Akshay Iyer-190070006
Minor
Hackerrank id- h190070006

Honour code-
I pledge on my honour and the Gita, that I have not used(except assignment 1) and will not use any unfair means( give or receive unauthorised assistance of any kind ) during this examination, or any other future examination or assignment in this course. I shared my code with a friend in assignment 1, which I THOROUGHLY regret, and which I sincerely beg your forgiveness for.

Q1

1.

I got everything right with my third algorithm.

In the first try, I successfully did everything right, but I created an additional 2D vector apart from the input, which has the changing values which we print at the end on the output window. Thus auxiliary space wasn't O(1), even though my program worked very well.

In my second try, I used O(1), but I had a time complexity of O(m^2*n). This was because to check if every column should be 0 or not, I searched above and below that element everytime.

In my third try, every time I found a 1 in a row or column, I made the first element of that row/ column 1. This removed the need to keep checking the elements of the same columns again and again every time we check for the values of outputs in the same column.
This caused some issues especially when it came to the element v[0][0], as it was the indicator for the top row, as well as the leftmost column, having all 1's. Hence just for that one indicator element, I created an additional variable k. One of them checked for columns, the other for rows having all 1's in the output.

(The history of all my tries is on hackerrank. )

2.

No better algorithm in terms of auxiliary space O(1). This is as we will have to define loop variables. We may also need to define some other temporary variables like my k. But at the end of the day as loop variables are definitely needed, we can't go below O(1).

Also our time complexity cannot go below O(m*n). This is as let us assume that we have a 1 at the bottom rightmost element. We will have to check it last. So in the worst case, we have to check all the elements before it, and the last element too. We see that it is 1, hence then we'll appropriately print the other 1's. Since, we have a total of m*n elements, and in the worst case we check all of them, our minimum time complexity is O(mn) only.

Q2

We will use the concept of depth first search in this question. Directed acyclic graph. Thus following only parent to children order, we cannot encounter the same node twice.

Create a vector A having [ak,ak-1,…….a1]

First we will start at any node and first find node a1. Then we will pop a1(the last element) from the vector, and keep searching ahead for a2 by carrying out dfs recursion only in the subtree which has a1 as its root.

Then we will do a depth first search for a2, starting at node a1. In every iteration of dfs, if we see that the child of that node is one out of a1,a3,….an except a2, then we do not do dfs of that child in the iteration/recursion.

If we find a2 as one of the children in the recursion, we only carry out the recursive dfs for that child only and not the others. We also pop a2 from the vector appropriately.

We do the same for a3, a4 etc.

Thus we keep going. If at some point our recursion is complete, (we have searched all the nodes avoiding the other a's as children appropriately) and we have not found ak, we have failed and there is no such path. If we do come across ak eventualy, we have succeeded.

Pseudo code-

```
Boolean find= False;     (This will become true only if we find all the elements of A in the desired
                          order, as then A will become an empty vector at the end)
dfs(int a)
{
        For I in children of a
        {
        if(I=A.last element)
        {
                A.pop_back;
                if(A.size()==0)
                      find=True;
                dfs(I);
        }
        else
        {
                If(I not in A)
                        Dfs A
                else
                        continue;     (goes to the next iteration of the for loop, and skips this child)

        }
}
```

As we see here, we are traversing every node only once in the absolute worst case, yet we are able to check if the order exists or not.
But we check if every node is in the vector A or not. Hence that takes O(k) to search, even though as I'm popping elements the size of the vector will keep decreasing.
Hence my time complexity is linear with O(k(n+m)).

We can store all the current A's in a new set/sorted vector A1 also. Thus searching if that element is in A will take O(log k) at worst.

Hence we can get a time complexity of O(log k(m+n)).

Q3

1.

In my algorithm, I carry out a dfs run-through of the graph, and every time I encounter a candy shop, I add the height of the tree to a global variable sum, which will give me the sum of distances from all candy shops.
The dfs function takes arguments as n(node that we consider as the head of the tree), and h(current height of the tree).
I do the dfs recursively inside the dfs function, and as we advance deeper, I send h+1 as the argument every time as the height increases ny 1.
Every time a candy shop is met, we add the distance of it form the head, into the total sum.

I used a for loop, and carried out the given dfs on every node once, treating it as the head, and hence finding the total distance of all of the candy shops from it. I then used mini and mining variables to find the minimum of all of these sums, and the head that will give the minimum sum.

I worked on this program for quite a while, but a runtime error occurs which I don't know how to fix, and the deadline is extremely close so I can't rectify it. But my logo cis definitely correct pls give me partial marks fro this question.

2.

The time complexity of my algorithm is O(n^2).

This is as, iterating through all the nodes of the graph once, considering one of the nodes as the head, takes O(n) time. This is as we need to go to every one of the n nodes once.

As we do n such iterations, considering each of the nodes once as the head of the tree, our time complexity gets multiplied by n. Hence our final time complexity is O(n^2).

3.

NO it is not necessary that she lives on a block with a candy shop. A counter example is given in the sample testcase itself, where 1,3,5 are the blocks with candy, but 2 is where she lives as it is at the minimum total distance from all 3.

Intuitively, if multiple candy shops are at the same distance from each other, say like they were on an equilateral triangle's vertices, or on the circumference off a circle, its makes sense that he point oo minimum distance will be at or very close to the center, and not at any one of the candy shops on the circumference. Hence it is not necessary as stated above.

Please give me marks for this question I'm sure its a very minor error, and this is the first assignment where I haven't got enough time before the deadline to complete everything as the past week has been extremely hectic for me even in terms of other courses and their deadlines.