cm} $j \leftarrow 0$;
6: \hspace{1.2cm} while $j < m$ and $T[i + j] = P[j]$ do
7: \hspace{2.4cm} $j \leftarrow j + 1$;
8: \hspace{1.2cm} end while
9: \hspace{1.2cm} if $j = m$ then \hspace{1cm} $\triangleright$ A match!
10: \hspace{2cm} return $i$
11: \hspace{1.2cm} else if $0 < j < m$ then \hspace{1cm} $\triangleright$ A mismatch in the middle of $P$
12: \hspace{2cm} $i \leftarrow i + j$;
13: \hspace{1.2cm} else \hspace{1cm} $\triangleright$ A mismatch at the beginning of $P$
14: \hspace{2cm} $i \leftarrow i + 1$;
15: \hspace{1.2cm} end if
16: end while
17: return There is no substring of $T$ matching $P$
```

Define $k = i + j$. In a single execution of outer while loop, $k$ increases by at least 1. The possible maximum value of $k$ is $n$. Thus the running time is $O(n)$.

Students who are on the right track but got 8 (or under) for this question often made one (or more) of the following mistakes (but not limited to):

1. Wrong index;
2. Have not considered two mismatch cases;
3. Use for($i=0; i<n; i++)$ as outer loop (this is wrong since increment of $i$ depends on different cases);
4. Have no indication or wrong indication where inner loop or “if” condition statement ends;
5. No return or wrong return.

Solution 2:

If all characters in pattern $P$ are different, then failure function is trivial, i.t. $f(j)=0$ for $0 \leq j < n$.

Running KMP without preprocessing takes time $O(n)$.

---

**Algorithm 3** Pattern Matching

1. **Input:** Strings $T$ (text) with $n$ characters and $P$ (pattern) with $m$ characters
2. **Output:** Starting index of the first substring of $T$ matching $P$, or an indication that $P$ is not a substring of $T$
3. $i \leftarrow 0$;
4. $j \leftarrow 0$;
5. **while** $i < n$ **do**
6. \hspace{1em} **if** $P[j] = T[i]$ **then**
7. \hspace{2em} **if** $j = m - 1$ **then** $\triangleright$ A match!
8. \hspace{2em} **return** $i - m + 1$
9. \hspace{1em} **end if**
10. $i \leftarrow i + 1$;
11. $j \leftarrow j + 1$;
12. **else if** $j > 0$ **then** $\triangleright$ A mismatch in the middle of $P$
13. \hspace{1em} $j \leftarrow 0$;
14. **else** $\triangleright$ A mismatch at the beginning of $P$
15. \hspace{1em} $i \leftarrow i + 1$;
16. **end if**
17. **end while**
18. **return** There is no substring of $T$ matching $P$

---

Running time analysis can be found in the textbook.

These two solutions are essentially the same, differing in appearance.
9. (10pts) Consider the following algorithm for finding a minimum spanning tree in a connected weighted graph \((w)\) is considered as the weight function). Determine if the output of this algorithm is always a minimum spanning tree of the graph or not and explain how you know. (Hint: Recall the cycle property discussed in class.)

\[
\text{sort the edges into non increasing order } (w(e_1) \geq w(e_2) \cdots \geq w(e_m))
\]

\[
\text{Initialize } T \text{ to be equal to the set of all edges } E
\]

\[
\text{for } i = 1 \text{ to } m \text{ do}
\]

\[
\text{if } T - \{e_i\} \text{ is a connected graph then}
\]

\[
T = T - \{e_i\}
\]

\[
\text{end if}
\]

\[
\text{end for}
\]

\[
\text{return } T
\]

This is Reverse-Delete algorithm. Consider any edge \(e = (v,w)\) removed by Reverse-Delete. At the time that \(e\) is removed, it lies on a cycle \(C\); and since it is the first edge encountered by the algorithm in decreasing order of edge costs, it must be the most expensive edge on \(C\). Thus by circle property, \(e\) does not belong to any minimum spanning tree. So if we show that the output \((V,T)\) of Reverse-Delete is a spanning tree of \(G\), we will be done. Clearly \((V,T)\) is connected, since the algorithm never removes an edge when this will disconnect the graph. Now, suppose by way of contradiction that \((V, T )\) contains a cycle \(C\). Consider the most expensive edge \(e\) on \(C\), which would be the first one encountered by the algorithm. This edge should have been removed, since its removal would not have disconnected the graph, and this contradicts the behavior of Reverse-Delete.
10. (10pts) Finding patterns in DNA sequences is a common task in bioinformatics. A DNA sequence is composed of characters A, C, G and T representing adenine, cytosine, guanine and thymine respectively. The KMP algorithm is used when the text and the pattern are not too long. Before running KMP we must calculate the failure function of the pattern. We need to find the pattern “GACAGATGA” in a DNA sequence.

(a) Calculate the failure function for the given pattern.

(b) Following the KMP algorithm, what is the number of comparisons required for finding the pattern “GACAGATGA” in “GGTACCCGACAGATGACAGA”? Include a trace of the algorithm execution, similar to those shown in class, positioning the pattern beneath the beginning of the text and then at all its subsequent positions until found in the text.

Solution: (a)

<table>
<thead>
<tr>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P[ j ]</td>
<td>G</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>A</td>
</tr>
<tr>
<td>F[ j ]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
