Date
Page

Unit - II

11-07-19

Array, Records and Pointers

★ Linear Array :

- A linear Array is a list of finite no. of 'n' of homogeneous data, elements (same types) such that
i) The elements of the array are a reference resp. by an index set consisting of 'n' consecutive nos
ii) The elements of the array are stored ~~up~~ resp. in successive memory location.
- The no. 'n' of elements is called the length or size of the array. If not explacitely stated, we will assume the inder set consist of the integers 1, 2, 3, ... n
- In general, the length or the no. of data elements of the array can be obtained from index set by the formula

$$\boxed{\text{Length} = UB - LB + 1}$$

where, UB is the largest Index called upper bound and LB is the smallest Index called Lower bound of the array.

- Note that length = UB when LB = 1.
- The elements of an array 'A' may be denoted by subscript notation $A_1, A_2, A_3 \ldots A_n$.
Parenthesis notation $A_{(1)}, A_{(2)}, A_{(3)}, \ldots A_{(n)}$
Bracket notation $A[1], A[2], A[3], \ldots A[n]$.

Eg: DATA

1	247
2	46
3	429
4	135
5	87
6	156

fig: a

DATA

247	56	429	135	87	156.
1	2	3	4	5	6

fig: b.

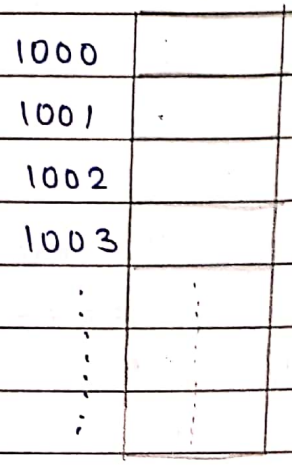
- Array 'DATA' is frequently pictured above in fig 'a' & 'b'. There are 6 elements in array of integers such that : DATA [1] = 247
  DATA [2] = 46
  DATA [3] = 429
  DATA [4] = 135
  DATA [5] = 87
  DATA [6] = 156.

Eg.: An automobile company uses an array auto to record the no. of automobiles sold each year from 1932 to 1984. Rather than beginning the index set with 1, it is more useful to begin the index set with 1932 to so that,

Length = 1984 - 1932 + 1
= 52 + 1
= 53.

★ Representation of Linear Array (LA) in memory :

- Let 'LA' be the linear array in memory of the computer. Recall that the memory of the computer is simply a seq. of address location in following fig.



Let us use the notation

LOC[LA(K)] = address of the element LA(K) of the array LA.

The element of LA are stored in succesive memory cells. Accordingly the comp. doesnot need to keep tract of the address of the every element of LA, but need to keep track only the address of 1st element of LA is denoted by base (LA).

And called the base address of LA. Using these address the comp. calculate the address of any element of LA using following formula.

formula :

$$\boxed{LOC[LA(K)] = Base(LA) + W(K - LB)}$$

where, W is the no. of the words per memory cell for the array LA.

K is subscript value.

- Consider the array 'AUTO' which record the no of automobiles sold each year from 1932 to 1984 as following fig.

200	1	1932
201	2	
202	3	
203	4	
204	1	1933
205	2	
206	3	
207	4	
208	1	1934
	2	
	3	
	4	
	.	

LOC (LA(K)) = Base(LA) + W (K - LB)

Base(Auto) = 200 ; W = 4

LOC (AUTO) (1932) = 200

LOC (AUTO) (1933) = 204

LOC (AUTO) (1934) = 208

Eg:

- LOC (AUTO) (1965) = 200 + 4(1965 - 1932)
  = 200 + 4(33)
  = 200 + 132 = 332.
- LOC (AUTO) (1998) = 200 + 4(1998 - 1932)
  = 200 + 4(66)
  = 464.

- LOC (AUTO) (2020) = 200 + 4(2020 - 1932).
  = 200 + 4(88)
  = 552
- LOC (AUTO) (2050) = 200 + 4 (2050 - 1932)
  = 200 + 4(118)
  = 672
- LOC (AUTO) (2100) = 200 + 4 (2100 - 1932)
  = 200 + 4 (168)
  = 872

15-07-19

* Traversing Linear Array :
  ~~Transverse~~ TRAVERSE [LA, UB, LB, K].

Here LA is linear array with lower bound LB & upper bound UB. This algorithm travels LA applying an operation travels to each elements of LA.

Let A be a collection of data elements stored in memory of the computer. Suppose we want to print the contents of each elements of A or suppose we want to count the number of element of A with given property this can be accomplish by traversing A i.e., by accessing & processing each elements exactly once.

- Steps:

i] [Initilize counter]
  Set K := LB.

ii] Repeat step 3 & 4 while
K ≤ UB.

iii] [visit elements] Apply process to LA[K].

iv] [Increase counter]
Set K := K+1
[END Step 2]

v] Exit.

★ Insert elements into Linear array.
INSERT [LA, K, N, ITEM].

Here LA is linear array with 'N' elements & K is positive integer such as that k is.
This algorithm is insert an elements ITEMS into the K position of LA.

• Steps :

i] Initilize Counter
Set J := N

ii] Repeat step 3 & 4 while
J ≥ K.

iii] [Move J^{th} element downward]
Set LA[J+1] = LA[J].

iv] [Decrease Counter]
Set J := J-1.
[End of step 2 loop].

v] [Insert Element]
Set LA[K] = ITEM.

vi] [Reset N]
Set N = N+1.

vii] Exit.

This algorithm INSERT data elements ITEM into the K position in linear array LA with N elements. The first four step create space in LA by moving downward one location each element from the K position. We emphasize that these elements are move in reverse order i.e, first LA of N.

LA(N-1) & last LA[k] otherwise date can be ~~ear~~ erase we first set J=N using J as counter decrease J each time the loop is executed until J reached K.

The next step 5 insert item into the array the space just created before exist from the algorithm the number 'n' increase by one for a new element.

16-07-19

A linear array is a collection of data elements in the memory of the comp. Inserting refers to the operation of adding another element to the collection of linear array.

Inserting an element at the end of linear array can be easily done provided the memory space allogated for the array is last in a to accomodate the addition element. On the other hand suppose we need to insertion element in the middle of array, then on the average half of the element must be moved downward to new locations to accomodate the new elements & keep the order of the other elements.

eg.:

LA

1	Brown
2	Davis
3	Johnson
4	Smith
5	Wagner

n=5

ITEM = Ford

LA

1	Brown
2	Davis
3	Ford
4	Johnson
5	Smith
6	Wagner

n=n+1
=5+1
=6.

ITEM = Taylor.

LA

1	Brown
2	Davis
3	Ford
4	Johnson
5	Smith
6	Taylor
7	Wagner

n=n+1
n=6+1
=7

* Deleting element from Array :

DELETE(LA, K, N, ITEM).

(Here LA is linear array with N elements & K is positive integer such that K <= N, this algorithm delete the k^{th} element of LA).

- Steps :

i] Set ITEM = LA[K].
ii] Repeat for J = K to N-1.
iii] [Move J^{th} element upward].
Set LA[J] = LA[J+1]
[End of loop].
iv] [Reset the number N of element].
Set N = N-1.
v] Exit.

Eg. :

	ITEM
1	Brown
2	Davis
3	Ford
4	Johnson
5	Smith
6	Taylor
7	Wagner

	After Deletion	
1	Brown	
2	Davis	
3	Ford	
4	Johnson	N = 7
5	Taylor	N = 7-1
6	Wagner	= 6.

A deletion refers to the operation of removing one of the element from linear array. Deleting an element at the end of array presents no difficulties but deleting an element somewhere in the middle of the array would req. that each subsiquent elements will move one location upward in order to fill up the array. Note that the terms downward refers to location with larger ~~relationship~~ subscript and the term upward refers to location with smaller subscript.

17-07-19

Searching Methods:

* Linear Searching.

LINEAR [DATA, N, ITEM, LOC]

Here DATA is a linear array with N elements & ITEM is a given item of info.

This algorithm finds the location 'LOC' of atom in data or set LOC = 0 if the search is unsuccessful.

• Steps:

i] [Insert ITEM at the end of DATA].
Set DATA (N+1) = ITEM.

ii] [Initialize counter] set LOC := 1.

iii] [search for ITEM]
Repeat while DATA [LOC] ≠ ITEM.
set LOC := LOC + 1
[END of LOOP]

iv] [successful ?] If LOC = N+1 then set LOC = 0.

v] Exit.

* Binary Searching:

Eg: Here data be the following sorted 13 elements array.
11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99.
We apply the binary search to DATA for diff. values of ITEM.

i] Suppose ITEM = 40

ii] The search for the atom in the array DATA where the values of DATA [BEG] & DATA [END] in each stage

of the algorithm are indicated by circle and the values of data of MID by square.

Specifically, BEG, END & MID will have the following succesive values.

i] BEG = 1

END = 13

MID = INT [BEG + END/2] = 1 + 13/2 = 7.

DATA [MID] = 55

ii] If ITEM ~~Item~~ 40 < 55, END has its value changed.

END = MID - 1

= 7 - 1 = 6.

Hence, MID = INT [BEG + END/2] = 1 + 6/2 = 7/2 = 3

& DATA [MID] = 30

iii] If ITEM 30 < 40, BEG has its value changed

BEG = MID + 1

= 3 + 1 = 4.

Hence, MID = INT [BEG + END/2] = [4 + 6/2] = 5.

& DATA [MID] = 40.

So, we have found ITEM in LOC = MID = 5.

B] Suppose ITEM = 80

i] BEG = 1

ii] END = 13

MID = INT [BEG + END/2] = 1 + 13/2 = 7.

DATA [MID] = 55.

ii] If ITEM 80 > 55, BEG has its value changed

BEG = MID + 1

= 7 + 1 = 8.

Hence, MID = INT [BEG + END/2] = [8 + 13/2] = 21/2 = 10.

& DATA [MID] = 77.

iii] If ITEM 80 < 77, BEG has its value changed.

BEG = MID + 1

= 10 + 1 = 11

Hence, DATA [BEG] = 80.

So, we have found ITEM in LOC = BEG = 11.

c] Suppose ITEM = 88

If ITEM = 80 < 88, BEG has its value changed.

BEG = MID + 1

= 11 + 1 = 12

& DATA [BEG] = 88

So, we have found ITEM in LOC = BEG = 12.

18-07-19

Eg: 13, 23, 27, 33, 51, 65, 77, 66, 85.

A] Suppose ITEM = 57.

i] BEG = 1

END = 8

MID = INT [BEG + END/2) = 1 + 8/2 = 4.

DATA (MID) = 33

ii] ITEM 57 > 33

BEG = MID + 1

= 4+1 = 5.

Hence, MID = INT (BEG + END/2) = (5 + 8/2) = 6

& DATA (MID) = 57.

B] Suppose ITEM = 23

i] BEG = 1

END = 8

MID = INT (BEG + END/2) = 1 + 8/2 = 4.

DATA (MID) = 33

ii] ITEM = 23 < 33

END = MID - 1

= 4 - 1 = 3

MID = INT (BEG + END/2) = 1 + 3/2 = 2

DATA (MID) = 23

c] Suppose ITEM = 27.

i] BEG = 1

END = 8

MID = INT (BEG + END/2) = 1 + 8/2 = 4

DATA (MID) = 33

ii] ITEM = 27 < 33

END = MID - 1

= 4 - 1 = 3

MID = INT (BEG + END/2) = 1 + 3/2 = 2

DATA (END) = 27

★ Algorithm :

BINARY (DATA, ITEM, LOC, N, LB, UB).

Here DATA is sorted array with lower bound LB & upper bound UB & ITEM is a given item of info. The variables BEG, END & MID denote resp. the beginning, end & middle location of a segment of elements of DATA. This algorithm find the location LOC (ITEM) in DATA or set LOC = NULL.

- Steps :

i] [Initialize segment variables]
Set BEG := LB, END := UB & MID = INT (BEG + END/2)

ii] Repeat step 3 & 4 while, BEG ≤ END & DATA (MID) ≠ ITEM.

iii] If ITEM < DATA (MID) then
set END = MID - 1
Else
set BEG = MID + 1.
(END of If Structure).

iv) Set MID = INT ((BEG + END)/2).
[END of step 2 loop)

v) If DATA (MID) = ITEM then
set LOC := MID
Else
set LOC := NULL
[End of If Structure].

vi] Exit

Bubble Sort:

Eg: (32), 51, 27, 85, 66, 23, 13, 57

32, (51), (27), 85, 66, 23, 13, 57

32, 27, (51), (85), 66, 23, 13, 57

32, 27, 51, (85), (66), 23, 13, 57

32, 27, 51, 66, (85), (23), 13, 57

32, 27, 51, 66, 23, (85), (13), 57

32, 27, 51, 66, 23, 13, (85), (57)

Path 1 Result: 32, 27, 51, 66, 23, 13, 57, 85.

(32), (27), 51, 66, 23, 13, 57, 85.

27, 32, 51, (66), (23), 13, 57, 85

27, 32, 51, 23, (66), (13), 57, 85

Path 2: 27, 32, 51, 23, 13, (66), (57), 85

Result: 27, 32, 51, 23, 13, 57, 66, 85

Path 3: 27, (32), (51), (23), 13, 57, 66, 85

Result: 27, ~~32~~, ~~51~~, ~~23~~, (13), 57, 66, 85

27, (32), (23), 13, 51, 57, 66, 85

27, 23, (32), (13), 51, 57, 66, 85

Path 4: 27, (23), (13), 32, 51, 57, 66, 85

(27), (13), 23, 32, 51, 57, 66, 85

Path 5: ~~(13)~~, (27), (23), 32, 51, 57, 66, 85

13, 23, 27, 32, 51, 57, 66, 85.

(2) ⟹ (99), (77), 40, 44, 80, 55, 33, 11, 22

77, (99), (40), 44, 80, 55, 33, 11, 22.

77, 40, (99), (44), 80, 55, 33, 11, 22

77, 40, 44, (99), (80), 55, 33, 11, 22

77, 40 44 , 80, (99) (55) 33 11 22

77 40 44 80 55 (99) (33) 11 22

77 40 47 80 55 33 (99) (11) 22

77 40 47 80 55 33 11 (99) (22)

(77) (40) 47 80 55 33 11 22 99

Path 1: 40 (77) (47) 80 55 33 11 22 99

40 47 77 (80) (55) 33 11 22 99

Path 2: 40 47 77 55 (80) (33) 11 22 99

40 47 77 55 33 (80) (11) 22 99

40 47 77 55 33 11 (80) (22) 99

40 47 (77) (55) 33 11 22 80 99

Path 3: 40 47 55 (77) (33) 11 22 80 99

40 47 55 33 (77) (11) 22 80 99

40 47 55 33 11 (77) (22) 80 99

40 47 (55) (33) 11 22 77 80 99

Path 4: 40 47 33 (55) (11) 22 77 80 99

40 47 33 11 (55) (22) 77 80 99

40 (47) (33) 11 22 55 77 80 99

Path 5: 40 33 (47) (11) 22 55 77 80 99

40 33 11 (47) (22) 55 77 80 99

(40) (33) 11 22 47 55 77 80 99

Path 6: 33 (40) (11) 22 47 55 77 80 99

33 11 (40) (22) 47 55 77 80 99

Path 7: (33) (11) 22 40 47 55 77 80 99

11 (33) (22) 40 47 55 77 80 99

11 22 33 40 47 55 77 80 99.

06-08-19

Bubble (DATA, N).

Repeat
1. Step 2 & 3 for K=1 to N-1.
2. Set PTR=1 [Initialize pass pointer PTR].
3. Repeat while PTR ≤ N-K [Execute Pass]
   a) If DATA [PTR] > DATA[PTR+1] than
      Interchange DATA [PTR] & DATA [PTR+1]
      [End of If structure]
   b) Set PTR = PTR + 1.
      [End of inner loop]
      [End of step 1 outer loop]
4. Exit.

Insertion sort:

Pass	$-\infty$ A[0]	77 [A[1]]	33 [A_2]	44 [A_3]	11 [A_4]	88 [A_5]	22 [A_6]	66 [A_7]	55 [A_8]
K=1	$-\infty$	(77)	33	44	11	88	22	66	55
K=2	$-\infty$	77	(33)	44	11	88	22	66	55
K=3	$-\infty$	33	77	(44)	11	88	22	66	55
K=4	$-\infty$	33	44	77	(11)	88	22	66	55
K=5	$-\infty$	33	44	77	(88)	88	22	66	55
K=6	$-\infty$	11	33	44	77	88	(22)	66	55
K=7	$-\infty$	11	22	33	44	77	88	(66)	55
K=8	$-\infty$	11	22	33	44	66	77	88	(55)
Sorted array	$-\infty$	11	22	33	44	55	66	77	88

Eg: 33, 51, 27, 85, 66, 23, 13, 57

Pass	$-\infty$	33	51	27	85	66	23	13	57
		$[A_0]$	A_1	A_2	A_3	A_4	A_5	A_6	A_7
k=1	$-\infty$	(33)	51	27	85	66	23	13	57
k=2	$-\infty$	33	(51)	27	85	66	23	13	57
k=3	$-\infty$	33	51	(27)	85	66	23	13	57
k=4	$-\infty$	27	33	51	(85)	66	23	13	57
k=5	$-\infty$	27	31	51	85	(66)	23	13	57
k=6	$-\infty$	27	31	51	66	85	(23)	13	57
k=7	$-\infty$	23	27	31	51	66	85	(13)	57
k=8	$-\infty$	13	23	27	31	51	66	85	(57)
Sorted array k=9	$-\infty$	13	23	27	31	51	57	66	85

07-08-19

⟹ Insertion (A, D)

(This algorithm sort the array A with N elements)

- Steps:

1. Set A[0] := $-\infty$ [initialize sentinel elements].
2. Repeat step 3 to 5 for K=2 ... N.
3. Set TEMP := A[K] & PTR := K-1.
4. Repeat while TEMP < A[PTR]
   a) Set A[PTR+1] := A[PTR] [Moves Element forward].
   b) Set PTR := PTR-1.
   [End of loop].
5. Set A[PTR+1] := TEMP (insert element in proper place).
   [End of step 2 loop]
6. Exit.

Selection Sort :

Eg: 77, 33, 44, 11, 88, 22, 66, 55

Pass	77 A[1]	33 A[2]	44 A[3]	11 A[4]	88 A[5]	22 A[6]	66 A[7]	55 A[8]
K=1, LOC=4	(77)	33	44	(11)	88	22	66	55
K=2, LOC=6	11	(33)	44	77	88	(22)	66	55
K=3, LOC=6	11	22	(44)	77	88	(33)	66	55
K=4, LOC=6	11	22	33	(77)	88	(44)	66	55
K=5, LOC=8	11	22	33	44	(88)	77	66	(55)
K=6, LOC=7	11	22	33	44	55	(77)	(66)	88
K=7, LOC=7	11	22	33	44	55	66	77	88

Eg: 32, 51, 27, 85, 66, 23, 13, 57

Pass	32 A[1]	51 A(2)	27 A(3)	85 A(4)	66 A(5)	23 A(6)	13 A(7)	57 A(8)
K=1, LOC=7	(32)	51	27	85	66	23	(13)	57
K=2, LOC=6	13	(51)	27	85	66	(23)	32	57
K=3, LOC=3	13	23	27	85	66	51	32	57
K=4, LOC=7	13	23	27	(85)	66	51	(32)	57
K=5, LOC=6	13	23	27	32	(66)	(51)	85	57
K=6, LOC=8	13	23	27	32	51	(66)	85	(57)
K=7, LOC=8	13	23	27	32	51	57	(85)	(66)
K=8, LOC=8	13	23	27	32	51	57	66	85

08-08-19

⟹ Selection Sort

MIN [A, K, N, LOC]

An array A is in memory, this procedure finds the location 'LOC' of the smallest elements among A[K], A[K+1] .... A[N].

1. Set MIN = A[K] & LOC = K [Initializes Pointer].
2. Repeat for J = K+1, K+2 ..... N;
   If MIN > A[J] then set MIN = A[J] &
   LOC := A[J] & LOC = J.
   [End of loop]
3. Return.

⟹ Selection Sort

SELECTION [A, N]

This algorithm sort the array A, with N elements.

1. Repeat step 2 & 3 for K = 1, 2, ... N-1.
2. Call MIN (A, N, K, LOC).
3. [Interchange A[K] & A[LOC]]
   Set TEMP := A[K]
   A[K] = A[LOC]
   A[LOC] = TEMP
   [End of Step 1 loop]
4. Exit.