Lecture on

Biconnected Component

Articulation Point

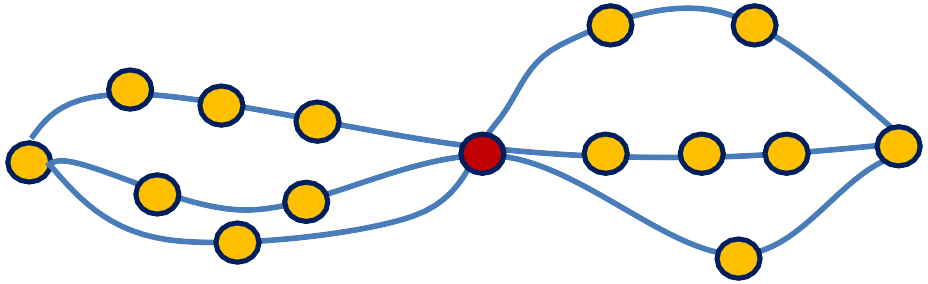
Articulation Point

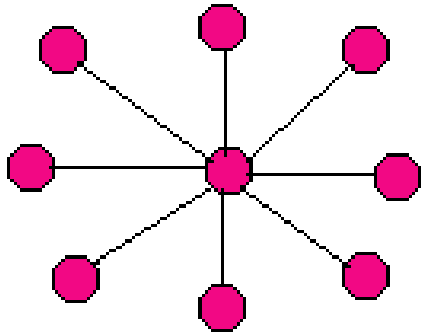
- **Definition:** A vertex ***x*** is said to be **articulation point** if ∃ ***u***,***v*** different from ***x*** such that every path between ***u*** and **v** passes through ***x***.

- A vertex in an undirected connected graph is an articulation point (cut vertex) if removing it (an edges through it) disconnects the graph.

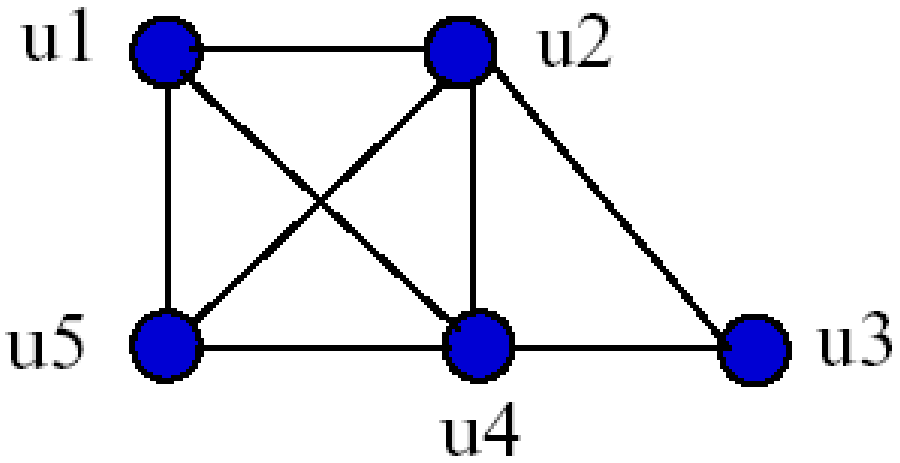
- Articulation point represents vulnerabilities in a connected network-single points whose failure would split the network into two or more disconnected components.

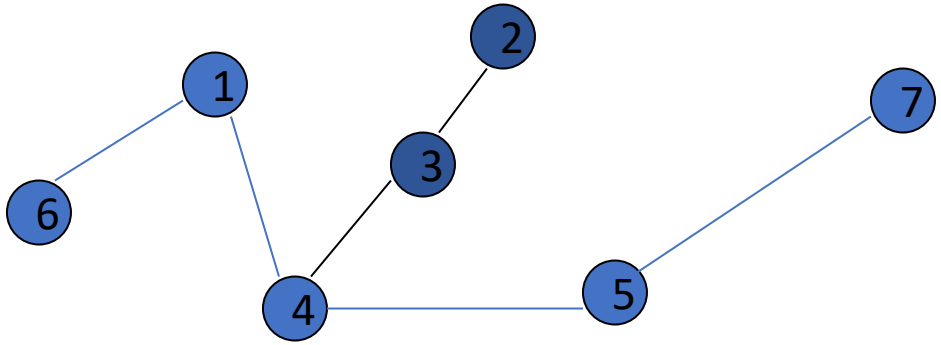
Articulation Point





$K_{1,8}$





Problem Statement

Observation: A graph is biconnected if none of its vertices is an articulation point.

AIM:

Design an **algorithm** to compute all **articulation points** in a given graph.

Simple Algorithm

A simple Approach is to one by one remove all vertices and see if removal of vertex cause disconnected graph .

1) **For every vertex v do following:**
   - **Take out the node and all passing edges**
   - **Find if we have only one component(using DFS)**
   - **If components =1 then node is not AP else node is AP**
2) **Print All APs**

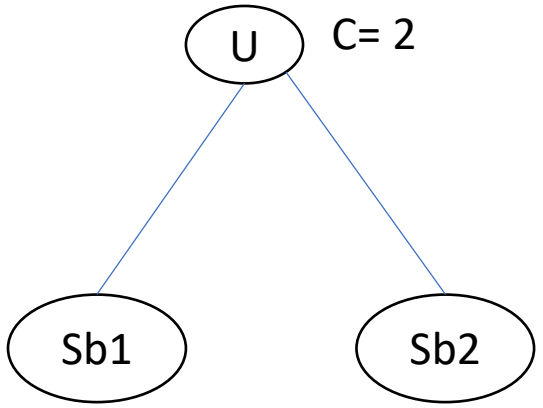
Simple Algorithm

Time complexity O(V* (V+E)) for a graph represented using the Adjacency List

How can we do better?

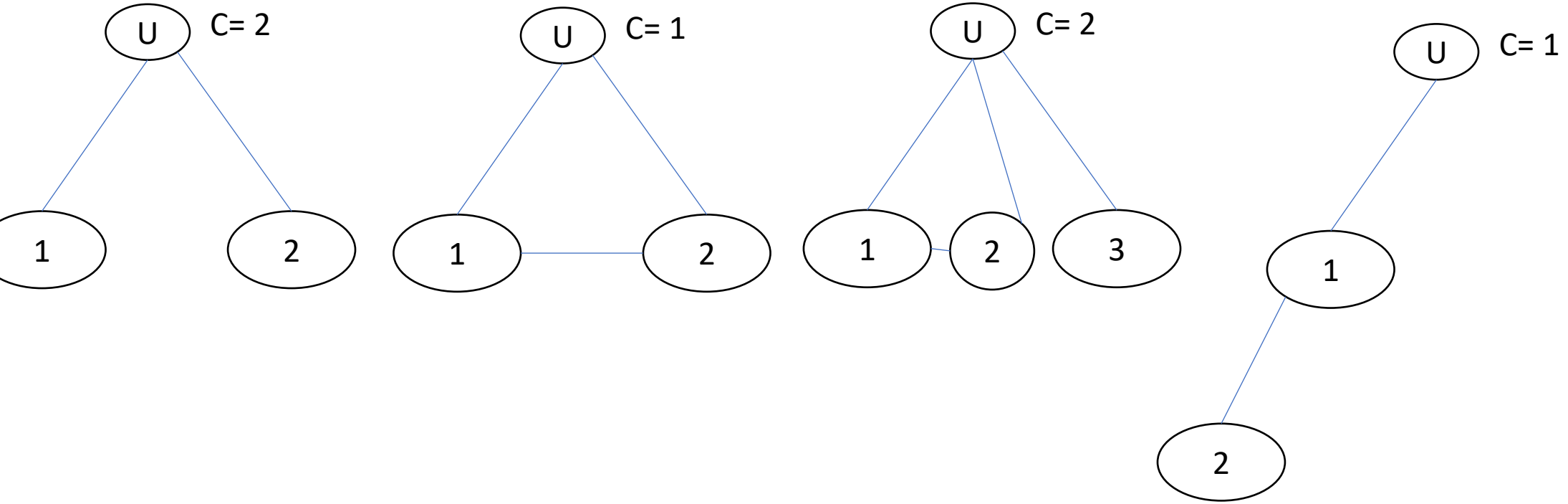
Conditions of an AP

Case 1: If node U is root of DFS tree and has at least 2 Children (Subgraph) then Node U is an Articulation Point(AP)



Conditions of an AP

Case 1: If node U is root of DFS tree and has at least 2 Children (Subgraph) then Node U is an Articulation Point(AP)



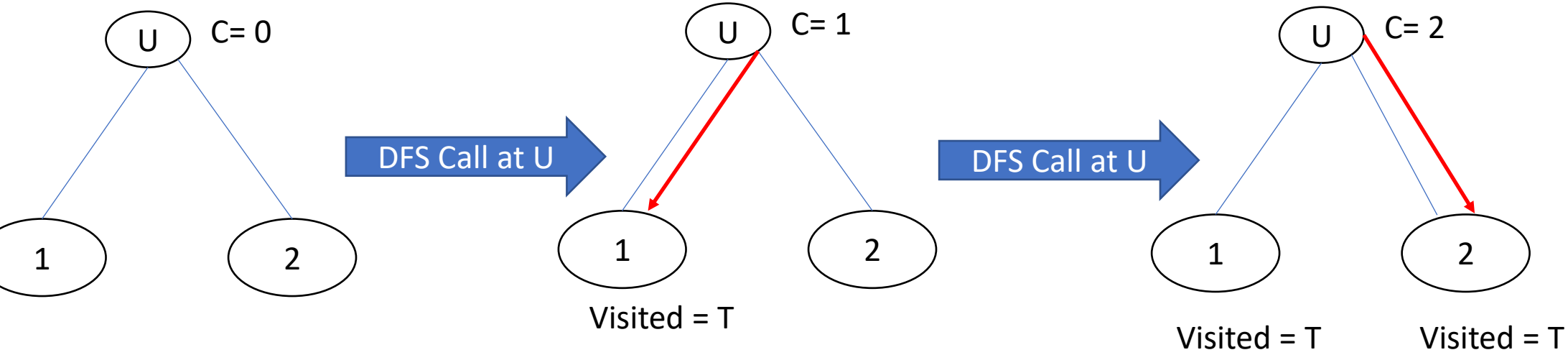
How to detect AP for Case 1

Case 1: If node U is root of DFS tree and has at least 2 Children (Subgraph) then Node U is an Articulation Point(AP)

1. **Keep children counter for each node**
2. **On calling DFS on each edge, mark the entire Subgraph**
3. **Call DFS on other edges only if node is not visited**
4. **Increment children count on each unique DFS call**
5. **If children (c) >1, then the node is AP**

Conditions of an AP

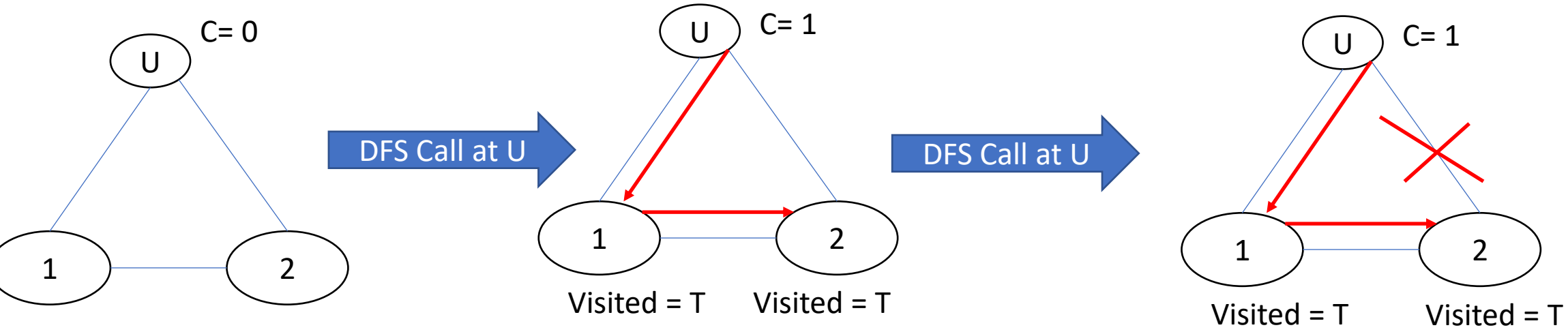
Case 1: If node U is root of DFS tree and has at least 2 Children (Subgraph) then Node U is an Articulation Point(AP)



Therefore U is AP

Conditions of an AP

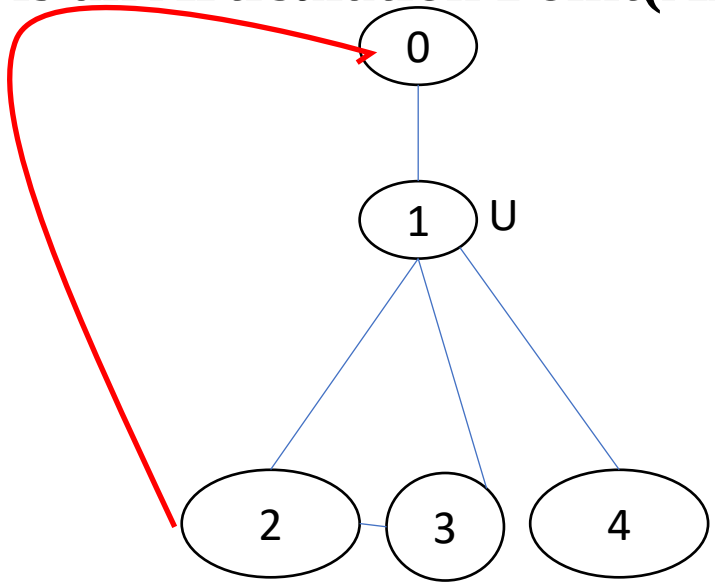
Case 1: If node U is root of DFS tree and has at least 2 Children (Subgraph) then Node U is an Articulation Point(AP)



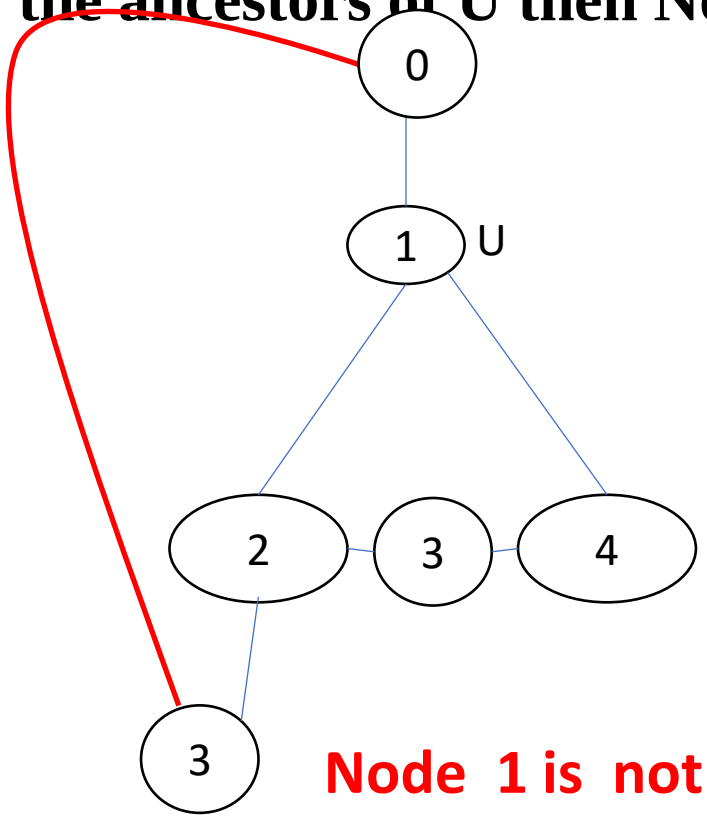
Therefore U is not an AP

Conditions of an AP

Case 2: If node U is not root of DFS tree and it has a child V such that no vertex in subtree rooted with V has back edge to one of the ancestors of U then Node U is an Articulation Point(AP)



Node 1 is AP



Node 1 is not AP

How to detect AP for Case 2

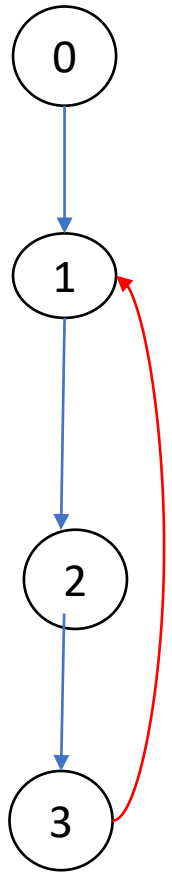
1. **We need to find the order of vertices from earliest to latest to detect back-edges.**

   We use timestamp to mark nodes with traversing values. We can do this by assigning discovery values.
2. **We need to maintain the earliest possible node accessible for a given node which will indicate if we have any back edge. For that we will assign low value for each node.**

Low[v] = min{discover[v], discover[w] : (u, w) is a back edge for some descendant u of v}

Background

Discovery Values and Low Values using DFS



We will use (discovery value , low value) using DFS

0	1	2	3
0	1	2	3

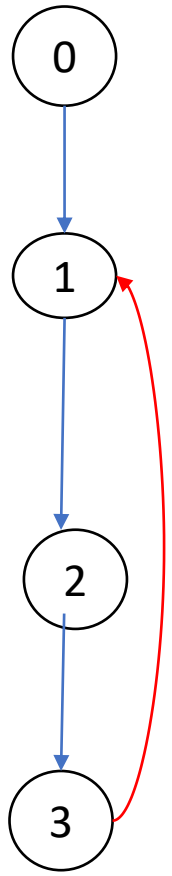
Discovery Time

0	1	2	3
0	1	2	3

Low Time = Lowest discovery node accessible

Background

Discovery Values and Low Values using DFS



We will use (discovery value , low value) using DFS

0	1	2	3
0	1	2	3

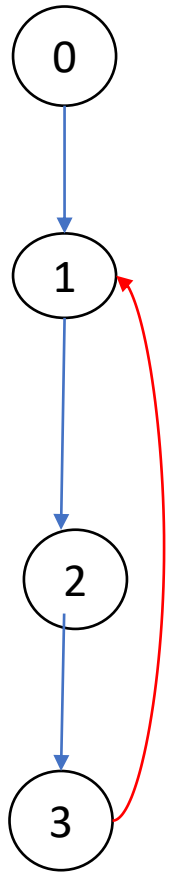
Discovery Time

0	1	2	1
0	1	2	3

Low Time = Lowest discovery node accessible

Background

Discovery Values and Low Values using DFS



We will use (discovery value , low value) using DFS

0	1	2	3
0	1	2	3

Discovery Time

0	1	1	1
0	1	2	3

Low Time = Lowest discovery node accessible

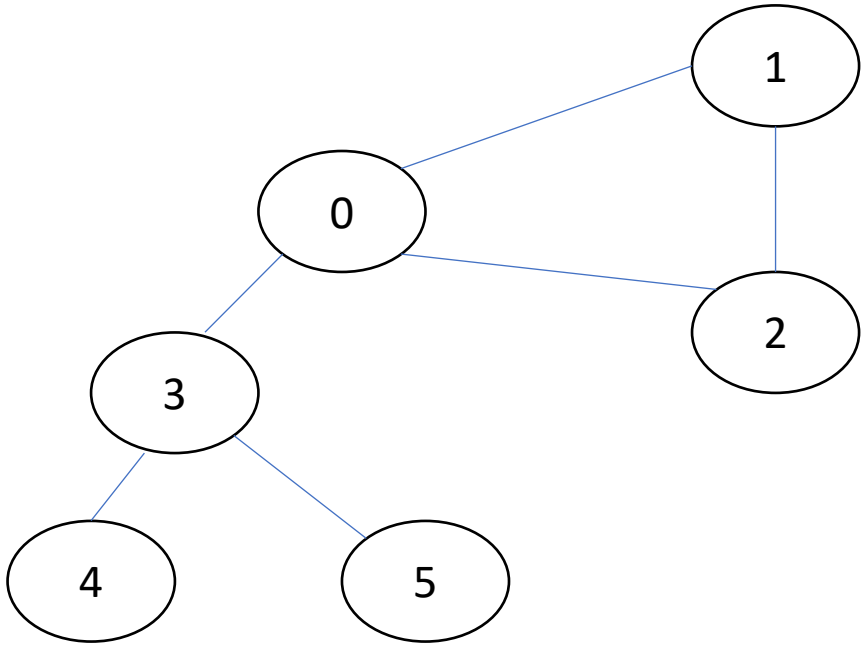
How to detect AP for Case 2

1. **We need to find the order of vertices from earliest to latest to detect back-edges.**

   We use timestamp to mark nodes with traversing values. We can do this by assigning discovery values.
2. **We need to maintain the earliest possible node accessible for a given node which will indicate if we have any back edge. For that we will assign low value for each node.**

Low[v] = min{discover[v], discover[w] : (u, w) is a back edge for some descendant u of v}

Algorithm on Example



DT

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

LT

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

Parent

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

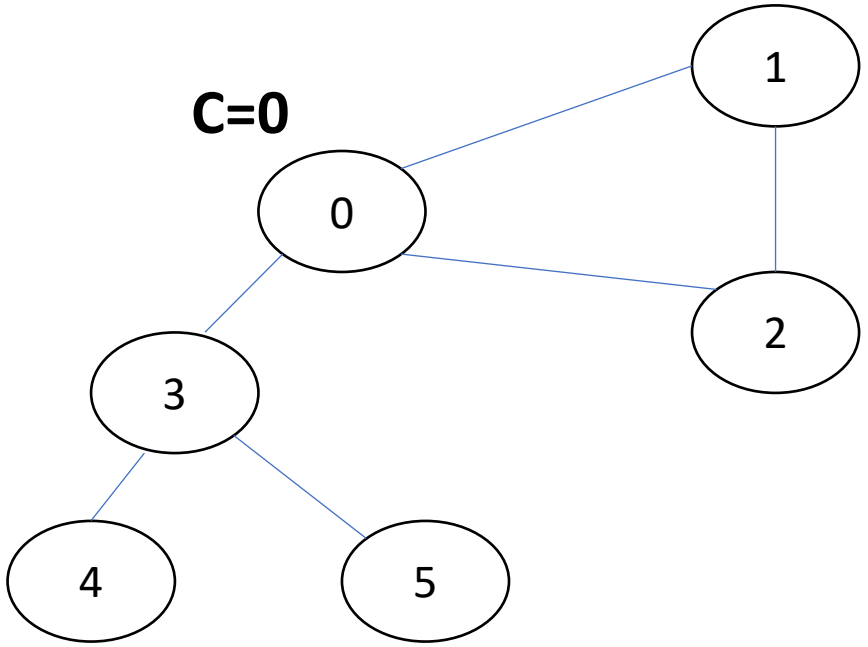
Let us assume Time = 0 and starting vertex = 0

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =0



DT

0	-1	-1	-1	-1	-1
0	1	2	3	4	5

LT

0	-1	-1	-1	-1	-1
0	1	2	3	4	5

Parent

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

Start of Child count (c) =0

DT for node 0 is as value of time and

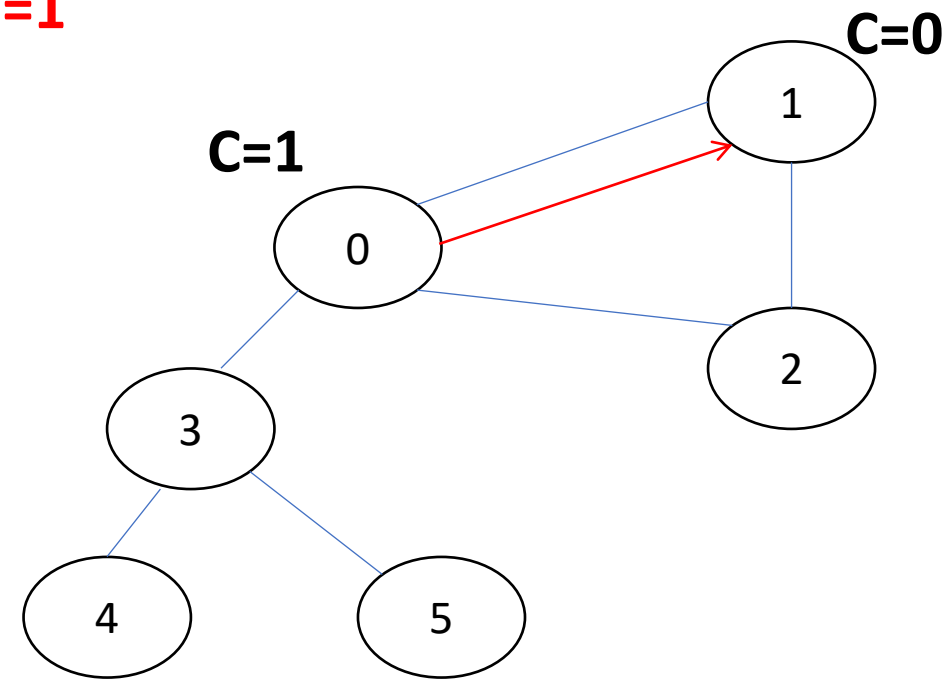
LT for node 0 is as value of time

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =1



DT

0	1	-1	-1	-1	-1
0	1	2	3	4	5

LT

0	1	-1	-1	-1	-1
0	1	2	3	4	5

Parent

-1	0	-1	-1	-1	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

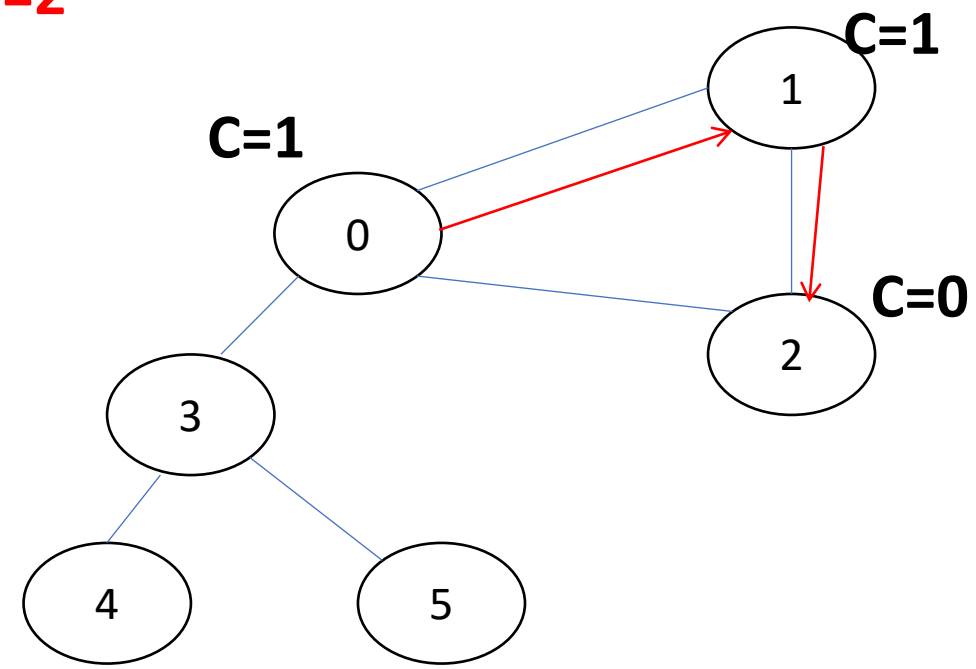
DT for node 1 = 1

And LT for node 1 = 1

Parent of Node 1 = 0

Algorithm on Example

Time =2



DT for node 2 = 2

And LT for node 2 = 2

Parent of Node 2 = 1

DT

0	1	2	-1	-1	-1
0	1	2	3	4	5

LT

0	1	2	-1	-1	-1
0	1	2	3	4	5

Parent

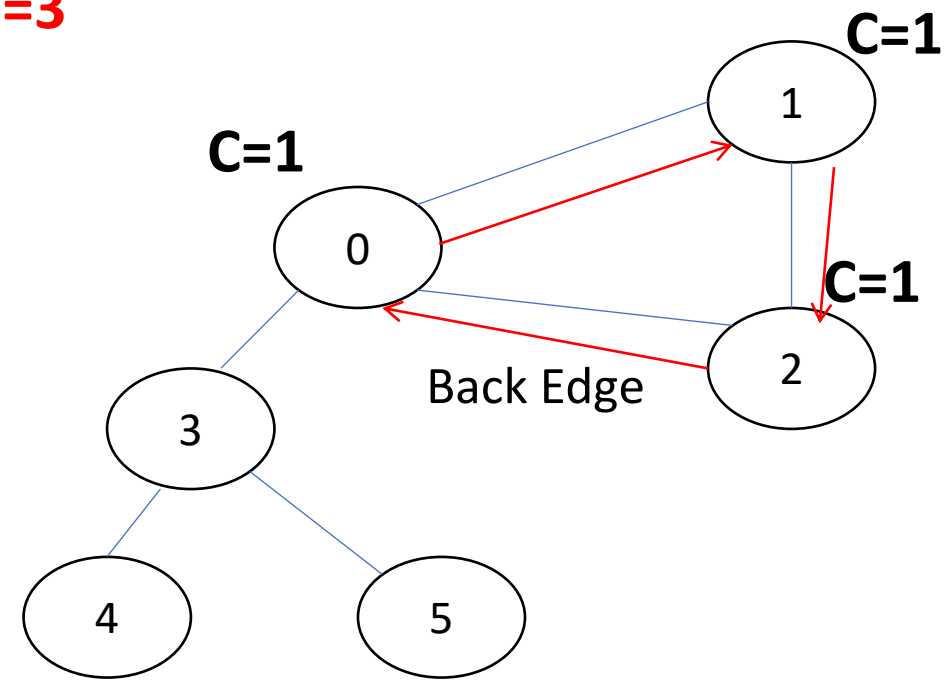
-1	0	1	-1	-1	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =3



DT

0	1	2	-1	-1	-1
0	1	2	3	4	5

LT

0	1	0	-1	-1	-1
0	1	2	3	4	5

Parent

-1	0	1	-1	-1	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

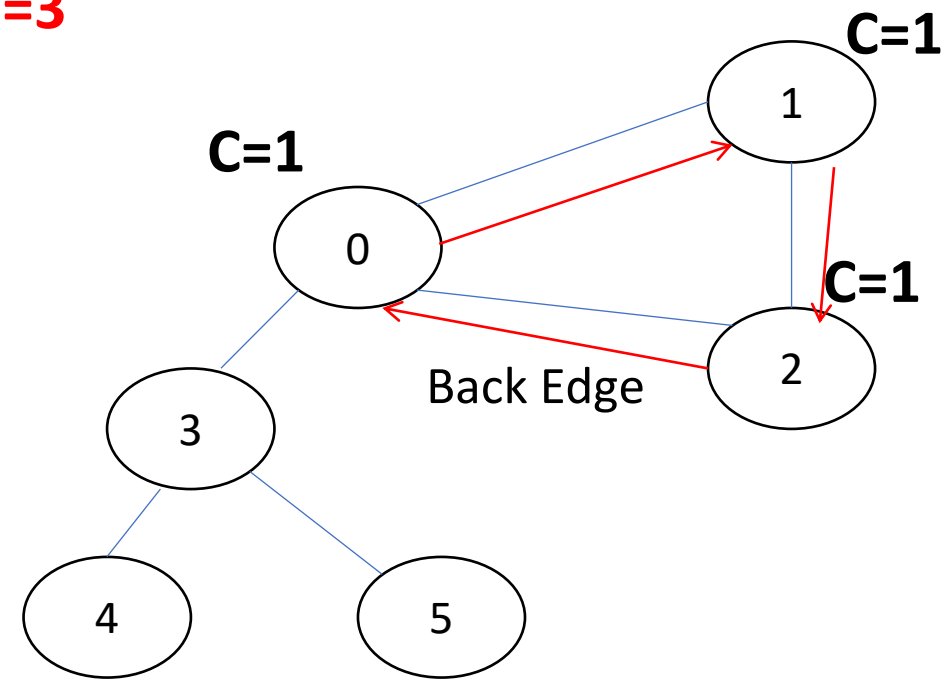
2 to 0 is already visited (Back edge)

LT of 2 updated

2 is not AP since no subgraph of 2

Algorithm on Example

Time =3



Backtrack of 2 to 1

Update LT of 1

DT

0	1	2	-1	-1	-1
0	1	2	3	4	5

LT

0	0	0	-1	-1	-1
0	1	2	3	4	5

Parent

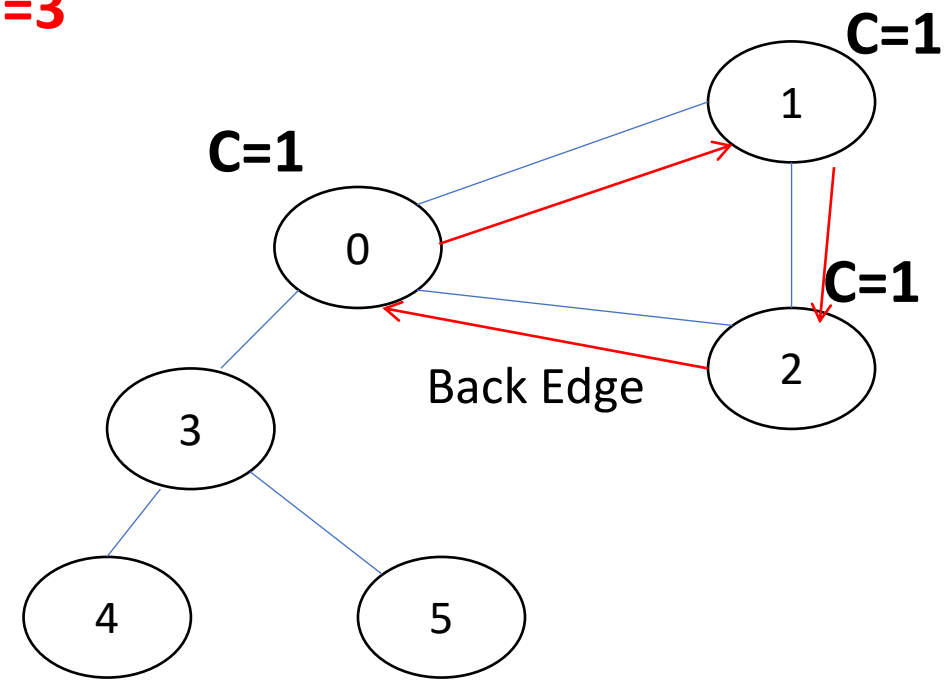
-1	0	1	-1	-1	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =3



DT

0	1	2	-1	-1	-1
0	1	2	3	4	5

LT

0	0	0	-1	-1	-1
0	1	2	3	4	5

Parent

-1	0	1	-1	-1	-1
0	1	2	3	4	5

Backtrack of 2 to 1

Update LT of 1

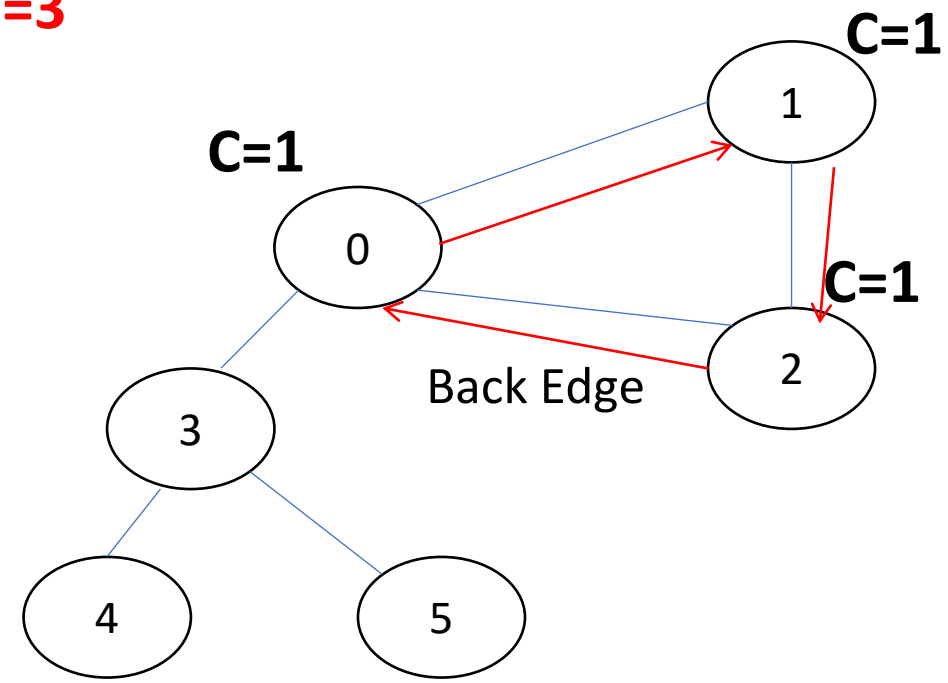
1 is not AP since only 1 subgraph of 1 having back edge

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =3



DT

0	1	2	-1	-1	-1
0	1	2	3	4	5

LT

0	0	0	-1	-1	-1
0	1	2	3	4	5

Parent

-1	0	1	-1	-1	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

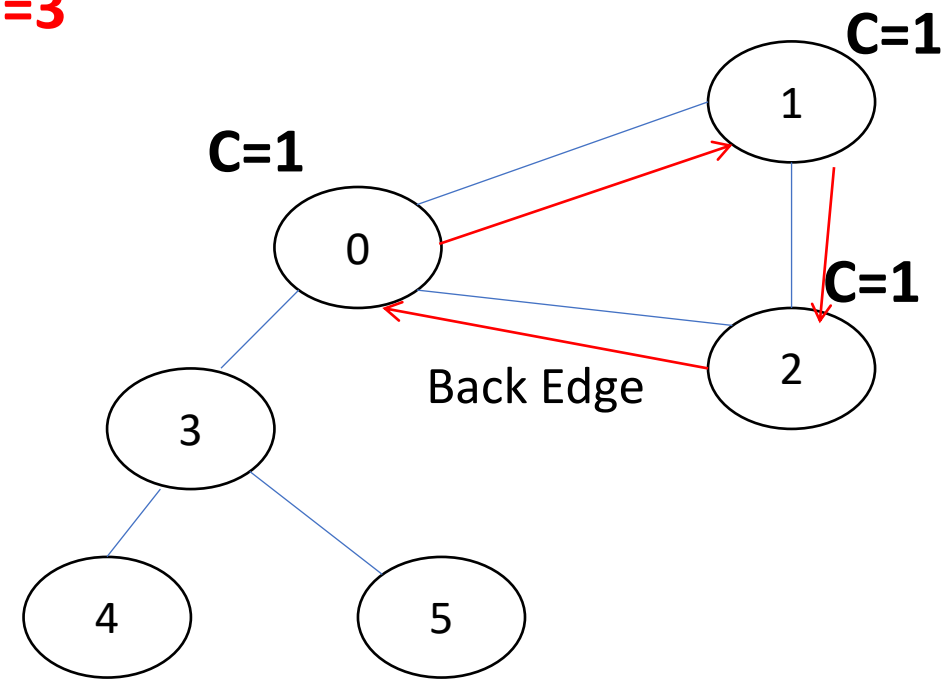
Backtrack of 1 to 0

Update LT of 0

0 is root node (parent of 0 is -1) and Child count (c) = 1 so till now this is not AP

Algorithm on Example

Time =3



0 to 2 already visited

DT

0	1	2	-1	-1	-1
0	1	2	3	4	5

LT

0	0	0	-1	-1	-1
0	1	2	3	4	5

Parent

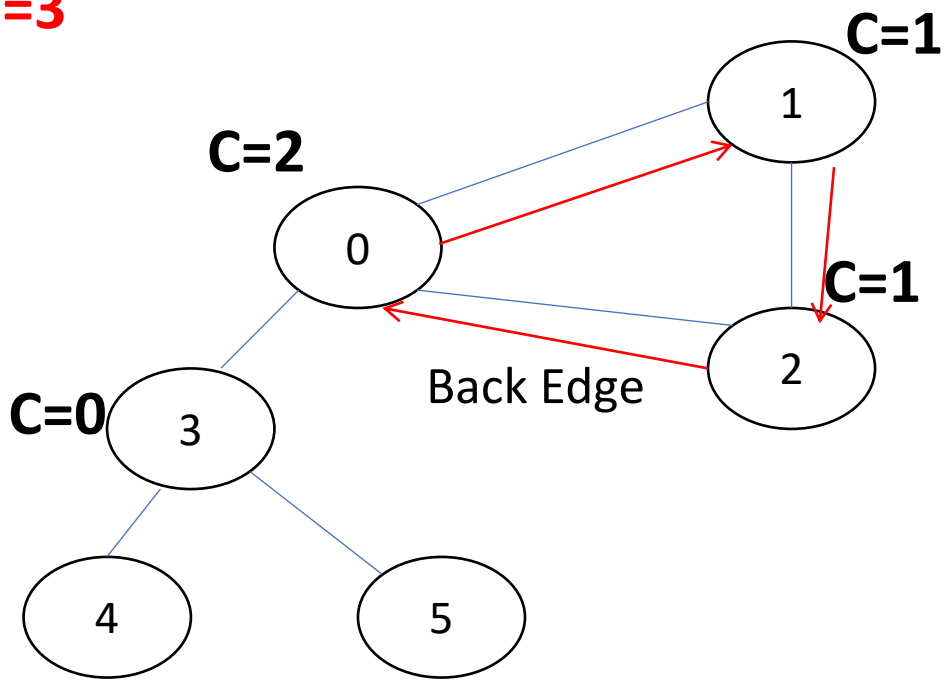
-1	0	1	-1	-1	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =3



DT

0	1	2	3	-1	-1
0	1	2	3	4	5

LT

0	0	0	3	-1	-1
0	1	2	3	4	5

Parent

-1	0	1	0	-1	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

Now, for 0 to 3:

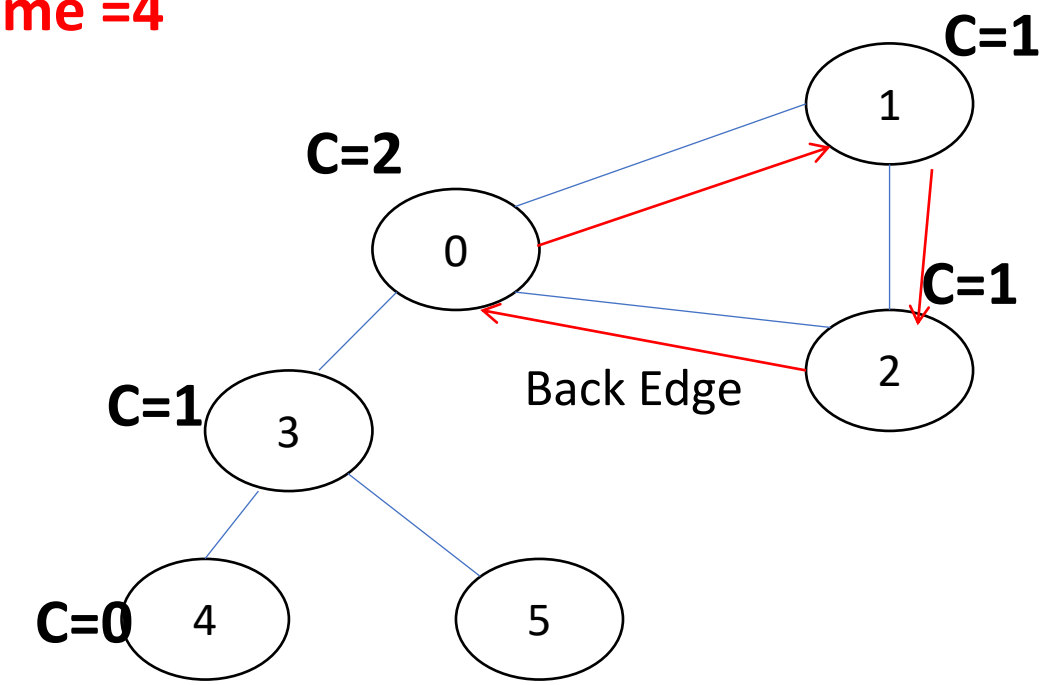
DT = 3

LT =3

Parent =0

Algorithm on Example

Time =4



Now, for 3 to 4:

DT = 4

LT =4

Parent =3

DT

0	1	2	3	4	-1
0	1	2	3	4	5

LT

0	0	0	3	4	-1
0	1	2	3	4	5

Parent

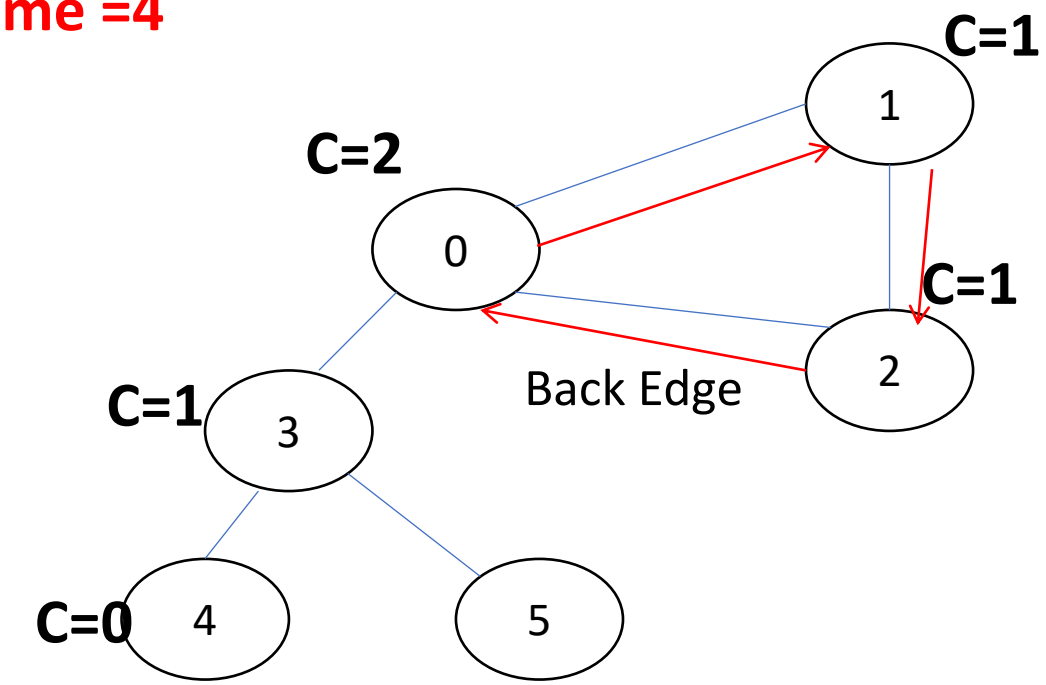
-1	0	1	0	3	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =4



Now, for 3 to 4:

DT = 4

LT =4

Parent =3

DT

0	1	2	3	4	-1
0	1	2	3	4	5

LT

0	0	0	3	4	-1
0	1	2	3	4	5

Parent

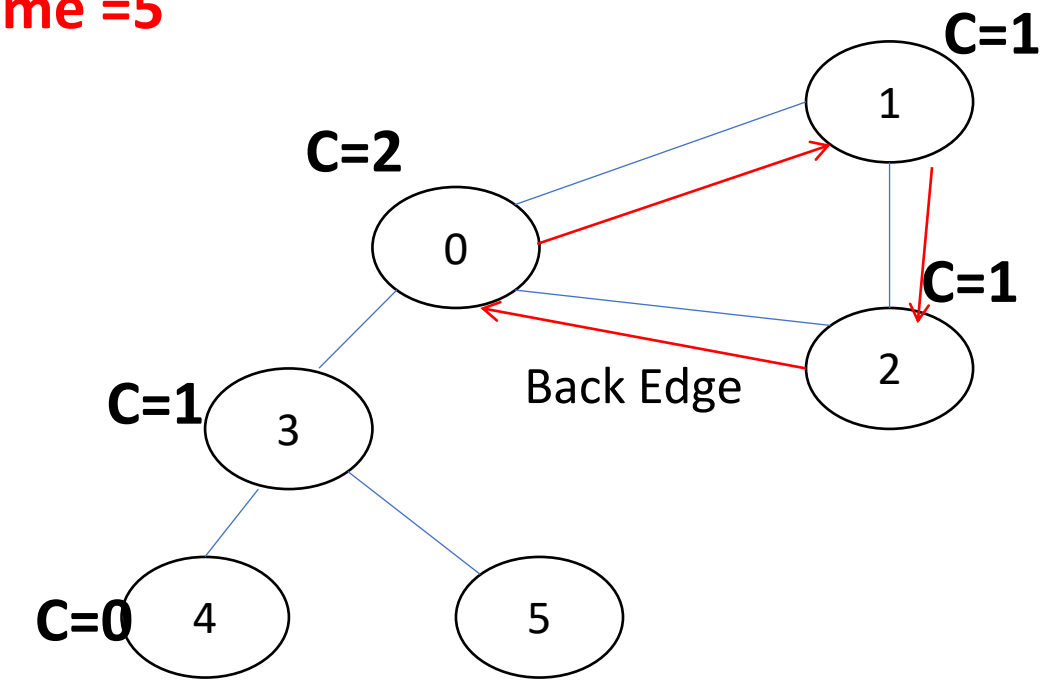
-1	0	1	0	3	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =5



4 is not an AP

Backtrack of 4 to 3

DT

0	1	2	3	4	-1
0	1	2	3	4	5

LT

0	0	0	3	4	-1
0	1	2	3	4	5

Parent

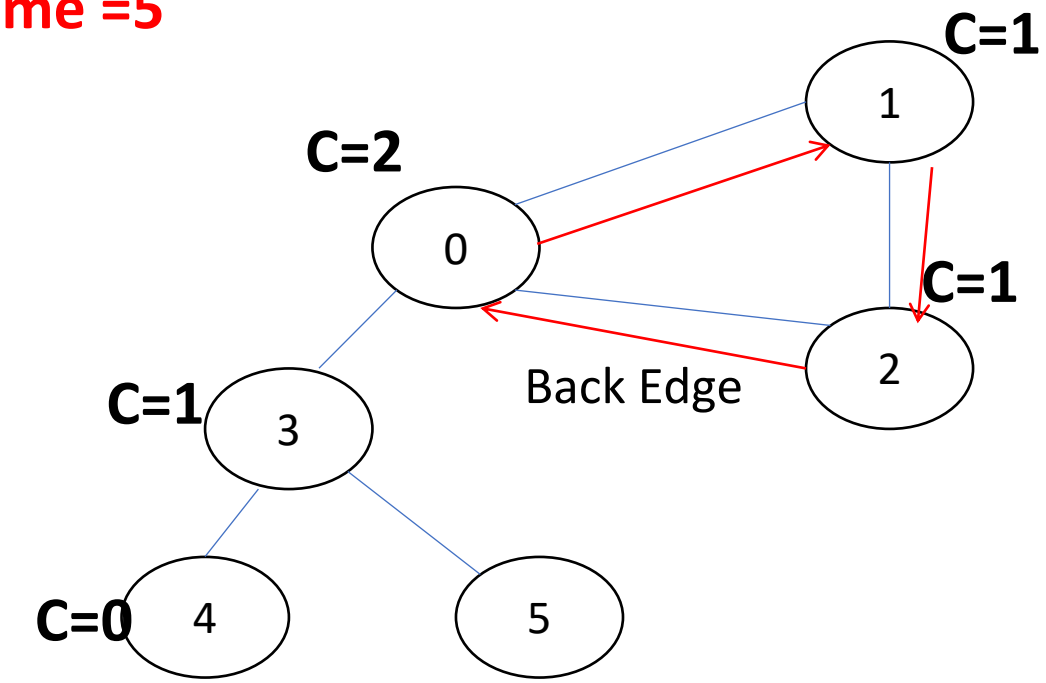
-1	0	1	0	3	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =5



4 is not an AP

Backtrack of 4 to 3

DT

0	1	2	3	4	-1
0	1	2	3	4	5

LT

0	0	0	3	4	-1
0	1	2	3	4	5

Parent

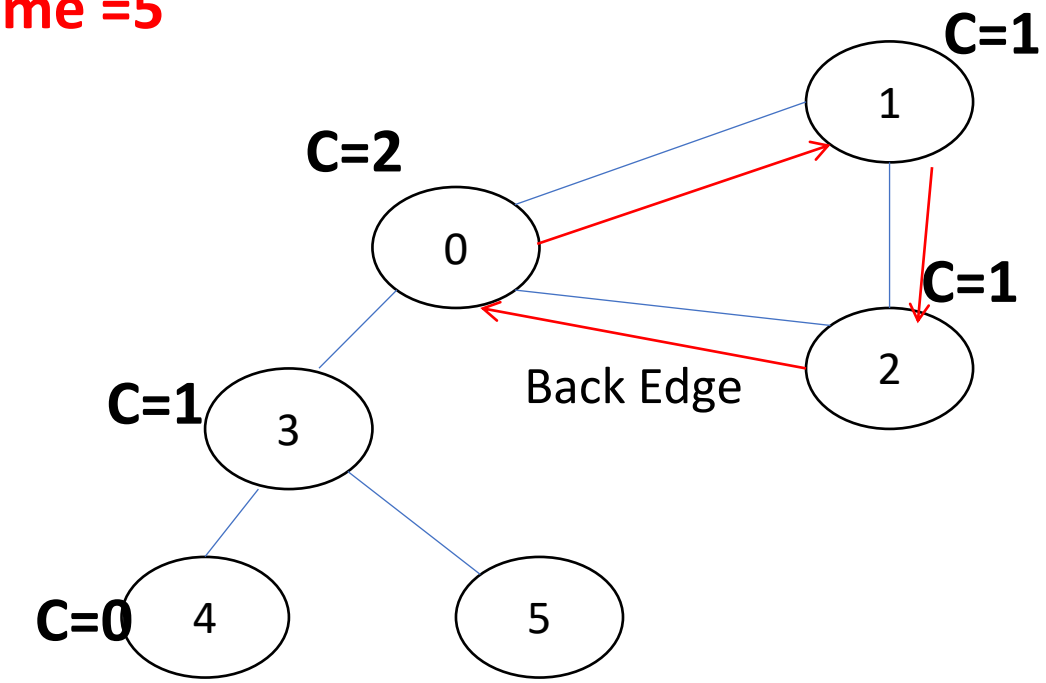
-1	0	1	0	3	-1
0	1	2	3	4	5

AP

F	F	F	F	F	F
0	1	2	3	4	5

Algorithm on Example

Time =5



DT

0	1	2	3	4	-1
0	1	2	3	4	5

LT

0	0	0	3	4	-1
0	1	2	3	4	5

Parent

-1	0	1	0	3	-1
0	1	2	3	4	5

AP

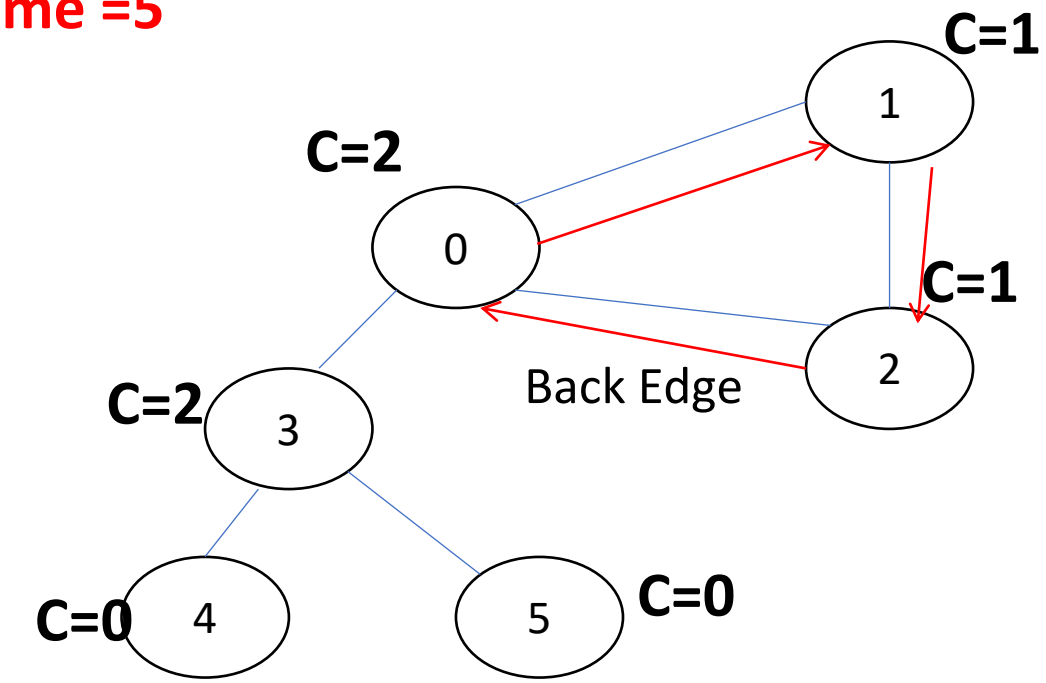
F	F	F	T	F	F
0	1	2	3	4	5

Update the low value of 3 = 3

Now 3 is AP since discovery time of 3 is less than equal to low value of 4

Algorithm on Example

Time =5



DT =5

LT =5

Parent =3

DT

0	1	2	3	4	5
0	1	2	3	4	5

LT

0	0	0	3	4	5
0	1	2	3	4	5

Parent

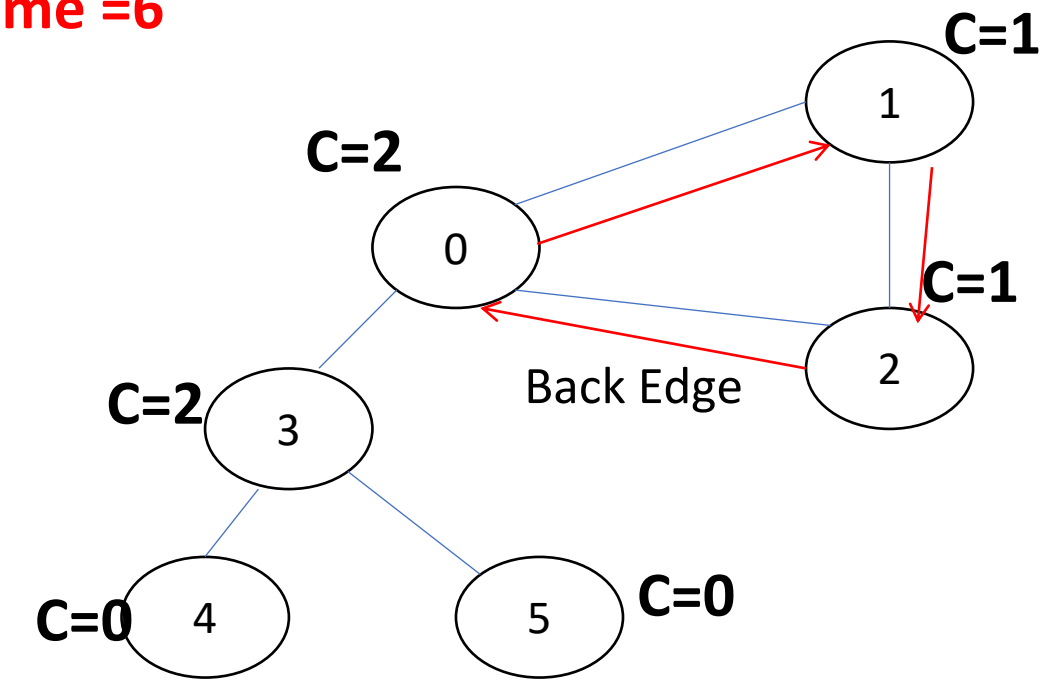
-1	0	1	0	3	3
0	1	2	3	4	5

AP

F	F	F	T	F	F
0	1	2	3	4	5

Algorithm on Example

Time =6



DT =5

LT =5

Parent =3

DT

0	1	2	3	4	5
0	1	2	3	4	5

LT

0	0	0	3	4	5
0	1	2	3	4	5

Parent

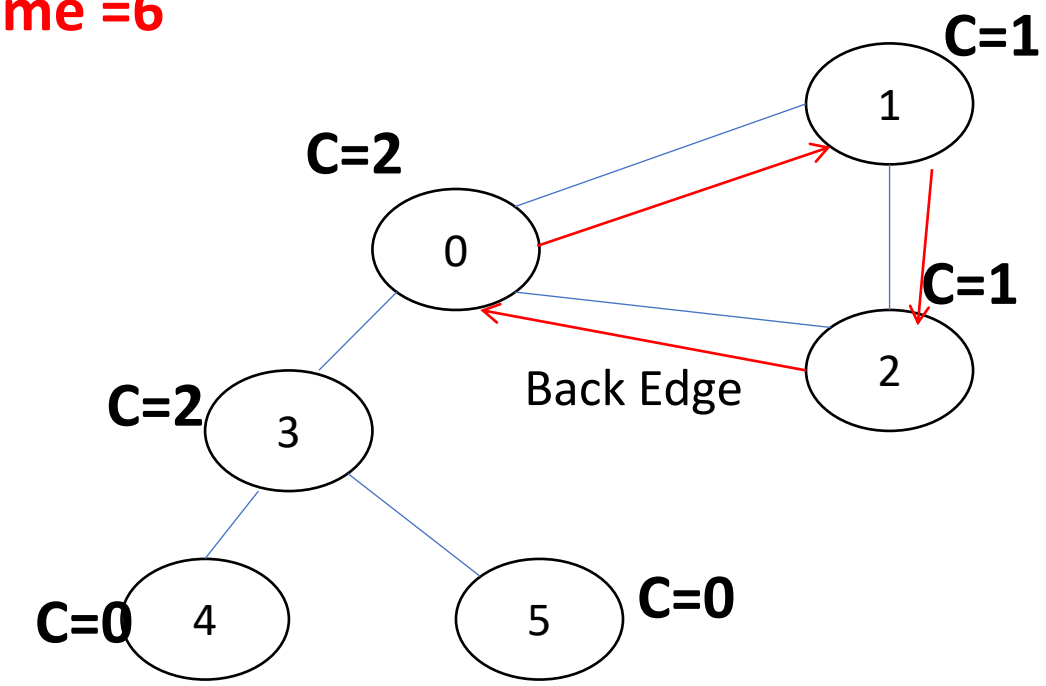
-1	0	1	0	3	3
0	1	2	3	4	5

AP

F	F	F	T	F	F
0	1	2	3	4	5

Algorithm on Example

Time =6



Backtrack to 3

Update LT of 3: No Update

DT

0	1	2	3	4	5
0	1	2	3	4	5

LT

0	0	0	3	4	5
0	1	2	3	4	5

Parent

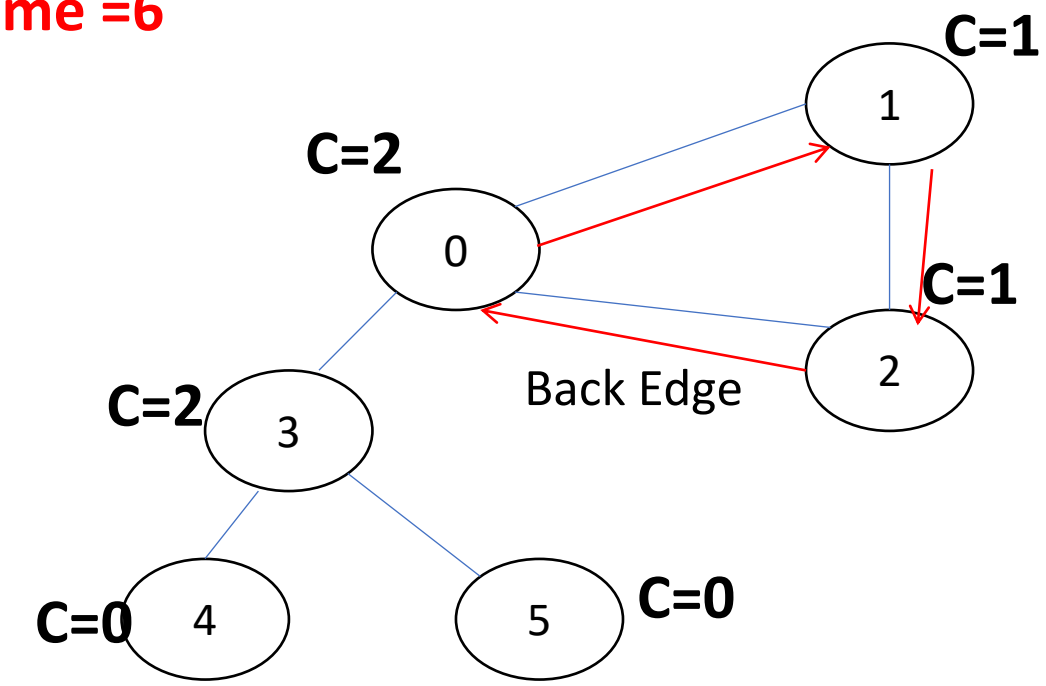
-1	0	1	0	3	3
0	1	2	3	4	5

AP

F	F	F	T	F	F
0	1	2	3	4	5

Algorithm on Example

Time =6



DT

0	1	2	3	4	5
0	1	2	3	4	5

LT

0	0	0	3	4	5
0	1	2	3	4	5

Parent

-1	0	1	0	3	3
0	1	2	3	4	5

AP

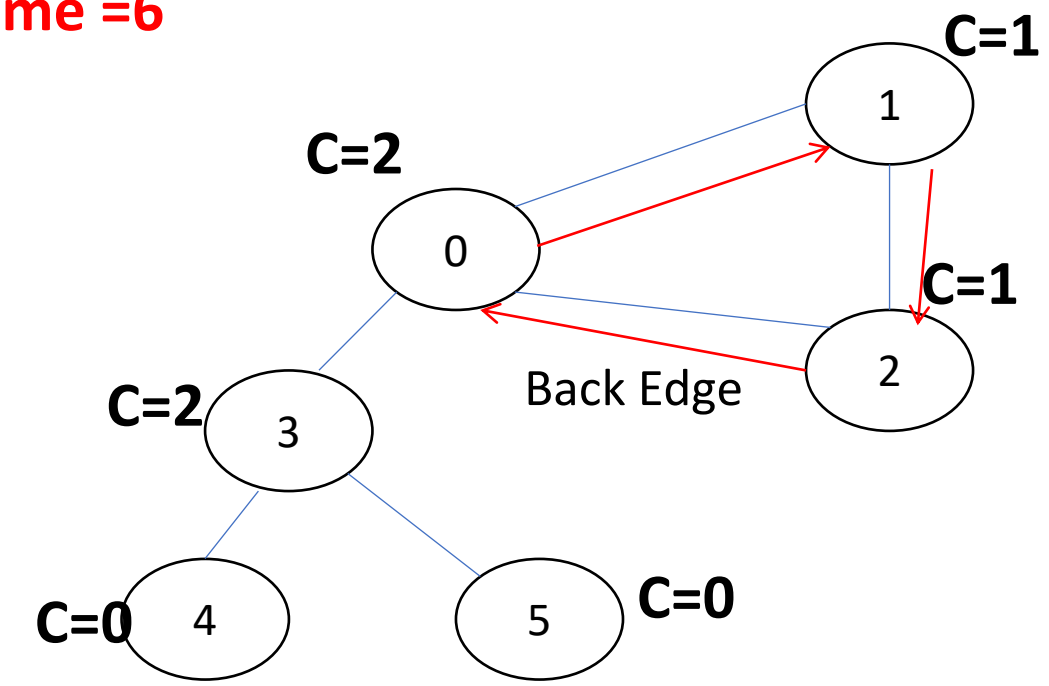
T	F	F	T	F	F
0	1	2	3	4	5

Backtrack to 0

Check 0 is AP: 0 is root node and Child count >1 therefore it is AP

Algorithm on Example

Time =6



Since all node visited so AP are 0 and 3

DT

0	1	2	3	4	5
0	1	2	3	4	5

LT

0	0	0	3	4	5
0	1	2	3	4	5

Parent

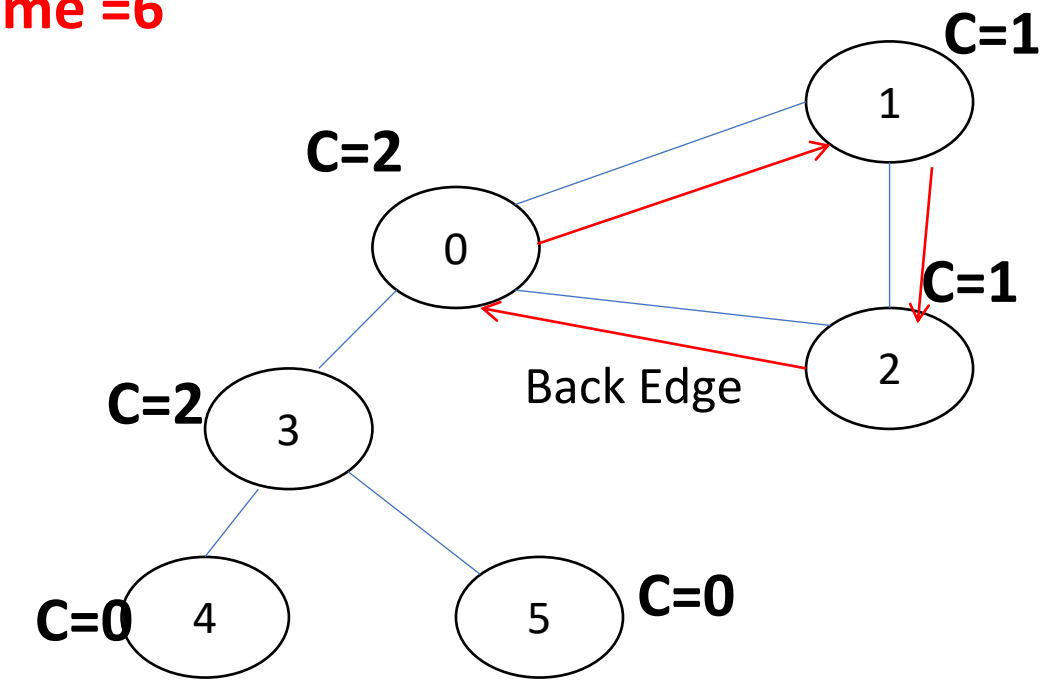
-1	0	1	0	3	3
0	1	2	3	4	5

AP

T	F	F	T	F	F
0	1	2	3	4	5

Algorithm on Example

Time =6



DT

0	1	2	3	4	5
0	1	2	3	4	5

LT

0	0	0	3	4	5
0	1	2	3	4	5

Parent

-1	0	1	0	3	3
0	1	2	3	4	5

AP

T	F	F	T	F	F
0	1	2	3	4	5

Since we have done single traversal of the graph so time complexity is (V+ E)

Do DFS traversal of the given graph

In DFS traversal, maintain a parent[] array where parent[u] stores the parent of vertex u.
To check if u is the root of the DFS tree and it has at least two children. For every vertex, count children. If the currently visited vertex u is root (parent[u] is NULL) and has more than two children, print it.

To handle a second case where u is not the root of the DFS tree and it has a child v such that no vertex in the subtree rooted with v has a back edge to one of the ancestors in DFS tree of u maintain an array disc[] to store the discovery time of vertices.

For every node u, find out the earliest visited vertex (the vertex with minimum discovery time) that can be reached from the subtree rooted with u. So we maintain an additional array low[] such that:
low[u] = min(disc[u], disc[w]) , Here w is an ancestor of u and there is a back edge from some descendant of u to w.

Thank You