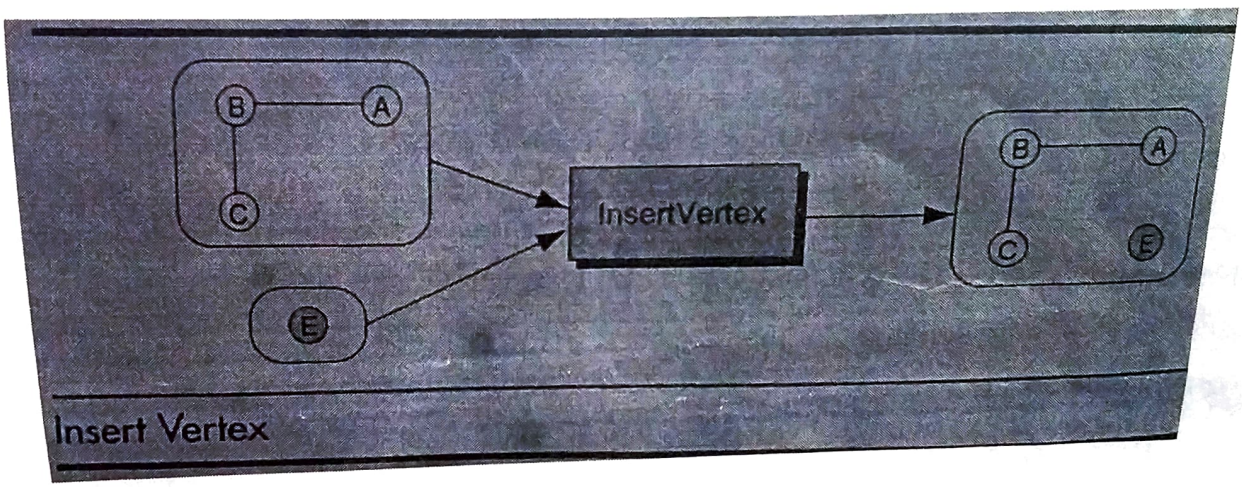
### 5.2.2 Operations

- Six primitive graph operations that provide the basic modules needed to maintain a graph

  - *Insert a vertex*
  - *Delete a vertex*
  - *Add an edge*
  - *Delete an edge*
  - *Find a vertex*
  - *Traverse a graph*

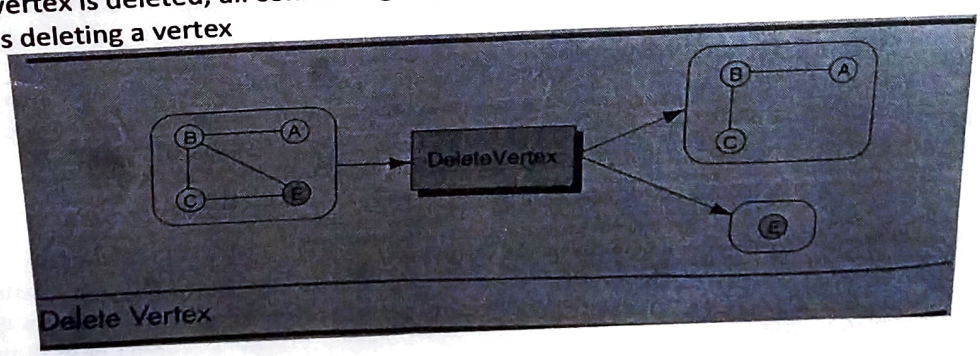
**Insert a vertex**

- It adds a new vertex to a graph
- When it is inserted, it is disjoint; that is, it is not connected to any other vertices in the list
- Inserting a vertex is just the first step in the insertion process
- After insertion, it must be connected
- Fig. shows a graph before and after a new vertex is added




**Delete a vertex**

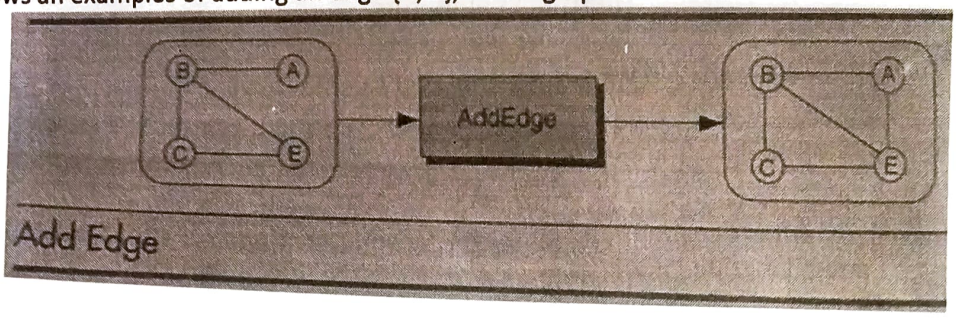
- Delete vertex removes a vertex from the graph
- When a vertex is deleted, all connecting edges are also removed
- Fig shows deleting a vertex




**Add edge**

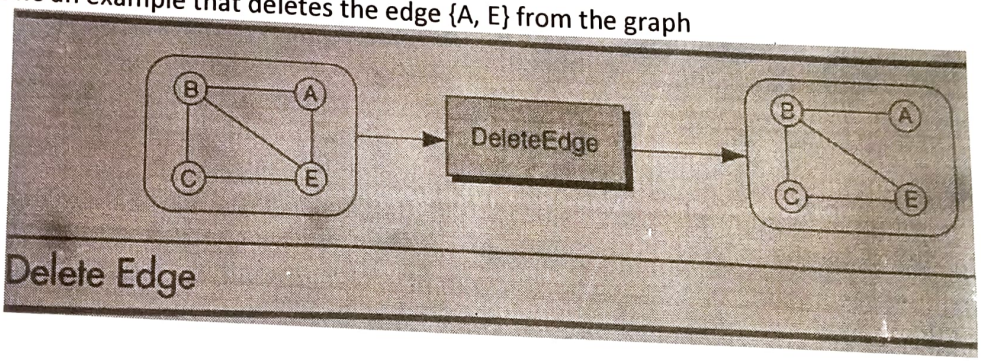
- Add edge connects a vertex to a destination vertex
- If a vertex requires multiple edges, add an edge must be called once for each adjacent vertex
- To add an edge, two vertices must be specified
- If the graph is a digraph, one of the vertices must be specified as the source and one as the destination

- Fig shows an examples of adding an edge {A, E}, to the graph



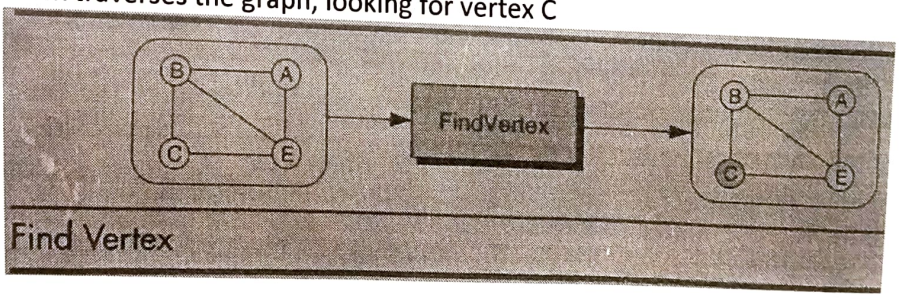

**Delete Edge**

- Delete edge removes one edge from a graph
- Fig. shows an example that deletes the edge {A, E} from the graph




**Find vertex**

- Find vertex traverses a graph, looking for a specified vertex
- If the vertex is found its data are returned
- If it is not found, an error is indicated
- In fig find vertex traverses the graph, looking for vertex C



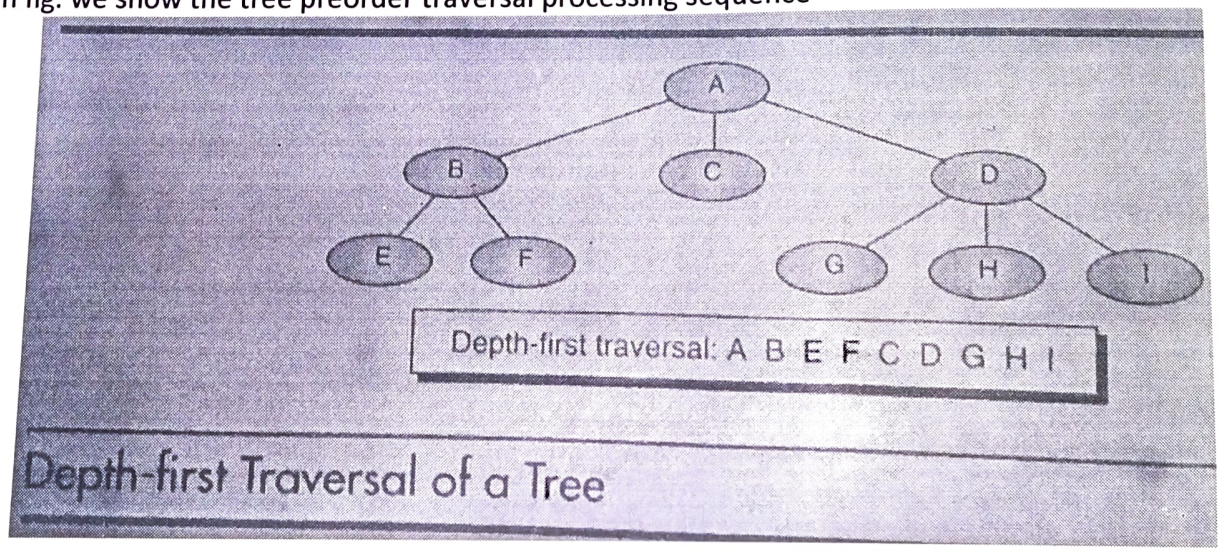

**5.2.3 Traverse graph**

- Atleast one application that requires that all vertices in a given graph be visited; that is, there is atleast one application that the graph be traversed
- Because a vertex in graph can have multiple parents, the traversal of a graph presents some problems not found in traversal of linear list and trees
- We must ensure that the vertex is visited only once
- The traditional solution is to include visited flag at each vertex
- Initially before traversal we set the visited flag in each vertex to off
- Then, as we traverse the graph, we set the visited flag to on to indicate that the data have been processed

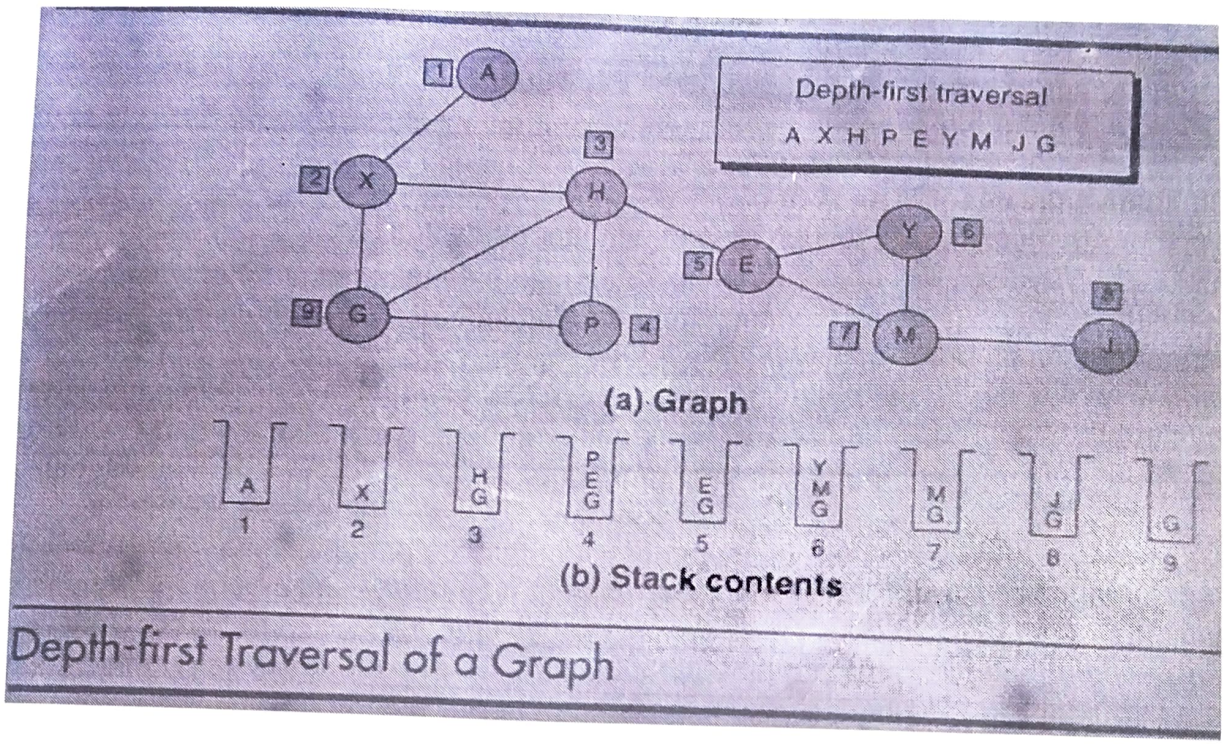
- Two standard graph traversals are:
    b. Depth first and
    c. Breadth first

**a. Depth first traversal**

- In the depth first traversal, we process all a vertex's descendents before we move to an adjacent vertex
- This is easy when the graph is a tree
- In fig. we show the tree preorder traversal processing sequence

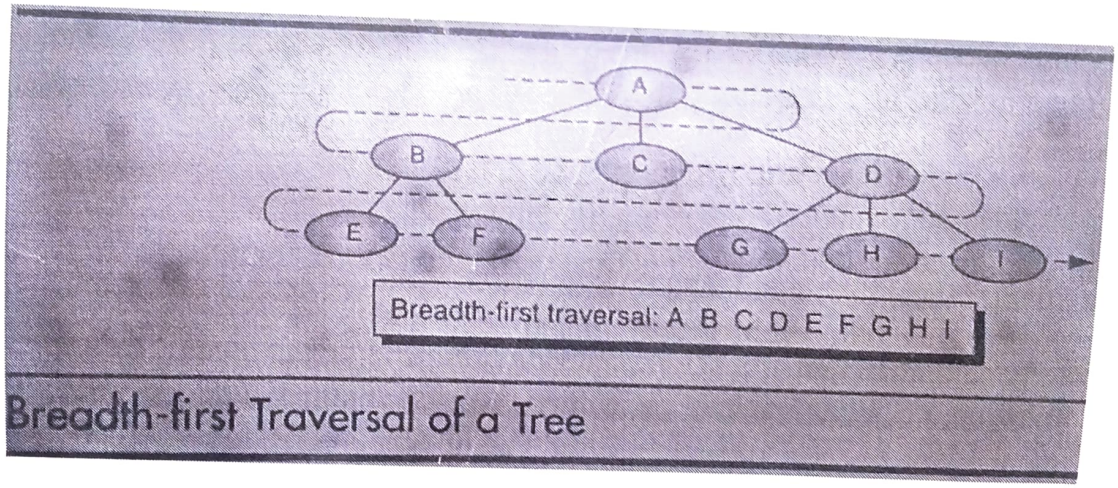



- The depth first traversal of a graph starts by processing the first vertex of the graph
- After processing the first vertex, we select any vertex adjacent to the first vertex and process it
- We process each vertex, we select an adjacent vertex until we reach a vertex with no adjacent entries
- The logic requires a stack (recursion) to complete the traversal
- The traversal processes adjacent vertices in descending, or last – in – first out (LIFO) order
- Trace a DFT through a graph in fig.
- The number in the box next to a vertex indicates the processing order
- The stacks below the graph show the stack contents as we work our way down the graph and then as we back out
    1. We begin by pushing the first vertex A, into the stack
    2. We then loop, pop the stack, and, after processing the vertex, push all of the adjacent vertices into the stack, process it and then push G and into the stack, giving the stack contents for step 3 as shown in fig (B) – HG
    3. When the stack is empty the traversal is complete

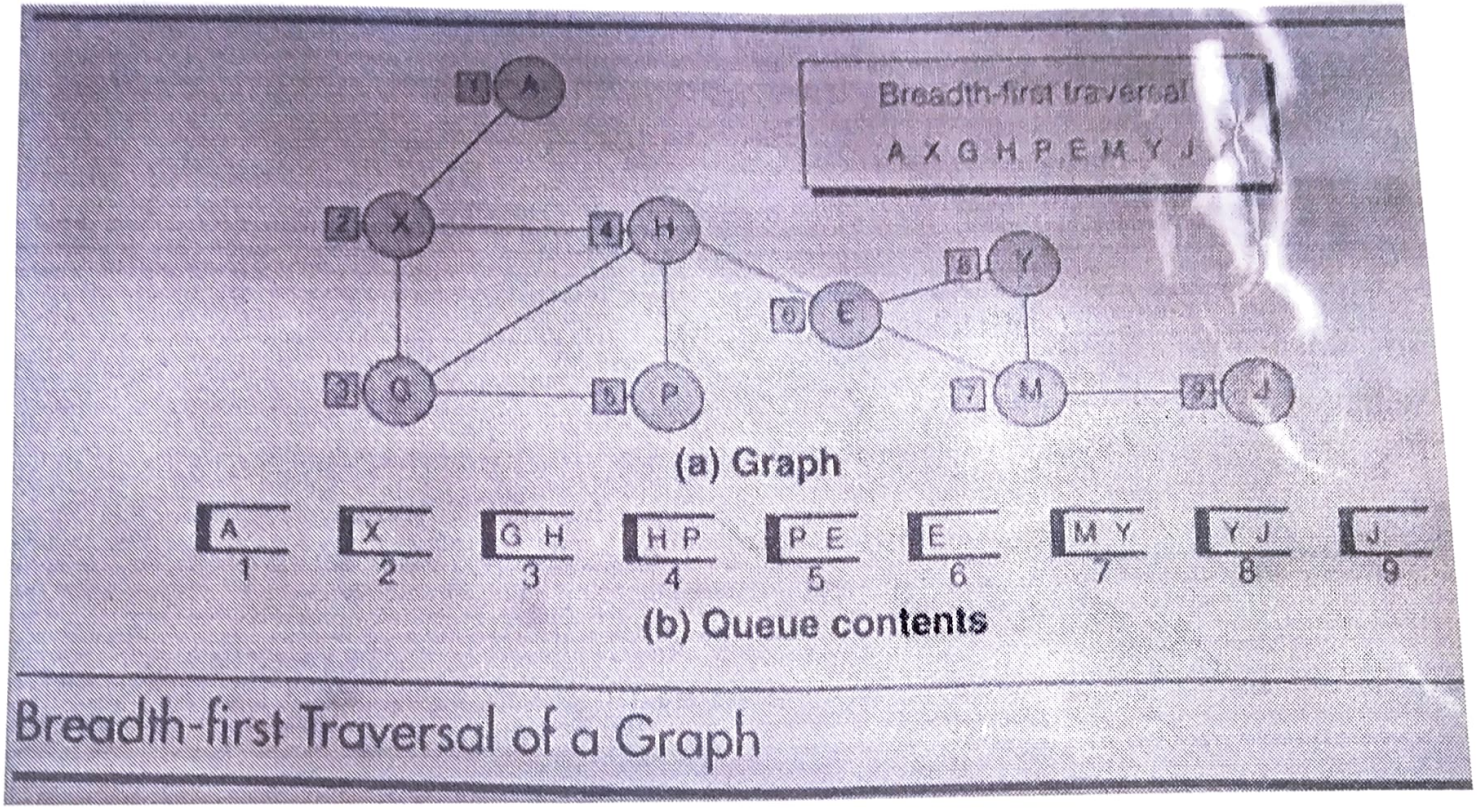



**Breadth first traversal**

- In the BFT of a graph, we process all adjacent vertices of a vertex before going to the next level
- Looking at the tree in fig, we see that its BFT starts at level 0 and then process all the vertices in level 1 before going to process the vertices in level 2




- The BFT of a graph follows the same concept
- We begin at starting vertex (A); after processing it we process all of its adjacent vertices (BCD)
- After we process all of the first vertex's adjacent vertices, we pick its first adjacent vertex(B) and process all of its vertices, and so forth until we are finished
- We show that the BFT uses a queue of its adjacent vertices in the queue
- Then select the next vertex to be processed, we delete a vertex from the queue and process it
- Lets trace the graph in fig.




Breadth first traversal of graph

1. We begin by enqueing vertex A in the queue
2. We then loop, dequeing the queue and processing the vertex from the front of the queue. After processing the vertex, we places of its adjacent vertices into the queue. Thus at step 2 in fig (b), we dequeue vertex X, process it, and then place vertices G and H in the queue. We are then ready for step 3, in which we process vertex G
3. when the queue is empty, the traversal is complete

In the BFT, all adjacent vertices are processed before processing the descendents of a vertex