DATE 02/08/2019

Data Structure:

The mathematical or logical model of a particular organization of data is called data structure.

Operations On a Data Structures:

1) Traverse
2) Search
3) Insertion
4) Deletion
5) Sorting
6) Merging.

Types Of Data Structures:

Primitive	Non-Primitive
- Integer	- Arrays
- Real	- Linked list.
- Char	
- Boolean	

They can be further classified as

1. Linear: Arrays, Linked list, stack, Queues.
2. Non-Linear: Graphs, Trees.

Be Positive...

DATE / /

* Linear Search Algorithm for an array.

LINEAR (DATA, N, ITEM, LOC).
↑

1. DATA [N+1] := ITEM
2. Set LOC := 1
3. Repeat while DATA [LOC] ≠ ITEM.
   LOC := LOC + 1
4. If LOC = N+1, Set LOC := 0
5. Exit.

* Binary Search Algorithm for an array.

BINARY (DATA, N, ITEM, LOC)
↑

1. BEG := 1.
2. END := N
3. MID := INT((BEG + END)/2)
4. Repeat while BEG ≤ END and DATA[MID] ≠ ITEM
   If ITEM < DATA[MID].
   Set END := MID - 1
   else, Set BEG := MID + 1

   MID := INT((BEG + END)/2).

5. If DATA[MID] = ITEM
   LOC := MID
   Else LOC := NULL
6. Exit.

DATE / /

* Sparse Matrices:

* Triangular Matrix:

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$n = f(j,k)$

$= \frac{j(j+1)}{2} + k.$

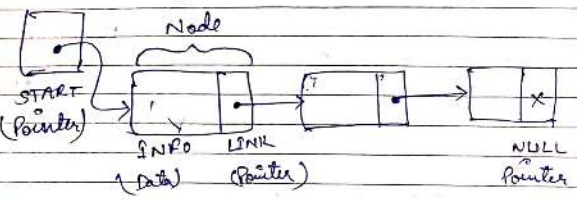
* Tridiagonal Matrix:

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix}$$

$3(j-2) + 2 + k - j + 2.0$

DATE / /

* **Linked List:** is a linear collection of data elements called nodes, where the linear order is maintained by pointers.



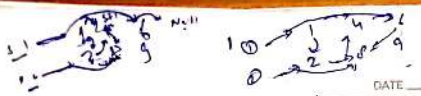
* TRAVERSE (INFO, LINK, START).
1. PTR := START
2. Repeat Steps 3 & 4 while PTR ≠ NULL
3. Apply PROCESS to INFO[PTR].
4. PTR := LINK[PTR].
5. Exit.

* SEARCH (INFO, LINK, START, LOC, ITEM)
1. PTR := START
2. Repeat Steps 3 & 4 while PTR ≠ NULL
3. If INFO[PTR] = ITEM
   LOC := PTR and Exit
   Else
   PTR := LINK[PTR].
4. LOC := NULL
5. Exit

DATE / /

* INSERT_BEG (INFO, LINK, START, ITEM, AVAIL).
1. If AVAIL = NULL
   Write "Overflow" and Exit
2. NEW := AVAIL
3. AVAIL := LINK[AVAIL].
3. INFO[NEW] := ITEM
   LINK[NEW] := START
4. START := NEW.
5. Exit

* INSERT (INFO, LINK, START, ITEM, LOC, AVAIL)
1. If AVAIL = NULL
   Write "Overflow" and Exit
2. NEW := AVAIL
   AVAIL := LINK[AVAIL].
3. INFO[NEW] := ITEM
   LINK[NEW] := LINK[LOC].
4. If LOC = NULL
   LINK[NEW] = START
   START := NEW
   Exit
5. LINK[NEW] := LINK[LOC]
   LINK[LOC] := NEW.
6. Exit.



DATE / /

Q. Join 2 sorted linked list in a new linked list in a sorted manner.

⇒ JOIN (INFO, LINK, START1, START2, START3).

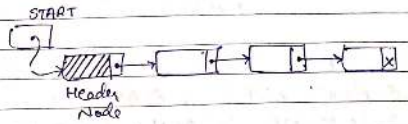
1. If INFO[START1] ≥ INFO[START2].
   START3 = START2.
   START2 = LINK[START2], START3 = LINK[START3]
2. Else
   START3 = START1

Be Positive...

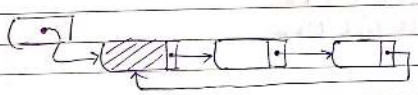
DATE / /

* Header Linked List:

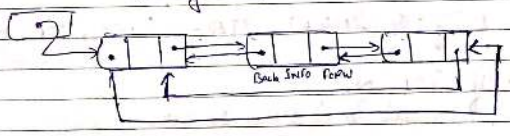
1. Grounded



2. Circular



* Doubly Linked List:
  2-way L.L.



* DEL_DLL (INFO, FORW, BACK, LOC, AVAIL)
1. If START = NULL write Underflow & Exit
2. FORW[BACK[LOC]] := FORW[LOC]
   BACK[FORW[LOC]] := BACK[LOC]

   FORW[LOC] := AVAIL
   AVAIL := LOC
3. Exit.

Be Positive...

DATE / /

* INS_DLL (INFO, FORW, BACK, START, ITEM, LOCA, LOCB, AVAIL)

1. If AVAIL = NULL, Write "Overflows" & Exit.
2. NEW := AVAIL, AVAIL := FORW [AVAIL].
3. INFO[NEW] := ITEM, BACK[NEW] := LOCA
   FORW[NEW] := LOCB

4. FORW [LOCA] := NEW
   BACK [LOCB] := NEW

5. Exit.

* Stack:

(Array)

INSERT_STACK (STACK, TOP, N, ITEM).

1. If TOP = N,
   Write "STACK IS FULL"
   & Exit.

2. TOP := TOP + 1
   STACK[TOP] := ITEM

3. Exit.

Be Positive...

DATE / /

* DEL_STACK (STACK, TOP, N, ITEM') (Array)

1. If TOP = 0
   Write "STACK IS EMPTY"
   & Exit.
2. ITEM := STACK[TOP].
   TOP := TOP - 1
3. Exit.

* INS_STACK (INFO, LINK, TOP, AVAIL, ITEM). (Linked list)

1. If AVAIL = NULL
   Write "Overflow" and Exit.
2. NEW := AVAIL, AVAIL := LINK[AVAIL].
3. INFO[NEW] := ITEM
   LINK[NEW] := TOP
   TOP := NEW.
4. Exit.

(Linked list)

* DEL_STACK (INFO, LINK, TOP, AVAIL, ITEM)

1. If TOP = NULL,
   Write "Underflow" & Exit.
2. ITEM := INFO[TOP]
3. TEMP := TOP
4. TOP := TOP[LINK]
5. LINK[TEMP] := AVAIL
   AVAIL := TEMP.
6. Exit.

Be Positive...

DATE / /

* INS_STACK1 (STACK, TOP1, TOP2, ITEM)
1. If TOP1 = TOP2 - 1
   Write "Overflow" & Exit.
2. TOP1 := TOP1 + 1
3. STACK[TOP1] = ITEM
4. Exit.

* INS_STACK2 (STACK, TOP1, TOP2, ITEM)
1. If TOP2 = TOP1 + 1
   Write "Overflow" & Exit
2. TOP2 := TOP2 - 1
3. STACK[TOP2] = ITEM
4. Exit.

* DEL_STACK1 (STACK, TOP1, TOP2, ITEM).
1. If TOP1 = 0
   Write "Underflow" & Exit.
2. ITEM := STACK[TOP1]
3. TOP1 := TOP1 - 1
4. Exit.

* DEL_STACK2 (STACK, TOP1, TOP2, ITEM, N)
1. If TOP2 = N+1
   Write "Underflow" & Exit.
2. ITEM := STACK[TOP2]
3. TOP2 := TOP2 + 1
4. Exit.

Be Positive...

DATE / /

* Applications of Stacks:
1. Evaluation of arithmetic expression.

Q: 5 * (6+2) - 12/4
P: 5, 6, 2, +, *, 12, 4, /, -
V: 37

* POSTFIX (Q, P)

1. Add ')' at the end of expression Q and Push '(' onto the top of stack.
2. Scan Q from left to right till the stack is not empty.
   a) If an operand is encountered, Add it to P.
   b) If an '(' is encountered, push it onto the top of stack.
   c) If an ⊛ operator is encountered
      i) Repeatedly pop all operators and add to P whose precedence (i.e. same or higher) is precedence of the operator ⊛.
      ii) Push ⊛ operator on the top of the stack.
   d) If ')' is encountered
      i) Repeatedly Pop all operators and Add to P till '(' is encountered.
      ii) Drop '(',
3. Exit.

Be Positive...

12 , 4

DATE

* EVALUATE (P, Value)

1. Add ')' as sentinel at the end of P.
2. Scan P from left to right until ')' is encountered.
3. If an operand is encountered, Push it onto the top of stack.
4. If an operator ⊛ is encountered
   a) Pop two elements from top of stack, 'A' being the topmost and 'B' the next to top.
   b) Apply B⊛A and Push the result on top of stack.
5. Set Value equal to result on stack and Exit.

* INS_QUEUE (FRONT, REAR, ITEM, INFO, AVAIL, LINK)

1. If AVAIL = NULL
   Write "Overflow" & Exit.
2. If FRONT = NULL
   NEW := AVAIL &, AVAIL := LINK[AVAIL].
   FRONT := REAR := NEW
3. Else
   NEW := AVAIL , AVAIL := LINK[AVAIL]
   LINK[REAR] := NEW.
   REAR := NEW.
   INFO[REAR] = ITEM
   LINK[REAR] = NULL.

Be Positive...

DATE

QUICKSORT

(44)	22	11	55	88	99	22	33	66	77
33	22	11	55	88	99	22	(44)		
			(44)	88	99	22	55		
			22	88	99	(44)			
33	22	11	22	(44)	(99)	88	55	66	77
1	2	3	4	5	6	7	8	9	10
					(77)	88	55	66	(99)

* QUICKSORT (A, N)

1. TOP := NULL
2. If N > 1
   TOP := TOP + 1
   LOWER[1] = 1 ; UPPER[1] = N
3. Repeat Steps 4 to 7 while TOP ≠ NULL
4. BEG := LOWER[TOP]
   END := UPPER[TOP]
   TOP := TOP - 1
5. Call QUICK (A, N, BEG, END, Loc)
6. If BEG < Loc - 1
   TOP := TOP + 1
   LOWER[TOP] := BEG
   UPPER[TOP] := Loc - 1
7. If Loc + 1 < END
   TOP := TOP + 1
   LOWER[TOP] = Loc + 1
   UPPER[TOP] = END
8. Exit.

Be Positive...

DATE / /

* QUICK (A, N, BEG, END, LOC)
1. LEFT := BEG, RIGHT := END, LOC := BEG.
2. (a) Repeat while A[LOC] ≤ A[RIGHT] and LOC ≠ RIGHT
RIGHT := RIGHT - 1.
(b) If LOC = RIGHT, Return.
(c) If A[LOC] > A[RIGHT]
i) //Swap
TEMP := A[LOC]
A[LOC] := A[RIGHT]
A[RIGHT] := TEMP
ii) LOC := RIGHT
iii) Go to Step 3.

3. (a) Repeat while A[LEFT] ≤ A[LOC] and LOC ≠ LEFT
LEFT := LEFT + 1
(b) If LOC = LEFT, Return.
(c) If A[LEFT] > A[LOC]
i) //SWAP
ii) LOC := LEFT
iii) Go to Step 2.

* TOWER OF HANOI

```
Tower Of Hanoi (N, start, aux, end)
{
    If (N == 1)
        Print ("Move " + start + " to " + end);

    Else
    {
        Tower Of Hanoi (N-1, start, end, aux);
        Tower Of Hanoi (1, start, aux, end);
        Tower Of Hanoi (N-1, aux, start, end);
    }
}
```

Be Positive...

F(n) = f...

DATE / /

* Priority Queue Using Arrays &
& Priority Queue Using Linked Lists

* Questions:

1) Reversing linked list
2) Detect loops & remove loops
3) Quicksort DLL (Doubly linked list)
4) Queue using DLL
5) All stack questions & assignment.

Be Positive...

DATE / /

TREE

PREORDER (INFO, LEFT, RIGHT, ROOT).

1. TOP := 1, STACK[1] := NULL, PTR := ROOT
2. Repeat steps 3 to 5 while PTR ≠ NULL
3. Apply PROCESS to INFO[PTR].
4. If RIGHT[PTR] ≠ NULL
   TOP := TOP+1, STACK[TOP] := RIGHT[PTR].
5. If LEFT[PTR] ≠ NULL
   PTR := LEFT[PTR].
   Else.
   PTR := STACK[TOP], TOP := TOP-1.
6. Exit.

INORDER (INFO, LEFT, RIGHT, ROOT)

1. TOP := 1, STACK[1] := NULL, PTR := ROOT
2. Repeat while PTR ≠ NULL
   a) TOP := TOP+1, STACK[TOP] := PTR
   b) PTR := LEFT[PTR]
3. PTR := STACK[TOP], TOP := TOP-1
4. Repeat steps 5 to 7 while PTR ≠ NULL
5. Apply PROCESS to INFO[PTR].
6. If RIGHT[PTR] ≠ NULL
   a) PTR := RIGHT[PTR]
   b) Go to Step 2.
7. PTR := STACK[TOP], TOP := TOP-1.
8. Exit.

Be Positive...

DATE / /

POSTORDER (INFO, LEFT, RIGHT, ROOT).

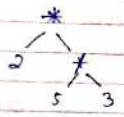
1. TOP := 1, STACK[1] := NULL, PTR := ROOT
2. Repeat steps 3 to 5 while PTR ≠ NULL
3. TOP := TOP+1, STACK[TOP] := PTR
4. If RIGHT[PTR] ≠ NULL
   TOP := TOP+1, STACK[TOP] := -RIGHT[PTR].
5. PTR := LEFT[PTR]
6. PTR := STACK[TOP], TOP := TOP-1
7. While PTR > 0
   a) Apply PROCESS to INFO[PTR]
   b) PTR := STACK[TOP], TOP := TOP-1
8. If PTR < 0
   a) PTR := -PTR
   b) Go to Step 2
9. Exit.

CREATE(INFO, LEFT, RIGHT, ROOT, AVAIL, LINK, ITEM)

1. If AVAIL = NULL
   write "Overflow" and Exit.
2. NEW := AVAIL
3. AVAIL := AVAIL[LINK].
4.

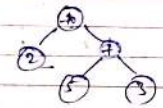
Be Positive...

* Expression Tree:



2 + (5 + 3)

2 5 3 + +



CREATE (EXP, ROOT)
1. Convert to postfix.
2. Create Node and add to stack if numbers.
3. If operator create node op.
4. Pop 2 nodes a & b.
5. Put b as left of op and a as right of op.
6. Push op to stack.
7. Return op.

Repeat while stack ≠ NULL

Be Positive...

* FIND_BST (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)
1. If ROOT = NULL, LOC := NULL, PAR := NULL & return.
2. If ITEM = INFO[ROOT], LOC := ROOT, PAR := NULL & return.
3. If ITEM < INFO[ROOT]
   SAVE := ROOT, PTR := LEFT[ROOT].
   Else
   SAVE := ROOT, PTR := RIGHT[ROOT].
4. Repeat Steps 5 to 7 while PTR ≠ NULL
5. If ITEM = INFO[PTR]
   LOC := PTR, PAR := SAVE & Return
6. If ITEM < INFO[PTR]
   SAVE := PTR, PTR := LEFT[PTR].
   Else
   SAVE := PTR, PTR := RIGHT[PTR].
7. LOC := NULL, PAR := SAVE
8. Return.

* INS_BST (INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM)
1. Call FIND(, , , , , LOC, PAR)
2. If LOC ≠ NULL Write 'Item already present' & Exit
3. If AVAIL = NULL, Write 'Overflow' & Exit.
4. NEW := AVAIL, AVAIL := LINK[AVAIL]
5. INFO[NEW] := ITEM, LEFT[NEW] := NULL, RIGHT[NEW] := NULL
6. If PAR = NULL
   ROOT := NEW
   Else if ITEM < INFO[PAR].
   LEFT[PAR] := NEW.
   Else
   RIGHT[PAR] := NEW
7. Exit.

Be Positive...

DATE / /

* DEL - BST (INFO, LEFT, RIGHT, ROOT, ITEM, AVAIL)
1. Call FIND(" , " , " , " , " , LOC, PAR)
2. If LOC = NULL, 'Write item not present' & Exit.
3. If LEFT[LOC] ≠ NULL and RIGHT[LOC] ≠ NULL
   Call CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   Else
   Call CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
4. LEFT[LOC] := AVAIL.
   AVAIL := LOC.
5. Exit.

CASE A (INFO, LEFT, RIGHT, ROOT, LOC, PAR)
1. If LEFT[LOC] = NULL and RIGHT[LOC] = NULL
   CHILD := NULL
   Else If
   LEFT[LOC] ≠ NULL
   CHILD := LEFT[LOC]
   Else
   CHILD := RIGHT[LOC]
2. If PAR = NULL
   ROOT := CHILD
   Else
   If LEFT[PAR] ≠ NULL
   LEFT[PAR] := CHILD
   Else
   RIGHT[PAR] := CHILD
3. Exit.

Be Positive...

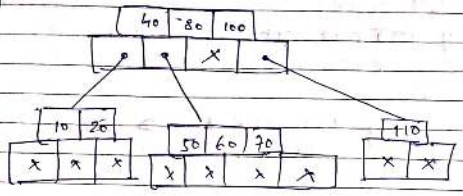
CASE B (INFO, LEFT, RIGHT, ROOT, LOC, PAR)
1. SAVE := LOC, PTR := RIGHT[LOC]
2. REPEAT while LEFT[PTR] ≠ NULL
   SAVE := PTR, PTR := LEFT[PTR].
3. SUC := PTR, PARSUC := SAVE
4. Call CASEA(, SUC, PARSUC)
5. If PAR = NULL
   ROOT := SUC
   Else
   If LEFT[PAR] = LOC
   LEFT[PAR] := SUC
   Else
   RIGHT[PAR] := SUC.
6. LEFT[SUC] := LEFT[LOC]
   RIGHT[SUC] := RIGHT[LOC].
7. Exit.

Be Positive...

DATE / /

* m-way Trees:
(m-1) keys
m children

4-way Tree



* B-Trees:

$\left[\frac{m-1}{2}\right]$ keys except for root node

more m children

All leaf nodes at same level.

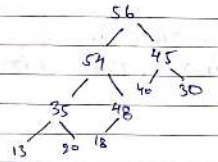
Tree grows upwards.

Be Positive...

DATE / /

* Heap Sort:

35, 56, 45, 20, 18, 40, 30, 15, 54, 48



* INSERT (TREE, N, ITEM)
1. N := N+1, PTR := N
2. Repeat 3-5 while PTR > 1
3. PAR := [PTR/2]
4. If ITEM < TREE[PAR]
TREE[PTR] := ITEM & return
5. TREE[PTR] := TREE[PAR], PTR := PAR
6. TREE[1] := ITEM.
7. Return

* DELETE (TREE, N, ITEM)
1. ITEM := TREE[1], LAST := TREE[N], N := N-1
2. PTR := 1, LEFT := 2, RIGHT := 3
3. Repeat 4-6 while RIGHT < N
4. If LAST ≥ TREE[RIGHT] and LAST ≥ TREE[LEFT]
TREE[PTR] := LAST & Return
5. If TREE[LEFT] ≥ TREE[RIGHT]
TREE[PTR] := TREE[LEFT]
PTR := LEFT

Be Positive...

DATE ___/___/___

Else
TREE[PTR] := TREE[RIGHT]
PTR := RIGHT

6. LEFT := PTR * 2 , RIGHT := LEFT + 1

7. If LEFT = N and LAST < TREE[LEFT]
TREE[PTR] := TREE[LEFT] , PTR := LEFT

8. TREE[PTR] := LAST
9. Return.

Be Positive...

DATE ___/___/___

Be Positive...

Else

TREE[PTR] := TREE[RIGHT]

PTR := RIGHT

6. LEFT := PTR * 2 , RIGHT := LEFT + 1

7. If LEFT = N and LAST < TREE[LEFT]

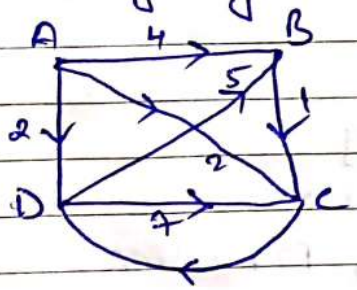
TREE[PTR] := TREE[LEFT] , PTR := LEFT

8. TREE[PTR] := LAST

9. Return.

* Graphs:

Adjacency Matrix



$$G = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$G^2 =$

$G^3 =$

$G^4 =$

Total paths = $G + G^2 + G^3 + G^4$.

* Warshall's Algo:

```
1. FOR I = 1 to N
     FOR J = 1 to N
       If G[I,J] = 1
          P[I,J] = 1
     Else
          P[I,J] = 0

2. FOR K = 1 to N
     FOR I = 1 to N
       FOR J = 1 to N
          P[I,J] = P[I,J] ∨ ( P[I,K] ∧ P[K,J] )
```

* Dijkstra's Algo:

For weighted graphs:

$$W = \begin{bmatrix} 0 & 4 & 7 & 2 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

```
1. For I = 1 to N
     FOR J = 1 to N
       If W[I,J] = 0
          C[I,J] = ∞
     Else
          C[I,J] = W[I,J].
```

DATE / /

2. FOR K=1 to N
   FOR I=1 to N
      FOR J=1 to N

$$C[I,J] = \min\left(C[I,J], (C[I,K] + C[K,J])\right)$$