CLASSIFICATION OF DATA STRUCTURES

Stacks

- A stack is a linear data structure in which insertion and deletion of elements are done at only one end, which is known as the top of the stack.
- Stack is called a last-in, first-out (LIFO) structure because the last element which is added to the stack is the first element which is deleted from the stack.
- In the computer's memory, stacks can be implemented using arrays or linked lists.
- Figure 2.3 shows the array implementation of a stack.
- Every stack has a variable top associated with it. top is used to store the address of the topmost element of the stack.
- It is this position from where the element will be added or deleted.
- There is another variable MAX, which is used to store the maximum number of elements that the stack can store.
- If top = NULL, then it indicates that the stack is empty and if top = MAX–1, then the stack is full.

A	B	C	D	E					
0	1	2	3	top = 4	5	6	7	8	9

Figure 2.3 Array representation of a stack

CLASSIFICATION OF DATA STRUCTURES

A	B	C	D	E					
0	1	2	3	top = 4	5	6	7	8	9

Figure 2.3 Array representation of a stack

Stacks

- In Fig. 2.3, top = 4, so insertions and deletions will be done at this position.
- Here, the stack can store a maximum of 10 elements where the indices range from 0–9.
- In the above stack, five more elements can still be stored.
- A stack supports three basic operations: push, pop, and peep.
- The push operation adds an element to the top of the stack.
- The pop operation removes the element from the top of the stack.
- And the peep operation returns the value of the topmost element of the stack (without deleting it).
- However, before inserting an element in the stack, we must check for overflow conditions.
- An overflow occurs when we try to insert an element into a stack that is already full.
- Similarly, before deleting an element from the stack, we must check for underflow conditions.
- An underflow condition occurs when we try to delete an element from a stack that is already empty.

CLASSIFICATION OF DATA STRUCTURES

Queues

- A queue is a first-in, first-out (FIFO) data structure in which the element that is inserted first is the first one to be taken out.
- The elements in a queue are added at one end called the rear and removed from the other end called the front.
- Like stacks, queues can be implemented by using either arrays or linked lists.
- Every queue has front and rear variables that point to the position from where deletions and insertions can be done, respectively.
- Consider the queue shown in Fig. 2.4.

Front					Rear				
12	9	7	18	14	36				
0	1	2	3	4	5	6	7	8	9

Figure 2.4 Array representation of a queue

CLASSIFICATION OF DATA STRUCTURES

Queues

- Here, front = 0 and rear = 5.
- If we want to add one more value to the list, say, if we want to add another element with the value 45, then the rear would be incremented by 1 and the value would be stored at the position pointed by the rear.
- The queue, after the addition, would be as shown in Fig. 2.5.
- Here, front = 0 and rear = 6.
- Every time a new element is to be added, we will repeat the same procedure.

Front						Rear			
12	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

Figure 2.5 Queue after insertion of a new element

CLASSIFICATION OF DATA STRUCTURES

Queues

- Now, if we want to delete an element from the queue, then the value of front will be incremented.
- Deletions are done only from this end of the queue.
- The queue after the deletion will be as shown in Fig. 2.6.

	Front					Rear			
	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

Figure 2.6 Queue after deletion of an element

CLASSIFICATION OF DATA STRUCTURES

Trees

- A tree is a non-linear data structure which consists of a collection of nodes arranged in a hierarchical order.
- One of the nodes is designated as the root node, and the remaining nodes can be partitioned into disjoint sets such that each set is a sub-tree of the root.
- The simplest form of a tree is a binary tree.
- A binary tree consists of a root node and left and right sub-trees, where both sub-trees are also binary trees.

CLASSIFICATION OF DATA STRUCTURES

Trees

- Each node contains a data element, a left pointer which points to the left sub-tree, and a right pointer which points to the right sub-tree.
- The root element is the topmost node which is pointed by a ‘root’ pointer.
- If root = NULL then the tree is empty.
- Figure 2.7 shows a binary tree, where R is the root node and T1 and T2 are the left and right subtrees of R.
- If T1 is non-empty, then T1 is said to be the left successor of R.
- Likewise, if T2 is non-empty, then it is called the right successor of R.

CLASSIFICATION OF DATA STRUCTURES

Trees

- In Fig. 2.7, node 2 is the left child and node 3 is the right child of the root node 1.
- Note that the left sub-tree of the root node consists of the nodes 2, 4, 5, 8, and 9.
- Similarly, the right sub-tree of the root node consists of the nodes 3, 6, 7, 10, 11, and 12.

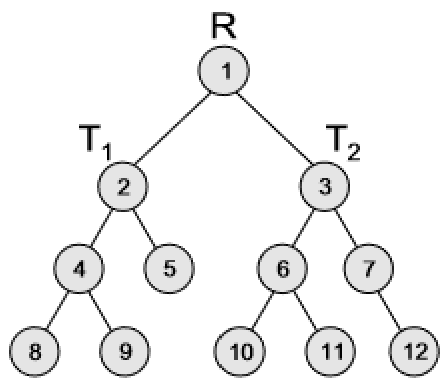

Figure 2.7 Binary tree

CLASSIFICATION OF DATA STRUCTURES

Graphs

- A graph is a non-linear data structure which is a collection of vertices (also called nodes) and edges that connect these vertices.
- A graph is often viewed as a generalization of the tree structure, where instead of a purely parent-to-child relationship between tree nodes, any kind of complex relationships between the nodes can exist.

CLASSIFICATION OF DATA STRUCTURES

Graphs

- In a tree structure, nodes can have any number of children but only one parent, a graph on the other hand relaxes all such kinds of restrictions.
- Figure 2.8 shows a graph with five nodes.
- A node in the graph may represent a city and the edges connecting the nodes can represent roads.
- A graph can also be used to represent a computer network where the nodes are workstations and the edges are the network connections.