DS. Data Structures

T: **Arrays.**

→ Contiguous & Static.

- One dimensional Array:

A [lb .... ub] → Array.

$$\boxed{Loc(A[i]) = L_o + (i - lb) * c}$$

$$\boxed{\text{Size of Array} = ub - lb + 1}$$

- Two dimensional Array:

$A[\underbrace{b_1 \cdots u_1}_{Row}, \underbrace{b_2 \cdots u_2}_{col}]$

Rows = $u_1 - b_1 + 1$
Cols = $u_2 - b_2 + 1$

* 2 methods of storing : {CMO, RMO}.

- RMO:

$$\boxed{Loc(A[i,j]) = Base + ((i-b_1)*(u_2-b_2+1) + (j-b_2))*c}$$

- Skip $(i-b_1)$ Rows
- Skip $(j-b_2)$ elements.

- CMO:

$$\boxed{Loc(A[i,j]) = Base + ((j-b_2)*(u_1-b_1+1) + (i-b_1))*c}$$

- Skip $(j-b_2)$ Cols
- Skip $(i-b_1)$ elements.

- Sparse matrix:

Most of the elements are 0 or NULL.

- upper Triangular matrix
- Lower Triangular matrix
- Tri-diagonal matrix
- Toeplitz matrix

Upper Triangular : $a_{ij} = 0 \ (i > j)$

Lower Triangular : $a_{ij} = 0 \ (i < j)$

Trick to solve array Qsn ⇒ Take small values of (n), & draw example, then try to match from option, or derive from them.

- **Upper Triangular Matrix:**

$$
\begin{array}{c|cccc|l}
 & 1 & 2 & 3 & 4 & \\
1 & a_{11} & a_{12} & a_{13} & a_{14} & \to (n) \text{ elements} \\
2 & 0 & a_{22} & a_{23} & a_{24} & \to (n-1) \text{ elements} \\
3 & 0 & 0 & a_{33} & a_{34} & \to (n-2) \text{ element} \\
4 & 0 & 0 & 0 & a_{44} & \to 1 \text{ element} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 1 & 2 & 3 \cdots & n & 4\times 4
\end{array}
$$

- RMO:

$$(i,j) \Rightarrow \underbrace{n + (n-1) + \cdots + (n-i+2)}_{\text{Before reaching 'i'th Row.}} + \underbrace{(j-i)}_{\text{in 'i'th Row}}$$

$$\Rightarrow n(i-1) - (1+2+3+\cdots(i-2)) + (j-i)$$

$$\boxed{(i,j) \Rightarrow n(i-1) - \frac{(i-2)(i+1)}{2} + (j-i)}$$

↳ this is for the array: $A[1 \cdots n]$

NOTE

Let the array be $A[0 \cdots n-1]$ (Like in C)

then,

$$\boxed{\begin{aligned}(i,j) &= n(i) - \frac{i(i-1)}{2} + (j-i) \\ &= n(i-lb) - \frac{(i-lb-1)(i-lb)}{2} + (j-i)\end{aligned}}$$

- CMO:

$$(i,j) \Rightarrow \underbrace{\frac{j(j-1)}{2}}_{\text{Skip } (j-1) \text{ Col.}} + \underbrace{(i-1)}_{\text{in the } j^{th} \text{ Col.}}$$

$$\boxed{(i,j) = j(j-1)/2 + (i-1)}$$

* Same can be done for $A[0 \cdots n-1]$ or $A[l \cdots u]$.

- **For Lower Triangular Matrix:**

RMO[Upper] = CMO[Lower] when $[i \leftrightarrow j]$

CMO[Upper] = RMO[Lower] when $[i \leftrightarrow j]$

T: HASHING.

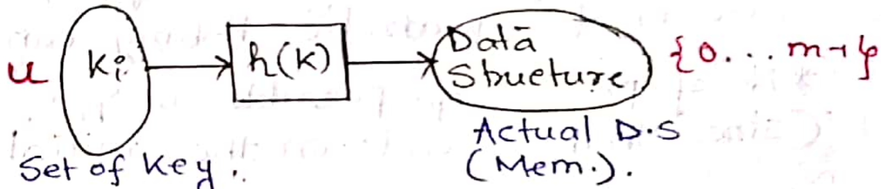
Searching time of some DS are :
(i) unsorted array → $\Theta(n)$
(ii) sorted array → $\Theta(\log n)$
(iii) Linked List → $\Theta(n)$
(iv) Binary Tree → $\Theta(n)$
(v) Binary Search Tree → $\Theta(n)$
(vi) Balanced BST → $\Theta(\log_2 n)$
(vii) Priority Queue (Heaps) → $\Theta(n)$.

⊛ Hashing decreases Time complexity, but as a result space complexity increases Rapidly.

* if we use hashing then on avg : $\Theta(1)$ time.

• **Direct Addr Table :**
Basic concept is to put : $\boxed{\text{(key) in A[key]}}$.
* Limitation : If say List = {1000006, 1000000}
∴ A[1000006] = 1000006;
* 1M space Reqd for just 2 numbers.
* only use when Range is very small.

• **Hashing :**



Generally : $\boxed{h(K) \simeq K \bmod m.}$

• **Collision Problem :**
* when 2 or more Keys map onto the same place ie $[h(K_i) = h(K_j)]$ then collision.

Solution :
(i) Better hash fn.
(ii) Chaining (good for key deletion)
(iii) Open addrsing (bad for deletion).
 • Linear Probe
 • Quadratic Probe
 • Double Hashing.

Load factor
$$\boxed{\alpha = \frac{n \rightarrow \#\text{elements}}{m \rightarrow \text{Size of Table}}}$$

• **Chaining :**
Insertion → $\Theta(1)$
Search → $\Theta(n)$
Deletion → $\Theta(n)$

* Avg Search = $\Theta(1+\alpha)$.

* deletion is easy in chaining.
* pointers Reqd (more Space)

- **Linear Probing:**

hash fn: $h: \{K_1, K_2 \cdots K_n\} \rightarrow \{0, 1, 2, \cdots m-1\}$.

$$\boxed{h(k, i) = (h(k) + i) \bmod m.}$$

Probe Seq: $h(k,0) \xrightarrow{Col.} h(k,1) \xrightarrow{Col} h(k,2) \cdots$ so on.

possible probe seq.:

$$\left.\begin{array}{l} 0, 1, 2, \cdots m-1 \\ 1\ 2 \cdots m-1, 0 \\ 2\ 3 \cdots m-1, 0, 1 \\ \vdots \\ m-1, 0\ 1, \cdots m-2 \end{array}\right\}$$

∴ Total of 'm' probe seq. are possible.

* Suffers from ⇒
- primary clustering
- Secondary clustering.

- **Quadratic Probing:**

$$\boxed{h'(k) = (h(k) + c_1 i + c_2 i^2) \bmod m.}$$

* depending on the values of c_1, c_2 & 'm', success of quadratic Probing can be judged.

* No of probe seq. possible is 'm'.
(Since, that depends on the initial value of m).

- **Double Hashing:**

* In double hashing, # probe seq. possible = m^2.

* doesn't ~~support~~ suffer from both primary & secondary clustering.

$$\boxed{h'(k) = (h_1(k) + i\, h_2(k)) \bmod m.}$$

- **What is Secondary Clustering.?**

* if two keys happen to have same initial probe, then the rest of the probe seq. will be same.

$K_1 \rightarrow (1, 3, 7, \cdots)$

$K_2 \rightarrow (1, 3, 7, \cdots)$.

Tree.

- **Binary Tree**: Each node having atmost 2 children.
- **Full Binary Tree**: All Levels are full.
- **Complete Binary Tree**: All Levels are full, except possibly the last level which is a left as possible.
- **K-ary Tree**: Internal node having K children.
  [All Levels need not be filled].
- **Important Reln in K-ary Tree.**

$$\boxed{i+l = ik+1}$$

($+1$ ← Root)

$\Rightarrow l = i(K-1)+1.$

$K \rightarrow$ deg. of internal node
$i \rightarrow$ no of internal node.
$l \rightarrow$ no of leaf nodes.

- **Reln b/w Leaf node & internal node in Binary Tree:**

Let $n_1 \Rightarrow$ internal nodes w/ 1 child.
Let $n_2 \Rightarrow$ internal nodes w/ 2 children.

∴ We can say:

$$\boxed{2*n_2 + 1*n_1 + 1 = n_1+n_2+L.}$$

$\Rightarrow 2n_2+1 = n_2+L$

$\Rightarrow \boxed{n_2+1 = L}$

this is valid for any Binary Tree.

- **Basic formulas:**

• $n \Rightarrow$ no of nodes. • $h \Rightarrow$ height • d = depth.
• b = branch factor ($b=2$ ∴ Binary Tree).

① For Complete Binary Tree:

$$\boxed{2^h \leq n \leq 2^{h+1}-1}$$

* Taking $\log_2$ on both sides.

$\log 2^h \leq \log n \leq \log 2^{h+1}-1$

$h \leq \log_2 n$

$\boxed{h = O(\log n)}$ → Complete, Full, Balanced.

otherwise $O(n)$.

② For full Binary Tree: $\boxed{n = 2^{h-1}+1}$

③ Sum of ht of all nodes is bounded by $O(n)$.

* $\boxed{h_{min} = \log_2 n \; ; \; h_{max} = n-1}$

- **Rank of a Node :** It is defined as . . .

$$\boxed{\text{Rank}(i) = (\text{No of nodes in Left SubTree}) + 1}$$

- **Array Representation :**

Root ← 1 LC → $2i$

Parent ← $\lfloor i/2 \rfloor$ RC → $2i+1$.

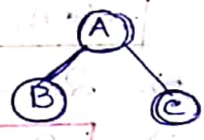
◦ $\boxed{\begin{aligned} \text{Max Array Size} &= 2^n - 1 \quad (\text{not complete / full}) \\ &= 2^{\lceil \log_2 (n-1) \rceil} - 1 \quad (\text{full / comp.}) \end{aligned}}$

- **No of Tree (Binary) Possible to 'n' nodes :**

for unlabelled nodes → $\boxed{\frac{1}{n+1}\binom{2n}{n}} = C_n$

for labelled → $\boxed{\frac{1}{n+1}\binom{2n}{n} * n!} = C_n * n!$

- **Tree Traversals :**
  - in Order. (L - Root - R) (BAC)
  - pre Order (Root - L - R) (ABC)
  - post Order (L - R - Root). (BCA)



* NOTE

in order

pre order

post order.

* Time Complexity → $\boxed{T(n) = O(n)}$

* **Double Order :**

```
DO(t) {
   if (t) {
      print(t);
      DO(t→L);
      print(t)
      DO(t→R);
   }
}
```

→ inorder
→ pre order

* **Triple Order :**

```
TO(t) {
   if(t) {
      print(t);
      TO(t→L);
      print(t);
      TO(t→R);
      print(t);
   }
}.
```

→ pre order
→ post order
→ inorder.

- ## Binary Search Tree.

For every node, the keys in Left subTree are less than the key of current & the right subTree nodes are greater.

* In BST, inorder always gives sorted order.

NOTE

No of binary Tree possible ← $\boxed{\frac{1}{n+1}\binom{2n}{n}}$

* if structure is given, ← only 1 BST possible.

- ## Insertion:

if skewed ⇒ $T(n) = O(n)$. → Worst Case.

if Balanced ⇒ $T(n) = O(h)$

$= O(\log n)$.

- ## Deletion:

(i) No child → Just Remove.

(ii) One child → delete the node, & make conn. w/ grand parent of of the disconnected node.

(iii) Two child ⇒ 2 methods are there:

(a) Replace w/ Inorder successor.

(b) Replace w/ Inorder predecesor.

* Inorder Successor

min element in Right SubTree.

* Inorder Predecessor.

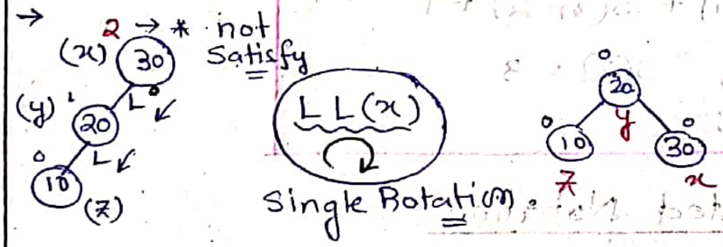
Greatest element in Left SubTree.

- ## AVL Tree (Adelson Velsky Landis):

Height Balanced Binary Search Tree.

$\boxed{\text{Balance Factor} : h_L - h_R}$ * (-1, 0, 1) acceptable B.F.

→



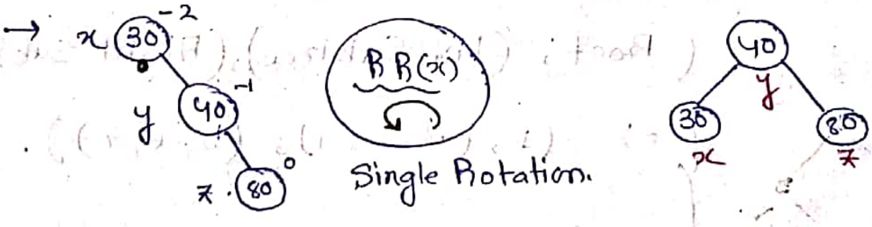
Single Rotation.

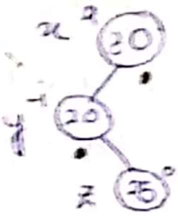
$\boxed{x \leftrightarrows y \quad z = \text{no change}}$

↑

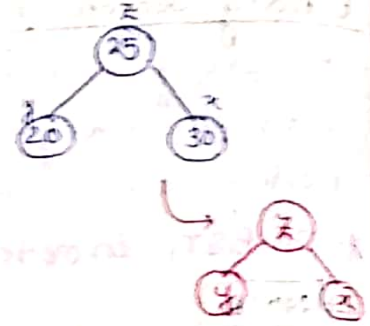
Shortcut.

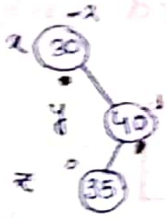
→



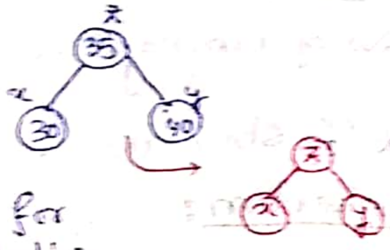


RL(x)
double Rotation





LR(x)
double Rotation.



$\boxed{\begin{array}{l} x \Leftrightarrow z \\ y \text{ unchanged} \end{array}}$ → Shortcut for single Rotation

⇒ After inserting an element, we travel from newly inserted node towards root, and ~~verify~~ verify if any node is imbalanced.
If imbalance is there, then go from imbal. node to newly inserted node, then we understand what type of imbalance occurs.

- Operations on AVL Tree:
  - insertion / deletion of one element = $O(\log_2 n)$
  - Construction w/ 'n' elements = $O(n \log_2 n)$.
- Min/Max nodes in AVL Tree:

* height will be given = h.

Let $n(h)$ = max nodes in AVL w/ h ht.

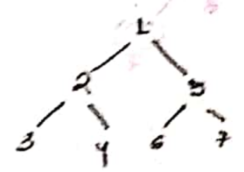
$$\therefore \boxed{\begin{array}{l} n(h) = 2n(h-1) + 1 \\ \quad \rightarrow 2^{h+1} - 1 \end{array}}$$

Let $s(h)$ = min nodes in AVL ht 'h'.

$$\boxed{\begin{array}{l} s(h) = s(h-1) + s(h-2) + 1 \\ s(0) = 1 \; ; \quad s(2) = 3 \\ s(1) = 2 \; ; \end{array}}$$

\# NOTE ⇝ Nested Notation:

Notation : (Root , (Left SubTree), (Right SubTree



⇒ (1, (2, 3, 4), (5, 6, 7)).

T8 **Stacks & Queue:**

- **Stack**

→ Linear Data Structure.

→ (LIFO, FILO)

* Orderd List
* Insertion & deletion done at one end. (TOP)

Applications:

→ Stack Permutation

→ Infix to Prefix

→ Infix to Postfix.

→ Expression Tree.

→ Recursion.

- **Stack Permutation:**

Push elements in given order, the pop elements in any order at any time.

$$\boxed{\text{Total no of stack Permutation} = \frac{1}{n+1}\binom{2n}{n}}$$

- **Recursion:**

• $T(n) \leftarrow$ no of func. calls for $f(n)$.

• $S(n) \leftarrow$ max. depth of runtime stack.

* tail Rec → Rec. at Last stmt.
* head Rec → Rec at first stmt.
* head-tail → Rec both at 1st & Last.

- **Ackermann function:**

$$A(x,y) = y+1 \quad : x=0$$
$$= A(x-1, y) : y=0$$
$$= A(x-1, A(x, y-1)) \quad \text{otherwise.}$$

Nested Recursion

Shortcut:

$$\boxed{\begin{aligned} A(0,y) &= y+1 \\ A(1,y) &= y+2 \\ A(2,y) &= 2y+3 \\ A(3,y) &= 2^{y+3}-3. \end{aligned}}$$

- **Tower of Hanoi.**

```
TOH (n, L, M, R) {
    if (n = 1)  L → R;
    else {
        TOH(n-1, L, R, M); ← Last 2 : (3rd, 4th ↔)
        L → R          ← (2nd → 4th);
        TOH((n-1), M, L, R); ← (2nd ↔ 3rd);
    }
}
```

$$\boxed{\begin{aligned} T(n) &= 2T(n-1)+1. \\ &= O(2^n) \end{aligned}}$$

$\boxed{\text{NOTE}}$

No of fn call : (2^n-1)

After how many fn call 1st move occurs : n.

After how many fn call Largest disk placed : $n+1$.

<u>Infix, Postfix, Prefix :</u>

infix ← operator in b/w operands (a+b)

Postfix ← operator after operands (ab+)

Prefix ← operator before operands. (+ab) .

Data Structure.	Time Complexity.								Space
	Average				Worst				Worst.
	Access	Search	Insert	Delete	Acc.	Search	insert	delete	
Array (Un)	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$O(1)$	$O(n)$	$O(1)$	$O(n)$.	$O(n)$
Array (Sorted)	$\Theta(1)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$	$O(1)$	$O(\log_2 n)$	$O(n)$	$O(n)$.	$O(n)$.
Stack.	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$.
Queue.	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Linked List.	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Hash Table	–	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	–	$O(n)$	$O(n)$	$O(n)$	$O(n)$
BST	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL Tree.	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log(n))$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$.

Scanned with CamScanner