Linear Data Structures and their representation

Marwadi University
Department of CE/IT

Unit 2
Linear Data Structures and their representation
Data Structure (01CE1301)

Array

Need: Storing marks of 20 students need 20 variables

Definition: *An array is a*

fixed size,

sequence collection of elements

of same data type.

Declaration and storing Array Elements

1. int a[3] = {1, 2, 5} [Compile Time]

   Data type   Variable name   size   data elements

2. int i, a[3]; [Run time]

```
int i, a[3];
 for (i=0 ;  i < 3; i++)
     a[i] = i;
```

3. int i, a[3]; [Run time]

```
int i, a[3];
for (i=0 ; i < 3 ; i++)
     scanf(“%d”,&a[i]);
```

Address of Array Elements

a[k] = BaseAddress [a] + w (k – lowerbound)

Where *k*, is the index of element

a, is the array variable name

w, is the number of words per memory location

lowerbound, is starting index

BaseAddress, is the starting address of array or address of first element of array

e.g.

index	0	1	2	3	4	5	6	7
a	10	20	30	40	50	60	70	80
Address	1000	1002	1004	1006	1008	1010	1012	1014

a [3] = 1000 + 2 (3 – 0)

= 1006.

Length of Array

Length = upperbound – lowerbound + 1

Where, upperbound is the last index of array
lowerbound is the first index of array

e.g.

index	12	13	14	15	16	17	18	19
a	10	20	30	40	50	60	70	80
Address	1000	1002	1004	1006	1008	1010	1012	1014

Length = 19 – 12 + 1 = 8.

Operations of Array { **traversal**, insertion, deletion, merging, searching }

1. **Traversal**

   A is the array

   lowerbound is starting index

   upperbound is last index

step 1: set i = lowerbound

step 2: Repeat step 3 to step 4 while i <= upperbound

step 3: print (A[i])

step 4: set i = i + 1

step 5: Exit.

Operations of Array { traversal, **insertion**, deletion, merging, searching }

2. Insertion → I) at the end of Array
→ II) middle of Array

(I) Insertion at the end of Array

Simply add new element at end of array

step 1: upperbound = upperbound + 1

step 2: a[upperbound] = new_value

step 3: Exit.

(II) Insertion at Middle of Array

Operations of Array { traversal, **insertion**, deletion, merging, searching }

(II) Insertion at Middle of Array

index	0	1	2	3	4		
a	10	20	30	40	50		

Insert at a[1], 45 POS=1

index	0	1	2	3	4		
a	10	20	30	40	50		

index	0	1	2	3	4	5	
a	10	20	30	40	50	50	

index	0	1	2	3	4	5	
a	10	20	30	40	40	50	

index	0	1	2	3	4	5	
a	10	20	30	30	40	50	

index	0	1	2	3	4	5	
a	10	20	20	30	40	50	

index	0	1	2	3	4	5	
a	10	45	20	30	40	50	

Operations of Array { traversal, **insertion**, deletion, merging, searching }

2. Insertion

- I) at the end of Array
- II) middle of Array

(II) Insertion at Middle of Array

insert (A, N, POS, new_value)

A, the array in which the element has to be inserted

N, number of elements in the array

POS, the position at which the element has to be inserted, and

new_value, the value has to be inserted

step 1: set i = N-1

step 2: Repeat step 3 to step 4 while i >= POS

step 3: set a[i+1] = a[i]

step 4: set i = i – 1

step 5: set N = N + 1

step 6: set a[POS] = new_value

step 7: Exit.

Operations of Array { traversal, insertion, **deletion,** merging, searching }

3. Deletion → I.) Last element
→ II.) middle element

(I) Delete last element

Simply delete last

step 1: upperbound = upperbound - 1

step 2: Exit.

(II) Delete middle element

Operations of Array { traversal, insertion, **deletion**, merging, searching }

(II) Delete middle element

index	0	1	2	3	4		
a	10	20	30	40	50		
		Delete, POS				POS=1	
index	0	1	2	3	4		i=POS
a	10	20	30	40	50		i=1
index	0	1	2	3	4		
a	10	30	30	40	50		i=2
index	0	1	2	3	4		
a	10	30	40	40	50		i=3
index	0	1	2	3	4		
a	10	30	40	50	50		i=4
index	0	1	2	3	4		
a	10	30	40	50	50		N=N-1

Operations of Array { traversal, insertion, **deletion**, merging, searching }

3. Deletion

I.) Last Element

II.) Middle Element

(II) Delete Middle Element

delete (A, N, POS)

A, the array in which the element has to be inserted

N, number of elements in the array

POS, the position at which the element has to be deleted

step 1: set i = POS

step 2: Repeat step 2 to step 4 while i < N-1

step 3: set a[i] = a[i+1]

step 4: set i = i + 1

step 5: set N = N – 1

step 6: Exit.

Operations of Array { traversal, insertion, deletion, **merging**, searching }

4. Merging

two unsorted array

A

10	50	20	40

B

30	5	15	9

C =

A	B

10	50	20	40	30	5	15	9

two sorted array

A

10	20	40	50

B

5	9	15	30

C = if A < B, then A else B; for all elements

5	9	10	15	20	30	40	50

Operations of Array { traversal, insertion, deletion, merging, **searching** }

5. Searching

- (I) Linear Search
- (II) Binary Search

(I) Linear Search

linear_search (A, N, Value, POS)

A, is the array
N, number of elements in the array
Value, is the data which we want to search
POS, is the index of searched data

step 1: set POS = -1 , i = 0

step 2: Repeat steps 3 while i < N

step 3: if A[i] = Value then

set POS = i

print POS

goto step 5

step 4: print "Value is not present in Array"

step 5: Exit.

Operations of Array { traversal, insertion, deletion, merging, **searching** }

(II) Binary Search

binary_search (A, lb, ub, Value, POS)

A, is the array
lb, is the starting index of array
ub, is the last index of array
Value, is the data which we want to search
POS, is the index of searched data

step 1: set POS = -1, beg = lb, end = ub
step 2: Repeat steps 2 to step 4 while beg <= end
step 3: set mid = (beg+end) / 2
step 4: if A[mid] = Value
    POS = mid
    print POS
    goto step 6
  if Value < A[mid]
    end = mid – 1
  else beg = mid + 1
step 5: if POS = -1, then
    print "Value is not present in Array"
step 6: Exit.

1D Array for Inter-Function Communication

1. Passing Individual Element
   (i) Passing Data Value
   (ii) Passing Address

2. Passing the Entire Array

1D Array for Inter-Function Communication

1. Passing Individual Element

(i) Passing Data Value

Function declaration

```
void fun(int);
void main( )
{
      int a[5] = {1,2,3,4,5};
      fun(a[3]);
}
```

Calling function

Called function

```
void fun(int num)
{
      printf("%d",num);
}
```

****

(ii) Passing Address

```
void fun(int*);
void main( )
{
      int a[5] = {1,2,3,4,5};
      fun(&a[3]);
}
```

Function declaration

Calling function

Called function

```
void fun(int *num)
{
      printf("%d", *num);
}
```

1D Array for Inter-Function Communication

2. Passing the entire Array

Function declaration

```
void fun(int [ ]);
void main( )
{
     int a[5] = {1,2,3,4,5};
     fun(a);
}
```

Calling function

Called function

```
void fun(int b[ ])
{
     int i;
     for(i=0;i<5;i++)
          printf("%d,",b[i]);
}
```

Pointers and Arrays

```
int a[5] = {1,2,3,4,5};
int *ptr;
ptr = &a[0];
ptr = &a[2];
ptr = a;
```

Note that a[i] = *(a+i)

Pointers of Arrays

```
int *ptr[3];
int a=5, b=7, c=10;
ptr [0]= &a;
ptr [1]= &b;
ptr [2]= &c;
printf("%d", *ptr[1]);
```

2D Arrays

Declaration of 2D Array:

data_type array_name [rowsize] [colsize];

int marks[3][3];

marks			
	00	01	02
	10	11	12
	20	21	22

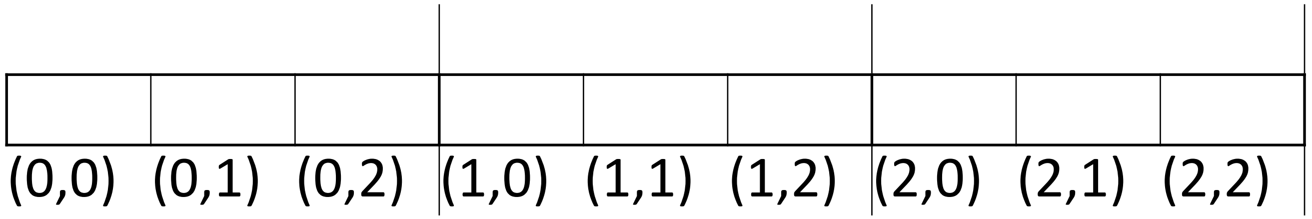
These elements are stored sequentialy; Two ways:

(i) Row Major Order (RMO)

(ii) Column Major Order (CMO)

2D Arrays

(i) Row Major Order (RMO)



A[M][N];

Address A[i, j] = BaseAddress + w*{ N(i-1) + (j-1)}

Where, w is the # of words stored per memory location

N is the number of elements in one Row (num of Columns)

i and j are the subscripts of array element

Example 1: A[20][5], Base Address=1000, number of words per memory location = 2, Compute the address of element, A[18,4]. Assume, the elements are stored in RMO.

2D Arrays

(I) Row Major Order (RMO)

Example 1: A[20][5], Base Address=1000, number of words per memory location = 2, Compute the address of element, A[18,4]. Assume, the elements are stored in RMO.

Solution:

Address of A[i, j] = BaseAddress + w * [N*(i-1) + (j-1)]

Address of A[18,4] = 1000 + 2 * [5 * (18-1) + (4-1)]

= 1000 + 2 * [85 + 3]

= 1176

Example 2: A[10][5], Base Address=2000, number of words per memory location = 2, Compute the address of element, A[8,5]. Assume, the elements are stored in RMO.

Ans:

Address of A[8,5] = 2078.

Example 3: A[10][10], Base Address=1000, number of words per memory location = 2, Compute the address of element, A[8,5]. Assume, the elements are stored in RMO.

Ans:

Address of A[8,5] = 1148.

2D Arrays

(II) Column Major Order (CMO)

2	4	6
1	3	5
3	6	9

2	1	3	4	3	6	6	5	9

For MxN Matrix,

Address of A[i, j] = BaseAddress + w * [M*(j-1) + (i-1)]

Example 1: A[10][10], Base Address=1000, number of words per memory location = 2, Compute the address of element, A[8,5]. Assume, the elements are stored in CMO.

Solution:

Address of A[i, j] = BaseAddress + w * [M *(j-1) + (i-1)]

Address of A[8,5] = 1000 + 2 * [10 * (5-1) + (8-1)]

= 1000 + 2 * [40 + 7]

= 1094

2D Arrays

- **Initialization of 2D Array**

  int A[2][3] = {1,2,3,4,5,6};

  int A[2][3] = { {1,2,3}, {4,5,6}};

  Only the size of first dimension can be omitted.

  int A[ ] [3] = { {1,2,3}, {4,5,6}};

  int A[2][3] = {0};

- **Accessing the Elements**
  - Using two FOR loops

2D Arrays

Operations on 2D Array

1) Transpose: A is MxN then B is NxM where, $B_{i,j} = A_{j,i}$

2) Sum: $C_{i,j} = A_{i,j} + B_{i,j}$

3) Difference: $C_{i,j} = A_{i,j} - B_{i,j}$

4) Product: A is M x N

   B is P x Q

   if N = P then $C_{i,j} = \sum A_{i,k} B_{k,j}$, for k = 1 to N

Sparse Matrices

- It is a matrix that has many elements with a value zero.

- For efficiently utilize the memory, data structure that take advantage of the Sparse structure should be used. (Stored only non-zero elements.)

- Represent using Array (Row and Column wise) or Linked List (Node and link wise).

Sparse Matrices

Lower-Triangular Matrix

All elements above the main diagonal have a value zero.

$A_{i,j} = 0$ where $i < j$

1	0	0	0
5	**3**	0	0
2	**7**	**-1**	0
3	**1**	**4**	**2**

Size (Number of Elements)

$= 1 + 2 + 3 + ... + N$

$= \sum i = N(N+1) / 2$

RMO : { 1, 5, 3, 2, 7,-1, 3, 1, 4, 2}

CMO : { 1, 5, 2, 3, 3, 7, 1, -1, 4, 2}

Sparse Matrices

Lower-Triangular Matrix

	a12	a13	a14	a15
		a23	a24	a25
			a34	a35
				a45
		a53		

RMO :

Address of A[5,3]= BA + w * { 1 ele 1st row + 2 ele 2nd row + 3 ele 3rd row + 4 ele 4th row + 2 ele 5th row}

Address of A[i,j]= BA + w * { 1 + 2 + ...+(i-1) + 2 ele 5th row}

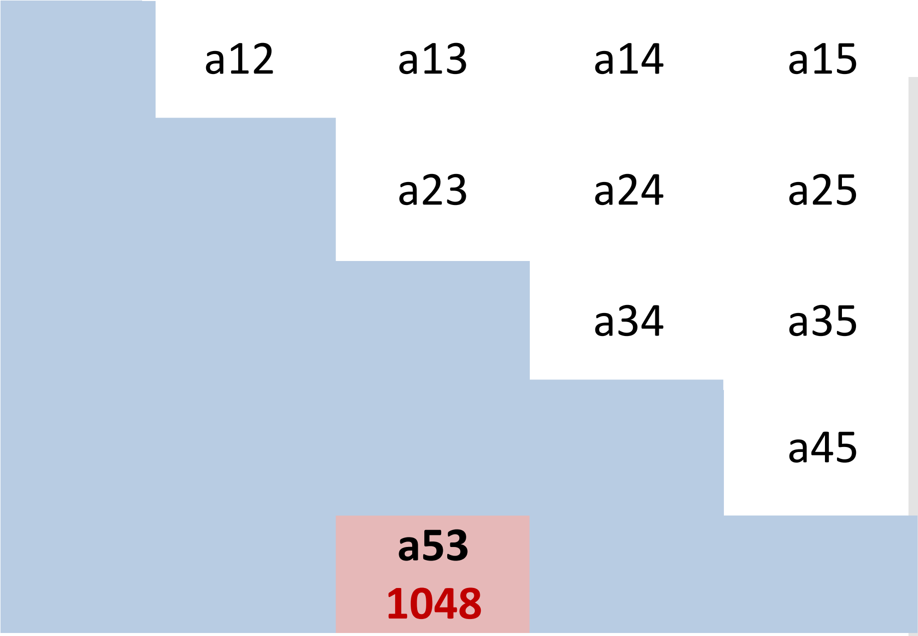
BA + w * { 1 + 2 + ...+(i-1) + (j-1)}

$\boxed{=BA + w*\{ i*(i-1)/2 + (j-1)\}}$

Sparse Matrices

Lower-Triangular Matrix

RMO :



Example:

BA = 1000

w = 4

Calculate Address of A[5,3]

Address of A[i,j]=BA +w*{ i*(i-1)/2 + (j-1)}

Address of A[5,3]=1000 + 4 * { 5*4/2 + (3-1) }

= 1000 + 4 * { 10 + 2 }

= 1000 + 48

= 1048

Sparse Matrices

Lower-Triangular Matrix

CMO :

a11 **1000**	a12	a13	a14	a15
a21 **1004**	**a22** **1020**	a23	a24	a25
a31 **1008**	**a32** **1024**	**a33** **1036**	a34	a35
a41 **1012**	**a42** **1028**	**a43** **1040**	**a44**	a45
a51 **1016**	**a52** **1032**	**A53** **1044**	**a54**	**a55**

Address of A[5,3]= BA + w * { 5 ele 1st col + 4 ele 2nd col + 2 ele 3rd col}

Address of A[i,j]= BA + w * { ∑ 1 to N - ∑ (1 to N-j+1) + (i-j) }

Address of A[5,3]=1000 + 4 * { ∑ 1 to 5 - ∑ (1to(5-3+1)) + (5-3) }

= 1000 + 4 *{ 15 – 6 + 2 } = 1000 + 44 = **1044**

Sparse Matrices

Uper-Triangular Matrix

All elements above the main diagonal have a value non zero.

$A_{i,j} = 0$ where $i > j$

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ 0 & \mathbf{3} & \mathbf{6} & \mathbf{7} \\ 0 & 0 & \mathbf{-1} & \mathbf{9} \\ 0 & 0 & 0 & \mathbf{3} \end{bmatrix}$$

Size (Number of Elements)

$= 1 + 2 + 3 + ... + N$

$= \sum i = N(N+1) / 2$

Tridiagonal Matrix

$A_{i,j} = 0$ where $|i-j| > 1$

$$\begin{bmatrix} 4 & 1 & & & \\ 3 & 2 & 5 & & \\ & 8 & 7 & 6 & \\ & & 9 & 4 & 1 \\ & & & 2 & 5 \end{bmatrix}$$

Size = N + (N-1) + (N-1)

= 3N – 2

RMO : {4, 1, 3, 2, 5, 8, 7, 6, 9, 4, 1, 2, 5}

CMO : {4, 3, 1, 2, 8, 5, 7, 9, 6, 4, 2, 1, 5}

Diagonal-Wise-Mapping: { 3,8,9,2, 4,2,7,4,5, 1,5,6,1}

Array Applications

- Widely used to implement mathematical vectors, matrices and other kinds of rectangular tables.
- Used to implement stack, queue, heap, hash table, string , etc..
- Can be used for dynamic memory allocation.