Data Structures and Algorithms (CS-2001)

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

School of Computer Engineering

4 Credit

Lecture Note

Chapter Contents

Sr #	Major and Detailed Coverage Area	Hrs
2	**Arrays**	2
	Arrays, Two-Dimensional Array, Address Calculation, Dynamically Allocated Arrays, Abstract Data Types, Polynomials, Matrix Addition and Multiplications, Sparse Matrix	

Arrays

Data Structures are classified as either **Linear** or **Non-Linear.**

- **Linear data structure:** A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached. Ex: Arrays, Linked Lists
- **Non-Linear data structure**: Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure. Ex: Trees, Graphs

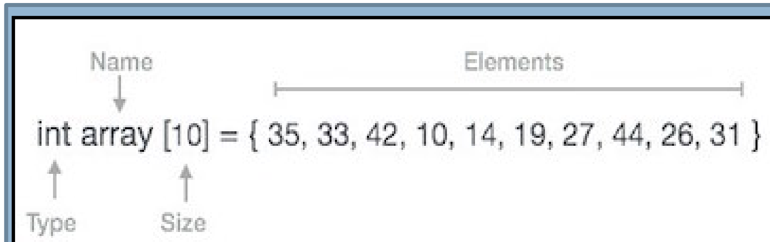
Arrays

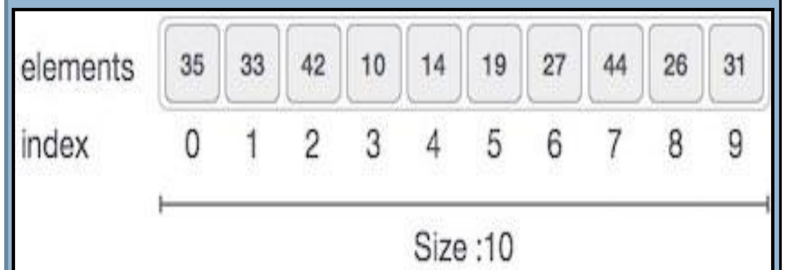
Array is a **container** which can hold fix number of items and these items should be of **same type**. Following are important terms to understand the concepts of Array. Arrays are of one-dimensional or multi-dimensional (i.e. 2 or more than 2)

- **Element –** Each item stored in an array.
- **Index –** Each location of an element in an array has a numerical index which is used to identify the element.

One-Dimensional Array

One-Dimensional array is also called as linear array and stores the data in a single row or column.





Lower Bound (LB) Upper Bound (UB)

Array Index	array[0]	array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]	array[8]	array[9]
Element	35	33	42	10	14	19	27	44	26	31
Address	100	102	104	106	108	110	112	114	116	118

Base Address

Array is of integer type so that element use 2 bytes and is the reason for the difference of 2 bytes (also called as **word Length** i.e w) in element's address. The elements are stored in successive memory location. Computer does not need to keep track of the address of every element of array, but need to keep track only the address of the first element of the array called the base address. Lower Bound (LB) is the smallest index and Upper Bound (UB) is the larget index. Length / Size / Range can be defined as **UB - LB + 1.** Memory location of the array can be tracked by using the base address. To calculate the memory location of an element in an array using the formula Location(array[k]) = Base Address + w(k – LB)

- Index starts with 0
- Array length/size/range is 10 (i.e. 9 – 0 + 1) which means it can store 10 elements.
- Each element can be accessed via its index. For example, we can fetch element at index 6 as 27.
- Address (array[6]) = 100 + 2 * (6 – 0) = 112 ?

1-D Array Address Calculation

Array of an element of an array say "A[i]" is calculated using the following formula:

Address of A [i] = BA + w * (i – LB)

Where,
BA = Base Address
w = Storage Size of one element stored in the array (in byte)
i = Subscript of element whose address is to be found
LB = Lower limit / Lower Bound of subscript, if not specified assume 0 (zero)

Example

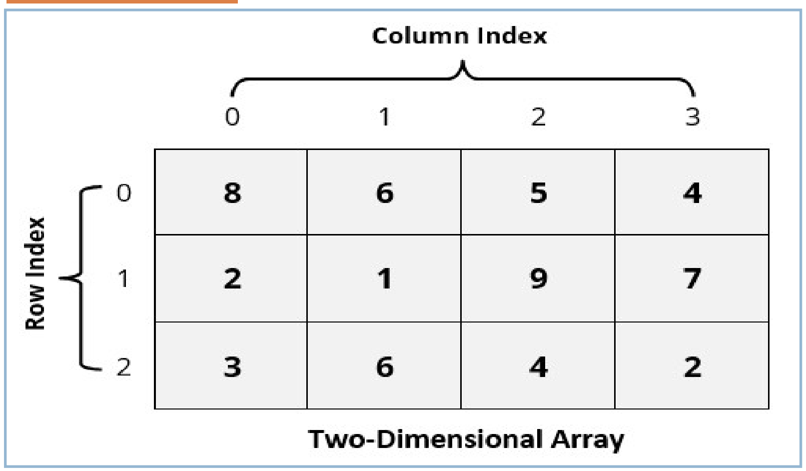
Problem: Given the base address of an array B[1300…..1900] as 1020 and size of each element is 2 bytes in the memory. Find the address of B[1700].

Solution:
The given values are: B = 1020, LB = 1300, W = 2, I = 1700
Address of A [i] = BA + w * (i – LB)
= 1020 + 2 * (1700 – 1300)
= 1020 + 2 * 400
= 1020 + 800 = 1820

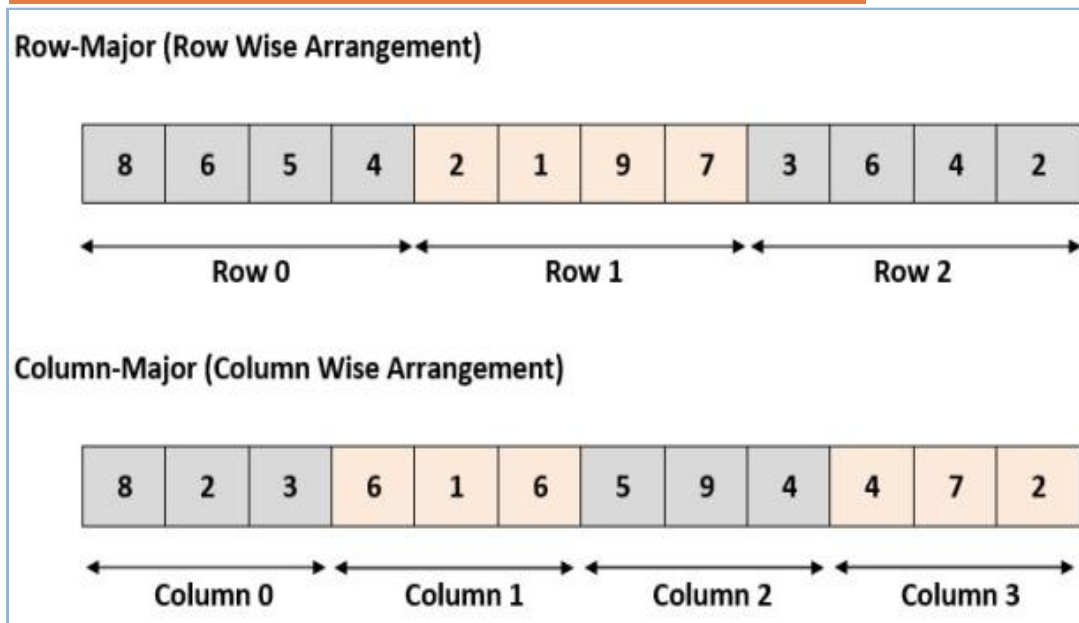
Two-Dimensional Array

C uses the so-called **array-of-arrays** representation to represent a multidimensional array. In this representation, a 2-dimensinal array is represented as a one-dimensional array in which each element is itself a one-dimensional array. In layman term, it is the collection of elements placed in rows and columns. For 2-dimensional, 2 types of memory arrangement i.e. **Row-Major arrangement** and **Column-Major arrangement**

2-D Array



Row-Major and Column-Major arrangement



Note : C supports row major order and **Fortran** supports column major order

Row major & Column order Address Calculation

- **Row-Major Order:** The address of a location in Row Major System is calculated as:
  Address of A [i][j] = BA + w * [n * (i – Lr) + (j – Lc)]
- **Column-Major Order:** The address of a location in Column Major System is calculated as:
  Address of A [i][j] = BA + w * [(i – Lr) + m * (j – Lc)]

Where:

BA = Base Address

i = Row subscript of element whose address is to be found

j = Column subscript of element whose address is to be found

w = Storage Size of one element stored in the array (in byte)

Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)

Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)

m = Number of row of the given matrix

n = Number of column of the given matrix

Row major & Column order Address Calculation cont…

Important: Usually number of rows and columns of a matrix are given (like A[20][30] or A[40][60]) but if it is given as A[Lr- – – – – Ur, Lc- – – – – Uc]. In this case number of rows and columns are calculated using the following methods:

Number of rows (m) will be calculated as = **(Ur – Lr) + 1**
Number of columns (n) will be calculated as = **(Uc – Lc) + 1**

And rest of the process will remain same as per requirement .

Example:

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

In figure there is 3x3 array and memory location start from 200 and each element takes 2 address. Calculate element at Array [3][1] for both row and column major.

Answer: Here m=n=3, i=3, j=1, w=2, base address=200.
Row-Major = BA + w * [n * (i – Lr) + (j – Lc)] = 200+2(3(3-0) + (1-0)) = ?
Column-Major = BA + w * [(i – Lr) + m * (j – Lc)] = 200+2((3-0) + 3*(1-0)) = ?

Class Work

An array X [**-15**..........10, 15...............40] requires one byte of storage. If beginning location is 1500 determine the location of X [15][20].

Answer: As you see here the number of rows and columns are not given in the question. So they are calculated as:

Number or rows say m = (Ur – Lr) + 1 = [10 – (- 15)] +1 = 26

Number or columns say n = (Uc – Lc) + 1 = [40 – 15)] +1 = 26

Column-Major :

The given values are: BA = 1500, w = 1 byte, i = 15, j = 20, Lr = -15, Lc = 15, m = 26

Address of A [i][j] = BA + w * [(i – Lr) + m * (j – Lc)]

= 1500 + 1* [26 * (15 – (-15))) + (20 – 15)]

= 1500 + 1 * [30 + 26 * 5] = 1500 + 1 * [160] = 1660

Row-Major:

The given values are: B = 1500, W = 1 byte, i = 15, j = 20, Lr = -15, Lc = 15, N = 26

Address of A [i][j] = BA + w * [n * (i – Lr) + (j – Lc)]

= 1500 + 1* [26 * (15 – (-15))) + (20 – 15)]

= 1500 + 1 * [26 * 30 + 5] = 1500 + 1 * [780 + 5] = 1500 + 785 = 2285

Dynamic Memory Allocation

The exact size of **array** is **unknown** until the **compile time** i.e. time when a compiler compiles code written in a programming language into a executable form. The size of array you have declared initially can be sometimes **insufficient** and sometimes **more than required**.

What?

The process of allocating memory during program execution is called dynamic memory allocation. It also allows a program to obtain more memory space, while running or to release space when no space is required.

Difference between static and dynamic memory allocation

Sr #	Static Memory Allocation	Dynamic Memory Allocation
1	User requested memory will be allocated at compile time that sometimes insufficient and sometimes more than required.	Memory is allocated while executing the program.
2	Memory size can't be modified while execution.	Memory size can be modified while execution.

Dynamic Memory Allocation Example

One-Dimensional Array

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int n, *list;
  printf("\nEnter the no of elements:");
  scanf("%d", &n);
  if (n < 1)
  {
    printf("Incorrect Value");
    return;
  }
  malloc(list, n * sizeof(int));
  if (!list)
   {
     printf("Insufficient Memory");
     return;
   }
  /* Allow the users to enter values*/
 /* print the values */
  free(list);
  return 0;
}
```

Two-Dimensional Array

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int rows, columns, **list;
  printf("\n Enter the no of rows and columns:");
  scanf("%d%d", &rows, &columns);
  if (rows < 1 || columns <1)
  {
    printf("Incorrect Value");
    return;
  }
  malloc(list, rows * sizeof(*list));
  if (!list)
   {
     printf("Insufficient Memory");
     return;
   }
  for (int i=0; i<rows; ++i)
  {
    malloc(list[i], columns * sizeof(**list));
  }
  if (!list)
   {
     printf("Insufficient Memory");
     return;
   }
 /* Allow the users to enter values*/
 /* print the values */
  free(list);
  return 0;
}
```

Dynamic Memory Allocation Example cont…

Find sum of n elements entered by user

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int n,i,*ptr,sum=0;
   printf("Enter number of elements: ");
   scanf("%d",&n);
   ptr=(int*)malloc(n*sizeof(int));
   // ptr=(int*)calloc(n, sizeof(int));
   if(ptr==NULL)
   {
      printf("Error! memory not allocated.");
      return;
   }
   printf("Enter elements of array: ");

   for(i=0;i<n;++i)
   {
      scanf("%d",ptr+i);
      sum+=*(ptr+i);
   }
   printf("Sum=%d",sum);
   free(ptr);
   return 0;
}
```

Pointer to array

Pointer to 1-D array

```
int i;
int a[5] = {1, 2, 3, 4, 5};
int *p = a;
for (i=0; i<5; i++)
{
 printf("%d,", a[i]);
 printf("%d\n", *(p+i));
}
```

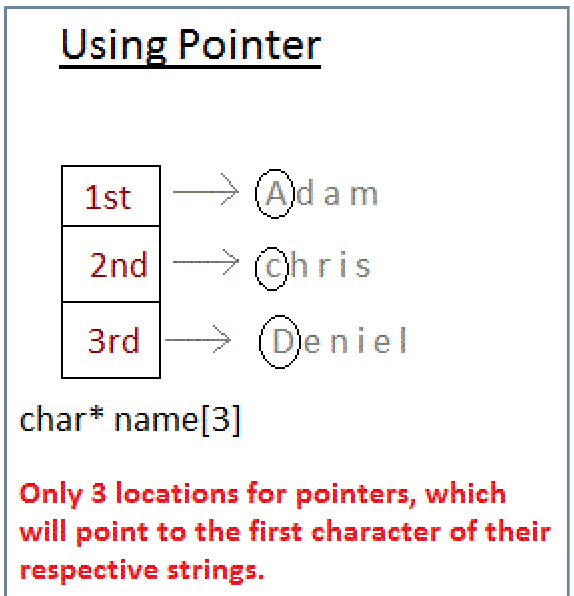
Pointer to 2-D array

```
int i,j;
int a[3,2] = {{1, 2}, {3, 4}, {1,5}};
int **p = (int **) a;
for (i=0; i<3; i++)
{
  for (j = 0;j<2;j++)
  {
    printf("%d,", a[i][j]);
    printf("%d\n", *(*(p+i)+j));
  }
}
```

Array of Pointers

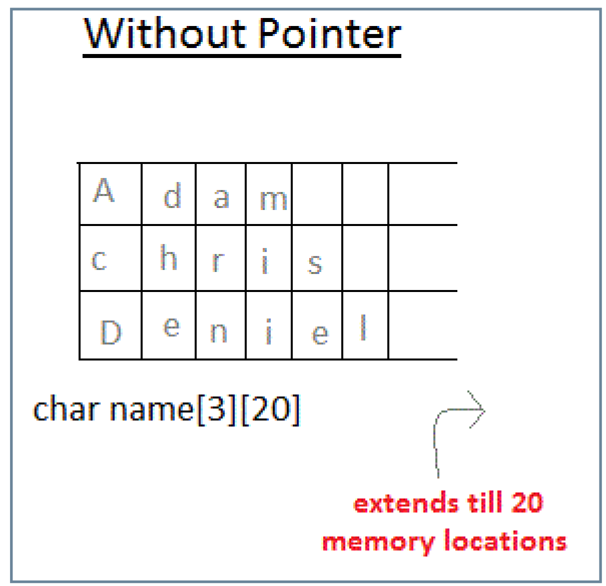
Array of pointers

```
char *name[3]={
            "Adam",
            "chris",
            "Deniel"};
```

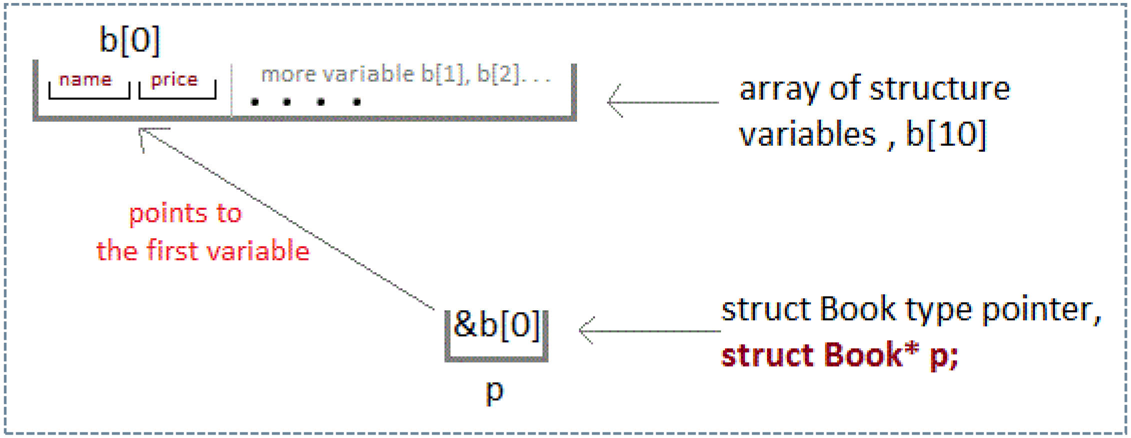


Array without pointers

```
char name[3][20]= {
            "Adam",
            "chris",
            "Deniel"};
```



Pointer to Structure




```
struct Book
{
 char name[10];
 int price;
}

int main()
{
 struct Book a;      //Single structure variable
 struct Book* ptr;   //Pointer of Structure type
 ptr = &a;

ptr->name = "Dan Brown";     //Accessing Structure Members
ptr->price = 500;

 struct Book b[10];   //Array of structure variables
 struct Book* p;      //Pointer of Structure type
 p = &b;
}
```

Array ADT

An abstract data type (**ADT**) is a mathematical model for data types where a data type is defined by its behavior (semantics) from the **point of view of** a **user of the data**, specifically in terms of **possible values**, **possible operations on data of this type**, and t**he behavior of these operations**. When considering **Array ADT** we are more concerned with the **operations** that can be performed on array.

Basic Operations

- **Traversal –** print all the array elements one by one.
- **Insertion –** add an element at given index.
- **Deletion** – delete an element at given index.
- **Search –** search an element using given index or by value.
- **Updation –** update an element at given index.
- **Sorting** – arranging the elements in some type of order.
- **Merging** – Combining two arrays into a single array
- **Reversing** – Reversing the elements

Default Value Initialization

In C, when an array is initialized with size, then it assigns following defaults values to its elements

- **bool –** false
- **char–** 0
- **float–** 0.0
- **double–** 0.0f
- **int–** 0

Basic Operation cont...

Insertion

Insert operation is to insert one or more data elements into an array. Based on the requirement, new element can be added at the beginning or end or any given position of array.

Let LA is a Linear Array (unordered) with N elements and K is a positive integer such that K<=N. Below is the algorithm where ITEM is inserted into the Kth position of LA. Procedure – **INSERT(LA, N, K, ITEM)**

1. Start
2. Set J=N
3. Set N = N+1 **/* Increase the array length by 1*/**
4. Repeat steps 5 and 6 while J >= K
5. Set LA[J+1] = LA[J] **/* Move the element downward*/**
6. Set J = J-1 **/* Decrease counter*/**
7. Set LA[K] = ITEM
8. Stop

Deletion

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Below is the algorithm to delete an element available at the Kth position of LA. Procedure - **DELETE(LA, N, K)**

1. Start
2. Set J = K
3. Repeat step 4 while J < N
4. Set LA[J] = LA[J+1] **/* Move the element upward*/**
5. Set N = N-1 **/* Reduce the array length by 1 */**
6. Stop

Basic Operation cont…

Search

You can perform a search for array element based on its value or position.

Consider LA is a linear array with N elements. Below is the algorithm to find an element with a value of ITEM using sequential search. Procedure - **SEARCH(LA, N, ITEM)**

1. Start
2. Set J=1 and LOC = 0
3. Repeat steps 4 and 5 while J < N
4. IF (LA[J] = ITEM) THEN LOC = J AND GOTO STEP 6
5. Set J = J +1
6. IF (LOC > 0) PRINT J, ITEM ELSE PRINT 'Item not found'
7. Stop

Updation

Update operation refers to updating an existing element from the array at a given position.

Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Below is the algorithm to update an ITEM available at the Kth position of LA. Procedure - **UPDATE(LA, N, K, ITEM)**

1. Start
2. Set LA[K] = ITEM
3. Stop

Basic Operation cont...

Traversal

Traversal operation refers to printing the contents of each element or to count the number of elements with a given property

Consider LA is a linear array with N elements. Below is the algorithm to print each element. Procedure - **TRAVERSE(LA, N)**

1. Start
2. Set J=1
3. Repeat steps 4 and 5 while J < N
4. PRINT LA[J]
5. Set J = J +1
6. Stop

Sorting

Sorting operation refers to arranging the elements either in ascending or descending way.

Consider LA is a linear array with N elements. Below is the **Bubble Sort algorithm** to sort the elements in ascending order. Procedure - **SORT(LA, N)**

1. Start
2. Set I = 0
3. Set J = 0
4. Repeat steps 5,6,7 and 8 while I < N
5. J = I + 1
6. Repeat steps 7 and 8 while j <N
7. IF LA[I] is > LA[J] THEN
8. Set TEMP = LA[I]; LA[I] = LA[J]; LA[J] = TEMP;
9. Stop

Basic Operation cont…

Merging

Merging refers to combining two sorted arrays into one sorted array. It involves 2 steps –

- Sorting the arrays that are to be merged
- Adding the sorted elements of both arrays to a new array in sorted order

LA1 is a linear array with N elements, LA2 is a liner array with M elements and LA3 is a liner array with M+N elements. Write the algorithm to sort LA1 & LA2 and merge LA1 & LA2 into LA3 & sort LA3 and print each element of LA3

1. Start

/* Assignment1 */

2. Stop

Reversing

Reversing refers to reversing the elements in the array by swapping the elements. Swapping should be done only half times of the array size

Consider LA is a linear array with N elements. Write the algorithm to reverse the elements and print each element of LA

1. Start

/* Assignment 2 */

2. Stop

Assignments

Assignment 3

LA is a linear array with N elements. Write the algorithm to finds the largest number and counts the occurrence of the largest number

1. Start
/* Assignment 3 steps */
2. Stop

Assignment 4

LA is a linear array with N elements. Write the algorithm to copy the elements from LA to a new array LB

1. Start
/* Assignment 4 steps*/
2. Stop

Assignment 5

LA is a linear array with N elements. Write the algorithm to transpose the array

1. Start
/* Assignment 5 steps */
2. Stop

Assignment 6

LA is a linear **sorted** array with N elements. Write the algorithm to insert ITEM to the array

1. Start
/* Assignment 6 steps */
2. Stop

Polynomials

Polynomial is an expression constructed from one or more variables and constants, using only the operations of addition, subtraction, multiplication, and constant positive whole number exponents. A term is made up of coefficient and exponent.

Examples –

- Polynomial with single variable P(X) = $4X^3 + 6X^2 + 7X + 9$ (**4,6,7** are coefficient & **3,2** are exponent)
- Polynomial with 2 variables P(X, Y) = $X^3 - 2XY + 1$ (**2** is coefficient & **3** is exponent)

Definition–

- Polynomial with single variable P(X) = $\sum_{k=0}^{n} a_k x^k = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$

A polynomial thus may be **represented using arrays**. Array representation assumes that the exponents of the given expression are arranged from 0 to the highest value (**degree**), which is represented by the subscript of the array beginning with 0. The **coefficients** of the respective exponent are placed at an appropriate index in the array. Considering single variable polynomial expression, array representation is

arr	9	7	6	4	(coefficients)
	0	1	2	3	(exponents)

Polynomial Addition

Consider LA is a linear array with N elements and LB is a linear array with M elements. Below is the algorithm for polynomial addition

1. Start
2. Set j= maximum of M or N
3. Create a sum array LSum[] of size J
4. IF (N is Greater Than or Equal to M) Then

    Copy LA[] to LSum[]

   else

    Copy LB[] to LSum[]
5. IF (N is Greater Than M) then

    Traverse array LB[] and LSum[i] = LSum[i] + LB[i]

   else

    Traverse array LA[] and LSum[i] = LSum[i] + LA[i] while i < j
6. PRINT LSum
7. Stop

Polynomial Multiplication

Given two polynomials represented by two arrays, below is the illustration of the multiplication of the given two polynomials.

Example :

Input: A[] = {5, 0, 10, 6} and B[] = {1, 2, 4}

Output: prod[] = {5, 10, 30, 26, 52, 24}

The first input array represents "$5 + 0x^1 + 10x^2 + 6x^3$"

The second array represents "$1 + 2x^1 + 4x^2$"

And output is "$5 + 10x^1 + 30x^2 + 26x^3 + 52x^4 + 24x^5$"

A	5	0	10	6	Coefficents
	0	1	2	3	Exponents

B	1	2	4	0	Coefficents
	0	2	2	3	Exponents

AXB

Prod	5	10	30	26	52	24	Coefficents
	0	2	2	3	4	5	Exponents

Polynomial Multiplication cont…

Algorithm:

prod[0.. m+n-1] Multiply (A[0..m-1], B[0..n-1])

1. Start
2. Create a product array prod[] of size m+n-1.
3. Initialize all entries in prod[] as 0.
4. Travers array A[] and do following for every element A[i]

    Traverse array B[] and do following for every element B[j]

        prod[i+j] = prod[i+j] + A[i] * B[j]

5. return prod[]
6. Stop

Class Exercise

Given three polynomials represented by arrays, write an algorithm to multiply those

Matrix Addition

Suppose A and B are two matrix arrays of order m x n, and C is another matrix array to store the addition result. i, j are counters.

Step 1: Start

Step 2: Read: m and n

Step 3: Read: Take inputs for Matrix A[1:m, 1:n] and Matrix B[1:m, 1:n]

Step 4: Repeat for i := 1 to m by 1:

Repeat for j := 1 to n by 1:

C[i, j] := A[i, j] + B[i, j]

[End of inner for loop]

[End of outer for loop]

Step 5: Print: Matrix C

Step 6: Stop

Matrix Multiplication

Suppose A and B are two matrices and their order are respectively m x n and p x q. i, j and k are counters. And C to store result.

Step 1: Start.

Step 2: Read: m, n, p and q

Step 3: Read: Inputs for Matrices A[1:m, 1:n] and B[1:p, 1:q].

Step 4: If n ≠ p then:

        Print: Multiplication is not possible.

    Else:

        Repeat for i := 1 to m by 1:

            Repeat for j := 1 to q by 1:

                C[i, j] := 0 [Initializing]

                Repeat k := 1 to n by 1

                    C[i, j] := C[i, j] + A[i, k] x B[k, j]

                [End of for loop]

            [End of for loop]

        [End of for loop]

    [End of If structure]

Step 5: Print: C[1:m, 1:q]

Step 6: Stop

Sparse Matrix

A **sparse matrix** is a two-dimensional array in which most of the elements are zero or null. It is a wastage of memory and processing time if we store null values or 0 of the matrix in an array. For example, consider a matrix of size 100 X 100 containing only 10 non-zero elements. In this matrix, only 10 spaces are filled with non-zero values and remaining spaces of matrix are filled with 0. That means, totally we allocate 100 X 100 X 2 = 20000 bytes of space to store this integer matrix and to access these 10 non-zero elements we have to make scanning for 10000 times.

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \end{bmatrix}$$

To avoid such circumstances different techniques such as **linked list** or **triplet** are used

Linked List Representation will be covered later

Sparse Matrix Triplet Representation

In this representation, only non-zero values are considered along with their row and column index values. In this representation, the 0th row stores total rows, total columns and total non-zero values in the matrix. For example, consider a matrix of size 5 X 6 containing 6 number of non-zero values. This matrix can be represented as shown in the image...

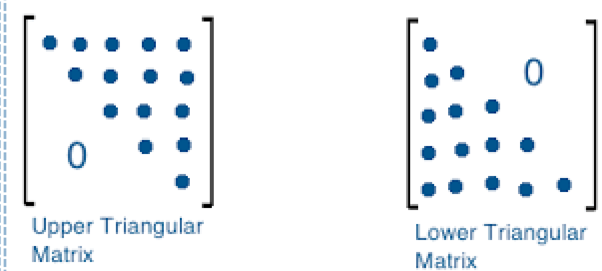
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

Rows	Columns	Values
5	6	6
0	4	9
1	1	8
2	0	4
2	2	2
3	5	5
4	2	2

In above example matrix, there are only 6 non-zero elements (those are 9, 8, 4, 2, 5 & 2) and matrix size is 5 X 6. We represent this matrix as shown in the above image. Here the first row in the right side table is filled with values 5, 6 & 6 which indicates that it is a sparse matrix with 5 rows, 6 columns & 6 non-zero values. Second row is filled with 0, 4, & 9 which indicates the value in the matrix at 0th row, 4th column is 9. In the same way the remaining non-zero values also follows the similar pattern.

Sparse Matrix cont...

N-square sparse matrices is called **triangular matrix**. A square matrix is called **lower triangular** if all the entries above the main diagonal are zero. Similarly, a square matrix is called **upper triangular** if all the entries below the main diagonal are zero.



Matrix where nonzero entries can only occur on the diagonal or on elements immediately above or below the diagonal is called a **tridiagonal matrix**

$$\begin{pmatrix} -4 & 1 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 1 & -4 \end{pmatrix}$$

Assignments

Assignment 7

Write an algorithm to add the originial sparse matrix with the transpose of the same matrix.

1. Start

/* Assignment 7 steps */

2. Stop

Assignment 8

Write an algorithm to multiply two sparse matrices.

1. Start

/* Assignment 8 steps*/

2. Stop

Assignment 9

9.1. Design an efficient data structure to store data for lower and upper triangular matrix.

9.2. Design an efficient data structure to store data for tri-diagonal matrix.

Assignment 10

Design an algorithm to convert a lower triangular matrix to upper triangular matrix.

1. Start

/* Assignment 10 steps */

2. Stop

Assignments

11. Write an algorithm that takes as input the size of the array and the elements in the array and a particular index and prints the element at that index.
12. Write down the algorithm to sort elements by their frequency.
13. Write down the algorithm to add two polynomials of single variable.
14. Write down the algorithm to multiply two polynomials of two variables.
15. A program P reads in 500 random integers in the range [0..100] presenting the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies?
16. Write down the algorithm to delete all the vowels in a character array.
17. Write down the algorithm to print all the elements below the minor diagonal in a 2-D array.
18. Write an algorithm to find a triplet that sum to a given value
19. Given an array arr, write an algorithm to find the maximum j – i such that arr[j] > arr[i]
20. Write an algorithm to replace every element in the array with the next greatest element present in the same array.

THANK YOU!

Home Work (HW)

1. Write an algorithm to find whether an array is subset of another array
2. Write an algorithm to remove repeated elements in a given array.
3. Write down the algorithm to find the k'th smallest and largest element in unsorted array
4. Write an algorithm to find the number of occurrence of k^{th} element in an integer array.
5. Write an algorithm to replace every array element by multiplication of previous and next
6. Consider the static array growing from both ends. Write the underflow and overflow condition for insertion and deletion from the perspective of both ends.
7. Given an array of integers, and a number 'sum'. Write an algorithm to find the number of pairs of integers in the array whose sum is equal to 'sum'.
   Examples: Input : arr[] = {1, 5, 7, -1}, sum = 6
   Output : 2 as Pairs with sum 6 are (1, 5) and (7, -1)

Home Work (HW)

8. Given an unsorted array, Write down the algorithm to find the minimum difference between any pair in given array.
   Input : {1, 5, 3, 19, 18, 25};
   Output : 1 as Minimum difference is between 18 and 19
9. Given an array of integers, Write down the algorithm to count number of sub-arrays (of size more than one) that are strictly increasing.
   Input: arr[] = {1, 2, 3, 4}
   Output: 6 as there are 6 sub-arrays {1, 2}, {1, 2, 3}, {1, 2, 3, 4}, {2, 3}, {2, 3, 4} and {3, 4}
10. Given an array of integers. All numbers occur twice except one number which occurs once. Write down the algorithm to find the number in O(n) time & constant extra space.
    Input: ar[] = {7, 3, 5, 4, 5, 3, 4};
    Output: 7
11. Given a 2D array of size m X n, containing either 1 or 0. As we traverse through, where ever we encounter 0, we need to convert the whole corresponding row and column to 0, where the original value may or may not be 0. Devise an algorithm to solve the problem minimizing the time and space complexity.

Supplementary Reading

- https://www.tutorialspoint.com/data_structures_algorithms/array_data_structure.htm
- http://www.geeksforgeeks.org/array-data-structure/
- https://www.hackerrank.com/domains/data-structures/arrays
- http://javarevisited.blogspot.in/2015/06/top-20-array-interview-questions-and-answers.html#axzz4myAupUvT
- https://www.youtube.com/playlist?list=PLqM7alHXFySEQDk2MDfbwEdjd2svVJH9p

FAQ

- **What is Array?**
  - Array is a collection of variables of same data type that share a common name.
  - Array is an ordered set which consist of fixed number of elements.
  - Array is an example of linier data structure.
  - In array memory is allocated sequentially so it is also known as sequential list.
- **1-D Array address calculation**

  Address of A[i] = BA + w * (i) where BA is the base address, w is the word length and i is the index
- **Row-Major order:** Row-Major Order is a method of representing multi dimension array in sequential memory. In this method elements of an array are arranged sequentially row by row. Thus elements of first row occupies first set of memory locations reserved for the array, elements of second row occupies the next set of memory and so on.

  Address of A [i][j] = BA + w * [n * (i – Lr) + (j – Lc)]

FAQ

- **Column-Major order:** Column-Major Order is a method of representing multi dimension array in sequential memory. In this method elements of an array are arranged sequentially column by column. Thus elements of first column occupies first set of memory locations reserved for the array, elements of second column occupies the next set of memory and so on.
  Address of A [i][j] = BA + w * [(i – Lr) + m * (j – Lc)]
- **Array ADT:** The array is a basic abstract data type that holds an ordered collection of items accessible by an integer index. These items can be anything from primitive types such as integers to more complex types like instances of struct. Since it's an ADT, it doesn't specify an implementation, but is almost always implemented by an static array or dynamic array.
- Time Complexity of 2-D array traversal = O(m*n)
- Time Complexity of 1-D array traversal = O(n)
- Time complexity of accessing an element of an 2-D array = O(m*n)
- Time complexity of accessing an element of an 1-D array = O(n)

FAQ

- **Ragged 2 D Arrays:**

```
char *font[] = {
(char []) {65,8,12,1,0,0x18, 0x3C, 0x66, 0xC3, 0xC3, 0xC3, 0xC3, 0xFF, 0xFF},
(char []) {39,2,3,4,9,0xC0,0xC0, 0xC0},
(char []) {46,2,2,4,0,0xC0,0xC0}};
```