Select the correct option for the following multiple choice questions or provide short answers accordingly:

1. Construct a standard trie over the following list of words: ape, ate, black, blue, atom, ball, blast, cat.
2. Construct a compressed/patricia trie over the following list of words: ape, ate, black, blue, atom, ball, blast, cat.
3. Construct a suffix trie for the string “mississippi”.

Solution:

```
  a
 /|
pe t
 /|
 e om
 /|
 ok st
```

```
b
 /|
all
 /|
a ue
```

```
c
 /|
at himp
```

Solution:
4. Tries are often used for prefix matching. Suppose you have a standard trie ‘T’ and a string ‘S’, and you want to find strings in ‘T’ that contain ‘S’ as the prefix. Describe a tree traversal algorithm that does this prefix search. To get full credit, the algorithm should output shorter/better matches first. For example, if S = ‘cat’ and T contains cat, car, bull, bear, stock, cater and caterpillar then the prefix search should output strings in the following order: cat, cater, caterpillar.

Solution: Begin searching for the prefix like a regular trie lookup. If the prefix can’t be found during this step then terminate. Otherwise once you have gone through all the characters in the prefix and are at some node ‘X’, check if the current node is stop node if it is then add it to your solution array. Next perform a breadth first search from there, and add the word to your array once you reach a stop word. During BFS, keep track of the words by adding both the prefix and the next node as a tuple in the queue. For example, once you have added cat, insert the following tuple in your queue: [(cat, node for e)]. On the next iterations the queue will be updated to [(cate, node for r)] and then [(cater, node for p)]. Also since cater is a complete word, add it to your solution array. Carry on with the BFS until you reach caterpillar, and add it your solution array.

5. What is the time complexity of constructing a trie for ‘N’ words of length at-most ‘L’ and then searching ‘K’ words of length at-most ‘M’ in this trie? You can assume that the words are from the English language.
   a) O(N + K)
   b) O(NL + KM)
   c) O(N)
   d) O(NK)

Solution: Constructing the trie would take O(NL) since there are N words and the length of the longest word would be L. Searching for K words would take a time of O(KM). Since these are two independent steps they take a total time of O(NL + KM)

6. How many unique minimum spanning trees does the following graph have? Please show all the MSTs.
solution: 6.
\( \binom{3}{2} \times 2 \)
7. Consider the undirected graph below:

(a) Using Prim's algorithm to construct a minimum spanning tree starting with node A, which one of the following sequences of edges represents a possible order in which the edges would be added to construct the minimum spanning tree? (circle all that apply)

a) (A, D), (A, B), (A, C), (C, F), (F, G), (G, E)
b) (A, D), (A, B), (A, C), (C, F), (G, E), (F, G)
c) (A, D), (A, B), (D, F), (F, G), (G, E), (F, C)
d) (A, D), (A, B), (D, F), (F, C), (F, G), (G, E)

(b) If we use Kruskal's Algorithm to construct our minimum spanning tree, list the sequence of edges in order of being added to the MST.

solution: ad
(b) (E, G), (C, F), (G, F), (A, D), (A, B), (D, F)

8. Which vertices may be removed to make the following graph no longer biconnected? (circle all that apply)
8. What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers? Draw an encoding tree. Based on your Huffman code, what would “decaf” be encoded into?

```
a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21
```

```
a:0000000  
b:0000001  
c:000001   
d:00001   
e:001   
f:001   
g:01   
h:1   
```

“decaf”:00001 0001 000001 0000000 001

Note that Huffman code for this problem is not unique. There are other possibilities.
10. When the frequencies are the first $n$ Fibonacci numbers ($n$ could be greater than 26 since the symbol set we are trying to encode is not limited to alphabet), how many bits will be needed to encode the least frequent character? Give brief explanation.

solution: we need to encode $n$ characters. In the encoding tree
  at depth 0, we have root.
  at depth 1, we have 1 character
  at depth 2, we have 1 character
  ...
  at depth $n-1$, we have 2 character
Thus, we need $n-1$ bits to encode the least frequent character.