**DATA STRUCTURES**

1. **Linked List**

***Single Linked List***

1. Creating a node.
2. Initializing
3. Making Relationships- creating linked list
4. Navigating
5. Inserting node between two nodes, Inserting node at end or at beginning.
6. Append node, traverse till end of list and add node. If list is not present, itself will act as first node.

***Double Linked List***

|NULL|value|ForwardPointer < = > |BackwardPointer|value|ForwardPointer < = > |BackwardPointer|value|NULL|

**1. Creating a structure:**

struct node

{

int val;

struct \*next;

struct\*prev;p

}NODE;

**2. Intializing values**

NODE N1,N2,N3;

N1.val=10;

N1.next=NULL;

N1.prev = NULL;

N2.val=10;

N2.next=NULL;

N2.prev = NULL;

N3.val=10;

N3.next=NULL;

N3.prev = NULL;

**3. Make relationships**

N1.next=&n2;

N2.next=&n3;

N2.prev=&n1;

N3.prev=&n2;

**4. Traversing**

Forward

head = &n1;

printf(“%d”,head->val);

head = head ->next;

Backward

head = head -> prev;

printf(“%d”,head->val);

***Circular Linked List***

|lastnodePointervalue|ForwardPointer < = > |BackwardPointer|value|ForwardPointer < = > |BackwardPointer|value|firstnodePointer|

nn->prev = head

nn->next = NULL

nn->next = temp

temp->prev = nn

1. **STACK**

Stack will grow from bottom to top.

When Bottom = = TOP ------ >>>> container is empty.

When TOP == Limit ----- >>>>>> container is full.

Stack size should be fixed.

While inserting into stack – need to know stack is full or not.

You can insert only when stack is not full. ---- PUSH

While deleting into stack - need to know stack is empty or not.

You can delete only when stack is not empty. ---- POP

1. **QUEUE**

If front = rear -- >> Empty queue

If rear = Max -- >> Queue is full

Inserting into queue --- Enqueue

Deleting from queue --- Dequeue

Display elements 0(Front) to top(Rear)

Data structures divided into two types:

**Primitive data structures**(int,float,char) – allows to store only single data type values.

And

**Non-Primitive data structures**(array,list,files). -allows to store multiple data type values.

*In list, there will be linear and non-linear:*

1. **Linear data structure**: data elements are arranged sequentially or linearly.

: Arrays,Linkedlists, stacks, queues.

1. **Linear data structure**: data elements are not arranged sequentially or linearly.

: Graphs, Trees.

Time complexity, space complexity are the basic design goals.

T(n):

1. n^(0) – O(1) --- if a step execute only once
2. n^\*d – O(n) --- if a step is repeated

Space complexity – the amount of memory it needs to run to completion.

O(1) – push pop

O(N) – iterating through loop

O(N^2) – nested loops

O(log N) – Logarithmic Time: Binary Search

O(N logN) – IN sorting algoritms: quick sort, merge sort

**TREE**

Nonlinear data structure like graph. It is a special form of graph having only one path between any nodes. Hierarchical data structure.

Binary Tree – A rooted tree in which no vertex has more than two children. Structure of it contains (data in the node, left child and right child).

*Nomenclature of tree*

1. Degree of a node – number of children of a root.
2. Level – levels grow from root to leaf. Root will have level 1.
3. Node – Number of nodes
4. Leaf – which do not have child
5. Sibling – same parent
6. Ancestor – parnet’s parent
7. Degree of tree – which has more number of children
8. Non Terminal

Full Binary tree or Strict Binary Tree – parent should have 0 or 2 children.

Perfect Binary Tree – if all its internal nodes have exactly 2 children. All leaf nodes are on the same level.

Complete binary tree – all nodes should be mostly filled on left side. ?? doubt

**Depth First Search**

Inorder (LNR)

Preorder (NLR)

postOrder(LRN)

**Binary Search Tree –** level order traversal

Insert ele 5,10,3,4,8,1,2,13

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 5 |  |  |  |  |  |
|  |  | 3 |  |  |  | 10 |  |  |  |
| 1 |  |  | 4 |  | 8 |  | 13 |  |  |
|  | 2 |  |  |  |  |  |  |  |  |

Logic: --------------------

Element greater than root will be placed on right side, or if element less than root will be placed on left side. Otherwise, if same element appeared this duplicate element can be ignored.

So to find for particular element, we need to traverse, it can be done by Depth First Search.

EX: 5,10,3,4,8,1,2,13

InOrder: (LNR)

1, 2, 3, 4, 5, 8, 10, 13 – sorted in accending order

PreOrder: (NLR)

5, 3, 1, 2, 4, 10, 8, 13

PostOrder: (LRN)

2, 1, 4, 3, 8, 13, 10, 5

Delete a node in BST

1. Delete a leaf node
2. Delete a node with one child
3. Delete a node with two children