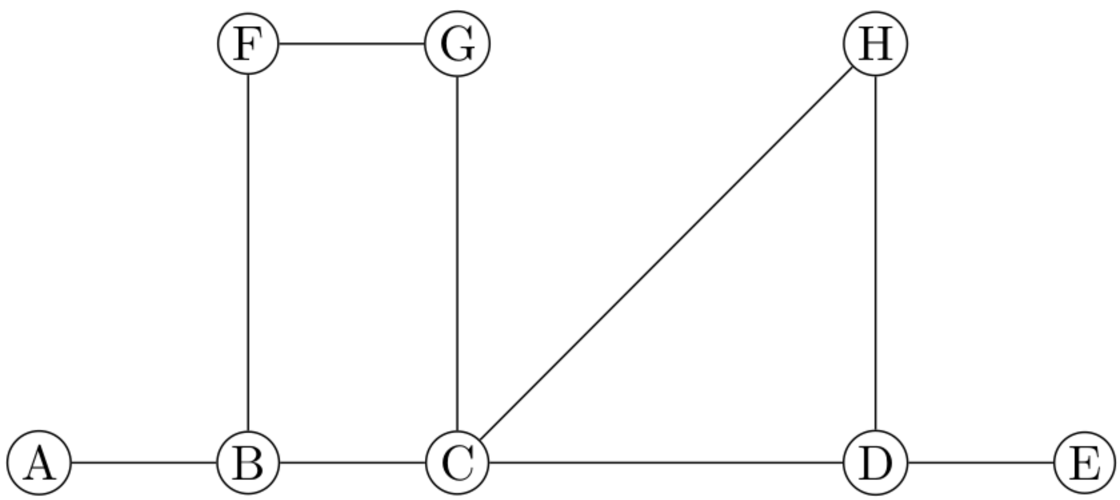
[CSL202] **2024-25-M**

TUTORIAL XII

Date: **Nov 22, 2024**.

1. Find the cut-edges in the following graph using the algorithm we have studied during the lectures. Recall that for each vertex u, we have stored a value $h[u]$, which helps to find cut-edges. Find $h[u]$ for every vertex u of G. Based on h values find the cut edges.



2. Find the cut-vertices in the above graph using the algorithm we have studied during the lectures. Find the h value for every vertex.

We have studied a linear time algorithm (Kosaraju's) to compute SCC's in a directed graph. Today we will learn a new (Tarjan's) algorithm to compute SCC's in a directed graph.

3. TRUE/FALSE: The number of strongly connected components in a directed graph is equal to the number of trees in a DFS forest.

Tarjan's algorithm

We store the following information at each node.

1. Discovery Time: Each vertex is assigned a discovery time when it is first visited in the DFS.
2. Low-Link Value: The smallest discovery time reachable from a vertex, including itself and its descendants. We use low-link[v] to store the value.
3. Stack: A stack is used to track the current path in the DFS. Vertices in the same SCC remain on the stack together.

Outline of the algorithm

1. For each unvisited vertex, perform a DFS
2. Assign a discovery time and low-link value to the vertex.
3. Push the vertex onto the stack and mark it as "onStack".
4. For each neighbor
   (a) If the neighbor is unvisited, recursively call the DFS on it.
   (b) Update the low-link value of the current vertex based on the neighbor's low-link value.
   (c) If the neighbor is on the stack, update the current vertex's low-link value to the minimum of its low-link value and the neighbor's low-link value.
5. After processing all neighbors of a vertex:
   (a) If the vertex's low-link value equals its discovery time, it is the root of an SCC.
   (b) Pop all vertices from the stack until the root vertex is reached, forming one SCC.