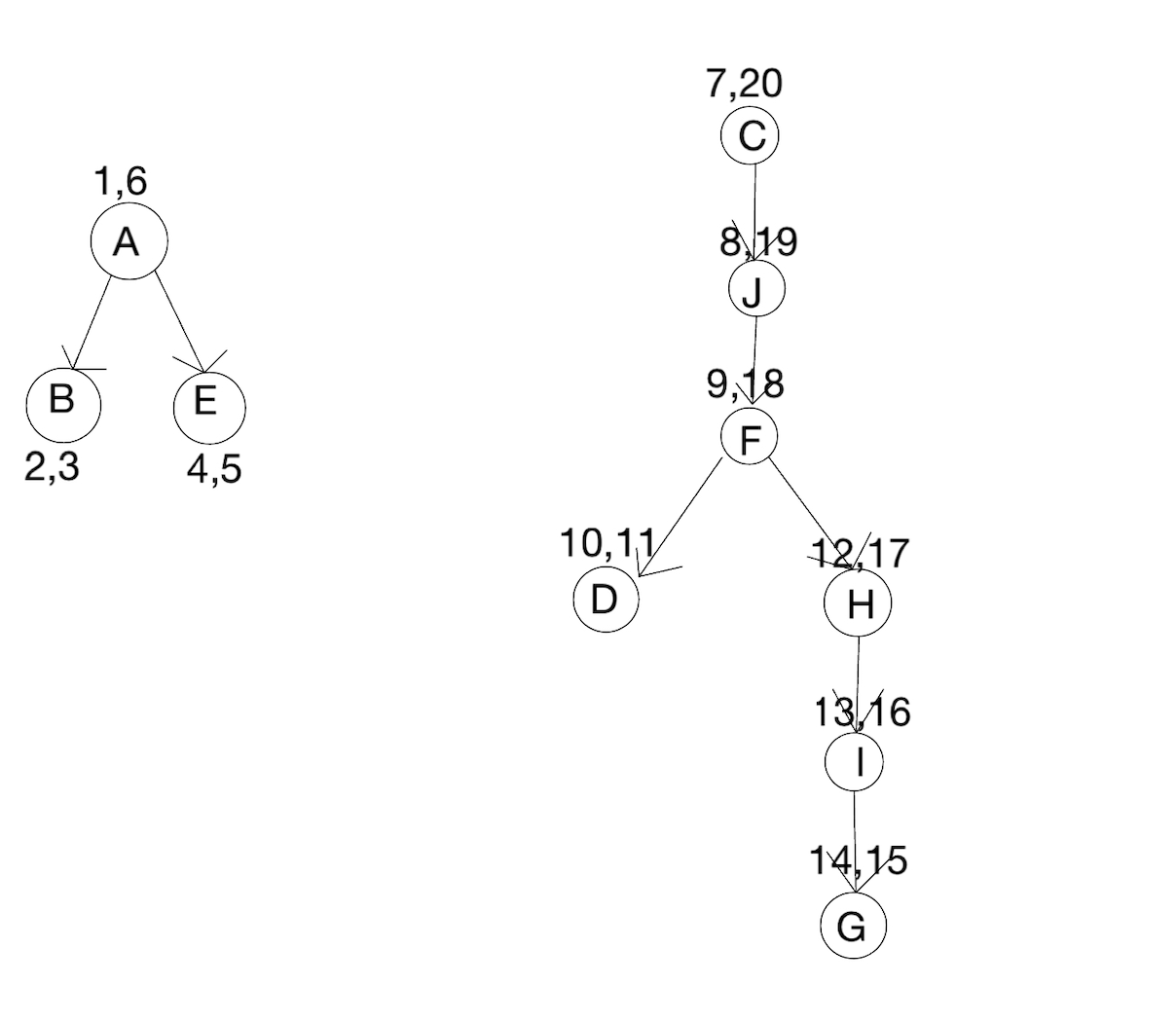
Problem Set 4

Hongrui(Ray) Liu

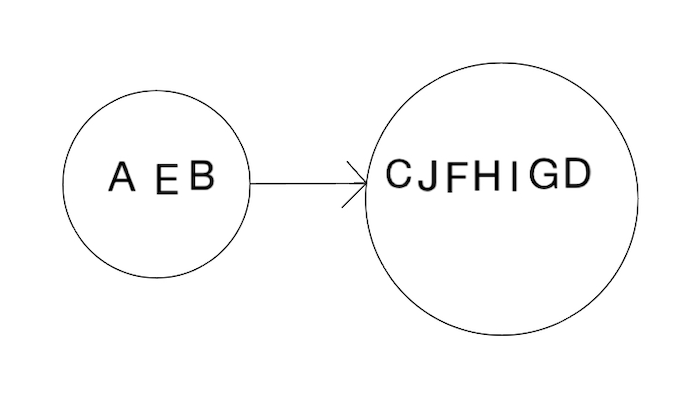
Oct 12, 2022

3.4.

1. According to the algorithm, we need to run the DFS on GR first, then we can have:



After the first DFS, the algorithm still requires a second DFS in the post visit number of the nodes: First, we start from C -> J ->F -> H -> I -> G ->D. Since the next ordered node is A and it’s not connected to any nodes in the previous strongly connected graph, it’s set to be another separate strongly connected graph. Afterwards, we start from A -> E ->B. Therefore, we can have 2 strongly connected graph : {CDJHIGD}, {AEB}



Since there are 2 strongly connected components in this graph, so we got 2 nodes in the metagraph which is shown in the figure.

1. Generally, we use the 2-coloring strategy to tell whether a graph is bipartite, and what occurred to me first is that every time when we want to paint the connected nodes, we can use a loop to traverse all the nodes connected but the time complexity is not linear. Therefore, we have to figure out another approach for this strategy.

In this case, DFS comes out first.

IsBipartite(G):

for all the v in V: //initialize all the nodes in the graph

set v as “unpainted”

//After initialization, we can start the DFS coloring search

for all the v in V:

if v is “unpainted”

{

if Paint(v, “blue”) == false

return false

}

//after the whole loop return true

return true

Paint(v, color):

v = color // paint the current node v with the color

for each edge connected to v( v, u) in E:

if( u is not “unpainted” && u.color != v.color) //this means the neighbor is colored

return false //the same, and we return false.

else

Paint(u, -color) //Paint the other color on this node

Explanation:

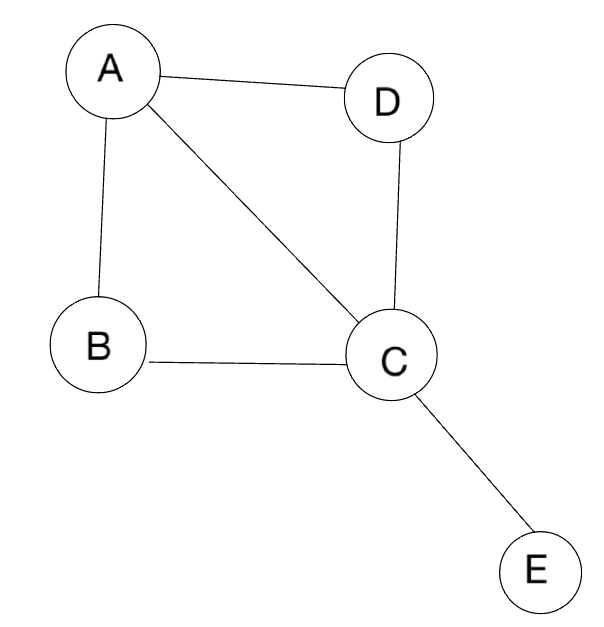
At first, we need to initialize all the nodes in the graph as “unpainted”, then we need to traverse all the “unpainted” nodes using DFS search in the for loop. If the color of any of the neighbors of the current traversed node is the same to the current traversed node, we return false. After the whole for loop, all the nodes should be traversed, and we return true since the color of all adjacent vertices is not the same which complies with the requirement for the bipartite graph.

The time complexity for this algorithm is : O(V+E) , since we are doing a DFS search with coloring strategy on this graph, and obviously, the running time for this algorithm is a linear one.

3.13

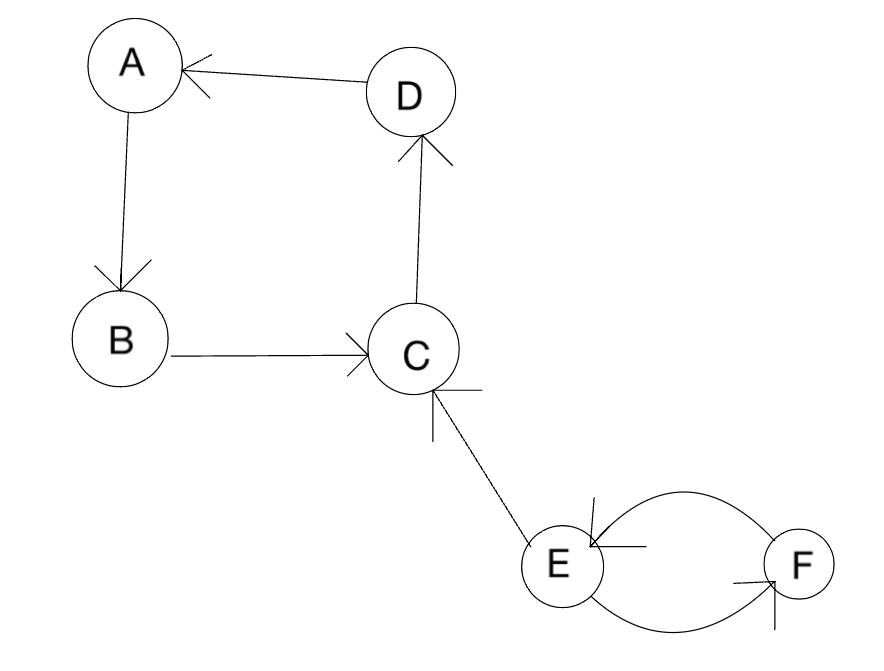
First, we need to run DFS on the graph, when we first traverse to the bottom of the graph, which means that the current node doesn’t have un-visited nodes. In this case, we can just remove this node from the graph, and it’s still connected.

For example:



So we run DFS on this graph with a source point on A, and we can have node E at the bottom of the tree. So we delete E from the graph and obviously it’s still connected.

If we want to make sure that adding one edge to the un-directed graph wouldn’t make the graph a strongly connected graph, we have to ensure that the two strongly connected components in the graph are disjointed, which means there are no edges between them. Next, if we add one edge between the two connected components, obviously it can not make the graph strongly connected since only nodes in one component can traverse all nodes in the graph. Here’s the example for it:



For this specific example, I added an edge between component{A,B,C,D} and {E,F}. In this case, the nodes in the component can reach all the nodes in the graph whole nodes in the component{A,B,C,D} can not reach the nodes in the other component. Therefore, at least 2 edges are needed to get the graph connected.