

Data Structures (unit 1)

1) Array operations

- Traversing An Array

- Step 1:- [initialization] set $I = \text{lower_bound}$
- Step 2:- Repeat Step 3 to 4
while $I <= \text{upper_bound}$
- Step 3:- Apply process to $A[I]$
- Step 4:- Set $I = I + 1$
[END OF LOOP]
- Step 5:- EXIT

- inserting an element in Array

- Step 1:- Set $\text{upper_bound} = \text{upper_bound} + 1$
- Step 2:- Set $A[\text{upper_bound}] = \text{val}$
- Step 3:- EXIT

.. (Algo. to append a new element to an existing array)

- Step 1:- [initialization] set $I = N$
- Step 2:- Repeat 3 and 4 while $I >= \text{pos}$
- Step 3:- Set $A[I+1] = A[I]$
- Step 4:- Set $I = I + 1$ [END OF LOOP]
- Step 5:- Set $N = N + 1$
- Step 6:- Set $A[\text{pos}] = \text{VAL}$
- Step 7:- EXIT.

.. (add new element to middle of an array)

• Delete element in a Array:-

- Step 1: Set upper-bound = upper-bound - 1
Step 2: EXIT
(Algo. to delete last element of an Array)
- Step 1: [initialization] Set I = POS
Step 2: Repeat step 3 and 4 while $I \leq N - 1$
Step 3: Set $A[I] = A[I + 1]$
Step 4: Set $I = I + 1$ [END OF LOOP]
Step 5: Set $N = N - 1$
Step 6: EXIT
(Algo to delete an element from the middle of an Array)

2) Linear search:-

- Step 1: [initialize] Set POS = -1
Step 2: [initialize] Set I = 1
Step 3: Repeat Step 4 while $I \leq N$
Step 4: IF $A[I] = VAL$
 SET POS = I
 PRINT POS
 GO TO Step 6
 [END OF IF]
 Set I = I + 1
 [END OF LOOP]
Step 5: IF POS = -1
 PRINT "VALUE IS NOT PRESENT
 IN AN ARRAY"
 [END OF IF]
Step 6: EXIT.

3)

Binary Search:

→ Step 1: [Initialize] Set BEG = lower-bound

END = upper-bound, POS = -1

Step 2: Repeat Step 3 and 4 while BEG <= END

Step 3: Set MID = (BEG + END) / 2

Step 4: IF A[MID] = VAL

SET POS = MID

PRINT POS

GO TO Step 6

ELSE IF A[MID] > VAL

SET END = MID - 1

ELSE

SET BEG = MID + 1

[END OF IF]

[END OF LOOP]

Step 5: IF POS = -1

PRINT "VALUE IS NOT PRESENT IN
IN THE ARRAY" [END OF IF]

Step 6: EXIT

4)

Sorting:

→ Step 1: Repeat Step 2 for I = N - 1

Step 2: Repeat For J = to N-1

Step 3: IF A[J] > A[J+1]

SWAP A[J] and A[J+1]

[END OF INNER LOOP]

[END OF OUTER LOOP]

Step 4: EXIT

5) Insertion Sort:-

- Step 1: Repeat Step 2 to 5 for $k = 1$ to $N - 1$
- Step 2: SET Temp = arr[k]
- Step 3: Set $J = k - 1$
- Step 4: Repeat while $\text{Temp} \leq \text{arr}[J]$
- set $\text{arr}[J + 1] = \text{arr}[J]$
 - Set $J = J - 1$
- [END OF INNER LOOP]
- Step 5: Set $\text{ARR}[J + 1] = \text{Temp}$
- Step 6: EXIT

6) Selection Sort:-

• Smallest (ARR, K, N, POS):-

- Step 1: [Initialize] Set small = arr[k]
- Step 2: [Initialize] Set pos = k
- Step 3: Repeat, for $J = k + 1$ to $N - 1$
- if $\text{small} > \text{arr}[J]$
 - Set $\text{small} = \text{arr}[J]$
 - Set $\text{pos} = J$
- [END OF IF]
- [END OF LOOP]
- Step 4: RETURN pos

• Selection (ARR, N):-

- Step 1: Repeat Step 2 and 3 for $k = 1$ to $N - 1$

Step 2: call smallest (ARR, k, N, pos)

Step 3: Swap $A[k]$ with $\text{ARR}[\text{pos}]$

[END OF LOOP]

Step 4: EXIT.

UNIT NO:- 21) Push Operation:-

→ Step 1: if $\text{TOP} = \text{MAX} - 1$

PRINT "OVERFLOW"

GOTO Step 4

[END OF IF]

Step 2: Set $\text{TOP} = \text{TOP} + 1$

Step 3: Set stack [TOP] = value

Step 4: END

2) Pop operation:-

→ Step 1: if $\text{top} = \text{NULL}$

PRINT "UNDERFLOW"

Goto Step 4 [END OF IF]

Step 2: Set $\text{val} = \text{stack} [\text{TOP}]$

Step 3: Set $\text{TOP} = \text{TOP} - 1$

Step 4: END.

3) Peek Operation:-

→ Step 4 Step 1: if $\text{top} = \text{NULL}$

PRINT "Stack is empty"

Goto Step 3.

Step 2: Return $\text{stack} [\text{TOP}]$

Step 3: END

UNIT NO:- 03 (Queue)

1) insert array elements in queue :-

→ Step 1: if rear = max - 1
 write "overflow"
 Goto step 4
 [END OF IF]

Step 2: if front = -1 and rear = -1
 set front = rear = 0
 else
 set rear = rear + 1

[end of if]

Step 3: set QUEUE [REAR] = NUM

Step 4: EXIT

2) Deletion in Queue :-

→ Step 1: if front = -1 OR FRONT > REAR
 WRITE UNDERFLOW
 ELSE
 SET VAL = QUEUE [FRONT]
 SET FRONT = FRONT + 1
 [END OF IF]

Step 2: EXIT.

3) Circular Insert Algorithm:-

→ Step 1: if (front == 0 AND rear == MAX - 1) or
(front == rear + 1)
 PRINT OVERFLOW
 Goto Step 5 [END OF IF]

Step 2: if front == -1
 set front = rear = 0
 else IF rear == MAX - 1
 set rear = 0
 else
 set rear = rear + 1 [end of IF]

Step 3 : set QUEUE [Rear] = NUM

Step 4: Print "element inserted"

Step 5: EXIT.

4) Circular Delete Algorithm:-

→ Step 1 :- if front == -1

 write UNDERFLOW

 GOTO step 4 [END OF IF]

Step 2:- set val = QUEUE [FRONT]

Step 3:- if front == Rear

 set front = rear = -1

 else IF front == MAX - 1

 set FRONT = 0

 else

 set Front = Front + 1

[END OF IF]

Step 4: EXIT.

5) Dequeue : insert :-

• Rear

→ Step 1:- if (front == 0 and rear == Max-1)

 OR (FRONT == Rear + 1)

 write OVERFLOW

 EXIT

Step 2:- if FRONT == -1

 set FRONT = REAR = 0

 else IF REAR == MAX - 1

 set REAR = 0

 else

Set REAR = REAR + 1

Step 3: Set Dequeue [Rear] = value

Step 4: EXIT.

• Front

→ Step 1:- if (Front == 0 and Rear == MAX - 1) OR
(Front == Rear + 1)
write OVERFLOW
EXIT

Step 2: if FRONT == -1

Set FRONT = REAR = 0

else IF FRONT == 0

Set FRONT = MAX - 1

else

Set FRONT = FRONT - 1

Step 3: Set Dequeue [FRONT] = value

Step 4: EXIT.

6) Dequeue: Deletion.

• Rear:

→ Step 1:- if Front == -1
write underflow
EXIT

Step 2: Set value = Dequeue [Rear]

Step 3: if FRONT == REAR

Set FRONT == REAR

else IF REAR == 0

Set REAR = MAX - 1

else

Set REAR = Rear - 1

Step 4: EXIT.

• Front :-

→ Step 1:- if front == -1
write UNDERFLOW
EXIT

Step 2: set value = Deque[FRONT]

Step 3:- if front == Rear

Set Front = Rear = -1

else if front == MAX-1

Set Front = 0

else

Set Front = Front + 1

Step 4:- EXIT.

Unit NO: 4 (Linked List)

1) Travelling a Linked List :-

→ Step 1:- [Initialize] set PTR = START

Step 2:- Repeat Step 3 and 4

while PTR != NULL

Step 3:- Apply Process - to PTR → DATA

Step 4:- Set PTR = PTR → NEXT

[END OF LOOP]

Step 5:- EXIT.

2) To print the number of nodes in a Linked List.

→ Step 1:- [Initialize] set count = 0

Step 2:- [Initialize] set PTR = START

Step 3:- Repeat Step 4 and 5

while PTR != NULL

Step 4:- Set count = count + 1

Step 5:- Set PTR = PTR → NEXT

[END OF LOOP]

Step 6:- WRITE COUNT

Step 7:- EXIT

3) Searching for a value in the Linked List:-

→ Step 1:- [Initialize] set PTR = START

Step 2:- Repeat Step 3 while PTR != NULL

Step 3:- If val = PTR → DATA

 set POS = PTR

 GOTO Step 5

ELSE

 SET = PTR → NEXT

[END OF IF] [END OF LOOP]

Step 4:- Set POS = NULL

Step 5:- EXIT.

4) inserting a node at beginning of the linked list:

→ Step 1: If $\text{AVAIL} = \text{NULL}$
 Write OVERFLOW
 GOTO step 7 [END OF IF]

Step 2: Set $\text{NEW-NODE} = \text{AVAIL}$

Step 3: Set $\text{AVAIL} = \text{AVAIL} \rightarrow \text{NEXT}$

Step 4: Set $\text{NEW-NODE} \rightarrow \text{DATA} = \text{VAL}$

Step 5: Set $\text{NEW-NODE} \rightarrow \text{NEXT} = \text{START}$

Step 6: Set $\text{START} = \text{NEW-NODE}$

Step 7: EXIT

5) inserting a node at end of the linked list:

→ Step 1: if $\text{AVAIL} = \text{NULL}$
 write OVERFLOW
 Go TO step 10 [END OF IF]

Step 2: Set $\text{NEW-NODE} = \text{AVAIL}$

Step 3: Set $\text{AVAIL} = \text{AVAIL} \rightarrow \text{NEXT}$

Step 4: Set $\text{NEW-NODE} \rightarrow \text{DATA} = \text{VAL}$

Step 5: Set $\text{NEW-NODE} \rightarrow \text{NEXT} = \text{NULL}$

Step 6: Set $\text{PTR} = \text{START}$

Step 7: Repeat step 8 while $\text{PTR} \rightarrow \text{NEXT} = \text{NULL}$

Step 8: Set $\text{PTR} = \text{PTR} \rightarrow \text{NEXT}$ [END OF LOOP]

Step 9: Set $\text{PTR} \rightarrow \text{NEXT} = \text{NEW-NODE}$

Step 10: EXIT.

6) insert a new node after a node that has value NUM.

→ Step 1:- if AVAIL = null
 write OVERFLOW
 GOTO Step 12 [END OF IF]

Step 2:- set NEW-NODE = avail

Step 3:- set avail = avail → next.

Step 4:- set new-node → data = val.

Step 5:- set PTR = START

Step 6:- set PREPTR = PTR

Step 7:- Repeat Step 8 and 8
 while PREPTR → DATA != NUM.

Step 8:- SET₁^{PEPTR} = PTR

Step 9:- Set PTR = PTR → NEXT

Step 10:- PREPTR → NEXT = new-node

Step 11:- set new-node → next = PTR

Step 12: EXIT.

7) inserting a Node before a given Node in Linked List:

→ Step 1:- if avail = null
 write OVERFLOW
 Go to step 12 [END OF IF]

Step 2:- set new-node = avail

Step 3:- set avail = avail → next.

Step 4:- set new-node → data = val

Step 5:- set PTR = START

Step 6:- set PREPTR = PTR

Step 7:- Repeat Step 8 and 9
 while PTR → DATA != NUM

Step 8:- set PREPTR = PTR

Step 9:- Set PTR = PTR → NEXT

Step 10:- PREPTR → NEXT = NEW-NODE

Step 11:- set NEW-NODE → NEXT = PTR

Step 12:- EXIT

8)

Deleting the 1st node of the linked list:-

→ Step 1:- if START = NULL
 Write UNDERFLOW
 Go to Step 5
 [END OF IF]

Step 2:- Set PTR = START

Step 3:- Set START = START → NEXT

Step 4:- FREE PTR

Step 5:- EXIT

9)

deleting the last node from the Linked List:-

→ Step 1:- if START = NULL.
 Write UNDERFLOW
 Go to Step 8 [END OF IF]

Step 2:- Set PTR = START

Step 3:- Repeat step 4 and 5
 while PTR → next != NULL

Step 4:- Set PREPTR = PTR

Step 5:- Set PTR = PTR → NEXT
 [END OF LOOP]

Step 6:- set PREPTR → NEXT = NULL

Step 7:- FREE PTR

Step 8:- EXIT.

10) deleting the Node after a given node in Linked List

→ Step 1:- if start = null
 write UNDERFLOW
 Goto Step 10 [END OF IF]
Step 2:- set PTR = START
Step 3:- set PREPTR = PTR
Step 4:- Repeat steps 5 and 6
 while PREPTR → DATA != NUM
Step 5:- set PREPTR = PTR
Step 6:- Set PTR = PTR → NEXT
 [END OF LOOP]
Step 7:- set TEMP = PTR
Step 8:- set PREPTR → NEXT = PTR → NEXT
Step 9:- FREE TEMP
Step 10:- EXIT.

11) Sort Linked List in Ascending order:-

→ Step 1:- if start = null or start → ~~DATA~~ = next = NULL
 write "List is too short to sort"
 Goto step 9 [END OF IF]
Step 2:- set PTR1 = START
Step 3:- Repeat Step 4 and to 7 while PTR1 ≠ NULL
Step 4:- set PTR2 = PTR1 → NEXT
Step 5:- Repeat Step 6 and 7 while PTR2 ≠ NULL
Step 6:- if PTR1 → DATA > PTR2 → DATA
 set TEMP = PTR1 → DATA
 set PTR1 → DATA = PTR2 → DATA
 Set PTR2 → DATA = TEMP
 [END OF IF]
Step 7:- Set PTR2 = PTR2 → NEXT
 [END OF INNER LOOP]
Step 8:- Set PTR1 = PTR1 → NEXT
 [END OF OUTER LOOP]
Step 9:- EXIT.

(12)

Merging the Node :-

→ Step 1 :- if list1 = NULL

Set Merged-list = list2

Go to Step 9 [END OF IF]

Step 2 :- if list2 = NULL

Set merged-list = list1

Go to Step 9 [END OF IF]

Step 3 :- Initialize pointers

Set PTR1 = list1

Set PTR2 = list2

Set merged-list = null

Set Last = NULL

Step 4 :- Repeat Step 5-7 while PTR1 ≠ NULL &
PTR2 ≠ NULL

Step 5 :- if PTR1 → Data ≤ PTR2 → DATA

Set Temp = PTR1

Set PTR1 = PTR1 → NEXT

ELSE

Set TEMP = PTR2

Set PTR2 = PTR2 → Next [END OF IF]

Step 6 :- if merged-list = NULL

Set merged-list = temp

Set last = temp

else

Set last → Next = temp

Set last = temp [END OF IF]

Step 7 :- [END OF LOOP]

Step 8 :- if PTR1 ≠ NULL

Set last → next = PTR1

else if PTR2 != NULL

set Last \rightarrow next = PTR2

[END OF IF]

Step 9:- EXIT

13) inserting a node at the beginning of the Doubly linked list :- ^{end}

→ Step 1:- if avail = null

write overflow

Go to step 9 [END OF IF]

Step 2:- set new-node = avail

Step 3:- set avail = avail \rightarrow next

Step 4:- set new-node \rightarrow data = val

Step 5:- set new-node \rightarrow prev = null

Step 6:- set PTR = START

Step 7:- Repeat Step 8 while PTR \rightarrow next != null

Step 8:- Set PTR = PTR \rightarrow next [END OF LOOP]

Step 9:- set PTR \rightarrow next = new-node

Step 10:- set New-node \rightarrow prev = PTR

Step 11:- EXIT

14) inserting a node at the beginning of the Doubly linked list :-

→ Step 1:- if avail = null

write overflow

Go to step 9 [END OF IF]

Step 2:- set new-node = avail

Step 3:- set avail = avail \rightarrow next

Step 4:- set new-node \rightarrow data = val

Step 5:- set new-node \rightarrow prev = null

Step 6:- set new-node \rightarrow next = start

Step 7:- set start \rightarrow prev = new-node

Step 8:- set start = new-node

Step 9:- EXIT

15)

Algorithm to insert a new node after a given node:-

→ Step 1 :- if avail = null

 write OVERFLOW

 Go to step 12 [END OF IF]

Step 2 :- set new-node = avail

Step 3 :- set avail = avail → next

Step 4 :- set new-node → data = val

Step 5 :- set PTR = START

Step 6 :- Repeat Step 7 while PTR → data != NUM

Step 7 :- Set PTR = PTR → next [END OF LOOP]

Step 8 :- set new-node → next = PTR → next

Step 9 :- set new-node → prev = PTR

Step 10 :- set PTR → next = new-node

Step 11 :- set PTR → next → prev = new-node

Step 12 : EXIT

16)

Algorithm to insert a new node before a given node:-

→ Step 1 :- If avail = null

 write OVERFLOW

 Go to step 12

[END OF IF]

Step 2 : set new-node = avail

Step 3 : set avail = avail → next

Step 4 :- Set new-node → data = val

Step 5 :- set PTR = START

Step 6 :- Repeat Step 7 while PTR → data != NUM.

Step 7:- set PTR = PTR \rightarrow Next
Step 8:- set new-node \rightarrow next = PTR
Step 9:- set new-node \rightarrow prev = PTR \rightarrow prev
Step 10:- set PTR \rightarrow prev = new-node
Step 11:- set PTR \rightarrow prev \rightarrow next = new-node
Step 12:- EXIT.

17) deleting the 1st node from a doubly linked list:-

→ Step 1:- If start = null
 write OVERFLOW
 Go to step 6 [END OF IF]
Step 2:- set PTR = START
Step 3:- set START = START \rightarrow next
Step 4:- set START \rightarrow PREV = NULL
Step 5:- FREE PTR
Step 6:- EXIT

18) deleting the last node from a doubly linked list:-

→ Step 1:- If start = null
 write OVERFLOW
 Go to Step 7 [END OF IF]
Step 2:- Set PTR = START
Step 3:- Repeat Step 4 while PTR \rightarrow next != null
Step 4:- set PTR = PTR \rightarrow next
 [END OF IF]
Step 5:- set PTR \rightarrow PREV \rightarrow next = null
Step 6:- Free PTR
Step 7:- exit

19)

deleting the node before a given node in doubly linked list

→ Step 1 :- if start = null
write underflow

Go to step 9 [END OF IF]

Step 2 :- set PTR = START

Step 3 :- Repeat Step 4 while PTR → data | = num

Step 4 :- set PTR = PTR → next [END OF LOOP]

Step 5 :- set TEMP = PTR → PREV

Step 6 :- set TEMP → prev → next = PTR

Step 7 :- set PTR → PREV = TEMP → PREV

Step 8 :- Free TEMP

Step 9 :- EXIT

20)

deleting the node after a given node in doubly linked list

→ Step 1 :- if start = null
write underflow

Go to step 9 [END OF IF]

Step 2 :- set PTR = START

Step 3 :- Repeat Step 4 while PTR → DATA | = num

Step 4 :- set PTR = PTR → next [END OF LOOP]

Step 5 :- set temp = PTR → next

Step 6 :- set PTR → next - temp → next

Step 7 :- set temp → next → prev = PTR

Step 8 :- Free temp

Step 9 :- EXIT.

Unit NO : 05 (Trees)

1) Pre-order Traversal :-

→ Step 1 :- Repeat Steps 2 to 4 while Tree != null
Step 2 :- write tree → data
Step 3 :- PreOrder (tree → left)
Step 4 :- PreOrder (tree → right)
[END OF LOOP]
Step 5 :- END.

2) In Order Traversal :-

→ Step 1 :- Repeat Steps 2 to 4 while TREE != null
Step 2 :- Inorder (tree → left)
Step 3 :- write tree → data
Step 4 :- inorder (tree → right)
[END OF LOOP]
Step 5 :- EXIT

3) Post order Traversal :-

→ Step 1 :- Repeat steps 2 to 4 while
Tree != null
Step 2 :- PostOrder (tree → left)
Step 3 :- PostOrder (tree → right)
Step 4 :- write Tree → data
[END OF LOOP]
Step 5 :- END.

4)

Operations on BST :-

→ Step 1:- if tree → data = val or tree = null RETURN Tree
ELSE
if val < tree → data
Return searchElement (tree → left , val)
else
Return searchElement (tree → right , val)
[END OF IF]
[END OF IF]

Step 2: END

5)

inserting a new node in a BST :-

→ Step 1: if tree = null
allocate memory for tree
set tree → data = val
set tree → left = tree → right
= null
else
if val < tree → data
insert (tree → left , val)
else
insert (tree → right , val)
[END OF IF] [END OF IF]

Step 2: END.

6) Deleting a Node from the BST :-

→ Step 1: if tree = null

 write "val not found in the tree."

else if val < tree → data

 delete (tree → left, val)

else if val > tree → data

 delete (tree → right, val)

else if tree → left and tree → right

 set temp = findLargestNode (tree → right)

 set tree → data = temp → data

 delete (tree → left, temp → data)

else

 set temp = tree

if tree → left = null and tree → right = null

 set tree = null

else if tree → left != null

 set tree = tree → left

else

 set tree = tree → right

[END OF IF]

FREE TEMP

[END OF IF]

Step 2: END

Unit NO: 06 (Graph)

1) Algorithm for Breadth-first Search:-

→ Step 1:- Set STATUS = 1 (ready state)
for each node in GT

Step 2:- enqueue the starting node A
and it's STATUS = 2
(waiting state)

Step 3:- Repeat steps 4 and 5 until
queue is empty

Step 4:- dequeue a node N . Process it
and set it's STATUS = 3

Step 5:- enqueue all the neighbours of
N that are in the ready
state (whose STATUS = 1) and set
their STATUS = 2

(waiting state)
[END OF LOOP]

Step 6:- EXIT.

2) Algorithm for depth - first search :-

→ Step 1:- set status = 1 (ready state) for each node in G

Step 2:- push the starting node A on the stack and set it's status = 2 (waiting state)

Step 3:- Repeat Steps 4 and 5 until the stack is empty.

Step 4:- Pop the top node N. Process it and set it's status = 3 (processed state)

Step 5:- Push on the stack all the neighbours of N that are in the ready state (whose status = 1) and set their status = 2 (waiting state)

[END OF LOOP]

Step 6:- EXIT