VII Abstract Data Type and Data Structure

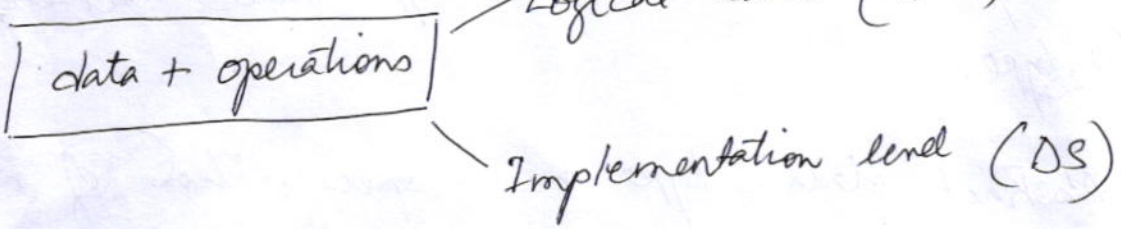
When an application requires a special kind of data which is not available as a built-in data-type, then it is the programmer's responsibility to implement his own kind of data. Here, the programmer has to specify how to store a value for that data, what are the operations that can meaningfully manipulate variables of that kind of data, amount of memory required to store a variable. The programmer has to decide all these things and accordingly implement them. Programmer's own data type is termed as abstract data type. The abstract data type is also alternatively called user-defined data type.

* Abstract data type is a specification of data types having some defined set of operations and which are independent of their implementation.

Example: Stack is an ADT, which can be implemented with an array or a linked list (both different data structure). There can be many different implementation of a given ADT.

⊛ Abstract data type is the logical picture of the data and the operations to manipulate the component elements of the data. Data structure is the actual ~~implementation~~ representation of the data during the implementation and the algorithms to manipulate the data elements. ADT is in the logical level and data structure is in the implementation level.

⊛ A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.



ARRAY

Definition

An array is a finite and ordered collection of homogeneous data elements. An array is finite because it contains only a limited number of elements, and ordered, as all the elements are stored one by one in contiguous locations of the computer memory in linear ordered fashion. All the elements of an array are of the same data type.

Operations on Array

① Traversing

This operation is used to visit all elements in an array.

```
TRAVERSE()
1.  i = LB
2.  Repeat while i ≤ UB
        1. Process A[i]
        2. i = i+1
3.  End while
4.  Stop
```

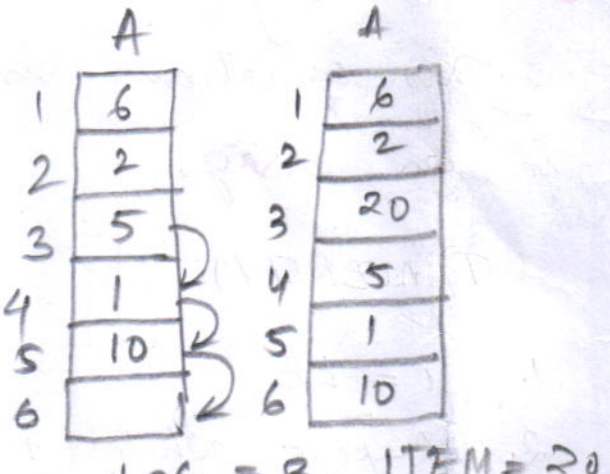
Here the Process() is a procedure which when called for, an element can perform an action. For example, display the element on the screen.

② Inserting

INSERT (A, N, LOC, ITEM)

Here A is a linear array with N elements and LOC is a positive integer such that LOC ≤ N. This algorithm inserts an element ITEM at the position LOC in A.

1. i = N
2. Repeat while i >= LOC
   1. A[i+1] = A[i] // move i^{th} element downward.
   2. i = i - 1
3. End while
4. A[LOC] = ITEM
5. N = N + 1 // Reset N.
6. Stop



LOC = 3. ITEM = 20

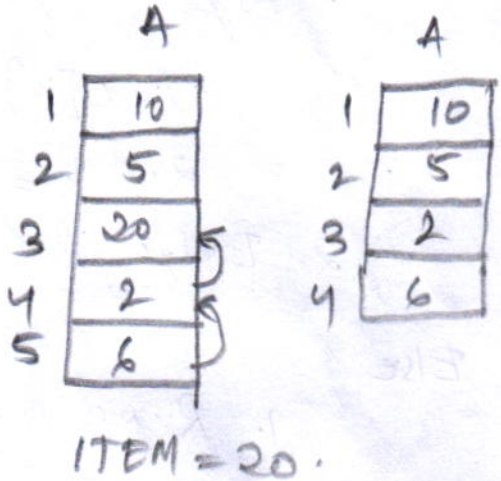
③ Deleting

DELETE (A, N, LOC, ITEM)

Here A is a linear array with N elements and LOC is a positive integer such that

$LOC \leq N$. This algorithm deletes the LOC^{th} element from A.

1. ITEM = A[LOC]
2. i = LOC
3. Repeat while $i < N$
   1. A[i] = A[i+1] // Move the $(i+1)^{th}$ element upward.
   2. i = i+1
4. End while
5. N = N-1 // Reset N
6. Stop.



ITEM = 20.

④ Merging

MERGE()

Here A and B are two linear sorted arrays with M and N elements respectively. This algorithm combines these two sorted arrays to form a new sorted array C with M+N elements.

1. I = 1, J = 1, k = 1
2. Repeat while (I ≤ M and J ≤ N)
   1. If (A[I] ≤ B[J]) then
      1. C[k] = A[I]
      2. I = I+1
   2 #se

1. C[K] = B[J]
2. J = J+1

3. Endif

3. k = k+1

4. Endwhile

5. If (I > M)

1. Repeat for L = J to N

1. C[k] = B[L]

2. k = k+1

2. Endfor

6. Else

1. Repeat for L = I to M

1. C[k] = A[L]

2. k = k+1

2. Endfor

7. Endif

8. Stop.

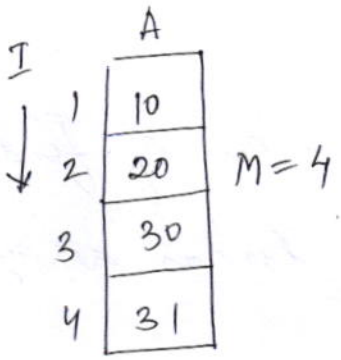
I > M → copy rest of B (J to N)

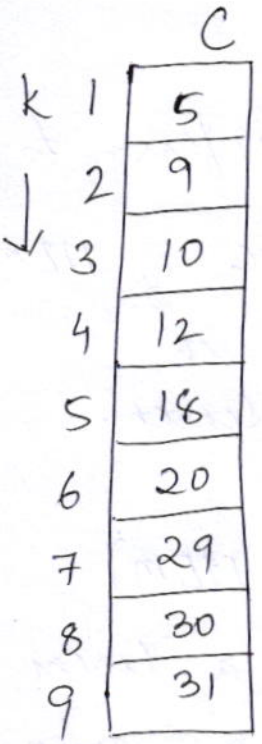
J > N → copy rest of A (I to M)

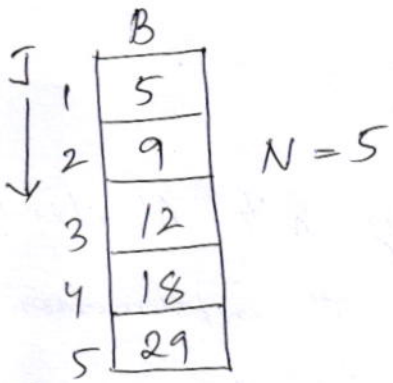
I → 1 to M (A)

J → 1 to N (B)

k → C.







Memory Representation of Array

Searching

Searching refers to the operation of finding the location LOC of ITEM in the linear array A.

↳ Linear Search
↳ Binary Search.

⊛ LINEAR (A, N, ITEM)

Here A is a linear array with N elements and ITEM is a given item of information.

The algorithm finds the location LOC of ITEM in A.

In this method, ITEM is compared with each element of A one by one.

1. LOC = 1
2. While (A[LOC] != ITEM) and (LOC <= N)
   1. LOC = LOC + 1
3. End While
4. If (LOC = N+1) then
   1. LOC = 0
5. End if.
6. return LOC.
7. Stop.

LOC is set to 0 if ITEM is not found.

Complexity of Linear search is $O(n)$.

(*) BINARY (A, N, ITEM)

Here A is a sorted array with N elements. ITEM is a given item of information. The variables BEG, END and MID denote the beginning, end and middle respectively of a segment of elements of A. This algorithm finds the location LOC of ITEM in A.

1. BEG = 1, END = N, MID = INT ((BEG + END)/2)
2. While (A[MID] != ITEM) and (BEG ≤ END)
    1. If (ITEM < A[MID]) then
        END = MID - 1
    2. Else BEG = MID + 1
    3. End if
    4. MID = INT((BEG + END)/2).
3. Endwhile
4. If (A[MID] = ITEM) then.
    LOC = MID.
5. Else LOC = 0.
6. End if.
7. Return LOC.
8. Stop.

Suppose A is an array which is sorted in increasing order. The binary search algorithm applied to our array A works as follows.

During each stage of the algorithm, the search of ITEM is reduced to a segment of elements of A:

$$A[BEG],\ A[BEG+1],\ \ldots,\ A[MID],\ \ldots,\ A[END]$$

Note that the variables BEG and END denote respectively the beginning and end locations of the segment under consideration. The algorithm compares ITEM with the middle element $A[MID]$ of the segment, where MID is obtained by

$$MID = INT\ ((BEG + END)/2)$$

If $A[MID] = ITEM$, then the search is successful and we can set $LOC = MID$, otherwise a new segment of A is obtained as follows:

(a) If $ITEM < A[MID]$, then ITEM can appear only in the left half of the segment.

$$A[BEG],\ A[BEG+1],\ \ldots,\ A[MID-1].$$

So we reset END = MID - 1 and begin Searching again.

(b) If ITEM > A[MID], then ITEM can appears only in the right half of the Segment:

A[MID+1], A[MID+2], ... , A[END].

So we reset BEG = MID + 1 and begin Searching again.

Initialey, we begin with the entire array A, ie we begin with BEG = 1 and END = N.

If the search is unsuccessful, we assign

LOC = 0.

Example:

Let A be the following sorted 13 element array A: 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99

Suppose ITEM = 40. The search for ITEM in the array A is shown below, where the values of A[BEG] and A[END] in each stage of the algorithm are indicated by circles and the value of A[MID] by square. Specifically BEG, END and MID will have the following successive values.

1. Initialey, BEG = 1 and END = 13 Hence.

MID = INT (1+B)/2 = 7 and so

A[MID] = 55.

2. Since 40 < 55, END has its value changed by

END = MID − 1 = 6 Hence.

MID = INT (1+6)/2 = 3 and so

A[MID] = 30.

3. Since 40 > 30, BEG has its value changed by

BEG = MID + 1 = 4 Hence.

MID = INT (4+6)/2 = 5 and so

A[MID] = 40.

We have found ITEM in location LOC = MID = 5

1.) (11) 22, 30, 33, 40, 44, [55], 60, 66, 77, 80, 88, (99)

2.) (11), 22, [30], 33, 40, (44), 55, 60, 66, ~~77~~, ~~80~~, 88, 99.

3.) 11, 22, 30, (33), [40], (44), 55, 60, 66, ~~77~~, ~~80~~, ~~88~~, 99

Binary Search for ITEM = 40.