Overview

- A Graph is a non-linear data structure, which consists of vertices(or nodes) connected by edges(or arcs) where edges may be directed or undirected.

- Graphs are a powerful and versatile data structure that easily allow you to represent real life relationships between different types of data (nodes). There are two main parts of a graph:

  - ✓ **The vertices (nodes) where the data is stored**
  - ✓ **The edges (connections) which connect the nodes**

- Vertices *i* and *j* are adjacent vertices iff (*i*, *j*) is an edge in the graph

Graph and Its Representations

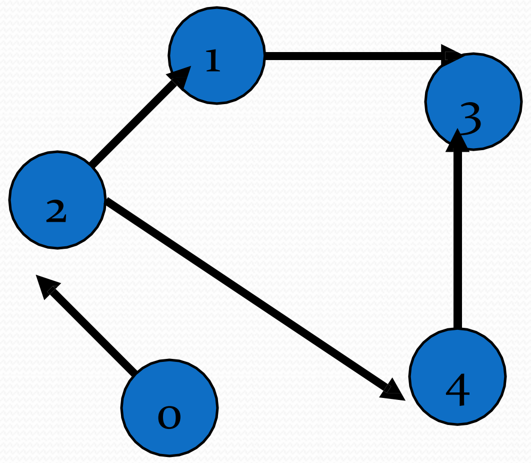
- The choice of graph representation is said to be situation specific. It totally depends on the type of operations to be performed and ease of use.

- Following the most commonly used representations of a graph.

1. Adjacency Matrix
2. Adjacency List

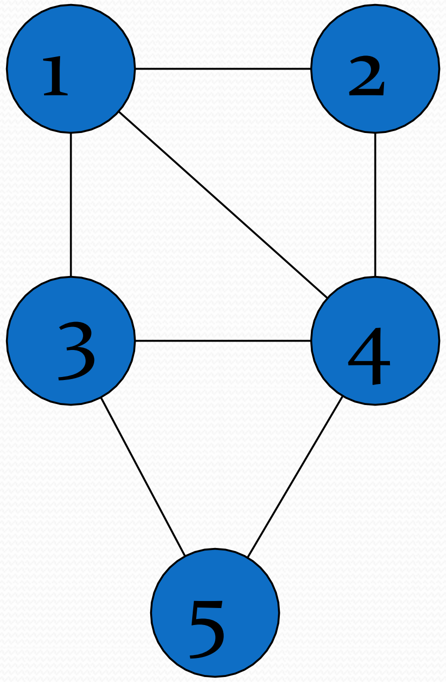
- *Adjacency Matrix*
  - A square grid of boolean values
  - If the graph contains N vertices, then the grid contains N rows and N columns
  - For two vertices numbered I and J, the element at row I and column J is true if there is an edge from I to J, otherwise false

Adjacency Matrix



$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & \text{false} & \text{false} & \text{true} & \text{false} & \text{false} \\ 1 & \text{false} & \text{false} & \text{false} & \text{true} & \text{false} \\ 2 & \text{false} & \text{true} & \text{false} & \text{false} & \text{true} \\ 3 & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} \\ 4 & \text{false} & \text{false} & \text{false} & \text{true} & \text{false} \end{array}$$

Adjacency Matrix



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

Advantages of using adjacency matrix are as follows:

- ➢ Easy to understand, implement and convenient to work with.
- ➢ Removing an edge involves time complexity of O(1).
- ➢ Queries like whether there is an edge from vertex „u‟ to vertex „v‟ are efficient and require time complexity, O(1).

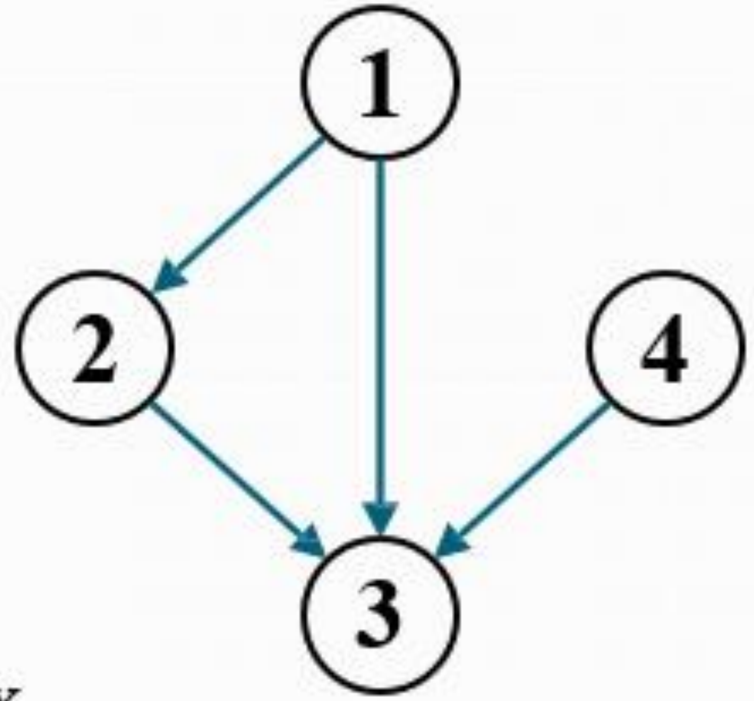
Disadvantages of using adjacency matrix are as follows:

- ▪ Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.
- ▪ If the number of nodes in the graph may change, matrix representation is too inflexible (especially if we don‟t know the maximal size of the graph

Adjacency List

- In this representation, every vertex of graph contains list of its adjacent vertices.

- Example:
  - Adj[1] = {2,3}
  - Adj[2] = {3}
  - Adj[3] = {}
  - Adj[4] = {3}
- Variation: can also keep a list of edges coming *into* vertex



Cont’d…

Advantages of using adjacency list are as follows:

- Addition of a vertex and connecting new vertices with the existing ones is easier.
- Has a space complexity of O(|V|+|E|).
- It allows us to store graph in more compact form and to get the list of adjacent vertices in O(1) time which is a big benefit for some algorithms.

Disadvantages of using adjacency list are as follows:

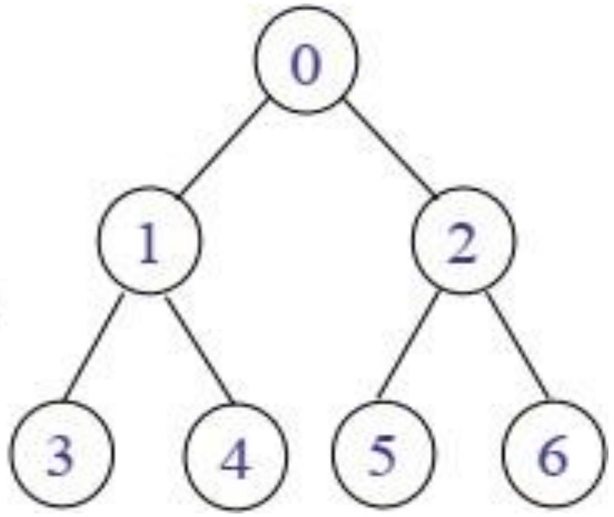
- Queries like whether there is an edge from vertex u to vertex v are not efficient and require time complexity, O(V).
- It does not allow us to make an efficient implementation, if dynamic change of vertices number is required.

Types Of Graphs

1. **Undirected Graph** all edges are undirected or If the connection is symmetric (in other words A is connected to B $\leftrightarrow$ B is connected to A), then we say the graph is undirected.
2. **Directed Graph** all edges are directed or If an edge only implies one direction of connection, we say the graph is directed.
3. **Weighted Graph**: A weighted graph is a graph where each edge has an associated numerical value, called **weight**.

Undirected Graphs

❖The pair of vertices representing any edge is unordered. Thus, the pairs (u,v) and (v,u) represent the same edge.



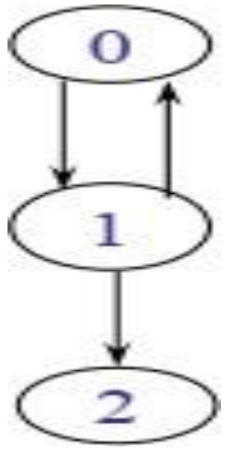
Example: a graph G2

- V(G2)={0,1,2,3,4,5,6}
- E(G2)={(0,1),(0,2), (1,3),(1,4),(2,5),(2,6)}
- G2 is also a tree that is a special case of graph
- An **undirected graph** is said to be **connected** if there is a path between every pair of vertices in the graph.

Directed Graphs (Digraph)

❖ each edge is represented by a ordered pairs <u,v>

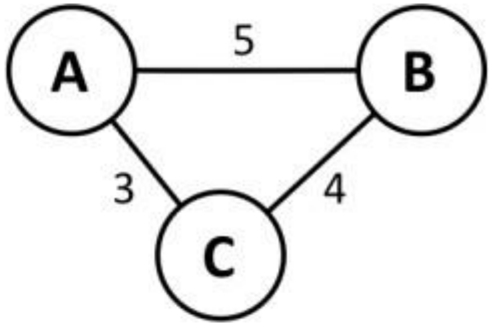
Example: a graph G3



- V(G3)={0,1,2}
- E(G3)={<0,1>,<1,0>,<1,2>}
- A directed path in a directed graph G is a sequence of distinct vertices, such that there is an edge from each vertex in the sequence to the next.

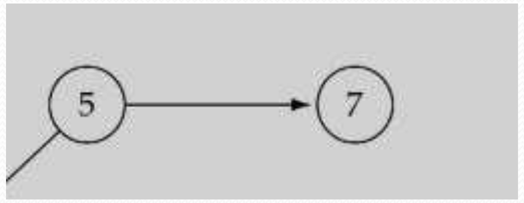
Weighted Graph

- A weighted graph is a graph where each edge has an associated numerical value, called weight. Weighted graphs may be either directed or undirected. The weight of the edge is often referred to as the “cost” of the edge. Example of weighted graph is:



Graph terminology

- Adjacent nodes: two nodes are adjacent if they are connected by an edge

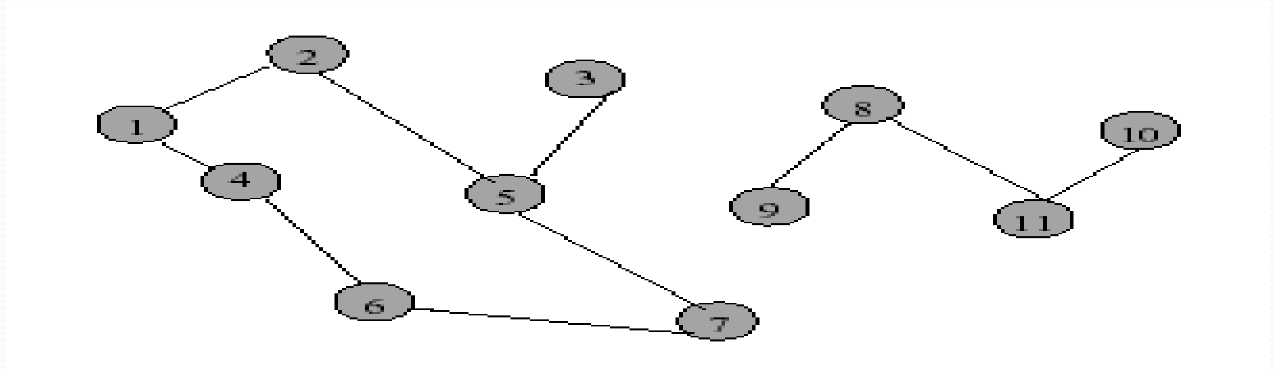


5 is adjacent to 7
7 is adjacent from

- Path: a sequence of vertices th atconnect two nodes in a graph
- A simple path is a path in which all vertices, except possibly in the first and last, are different.
- Complete graph: a graph in which every vertex is directly connected to every other vertex

Continued…

- A cycle is a simple path with the same start and end vertex.
- The **degree** of vertex *i* is the no. of edges incident on vertex *i*.



e.g., degree(2) = 2, degree(5) = 3, degree(3) = 1

Graph Traversal Algorithms

- ❖Graph traversal means visiting all the nodes of the graph.
- ❖We want to visit every vertex and every edge exactly once in some well-defined order.
- ❖There are two primary traversal algorithms:
  - *Breadth-First Search* (BFS) and
  - *Depth-First Search* (DFS).

DFS/BFS Search Algorithm

```
SEARCH(Node n):
1. Visit n.
2. Put n's children on a list (Stack or Queue).
3. For each child c in list:
   SEARCH(c)
```

- If we use a Stack and recursive, we get DFS
- If we use a Queue and iterative, we get BFS

BFS-Breadth First Search

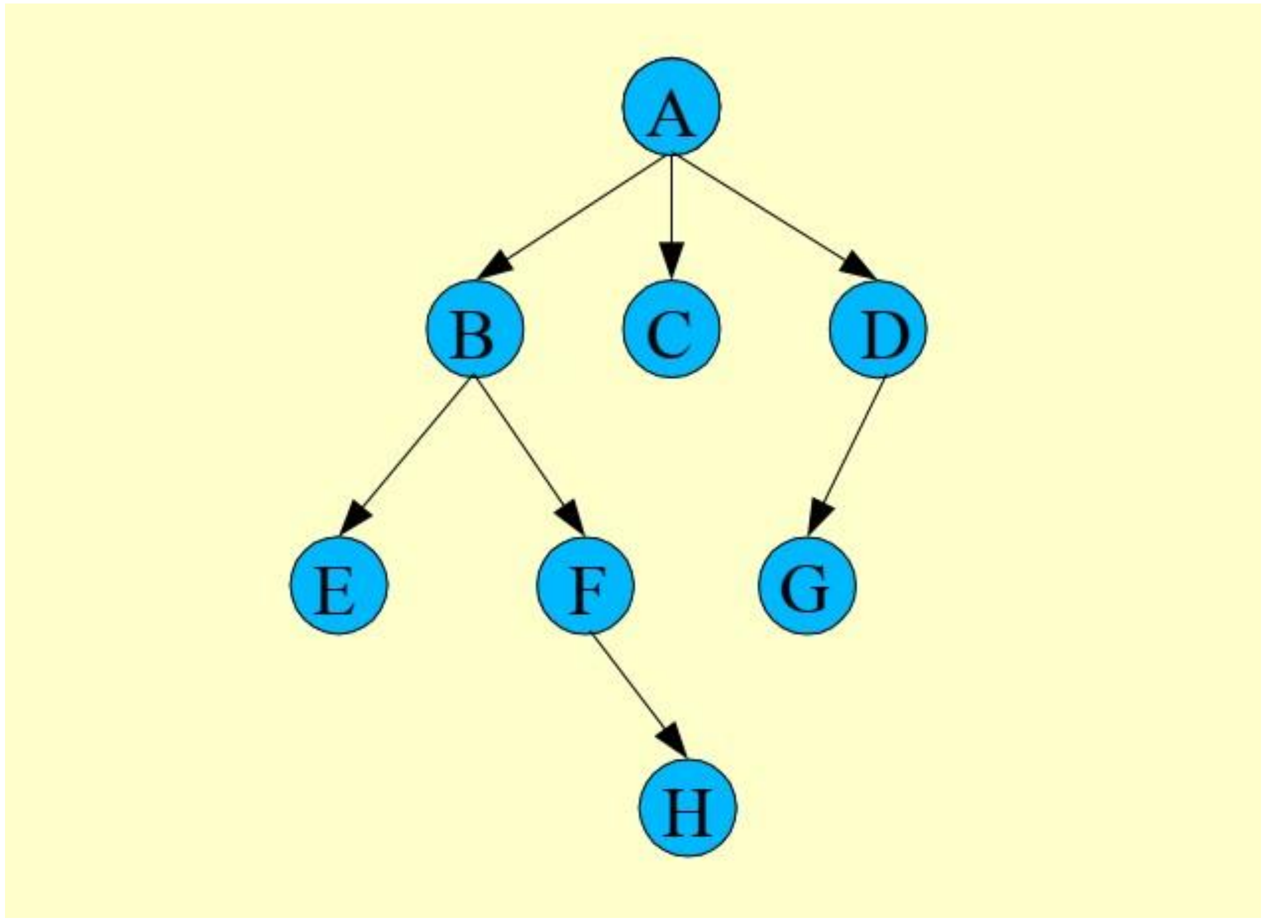
❖ **Idea: Visit all children before first grandchild.**

- **Visiting order:** Visit a vertex, then visit all of its neighbors, then visit all of the neighbors of its neighbors, . . .
- BFS is that a BFS is best used at finding the shortest paths in a given Graph
- Needs flags or someway to preventing infinite loops: **Weakness**

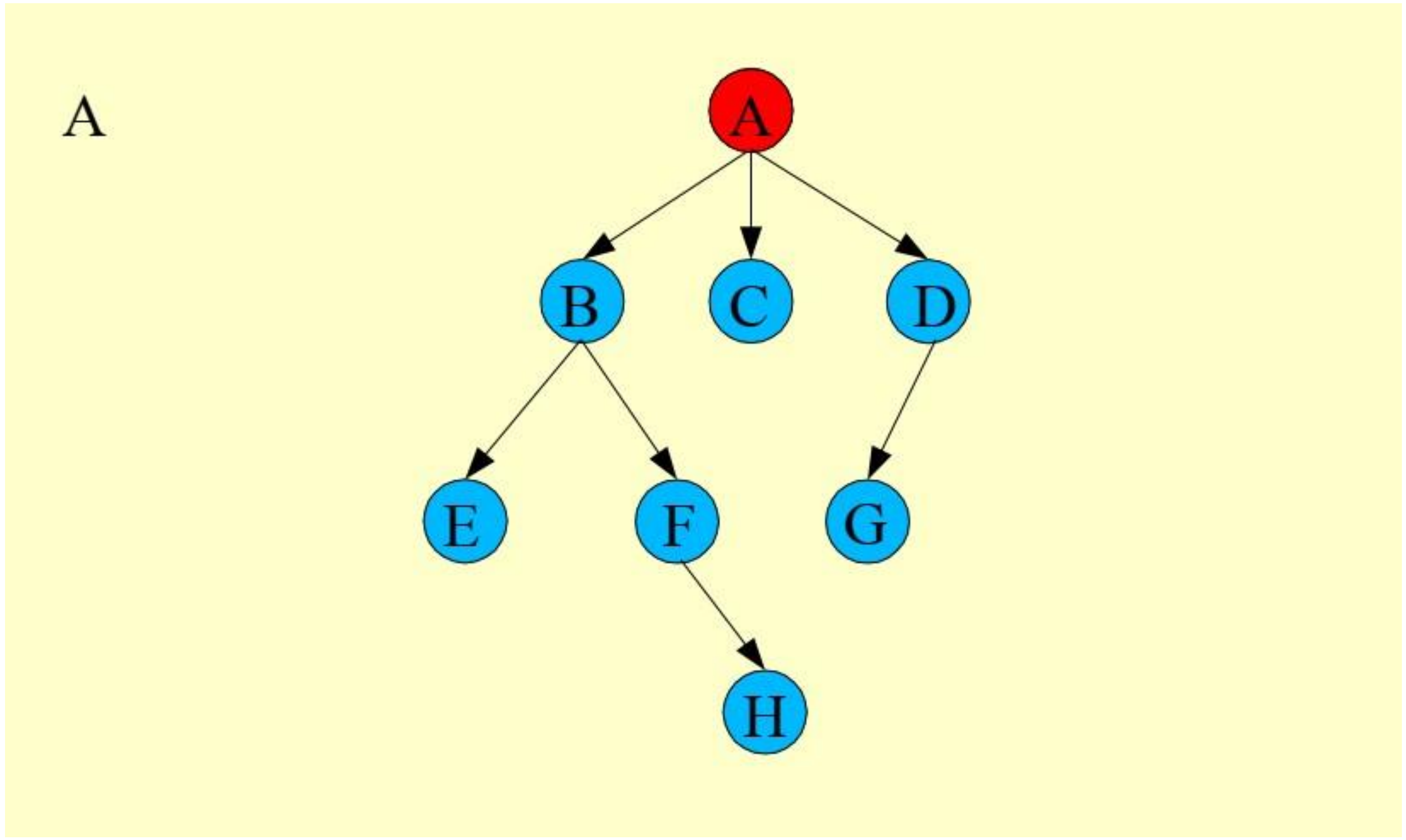
BFS Algorithm

```
create a queue Q
mark v as visited and put v into Q
while Q is non-empty
        remove the head u of Q
        mark and enqueue all (unvisited) neighbours of u
```

Example : BFS

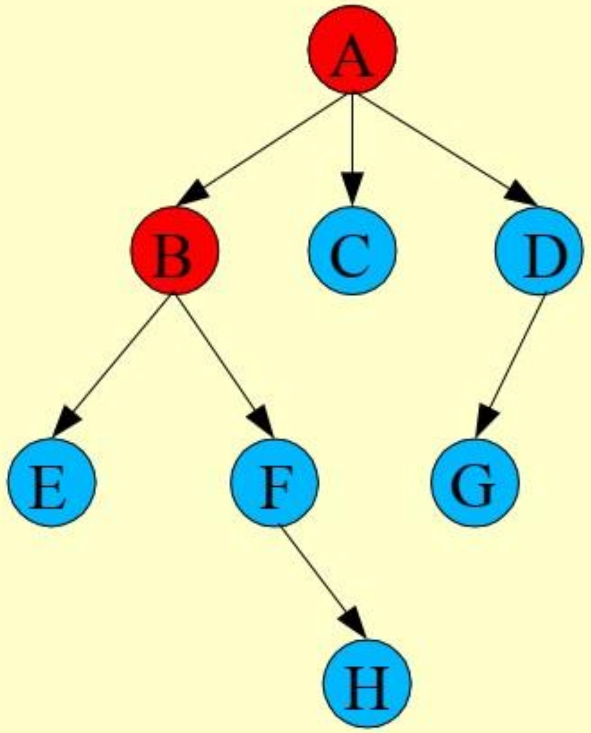


Cont'd…

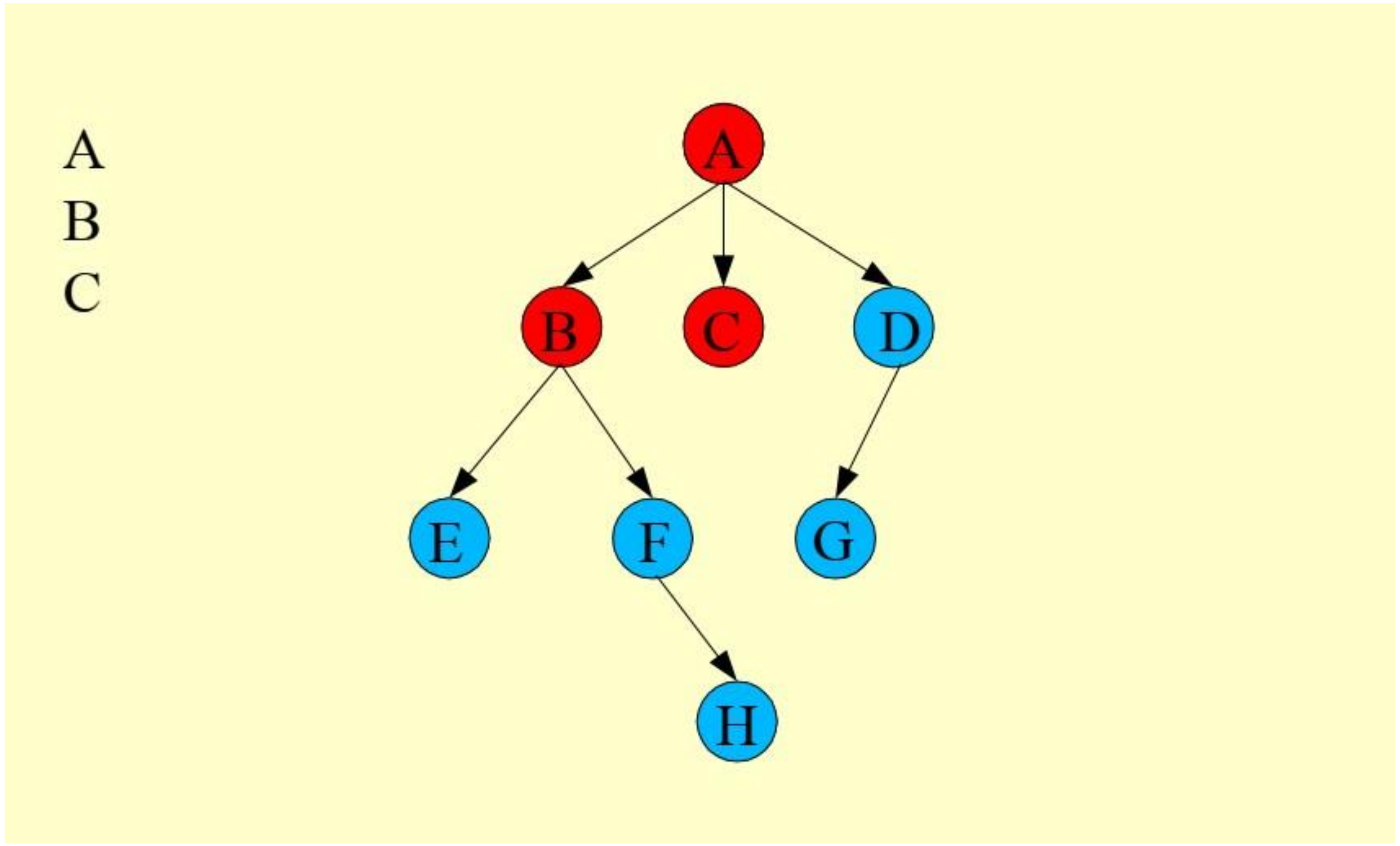


Cont’d…

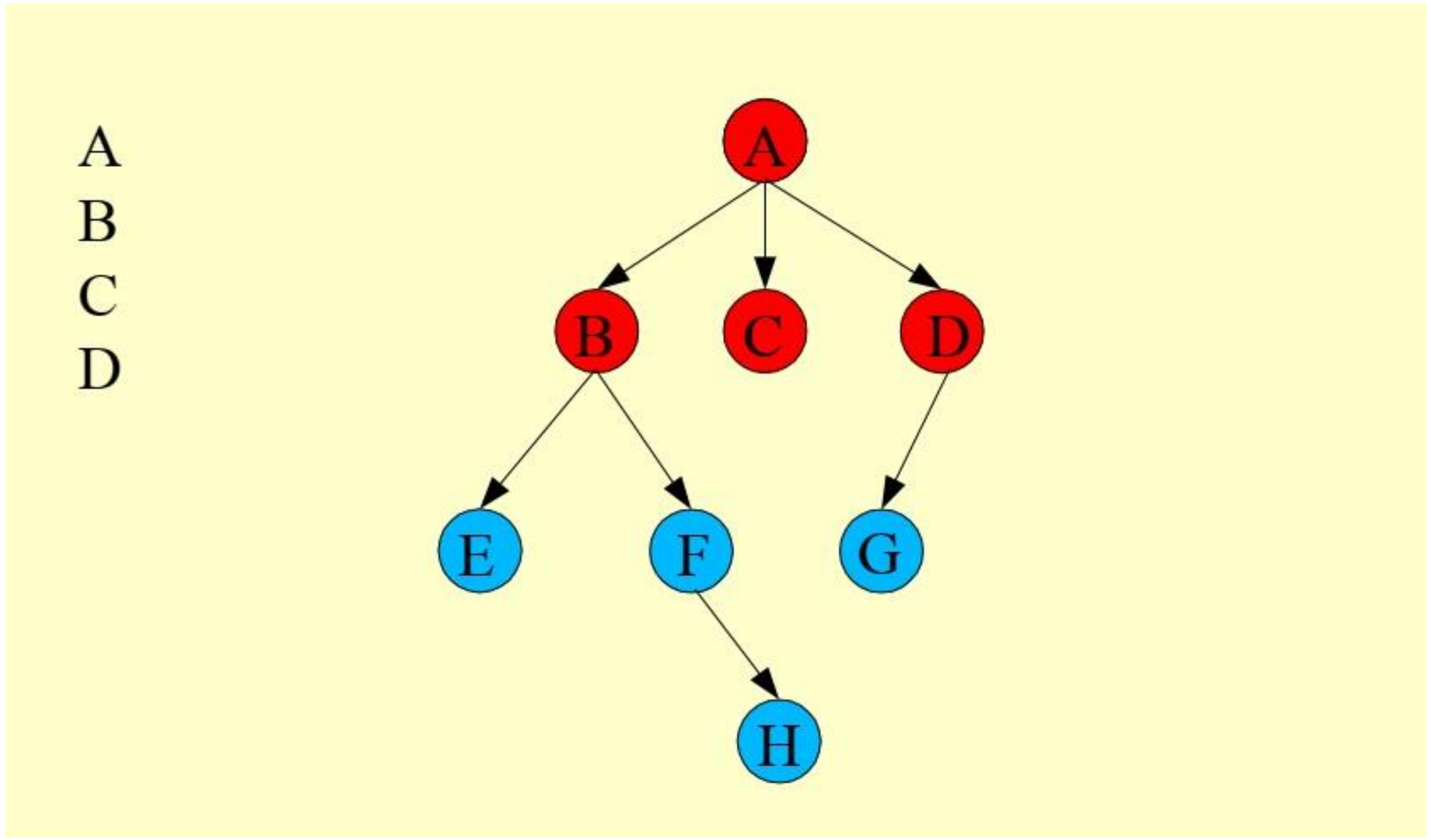
A
B



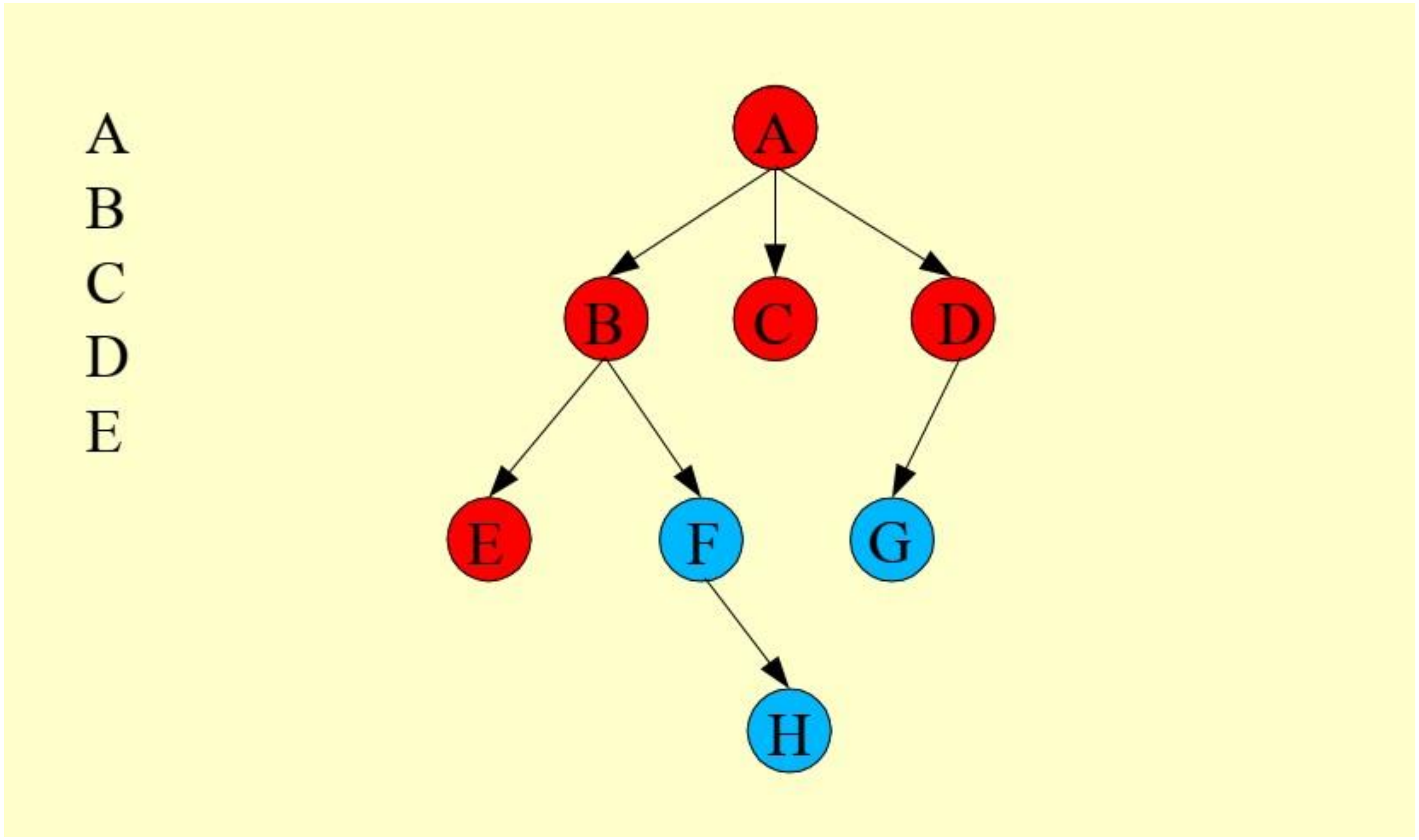
Cont’d…



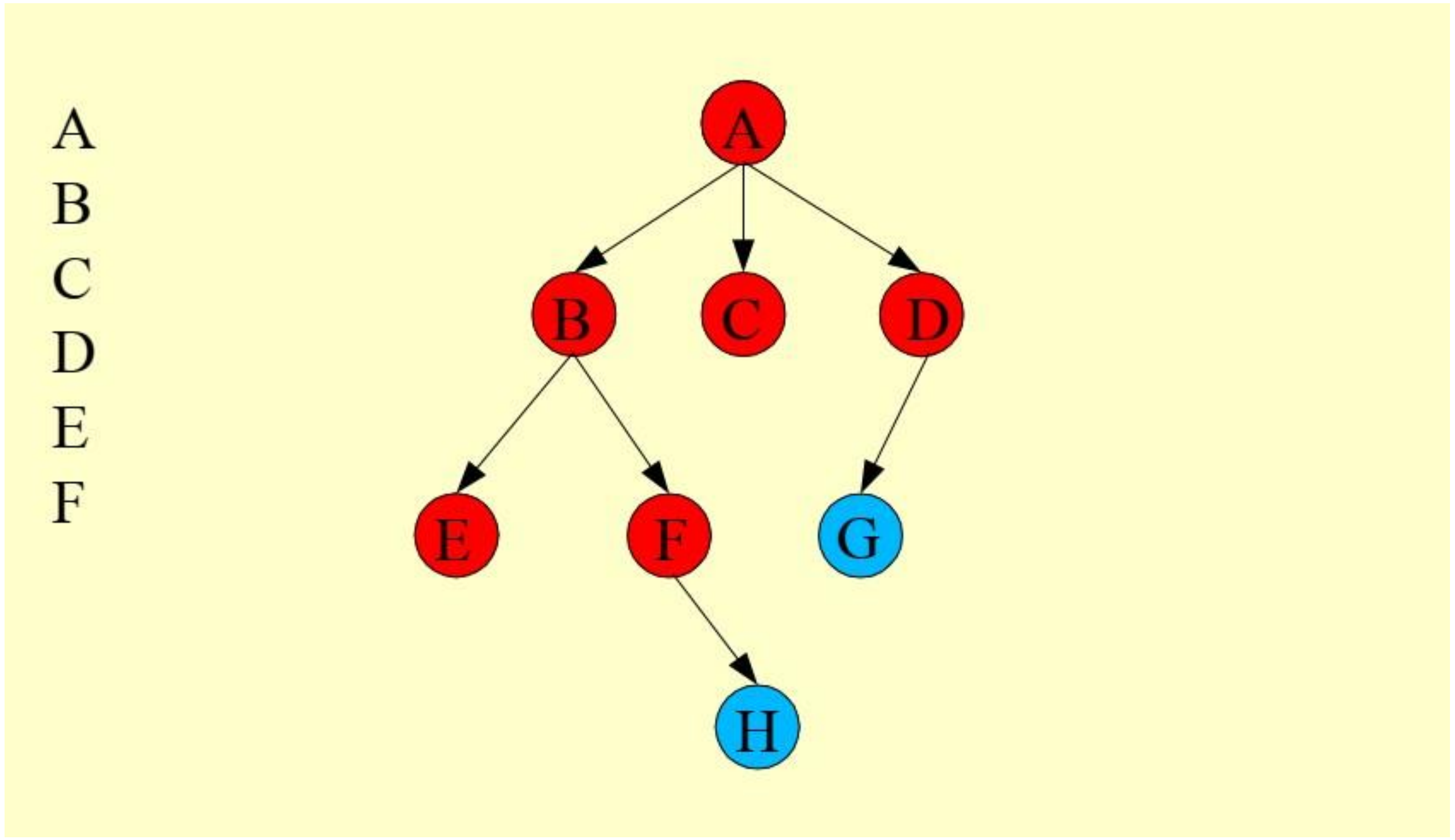
Cont'd…



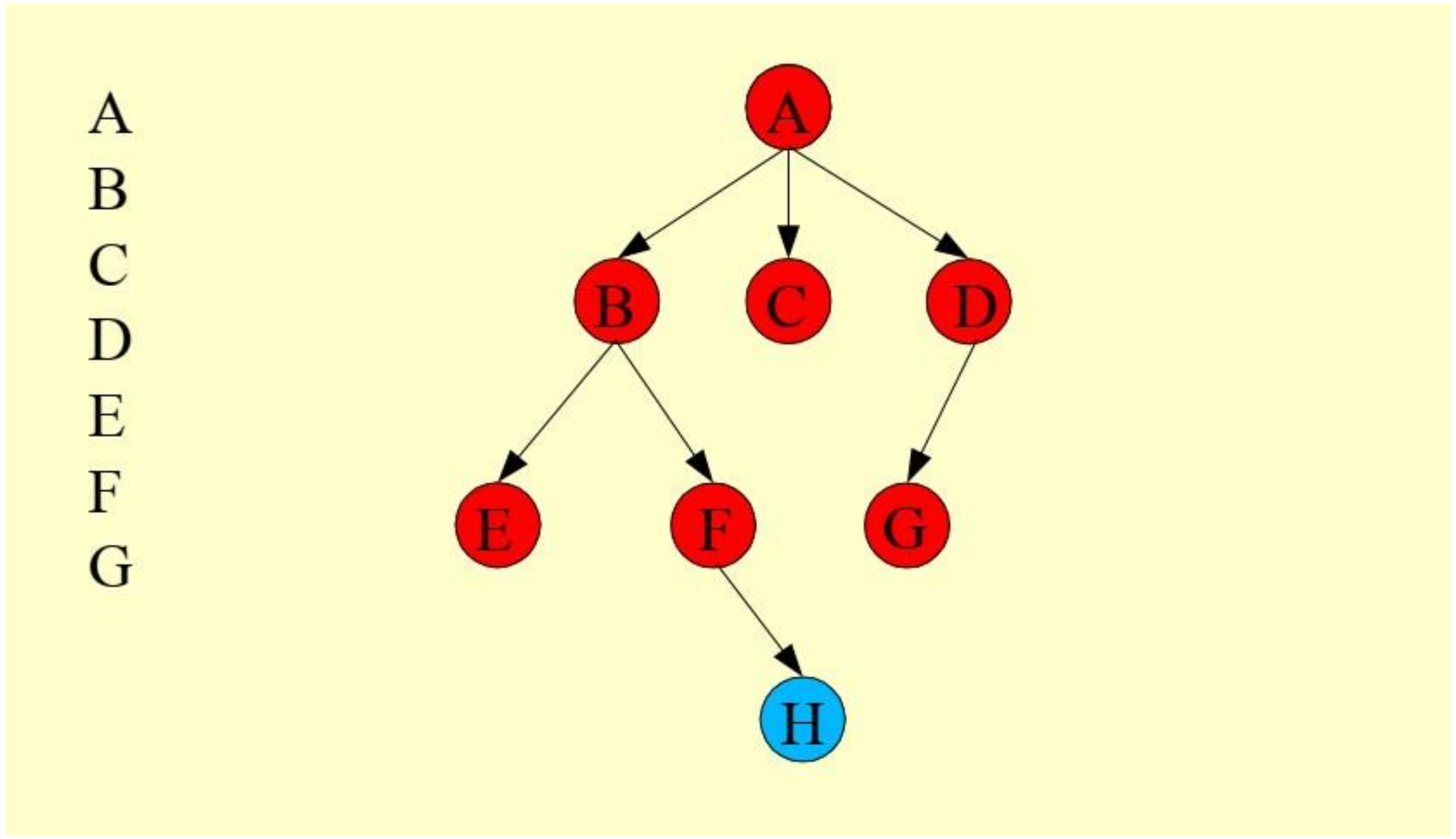
Cont'd…



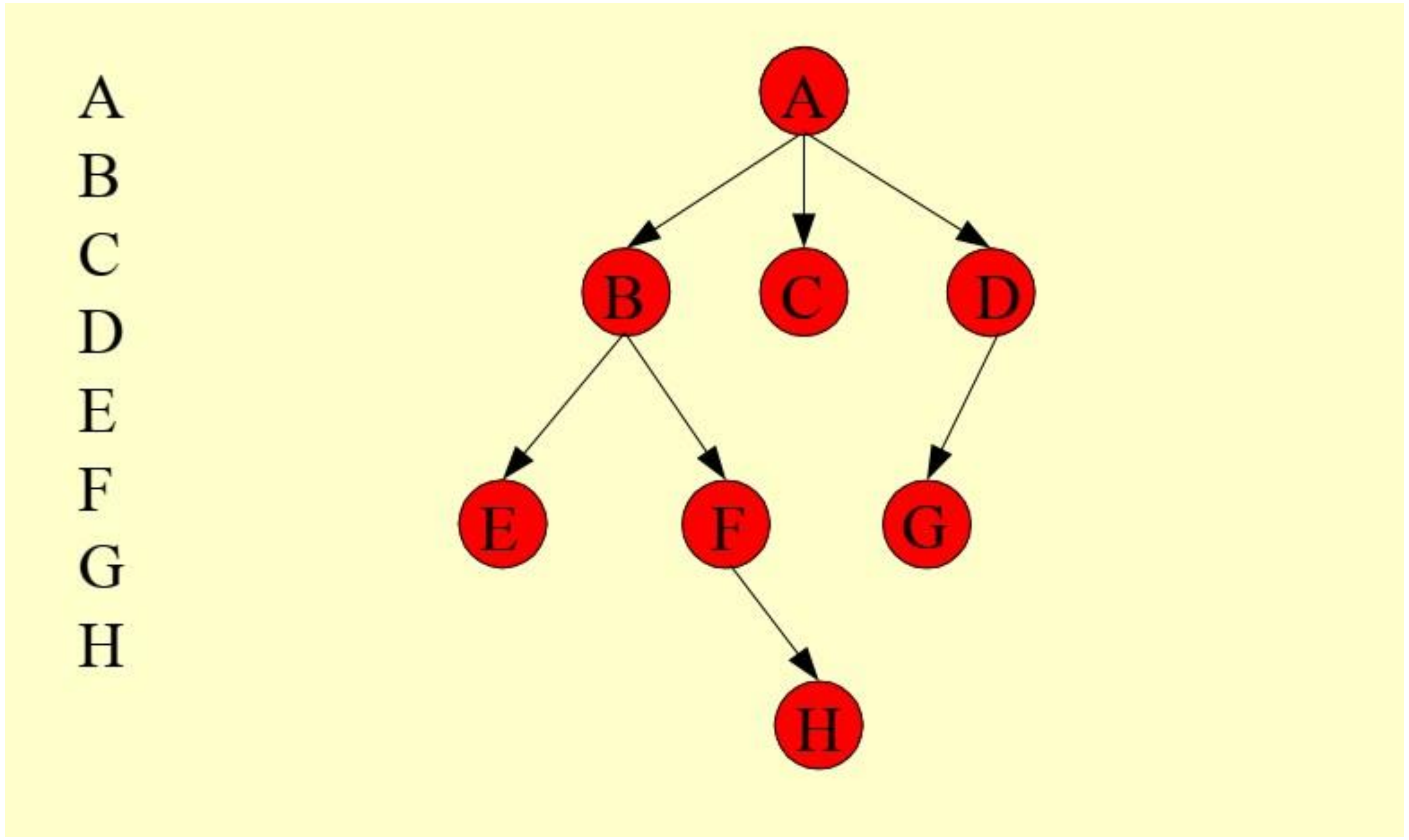
Cont’d…



Cont’d…



Cont’d…



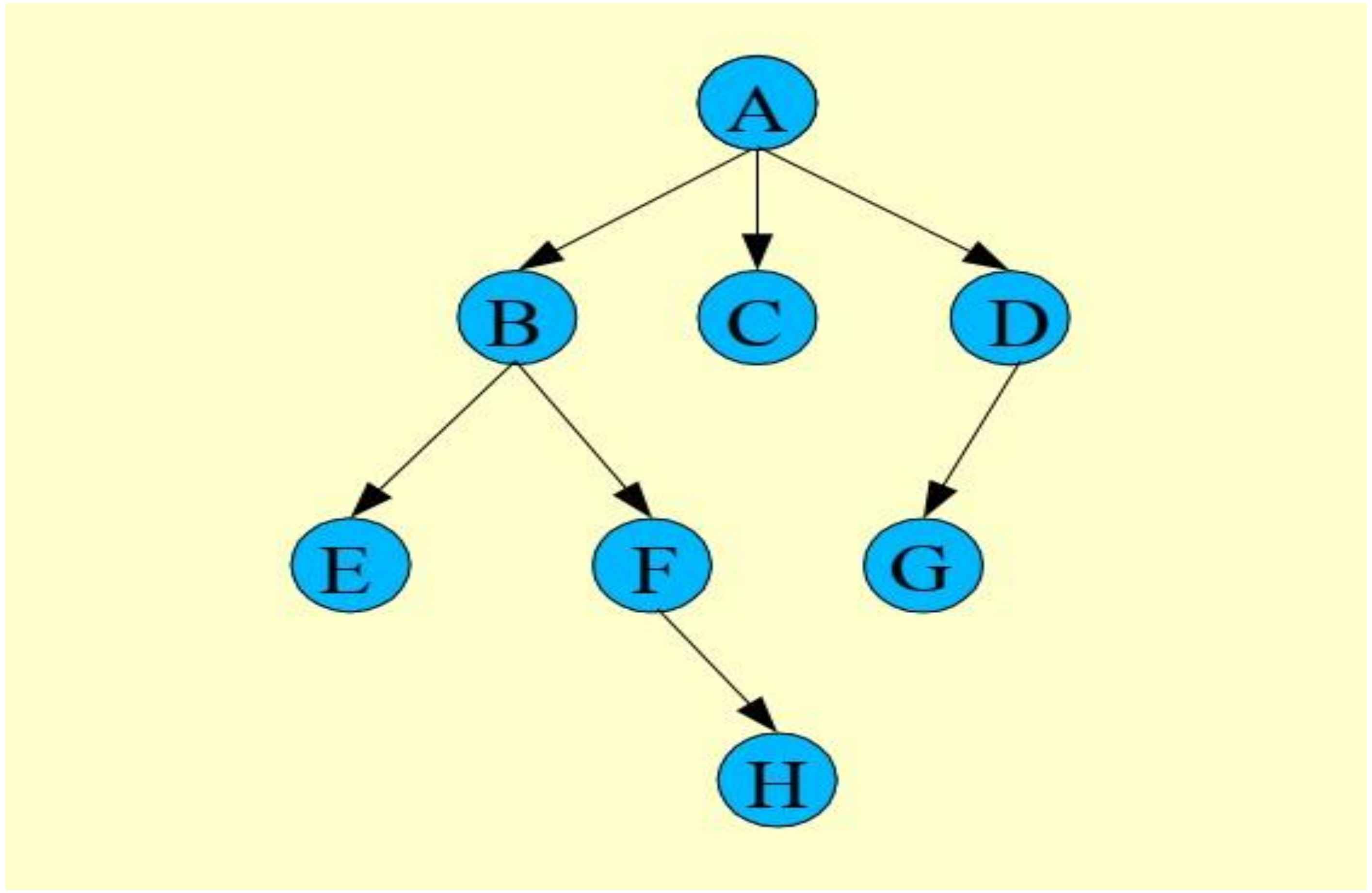
$$\begin{tikzcd}
 & & & A \arrow[dll] \arrow[d] \arrow[dr] & \\
 & B \arrow[dl] \arrow[dr] & & C & D \arrow[d] \\
E & & F \arrow[dr] & & G \\
 & & & H &
\end{tikzcd}$$

DFS-Depth First Search

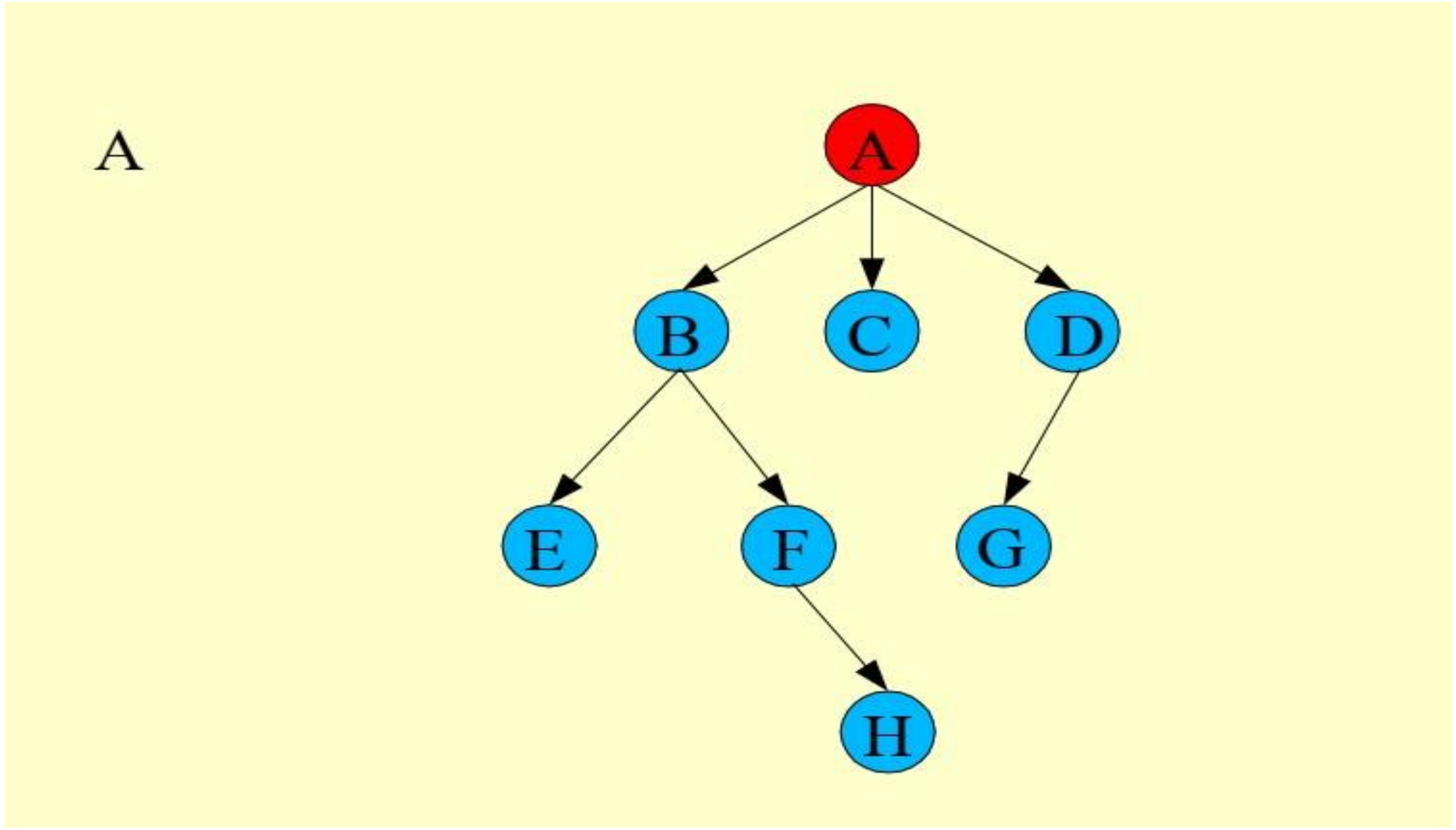
- ***Idea*: Follow a single path to its end, then backtrack**
- Is an algorithm for traversing or searching a tree, tree structure or graph.
- One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

DFS ALGORITHM USING STACK

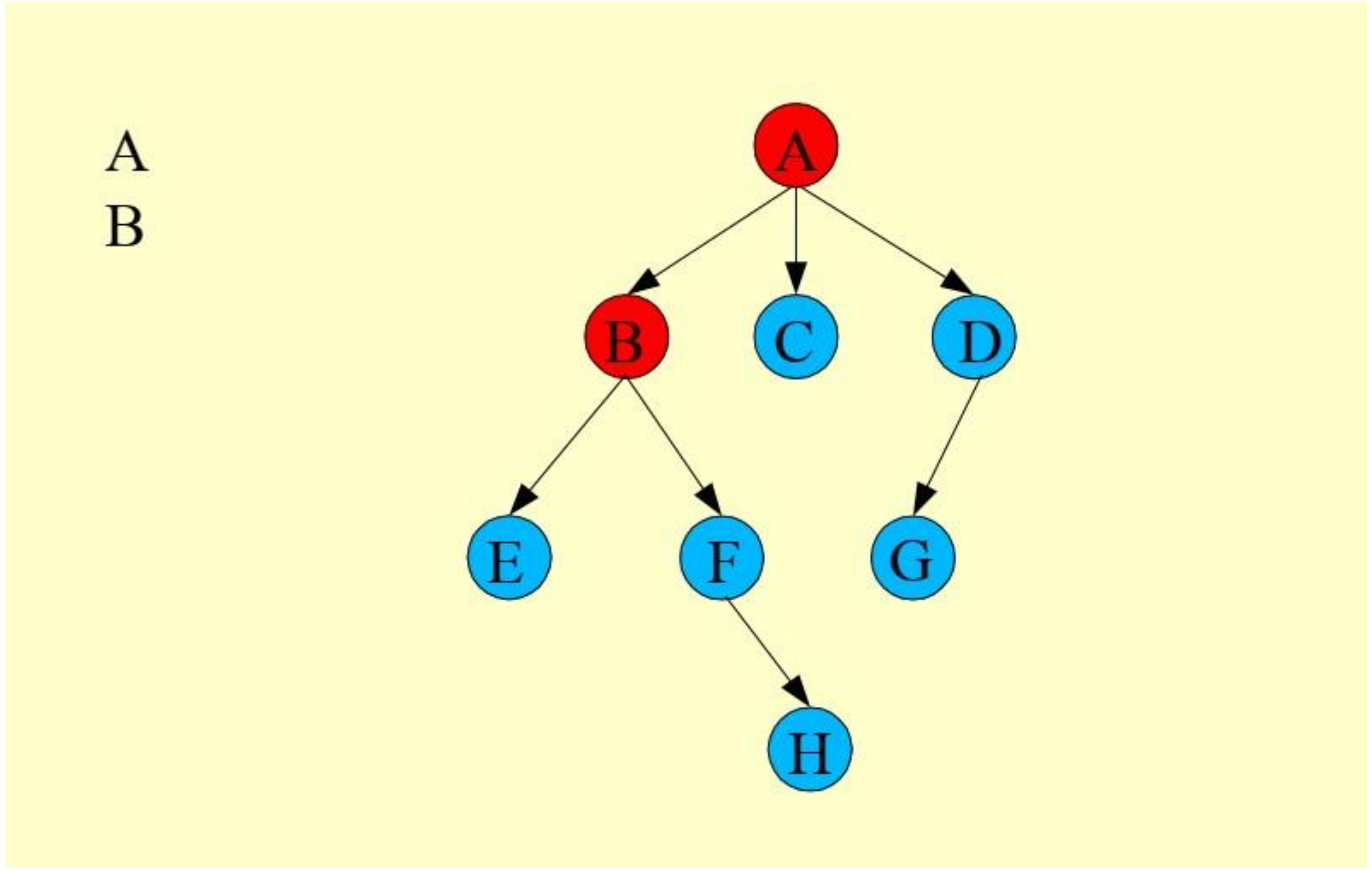
Example:DFS



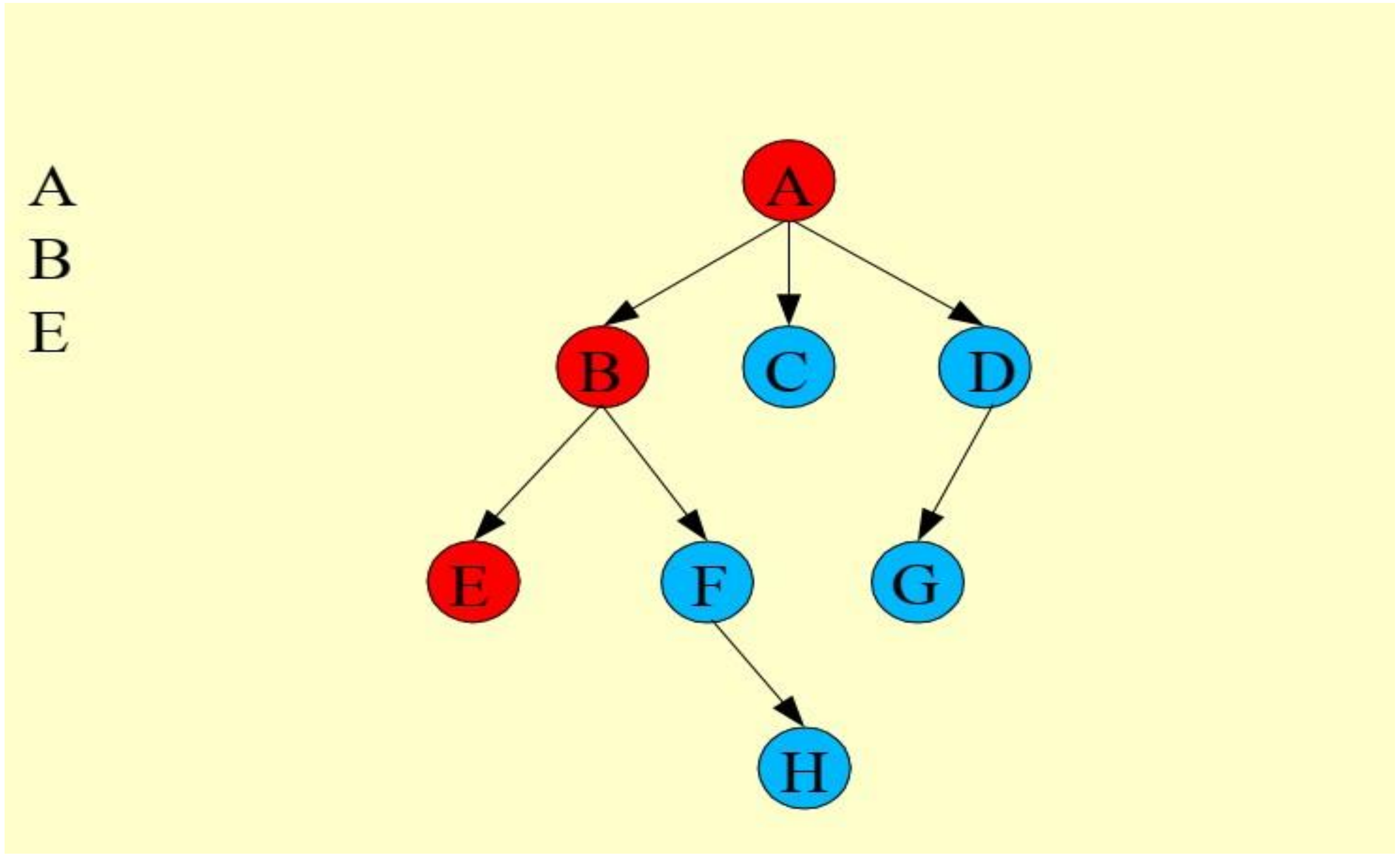
Cont’d…



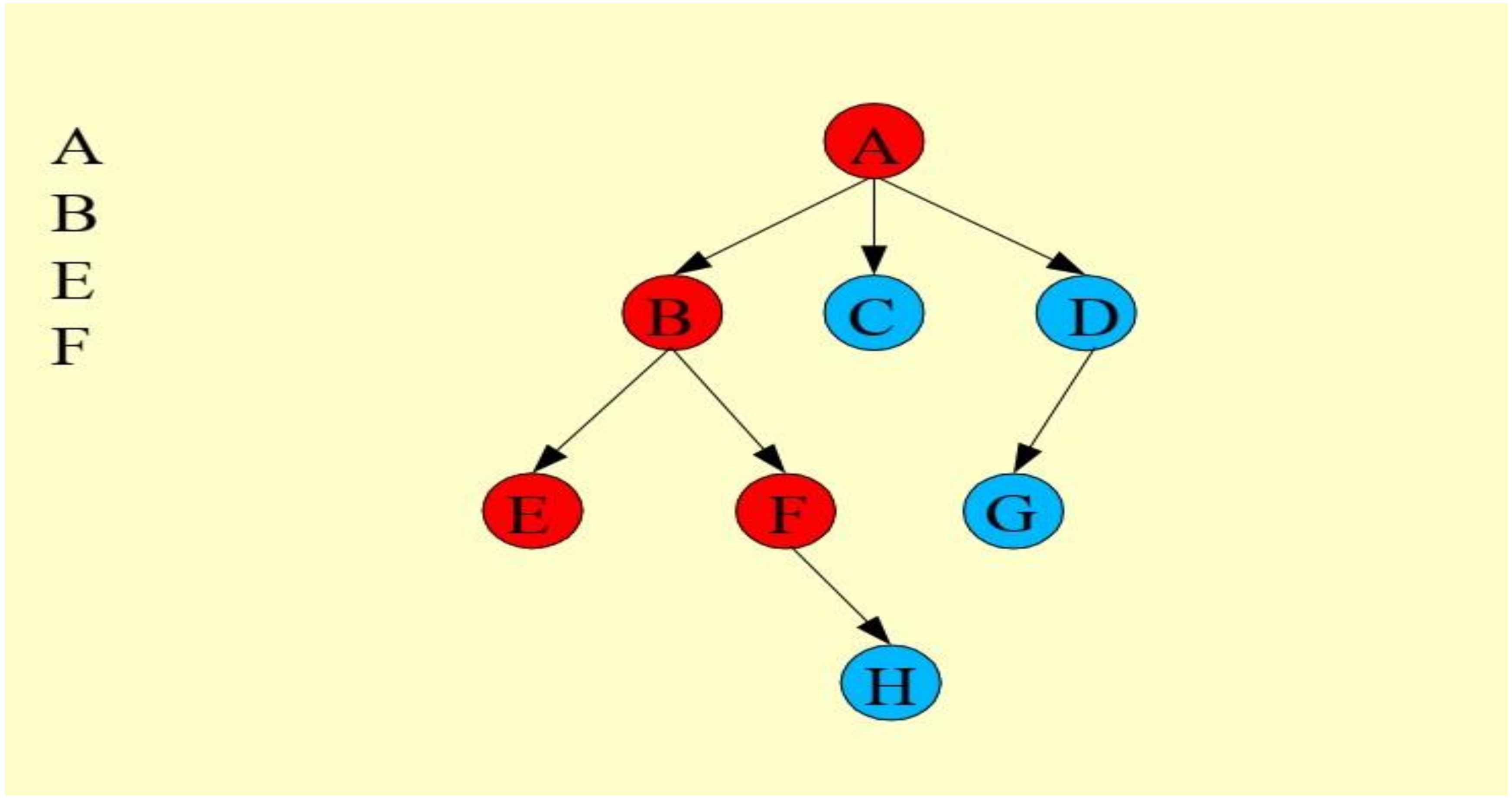
Cont'd…



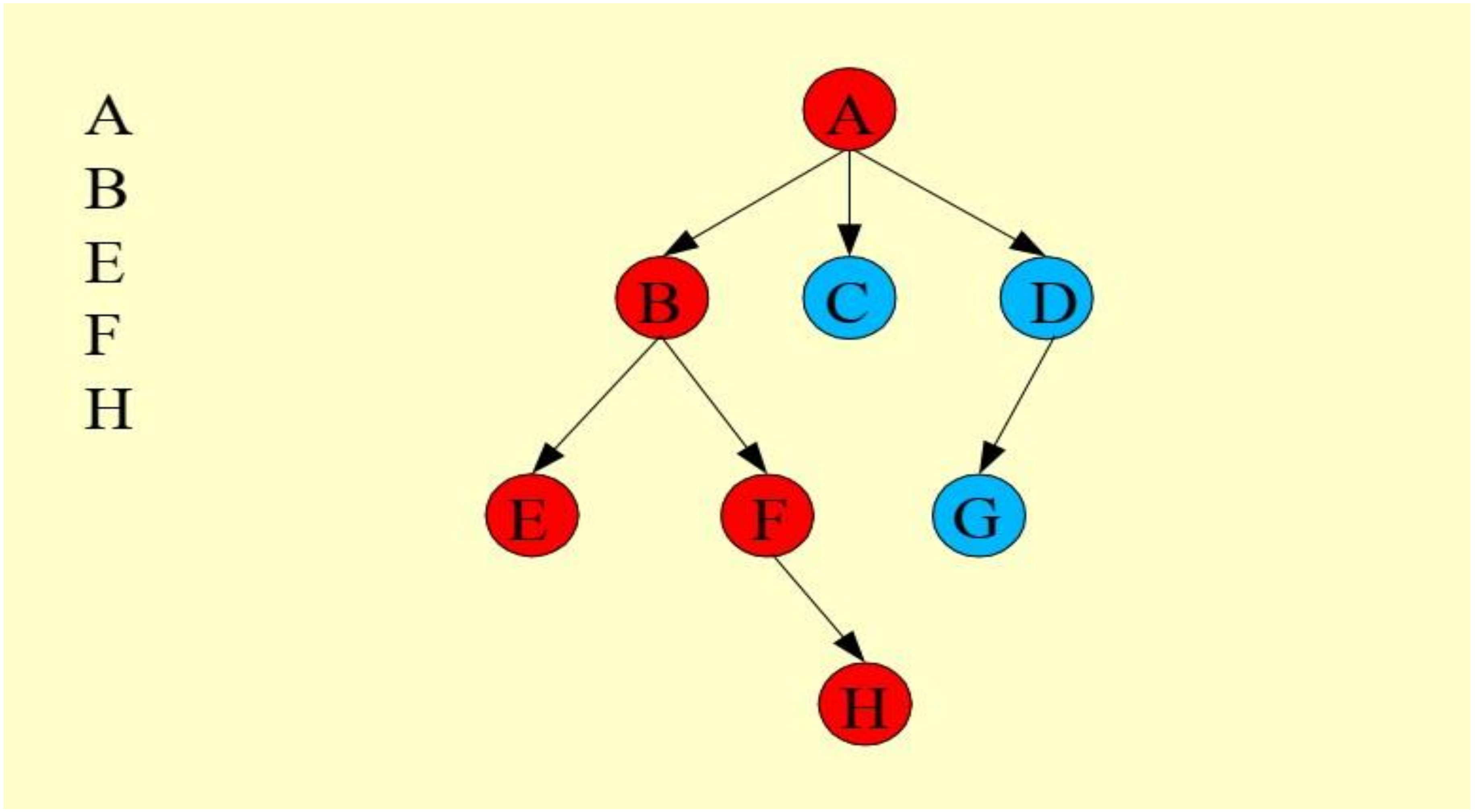
Cont'd…



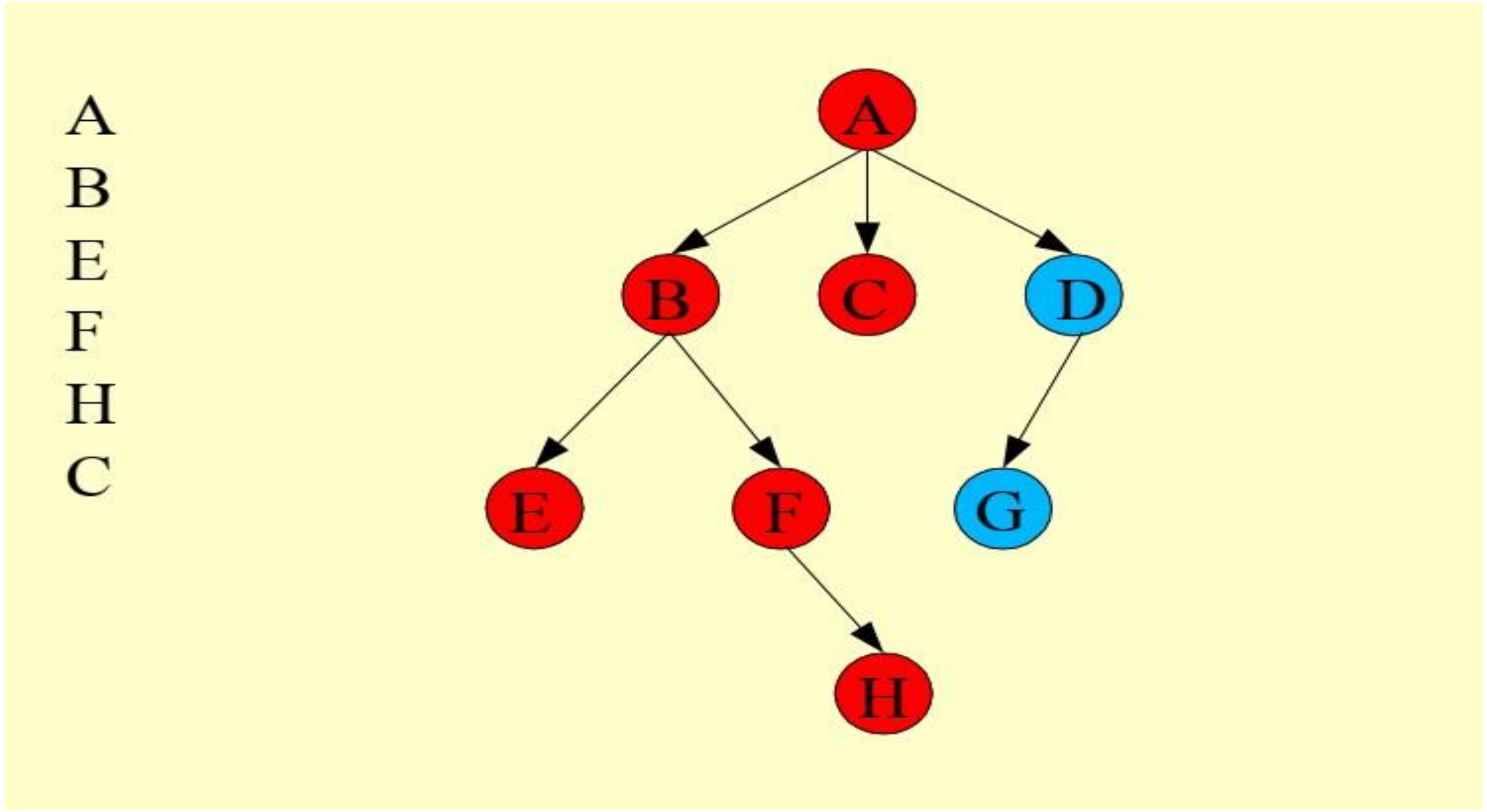
Cont’d…



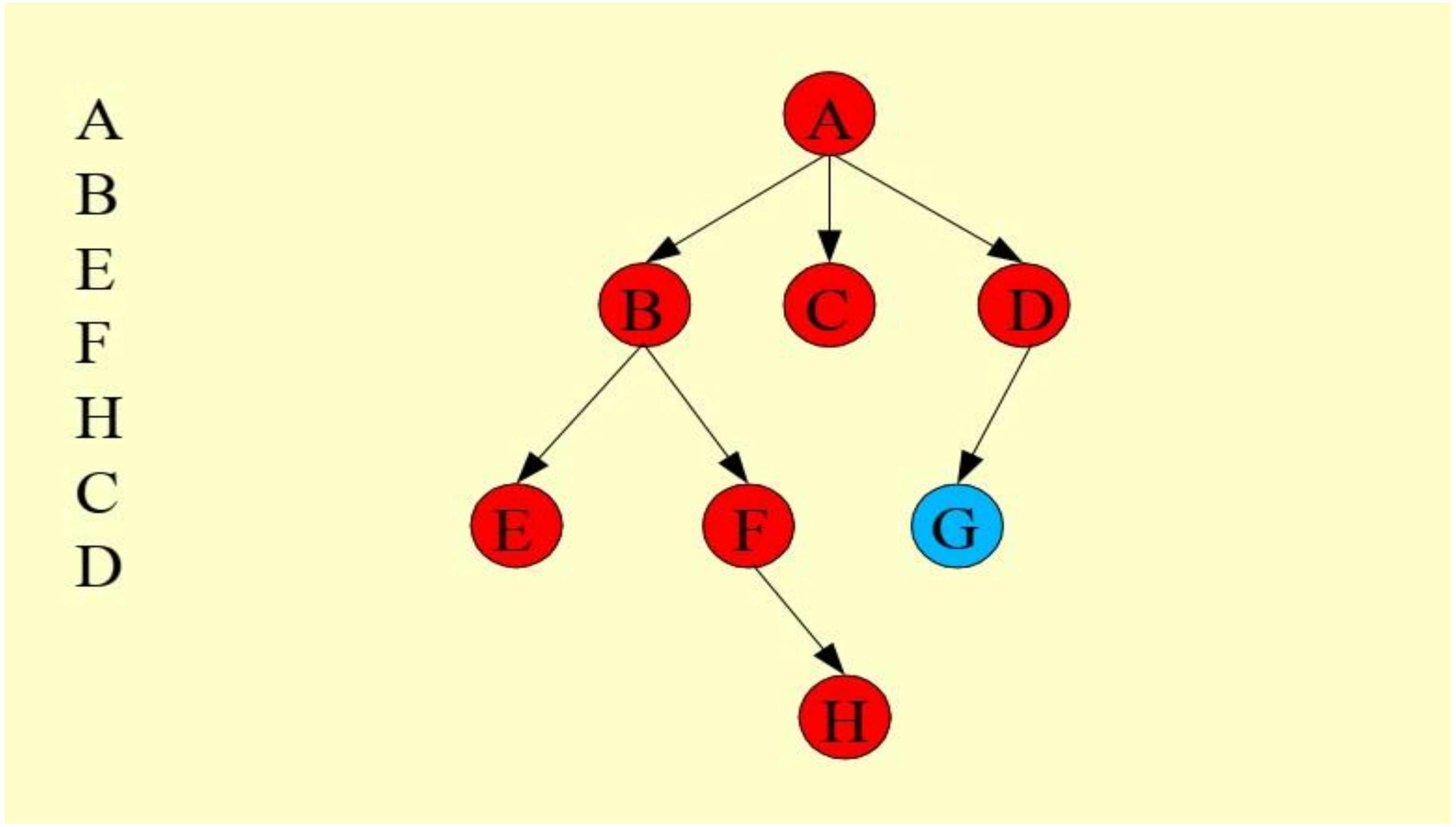
Cont'd…



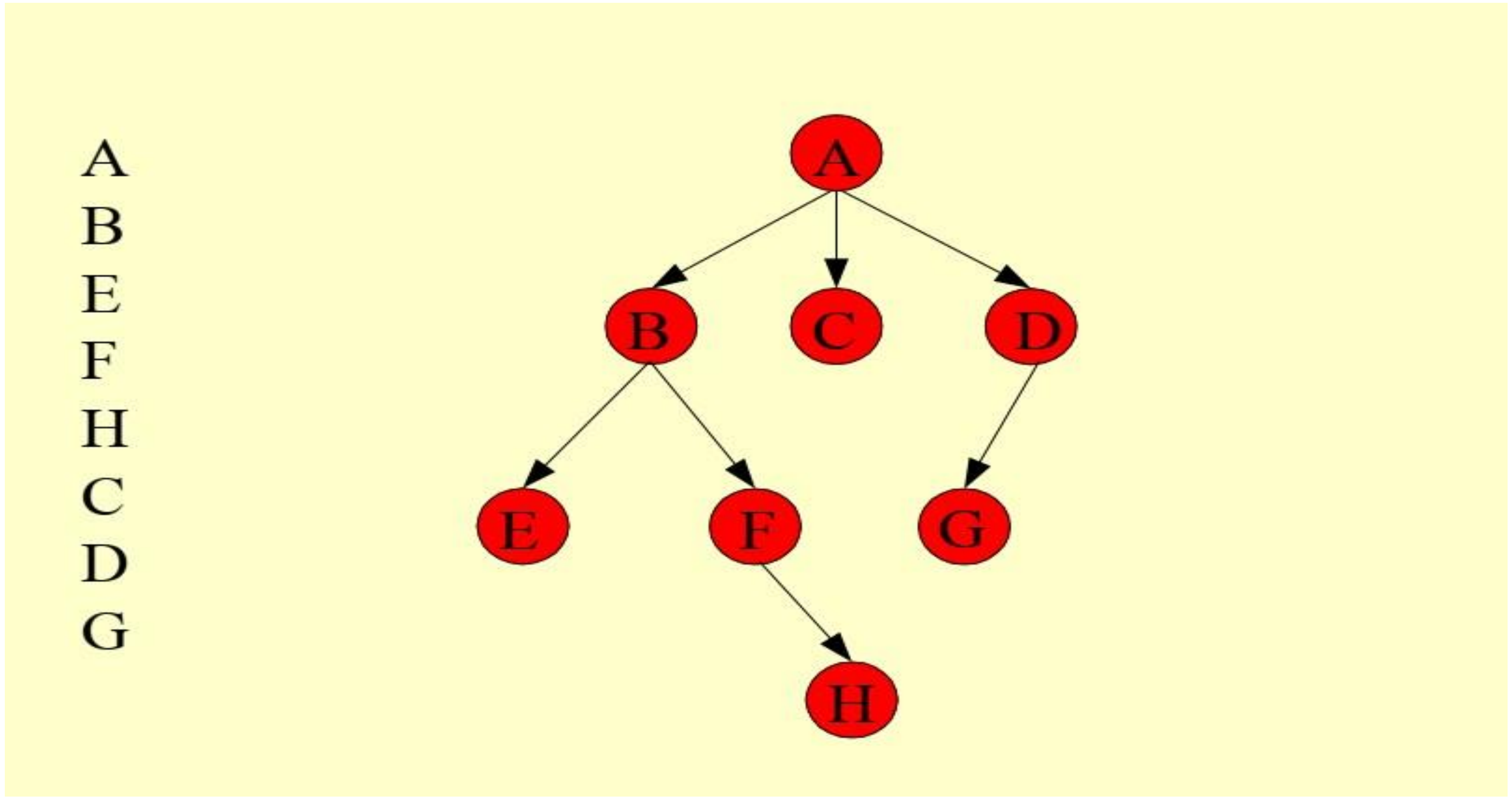
Cont’d…



Cont’d…



Cont’d…



Graph Traversal Algorithm Complexity

Time Complexity

- **BFS Vs DFS Time complexity :** O(V+E), when implemented using the adjacency list. Where V is number of vertices in the graph and E is number of edges in the graph.

- ***Space complexity* (memory)**

BFS :*Space complexity is O(|V|) as well - since at worst case you need to hold all vertices in the queue.*

Cont'd…

- **DFS**: Space Complexity depends on the implementation, a recursive implementation can have a O(h) space complexity [worst case], where h is the maximal depth of your tree. But Depth-first can waste time going down a "rabbit hole", when solution is high in tree.
- *Worst Case* for DFS will be the best case for BFS, and the *Best Case* for DFS will be the worst case for BFS.