D.S (Theory)

Page No.
Date

* Algorithmic notation
=> Steps, Go to step n, Exit
=> Comment's are written in [square bracket]
=> Variable names use capital letters, MAX
=> Assignment Statement :=
=> Read: write:

* Storing Steing
i] Record oriented, fixed length storage
In fixed-length storage each line of print is viewed as a record, where all record have same length. i.e where each record accomodates same number characters. These records have maximum length 80.

ii] Variable length storage with fixed maximum

iii] linked storage

* Character Data Type
i] Static: a variable whose length is defined before the program is executed and cannot change throughout program.

ii] Semistatic: a variable whose length may vary during execution of program as long as length does not exceed a maximum value determined by program before the program is executed:

Complexity of first pattern matching algorithm = O(n
Second = O(n)

iii] dynamic: A variable whose length can change during execution of program.

* String operations:
i] Substring (string, initial, length)
ii] Index (text, pattern)
iii] length (string)
iv] Concatenation $S_1 \| S_2$

* word Processing:
i] Insertion (text, position, string)
ii] DELETE (text, position, length)
iii] REPLACE (text, pattern1, pattern2)

* Two - Dimensional Array
Row - Major order
$LOC[A[J,K]] = BaseAd + w[N(J-1) + (K-1)]$
Column - Major order
$LOC(A[J,K]) = BaseAd + w[M(K-1) + (J-1)]$

* 3-D Array: Column - Major order
$(E_3 L_2 + E_2) L_1 + E_1$

* Pointer Array: An array PTR is called a pointer array if each element of PTR is pointer.

* Record: is a collection of related data items.

* Sparse Matrices: Matrices with relatively high proportion of zero entries are called Sparse Matrices.

* $L = UB - LB + 1$

  $E = k - LB$

* Garbage Collection: The operating system of a computer may periodically collect all deleted space onto free-storage list. This technique is called garbage collection.

* Memory allocation: The maintainance of linked list in memory assumes possibity of inserting new nodes into lists and hence requires some mechanism which provides unused memory space for the new nodes.

* Header linked list: It always contains a special node, called header node at beginning of list. There are two type's of header linked list.

  i) Grounded header list: Last node contain NULL pointer.

  ii) Circular header list: Last node points back to the header node.

* Two-way list: It can be traversed in two direction. Collection of nodes. Each node N is divided into three parts.
i] Information field INFO
ii] Pointer field FORW contain location of Next node.
iii] pointer field BACK contain location of preceding node.

* Polish Notation: in which operator symbol is placed before two operand.

* Reverse polish Notation: operator symbol is placed after two operand.

* Quick Sort: It is an algorithm of divide and conquer type. That is a problem of soring a set is reduced to problem of soring two smaller sets.

* Resursion: a function calls itself directly or indirectly.

* Deque: Elements can be added or removed at either end but not in middle. (Double-ended queue)

* Priority queue: Collection of elements such that each element has been assigned a priority.

Algorithm : Array

Page No.
Date

1] Traversing Array:

1. Set K := LB
2. Repeat step 3 and 4 while K ≤ UB
3. Apply PROCESS to LA[K]
4. Set K := K+1
   [End of step 2 LOOP]
5. Exit

2] Insertion

INSERT(LA, N, K, ITEM)

1. Set J := N
2. Repeat steps 3 and 4 while J ≥ K
3. Set LA[J+1] := LA[J]
4. Set J := J-1 [End of step 2 loop]
5. Set LA[K] := ITEM
6. Set N := N+1
7. Exit

3] Deletion

DELETE(LA, N, K, ITEM)

1. Set ITEM := LA[K]
2. Repeat for J = K to N-1:
   Set LA[J] := LA[J+1]
   [End of loop]
3. Set N := N-1
4. Exit

4] BUBBLESORT (DATA, N)

1. Repeat steps 2 and 3 for K = 1 to N - 1
2. Set PTR := 1
3. Repeat while PTR ≤ N - K :
   If DATA [PTR] > DATA [PTR+1], then:
   Interchange DATA[PTR] and DATA [PTR+1]
4. Set PTR := PTR + 1
5. Exit

5] LINEAR SEARCH (DATA, N, ITEM, LOC)

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

6] BINARYSEARCH(DATA, LB, UB, ITEM, LOC)

1. Set BEG := LB, END := UB

$$MID = INT\left(\frac{BEG + END}{2}\right)$$

2. Repeat setps 3 and 4 while
   BEG ≤ END and DATA[MID] ≠ ITEM
3. If ITEM < DATA [MID], then:
   Set END := MID - 1
   Else
   Set BEG := MID + 1
4. Set MID := INT $\left(\frac{BEG + END}{2}\right)$

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

7. LARGE + SecondLARGE (DATA, N, LOC1, LOC2)
1. Set FIRST := DATA [1]
SECOND := DATA [2]
LOC1 := 1
LOC2 := 2
2. If FIRST < SECOND, then:
a) INTERCHANGE FIRST and SECOND
b) Set LOC1 := 2 and LOC2 := 1
3. Repeat for K = 3 to N:
If FIRST < DATA [K], then:
a] Set SECOND := FIRST and
FIRST := DATA [K]
b] set LOC2 := LOC1 and
LOC1 := K
Else if SECOND < DATA[K], then:
set SECOND := DATA[K] and
LOC 2 := K.
4. Exit

8) First Pattern matching [Slow]

1. Set K := 1 and MAX := S - R + 1
2. Repeat steps 3 to 5 while K ≤ MAX:
3. Repeat for L = 1 to R:
   If P[L] ≠ T[K + L - 1], then:
   Go to step 5.
4. Set INDEX = K, and Exit. [Pass]
5. Set K := K + 1.
6. Set INDEX = 0. [Fail]
7. Exit

9) Second pattern matching [Fast]

1. Set K := 1 and $S_1 = \emptyset$.
2. Repeat steps 3 to 5 while $S_K \neq P$ and $K \leq N$
3. Read T_K
4. Set $S_{K+1} := F(S_K, T_K)$
5. Set K := K + 1
6. IF $S_K = P$, then:
   INDEX = K - LENGTH(P).
   Else:
   INDEX = 0.
7. Exit.

Algorith: Linked List

Page No.
Date

1] Traversing:

1. Set PTR := START
2. Repeat steps 3 and 4 while PTR ≠ NULL
3. Apply PROCESS to INFO[PTR]
4. Set PTR := LINK [PTR]
5. Exit

2] SEARCHING (Sorted)

SRCHSL (INFO, LINK, START, ITEM, LOC)

1. Set PTR := START.
2. Repeat step 3 while PTR ≠ NULL
3. If ITEM < INFO [PTR], then:
   Set PTR := LINK [PTR]
   Else if ITEM = INFO [PTR], then:
   Set LOC := PTR, Exit
   Else:
   Set LOC := NULL, Exit
4. Set LOC := NULL
5. Exit

3. Searching (Unsorted)

1. Set PTR := START
2. Repeat step 3 while PTR ≠ NULL
3. If ITEM = INFO [PTR], then:
   Set LOC := PTR, EXIT.
   Else:
   Set PTR := LINK [PTR]
4. Set LOC := NULL
5. Exit.

4) Insertion at beginning

INSFIRST(INFO, LINK, START, AVAIL, ITEM)

1. If AVAIL = NULL, then:
   write: OVERFLOW and Exit.
2. Set NEW := AVAIL and
   AVAIL := LINK[AVAIL]
3. Set INFO[NEW] := ITEM.
4. Set LINK[NEW] := START
5. Set START := NEW
6. Exit

5) Insertion after given node.

INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

1. If AVAIL = NULL, then:
   write: OVERFLOW, Exit
2. Set NEW := AVAIL
   AVAIL := LINK[AVAIL]
3. Set INFO[NEW] := ITEM
4. If LOC = NULL, then:
   Set LINK[NEW] := START
   START := NEW
   Else:
   Set LINK[NEW] := LINK[LOC]
   LINK[LOC] := NEW
5. Exit.

6] Insertion into sorted linked list.

FINDA (INFO, LINK, START, ITEM, LOC)

1. If START = NULL then:
   Set LOC := NULL and Return,
2. If ITEM < INFO[START], then:
   Set LOC := NULL and Return
3. Set SAVE := START
   PTR := LINK[START]
4. Repeat steps 5 and 6 while PTR ≠ NULL
5. If ITEM < INFO[PTR], then:
   Set LOC := SAVE, and Return.
6. Set SAVE := PTR and PTR := LINK[PTR]
7. Set LOC := SAVE.
8. Return.

INSERT(INFO, LINK, START, AVAIL, ITEM)

1. Call FINDA (INFO, LINK, START, LOC, ITEM)
2. Call INSLOC (INFO, LINK, START, AVAIL,)
3. Exit.

7] Deleting a Node following given Node.

DEL (INFO, LINK, START, AVAIL, LOC, LOCP)

1. If LOCP = NULL, then:
   Set START := LINK[START].
   Else
   Set LINK[LOCP] := LINK[LOC]
2. Set LINK[LOC] := AVAIL
   AVAIL := LOC
3. Exit

✱ Traversing Circular Header List

1. Set PTR := LINK [START]
2. Repeat Steps 3 and 4 while PTR ≠ START
3. Apply PROCESS to INFO[PTR]
4. Set PTR := LINK [PTR]
5. Exit

✱ Linked Representation of Queue:

1) INSERT (INFO, LINK, FRONT, REAR, AVAIL)
   1. If AVAIL = NULL, then
      write: OVERFLOW and Exit.
   2. Set NEW := AVAIL and
      AVAIL := LINK [AVAIL].
   3. INFO[NEW] := ITEM and
      LINK [NEW] := NULL
   4. If (FRONT = NULL) then,
      FRONT = REAR = NEW
      else Set LINK [REAR] := NEW and
      REAR = NEW
   5. Exit

2. DELETE (INFO, LINK, FRONT, REAR, AVAIL)
   1. if (FRONT = NULL) then, write: UNDERFLOW
   2. Set TEMP := FRONT
   3. ITEM := INFO (TEMP)
   4. FRONT := LINK [TEMP]
   5. LINK [TEMP] := AVAIL and
      AVAIL := TEMP
   6. Exit.

STACK

Page No.
Date

Array Representation:

* PUSH (STACK, TOP, MAXSTK, ITEM)
1. IF TOP = MAXSTK, then:
   Print: OVERFLOW, and Return.
2. Set TOP := TOP + 1.
3. Set STACK[TOP] := ITEM
4. Return.

* POP(STACK, TOP, ITEM)
1. If TOP = 0, then:
   Print: UNDERFLOW, and Return.
2. Set ITEM := STACK[TOP].
3. Set TOP := TOP - 1.
4. Return

Linked Representation:

* PUSH(INFO, LINK, TOP, AVAIL, ITEM)
1. If AVAIL = NULL, then:
   write: OVERFLOW and Exit.
2. Set NEW := AVAIL and AVAIL := LINK[AVAIL]
3. INFO[NEW] := ITEM
4. LINK[NEW] := TOP
5. TOP := NEW
6. Exit.

* POP(INFO, LINK, TOP, AVAIL, ITEM)
1. If TOP = NULL, then: write: UNDERFLOW
2. Set ITEM := INFO[TOP]
3. Set TEMP := TOP & TOP := LINK[TOP]
4. Set LINK(TEMP) = AVAIL & AVAIL = TEMP
5: Exit

* Factorial using Recursion:

FACT (Fact, N)

1. If N=0, then: set FACT:=1, Return
2. set FACT:=1.
3. Repeat for K=1 to N.
   Set FACT := K * FACT.
4. Return.

FACT(FACT,N)

1. If N=0, then: set FACT:=1, Return
2. CALL FACT(FACT, N-1)
3. Set FACT := N * FACT
4. Return

* FIBONACCI (FIB,N).

1. If N=0 and N=1, then: set FIB:= N
2. Call FIBONACCI (FIBA, N-2);
3. call FIBONACCI (FIBB, N-1);
4. set FIB:= FIBA + FIBB
5. Return.

Queue

Page No.
Date

* Array Representation:
* QINSERT(QUEUE, N, FRONT, REAR, ITEM)
1. If FRONT= 1 and REAR= N
   or if FRONT= REAR+1, then:
   write: OVERFLOW and Return.
2. If FRONT := NULL
   Set FRONT := 1 and REAR := 1
   Else if REAR = N, then:
   Set REAR := 1
   Else:
   Set REAR := REAR+1
3. Set QUEUE[REAR] := ITEM
4. Return

* QDELETE(QUEUE, N, FRONT, REAR, ITEM)
1. IF FRONT := NULL, then:
   write: UNDERFLOW, Return.
2. Set ITEM := QUEUE[FRONT]
3. IF FRONT = REAR, then:
   Set FRONT := NULL
   and REAR := NULL
   Else if FRONT = N, then:
   Set FRONT := 1
   Else:
   Set FRONT := FRONT+1.
4. Return.

* Fixed length string: we already declare the size of size of string to be stored in memory. It has disadvantages of memory waste.

* Variable length string: These type of string have ability to change their size during execution of progeam.

* Linked string: It stores the string in nodes.

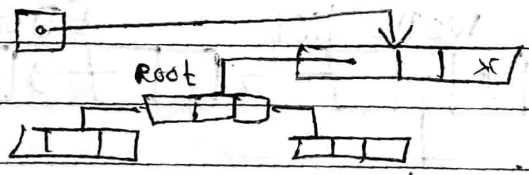
5 - Tree

Page No.
Date

* Tree: Data frequently contain a hierarchical relationship between various elements. The d.s which represent this relationship is called tree.

* Binary Tree : It is a finite set of elements, called nodes. such that:
i] T is empty (null tree or empty tree)
ii] T contains distinguished node R, called root of T, and remaining nodes of T form and ordered pair of disjoint binary trees T_1 & T_2.

* Header Node: Sometimes, special node called header node, is added to beginning of T. When this extra node is used, free pointer variable, which we will call HEAD, will point to header node and left pointer of header node points to root of T.



* Binary Search Tree: The value value at N (Middle) is greater than every value in left subtree of N and is less than every value in right subtree of N.

* HEAP: H is called Heap, if every node N of H has following property:
The value at N is greater than or equal to value at each of children of N.

Graph

* A Graph consist of two things:
1) A set V of elements called vertices
2) A set E of edges. $e = [u, v]$

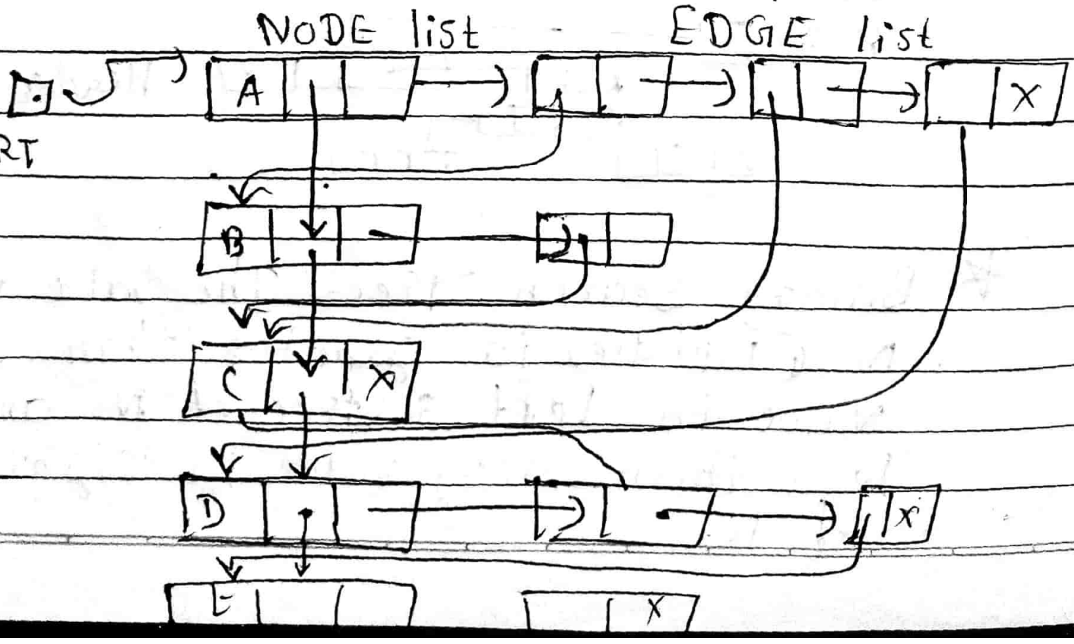
* Adjacency Matrix

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

* Path matrix:

$$p_{ij} = \begin{cases} 1 & \text{if there is path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

* Linked representation of Graph:



* Breadth-First Search: First we examine Starting node A. then we examine all neighbors of A. Then we examine all neighbors of neighbors of A and so on. Such that no node is processed more than one. This is accomplished by using a queue to hold nodes that are waiting to be processed.

* Depth first search: First we examine Starting node A. Then we examine each node N along a path P which begins at A; i.e we process neighbor of A. Then neighbour of neighbor of A.

* Selection Sort: first find smalled element and put it in first position then find second Smallest element and put it in second position and so on.

* Insertion Sort: Scans Array from A[1] to A[N], insert each element into its proper position.