# Graph Algorithms: Undirected Graphs

Unit 4: Lecture 05

# Undirected Graph

An undirected graph G=(V,E) is a collection of a finite set of vertices V and a set of edges E such that each edge in E is an **unordered pair of vertices**.

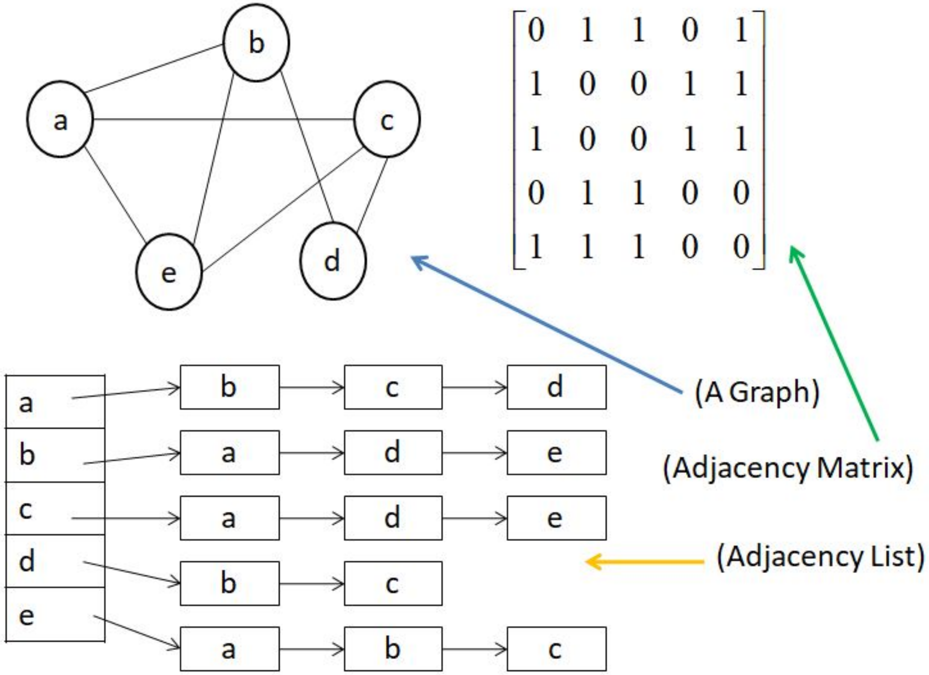
Thus, the **direction of an arc is not specified** and it implies that the movement between the vertices is bidirectional.

An undirected graph can also be represented as an **adjacency matrix** or an **adjacency list** (similar to a directed graph) however the number of elements in the matrix or in the list are increased as compared to a digraph.

# Undirected Graph (contd.)

**Example:**

The **weighted graphs** can also be represented accordingly- the weights on the arcs or within the list nodes.



# Minimum Cost Spanning Trees (MST)

In a weighted undirected graph, the **spanning tree** with minimum cumulative cost is called an **MST**.

**MST Property**: Let $G=(V,E)$ be a weighted connected graph and $U \subset V$. If $(u,v)$ is an edge of lowest cost such that $u \in U$ and $v \in (V-U)$ then there is an MST that includes $(u,v)$ as an edge.

Various methods follow the MST property. Two common methods are:

- Prim’s Algorithm (retains the tree)
- Kruskal’s Algorithm (grows a forest)

# Graph Traversal

The undirected graphs can be traversed using DFS or BFS (as of directed graphs).

**DFS for undirected graphs**:

Each tree in the forest is one **connected component** of the graph- if a graph is connected, it has only one tree in the DFS spanning forest.

For undirected graphs, there are only **tree arcs** and **back arcs** in the spanning forest.

No distinction between forward and back arcs and there can't be a cross arcs!

# DFS (contd.)

**Tree arc**- the arc (x,y) such that dfs(x) directly calls dfs(y) or vice versa

**Back arc**- the arc (x,y) such that neither dfs(x) nor dfs(y) calls the other directly but one calls the other indirectly (e.g., dfs(y) calls dfs(z) which calls dfs(x): y is an ancestor of x

Each node can be tracked with the help of **dfsno()** which increases successively for each vertex as the visit goes up.

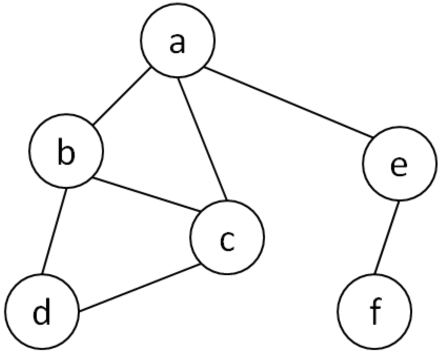
# DFS (contd.)

**Example:**

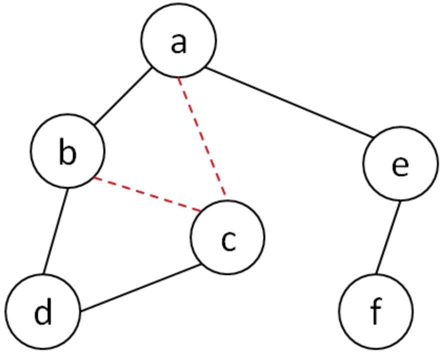
dfsno(a)=1, dfsno(b)=2,

dfsno(d)=3, dfsno(c)=4,

dfsno(e)=5, dfsno(f)=6




A Graph




DFS Tree

**Tree arcs** are represented by by solid lines in the DFS tree and **back arcs** are represented by red dotted lines.

# BFS

**BFS** involves the listing of all the nodes **adjacent to a node at once**- the approach searches the adjacent nodes **as broadly as possible**.

The spanning forest of BFS can be generated with **tree arcs** as well as **cross arcs** (no back arcs): every non-tree arc is a cross arc!

BFS procedure (next slide) **uses a queue** to store the visited vertices and lists the edges to a set generating the spanning forest.

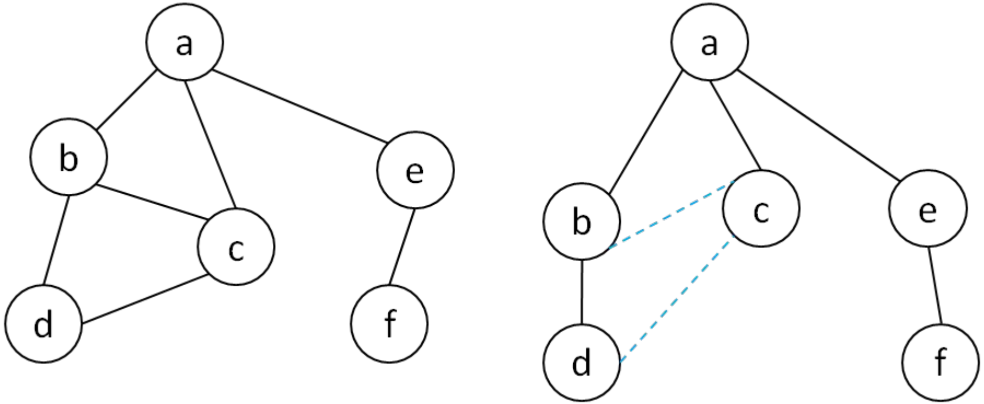
The **time complexity** of BFS is **O(max(n,e))** when adjacency list is used to represent the graph of n vertices and e arcs.

# BFS (contd.)

```
procedure bfs(v)
	mark[v]=visited
	enqueue(v,Q)		//Q is a queue of vertices
	while not empty(Q)
		x=front(Q)
		dequeue(Q)
		for each vertex y adjacent to x
			if(mark[y]=unvisited
				mark[y]=visited
				enqueue(y,Q)
				insert((x,y),T)		//T is the set of edges: spanning tree/forest
end procedure
```

# BFS (contd.)

**Example: tree arcs** are shown by solid lines and **cross arcs** are shown by dotted lines.




**Note:** It can be observed that both DFS and BFS produce the spanning trees for the given undirected graphs

Exercise- Can an MST be generated by DFS or BFS?

# Articulation Points

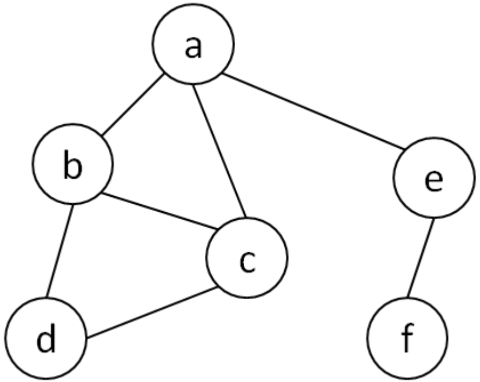
DFS or BFS can be used to find the connected components of a graph (the **connected components** are tree arcs of either spanning forest).

**Articulation Point-** an articulation point of a graph is a vertex v such that when it is removed from the graph, the connected components are divided into two or more components.

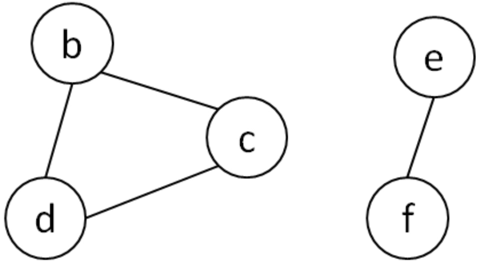
A connected graph with no articulation points is said to be **biconnected**.

# Articulation Points (contd.)

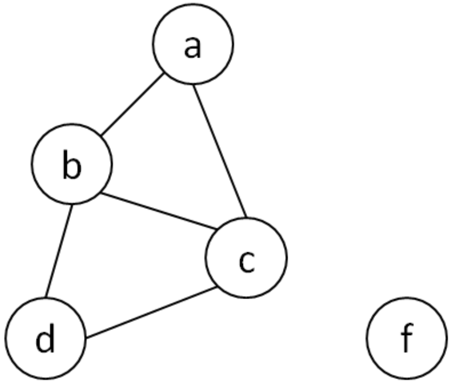
Example: In the given graph a and e are the articulation points




A Graph




After deleting a




After deleting e

# Articulation Points (contd.)

**How to find** the articulation points? **DFS is used** to find the articulation points using the following steps:

- Perform DFS and compute **dfsno()** for each vertex
- For each vertex v, compute **low(v)** which is the smallest dfsno() of v **or** of any vertex w reachable from v by following down 0 or more tree edges to a descendant x of v and then following a back arc (x,w)
  - **Compute low(v)** for all vertices v by visiting the vertices in a **postorder** traversal. While processing v, compute low(y) for every child y of v; low(v) is **minimum of**
    - dfsno(v),
    - dfsno(z) for any vertex z for which there is a back edge (v,z),
    - low(y) for any child y of v

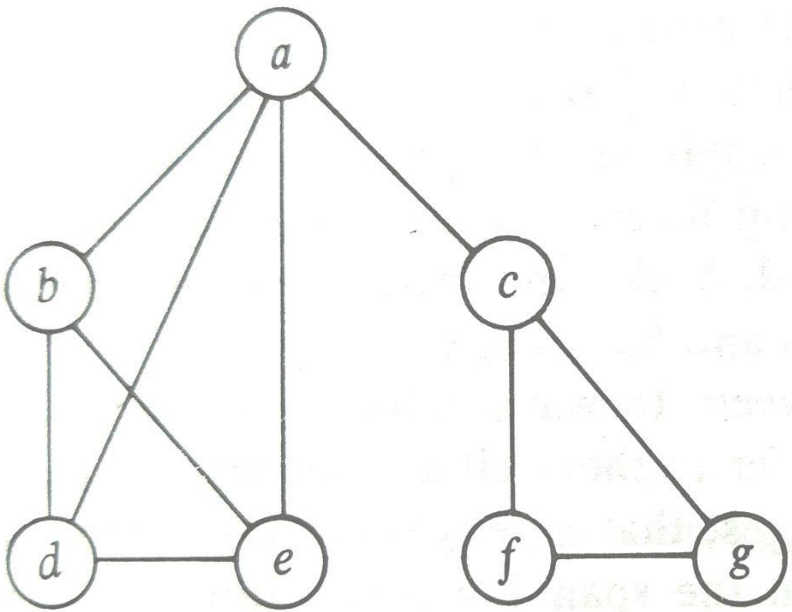
# Articulation Points (contd.)

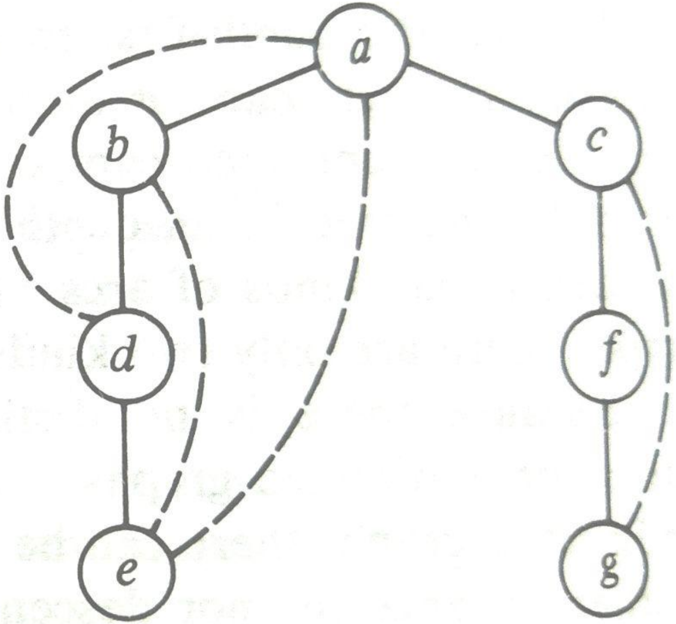
- Find the articulation points as:
    - The **root is an articulation point iff** it has 2 or more children
    - A vertex **v other than the root is an articulation point iff** there is some child w of v such that **low(w)>=dfsno(v)** (In this case v disconnects w and its descendants from the rest of the graph)

The **time taken** by the process is **O(e)**.

# Articulation Points (contd.)

**Example:** Consider the following graph with DFS tree-



# Articulation Points (contd.)

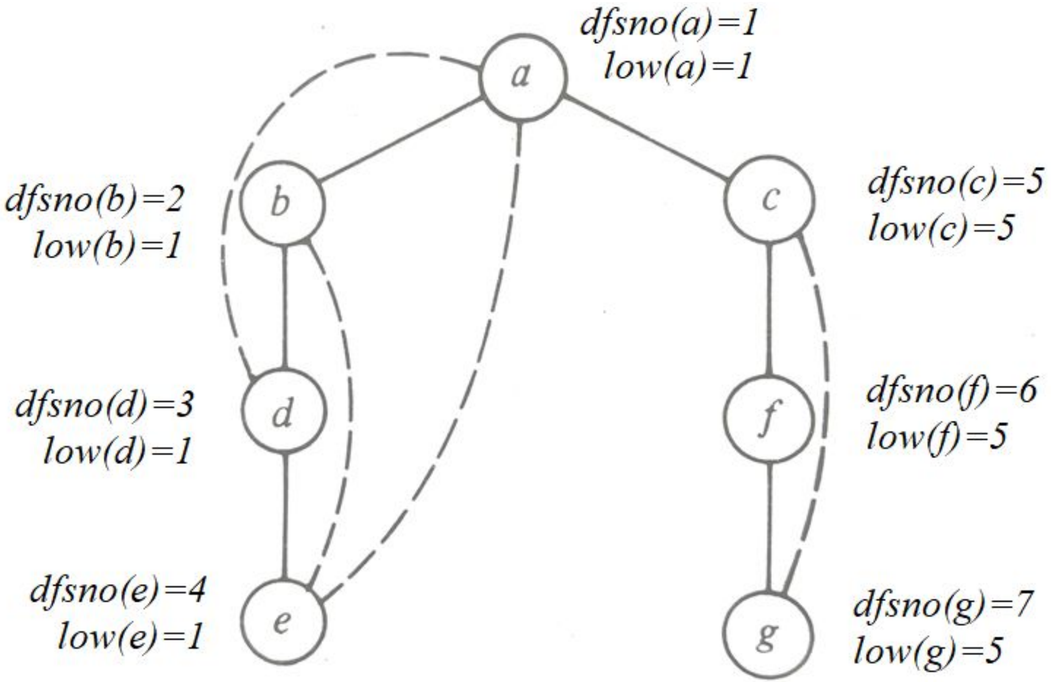
Example (contd.): {a,c} are the articulation points-

**a is the articulation point**

as it has 2 children,

**c is another** point as it has

1 child f with low(f)>=dfsno(c).



# Exercise

1. Compare Prim’s algorithm with Kruskal’s algorithm in terms of time and space complexity.
2. Write a procedure to find all the articulation points of a graph.
3. Discuss if it is possible to get an MST using DFS in any case. Illustrate with an example.

# Reference Book

Data Structures and Algorithms: Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, 10th Impression, Pearson Education, New Delhi