Sample Exam 3

(1) A bipartite graph $G = (V, E)$, is a graph such that V can be partitioned into two subsets V_1 and V_2 an no edge has both its vertices in the same subset. Prove that given a minimum spanning tree (mst) is a bipartite graph.

(2) There are a total of n courses you have to take, labeled from 0 to n-1.
Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]
Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses.
There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

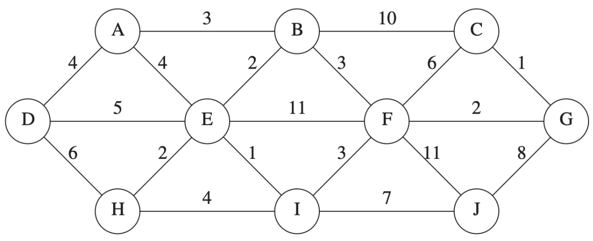
Example:
Input: 4, [[1,0],[2,0],[3,1],[3,2]]
Output: [0,1,2,3] or [0,2,1,3]

Given that we have a set of edges as an input what algorithm would you use to solve this in linear time $O(|E|)$?

(3) Given a list of non negative integers, arrange them such that they form the largest number. Now this is a relatively interesting sorting problem. Such that you have to sort the numbers such that the result is the largest number. Hence we have to create a custom compare function. Describe how you would compare two numbers x and y to solve this problem.

(4) Given the root of a m-ary tree, root, write an algorithm to determine the depth of the tree.

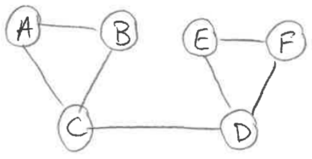
(5) Perform Kruskal's Algorithm on the following tree.



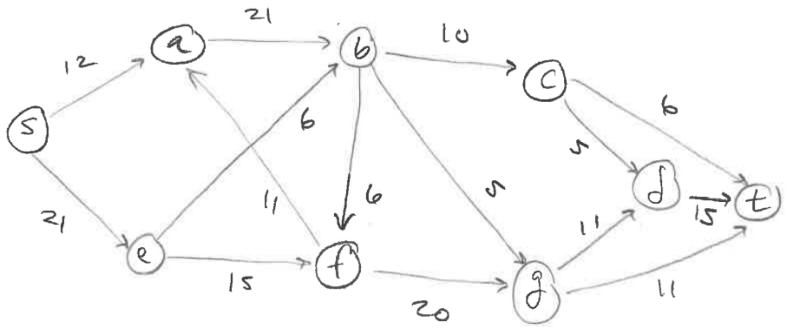
(6) What is the running time of the following code:

```
# x is a N X N array
# search runs in O(logn)
count = 0
for i in range(N):
    result = search(x[i], value)
    if result == value:
        count += 1
```

(7) Does the following graph have a euler circuit?



(8) Find the maximum flow of the following graph. List each augmenting path at each step the maximizes the flow. Draw the resulting residual and flow graphs.



(9) Given an unsorted array find the maximum distance between successive elements. Write an algorithm for this.

Solutions

(1) Since G is a MST that means we can treat it has a tree. Let us prove by induction that all trees of any size are bipartite.
Proof: By induction on the size of the tree (the number of nodes)
Basis: Start with a tree of size one. This tree contains only the root, this tree is trivially bipartite as we put the vertex in set partition while the other is empty.
Inductive Hypothesis: All trees of size n are bipartite.
Inductive Step: Lets take a tree of size $|T|$ and add a leaf, v, to increase the size by one. By the inductive hypothesis we know that $T - v$ is bipartite and since T is a tree and contains no cycles there is only one edge connecting v to a node u in T. Hence we can put v in the partition that u is not in. Hence a tree of size $|T| + v$ is still bipartite.
By induction we have proven that all trees are bipartite so any MST is also bipartite.

(2) We can use topological sorting since it runs in linear time.

(3) First what we will do is convert each number into a string. Because of lexicographical ordering ”9” > ”8” so the bigger integer is still greater. However we are not simply comparing if the two strings are greater. We have to order them such that the concatenation of the two numbers is the greatest possible number. We do this by writing a comparator as follows:

```
def comparator(x,y):
    xy = x + y
    yx = y + x
    if xy > yx:
        return 1
    elif xy == yx:
        return 0
    else:
        return -1
```

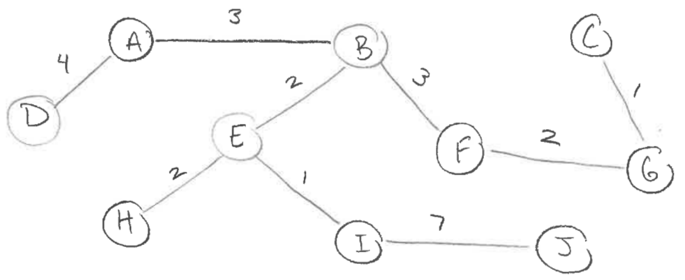
What this comparator does is given two numbers say ”89” and ”30” it determines what order of concatenating them results in the larger number. In this case ”8930” > ”3089” so ”89” should come before ”30” in the resulted sorted list.

(4) For this we would use depth-first search to search each path in the tree. This way we can find the maximum depth.

```
"""
Definition for a Node
class Node:
    def __init__(self, value, children):
        self.value = value
        self.children = children
"""

def max_depth(root):
    depth = 0
    for child in root.children:
        depth = max(depth, max_depth(child)
    return depth + 1
```

(5) After performing the algorithm we get the resulting graph. I will show each step during the review.

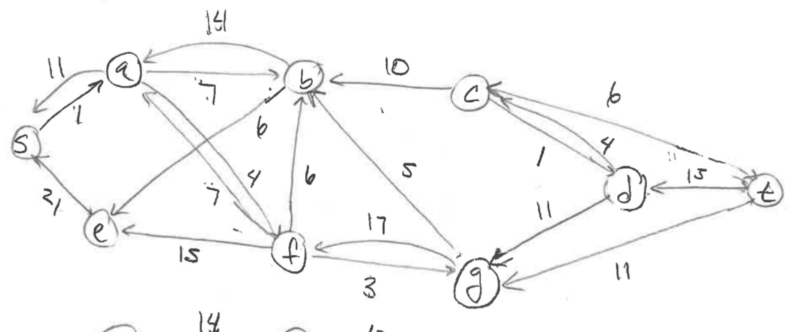


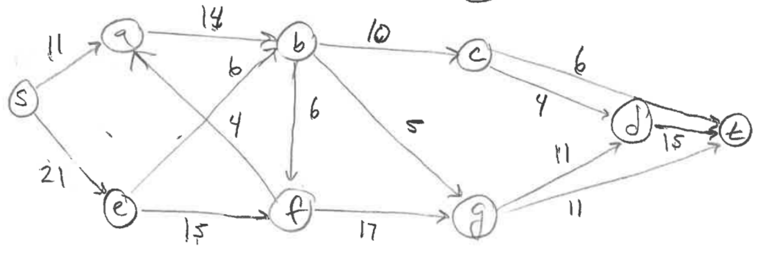
(6) The running time is $O(NlogN)$. Remember that in for loops it's the running time of the for loop times the running time of what happens inside the loop. So the for loop is $O(N)$ and since search is $O(logN)$ we do $O(N * logN) = O(NlogN)$

(7) Yes

(8) See the resulting graphs

s, e, f, g, t = 11
s, a, b, c, t = 6
s, e, b, f, g, d, t = 6
s, a, b, g, d, t = 5
s, e, f, a, b, c, d, t = 4





(9) In this case we simply do what the algorithm says we sort the array and find the maximal difference between successive elements:

```
def max_diff(arr):
    arr = sorted(arr)
    max_difference = 0
    for i in range(len(arr)-1):
        max_difference = max(max_difference, (arr[i+1]-arr[i]))

    return max_difference
```