Name: Sahasra Ranjan RollNo: 190050102

1. (i)

Statement: there is exactly one string s1 of length n in S such that f(s1) is undefined and exactly one string s2 of length n such that g(s2) is undefined.

By induction:

Base: f(1) = undefined, g(0) = undefined, f(01) is undefined, g(10)

Induction step:

for n-1, n-2 the statement is true (say)

f(s.01) is undefined if g(s) is undefined (only possible case for f() to be undefined) but g(s) is undefined only for a single input s of length n-2

similarly, g(s.0) is the only case when g is undefined when f(s) is undefined. But f() is undefined only for a unique value of s of length n-1. Therefore g() is undefined for a unique input of length n i.e. s.0

Conclusion:

The statement is true for n=0,1 and by induction, if the statement in true for n-1,n-2, it is true for n.

Hence for all n the statement is true

String can be represented as :

for f:

1s at 0, 3, 6, ... 3k positions
1,01,001,1001,01001

for g:

1s at 1, 4, 7, ... 3k+1 positions
0,10,010,0010,10010,010010

(position from right side)

(ii)

We will prove this by strong induction, inducting on the length of the string.
To prove: f(g(s)) = s [if g(s) is defined]& g(f(s)) = s [if f(s) is defined]

Base:

g(f(0)) = g(1) = 0
g(f(1)) (not defined)
f(g(0)) (not defined)
f(g(1)) = f(0) = 1
g(f(00)) = g(01) = 00
g(f(10)) = g(00) = 10
g(f(01)) = undefined
f(g(00)) = f(10) = 00
f(g(10)) = undefined
f(g(01)) = f(00) = 01

Induction Hypothesis:

for a k >= 2,
f(g(s)) = s (if g(s) is defined)
g(f(s)) = s (if f(s) is defined)

Induction Step:

for s of length k+1

if f(s) is defined:
If s = s'.0:

g(f(s)) = g(f(s'.0))
= g(g(s').0) if g(s') is defined
= f(g(s')).0
= s'.0 = s

else if s = s'.01
g(f(s) = g(f(s'.01))
= g(g(s').01) if g(s') is defined
= f(g(s')).01
= s'.01 = s

So, g(f(s)) = s

If g(s) is defined,
s = s'.0
f(g(s)) = f(g(s'.0))
= f(f(s').0) g(s) is defined so is f(s')
= g(f(s')).0
= s' by induction hypothesis.
= s'.0 = s

s = s'.01
f(g(s)) = f(g(s'.01))
= f(f(s').01) g(s) is defined so is f(s')
= g(f(s')).01
= s' by induction hypothesis.
= s'.0 = s

Therefore, f(g(s)) = s

Hence, proved.

2. (i)

Consider a graph with n vertices (1,2,..,n-1) and each constraint to be path. Weight of that path = C_ij. Now we can do a (directed) dfs/bfs to find if there exists a path b/w 0 and n-1. If yes then the sum is bounded else not bounded

Now to find the bound, we can find the minimum possible distance b/w 0 & n-1 using the shortest path algorithm. This distance will be the bound as max sum from 0 to n-1 must be less than all of the bounds. So max sum value will be the min of all the bounds.

Algo:

1. Keep dist[], with all inf
2. Create a topological order of all vertices
3. For every element in the ordering select element i:
   For all of its adj j:

dist[j] = min(dist[j], dist[u]+C_ij)

dist[n-1] will be the bound

(ii)
1. Build a new graph connecting I, I+1 with weight C_ij = M.
2. Do a DFS and reduce the C_ij to max sum from I to j (considering each max to be M)
3. We have n vertices and m+n edges. Now do the shortest path algo defined above to get the min path length from 0 to n-1. This will be the bound on sum from 0 to n-1.

4. (i)

To find a chordless odd cycle, we can do a dfs (linear) and mark all the nodes alternately (say red, blue)
Also keep a node stack to store the traversal.

If there is a path (backedge) u<-->v such that color(u) == color(v) then this is an odd cycle.

If they have a common neighbour, it is not a chordless odd cycle (cycle-3).

We have to delete both u and v one at a time. (linear). And for each time check if the remaining graph is bipartite(using the red black method mentioned before). If yes: nearly bipartite, else not nearly bipartite

If it is a chordless cycle:

It mean there is a backedge from u to v (both being same color)
Unstack the node traversal stack to get the nodes from v to u
This is our chordless odd cycle.

Delete these vertices and edges (linear)

Check if the graph left is bipartite.(linear, using the same dfs colouring algo) If no then it is "not" a nearly bipartite

if yes:

It is possible that a single node if common with many odd cycles, if this is the case we have to find and remove this node.

Since after deleting (marking it deleted to keep the edges intact) the nodes, we have a remaining graph. We already marked this graph (is bipartite). Now we can check the adj nodes where we got the edge with same color on both side. For both of the nodes, we have to check if all of its neighbour have same color or not.

Case1: for both the nodes, all of their neighbour have same same color, delete one of the nodes

Case2: Only one of them have neighbours with different color (delete that node)

Case3: both have neighbours with different colors (not nearly bipartite)

Also match these colors with the neighbours of the nodes in the odd cycle, it's trivial to return if nearly bipartite or not.

All of these step are standard dfs traversal or adj interation which can me completed in linear time i.e. O(n+m)

(ii) By smallest (no. of vertices).

Consider K_4 completely connected graph. If we delete any vertex, we will get an odd cycle

(iii)