Chapter 3: Array Processing

In this chapter you will learn all about:

- Defining and declaring arrays
- Array initialization and accessing array elements
- Time & Space Complexities – Quantifying the performance of your algorithm
- 1-D and 2-D array processing application examples

==

.

A. Arrays

Storage of multiple data items that have common data types/characteristics;

A.1 One-Dimensional Arrays

int x[100]; /* integer array capable of storing 100 integer values */

Visualize an array as a set of contiguous cells, with each cell taking one element. The first element is always referred to as, x[0] and the last element is x[99], in our above example;

Assume that an integer array **weight** of size 5 stores the values, 23, 79, 0, 12, 3. Then,

weight[0] stores 23, weight[1] stores 79, weight[2] stores 0, weight[3] stores 12 and weight[4] stores 3. Pay attention to the indices and how they are used.

So, as we learnt earlier, for an integer array, each cell has a capacity of 2 bytes to store an integer.

Different ways of Array Initialization:

int digits[10] = {1,2,2,3,4,4,5,6,9,0};

char colour[4] = {‘r’,’w’,’b’,’y’};

char mycolour[3] = “RED”; /* Error ? Something to learn! Check! */

char mycolour [] = “RED”;

So, digit[0]=1 and digit[9]=0 and colour[0]=’r’ and colour[3]=’y’

If uninitialized automatically the array values will be set to 0.

Tutorial 3.1:

- (a) What are the contents of this array digits? int digits[10] = {3,3,1};
- (b) Declare the following and measure the size of the strings using sizeof() and length of the string using strlen(). ***What do you observe in both the cases?***

  char mystring[] = “NUS”;

  strcpy(mystring, “Clementi”); /* Print and see */

  Point to observe: It is a **bad** practice to initialize a char array with a string literal.

Tutorial 3.2 **[Easy!]** *Compute an **average** of 100 floating point numbers and also compute the **deviation** of each number from the average using the formula:* $d_i = x_i - avg$, where x_i is the i-th element and avg is the average of all the floating point numbers. [**Discussion**]

A.2 Two-Dimensional Arrays

Same definition as one-dim arrays except that for each subscript we need to mention explicitly the size by a square bracket;

Example 3.1:

float my_table[25][25]; -> 625 elements starting from [0][0] to [24][24]

int age_height[10][10]; implies 100 elements starting from [0][0] to [9][9]

Other forms:

int values[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};

```
int values[3][4] = {
                        {1,2,3,4},
                        {5,6,7,8},
                        {9,10,11,12}
                        } ;
int values[3][4] = {
                        {1,2,3},
                        {4,5,6},
                        {7,8,9}
                        } ;
```

Elements [0][3], [1][3], [2][3] will have values 0 automatically assigned to them;

Q: *What if the number of elements is more than the size of the array defined?*

Example 3.2: Add all values of a 3 x 4 matrix by a constant integer value Y and multiply by int Q; *Part of the code is only shown;*

```
int i, j, Y, Q;
…
for(i = 0; i < 3;++i) {
        for(j = 0;j < 4;++j) { /* <- iterating over column entries for a given row */
        printf("Original value: %4d",values[i][j]);
        values[i][j] = (values[i][j] + Y)*Q;
        printf("Updated value: %4d\n",values[i][j]);
        } /* end of J loop */
} /* end of I loop */
```

==

NOTE:

Entries from user – You need two loops to fill in a two-dimensional matrix (m x n); Pseudo code given below:

```
For i = 1 to m
        For j = 1 to n
                printf("Input the value for A(%d,%d): ",i,j);
                scanf("%d",&A(i,j));
        endfor
        printf("\n"); /* for the next row entries */
endfor
```

A.3 Time & Space Complexities

Typical questions we would like to answer are as follows.

(a) How fast your program runs?

(b) How much data storage needed to hold your data for processing?

(c) Is it possible to accurately quantify the time taken by your program? If not, what is the best approximation/estimate?

(d) How to represent/quantify the estimated time?

Follow the rigorous discussions during the lecture for this part.

Note: *We will attempt to address the above, specifically the part (d), in all the examples in the next section.*

B. Application Examples using Arrays

Following will be **discussed** in the lecture sessions in detail. You can try implementing on your own following the lecture discussions.

1. Matrix Operations:

(a) Given a 2D matrix of size **N x N**, add all the elements in each row to generate a N x 1 vector; Discuss time & space complexities.

(b) Demonstrate adding two 2D Matrices of size N x N each. Discuss time & space complexities.

(c) Demonstrate two-dimensional matrix **multiplication** procedure (sequential algorithm). Discuss time & space complexities.

Define: Matrix **A (m x n)** and Matrix **B (n x m)**; Read the values of the matrices either from a file or from the user (when m and n are small);

Element C(i,j) is given by:

- C(i,j) = a(i,0) x b(0,j) + a(i,1) x b(1,j) + a(i,2) x b(2,j) + **. . .** + a(i,n) x b(n,j), i=0,1,…,m-1; j=0,1,…,n-1
- Above has to be repeated for every element – product matrix will have **m x m** elements;

Step-by-step pseudo code is given together with the order in which computations progress.

1. Fix a row say i;
2. for each element in that row i, compute C(i,j) for all j=1,…,n
3. repeat for every row i (repeat step 2);

Pseudo code:

```
For i = 0 to m-1
    For j = 0 to n-1
        C(i,j)=0;
        For k=0 to n-1
            C(i,j) = C(i,j) + A(i,k) x B(k,j) (**)
        Endfor
    Endfor
Endfor
```

Order of computation – C(0,0), C(0,1), …C(0,n); C(1,0), C(1,1),…C(1,n-1); . . . ; C(m-1,0), C(m-1,1),…,C(m-1,m-1);

Q: Let m = 100 and n = 100; If the computation time for (**) step takes 100 nsecs, what is the total time taken for computing the C matrix by the above logic?

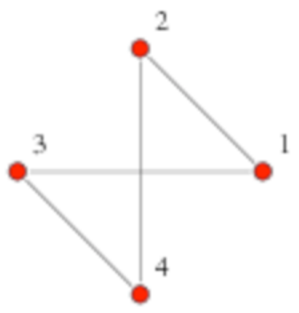
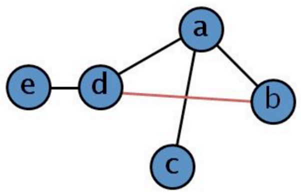
Question: What is the time complexity to determine A^3?

2. Computer network representation and analysis

A computer network can be represented as a graph. A compute node (PC/CPU) in your network is represented, as a node, or called as a vertex, in the graph. When two CPUs are connected by a link (communication links – wired or wireless) this link is represented as an edge in the graph. Thus, given a computer network (say, on your campus or even within your department) can be represented as a Graph ***G* = <*V*,*E*>**, where **V** is the number of vertices (representing computer nodes) and **E** is the number of edges (links between the nodes).

The graph G can be represented in the form of an ***adjacency matrix.*** Once an adjacency matrix is generated, we can work on this adjacency matrix to understand and compute several important metrics. The metrics to be computed are directly useful in several real-world applications.

Table 1. Examples of Adjacency Matrices

 $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$	 $\begin{array}{c\|ccccc} & a & b & c & d & e \\ \hline a & 0 & 1 & 1 & 1 & 0 \\ b & 1 & 0 & 0 & 1 & 0 \\ c & 1 & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 0 & 0 & 1 \\ e & 0 & 0 & 0 & 1 & 0 \end{array}$

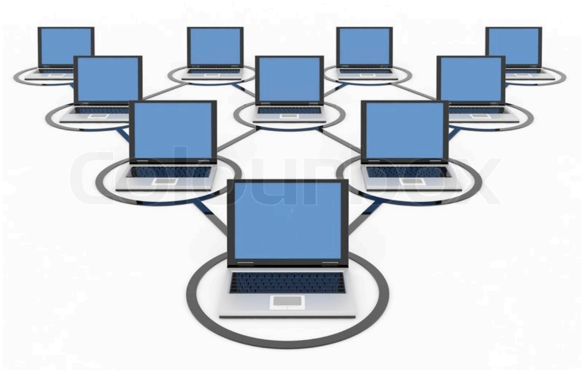
An adjacency matrix is a matrix that comprises entries 0 and 1. See examples of adjacency matrices shown in the above Table 1. The adjacency matrix, sometimes also called as the ***connection matrix****, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (v_i,v_j) according to whether v_i and v_j are adjacent or not.*

Pay attention to the diagonal entries. All must be zero.(Why?)

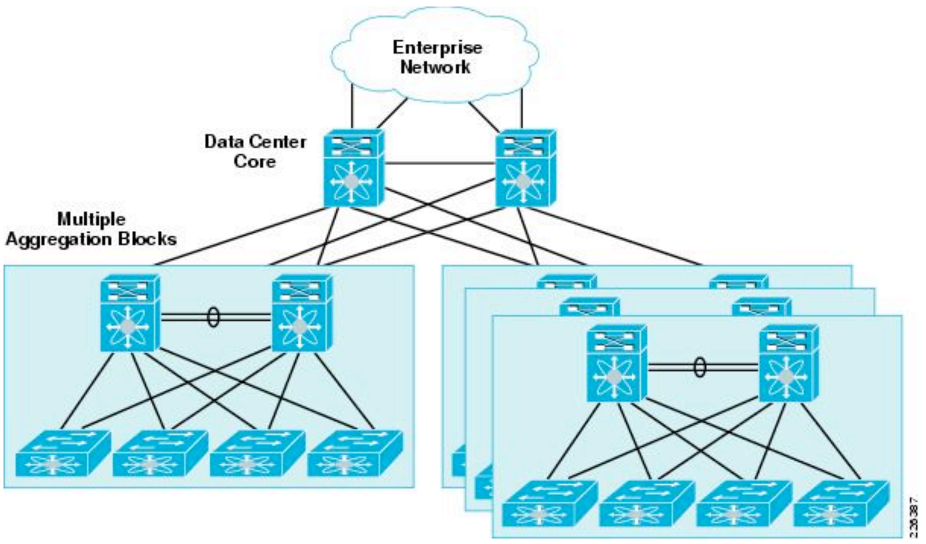
Q: What representation ***ensures*** *a* ***connected*** *graph?*

Tutorial 3.3: For the networks below, generate adjacency matrices.

(a) Network below is a typical small-size computer network, commonly found in our labs and small organizations.



(b) Network below is a typical enterprise network and considered highly scalable and a large-scale network. This is an example of hierarchical **Data Centre architectures**. On the lower right side consider only one plane.



On such graphs, we can compute several quantities such as, (i) path between any two nodes, (ii) ***diameter*** of a given network (*What is it?*), (iii) nodes prone to ***cyber-threats***(!!), and so on.

It may be noted that if we replace all the '1's in the adjacency matrix by a positive number this may represent any meaningful quantity. For example, a value 3 (called as "weight") on the link between nodes 2 and 5 may denote bandwidth available is 3 MB/s between nodes 3 and 5. See the tutorial problem below. Solution **hints** will be discussed and you need to implement this following problem as it involves only loops and decision-making statements.

Tutorial 3.4 [DIY]: Let **G**[5][5] = { 0,3,0,2,0, 3,0,0,2,2, 0,0,0,2,0, 2,2,2,0,0, 0,2,0,0,0}.

Determine the following.

(i) [Verify connectedness] Check if the graph is first a connected graph;

(ii) [Find connected pairs] Display all the **directly** connected nodes.

(iii) [Identifying Path] Identify a path between nodes 1 and 3;

(iv) [**Degree** of a node] This denotes the number of links connected to a node;

(v) [Distant Directly Connected Pairs] This is given by:

DDCP = **max**{ dist(i,j) | i,j in V are directly connected}

(vi) [Cyber-threat**] Find the node that is under high cyber-threat. To determine this, find the node that has **maximum** degree.

** - This is just one of the many ways! Usually on a tightly coupled node, administrators avoid storing sensitive and large amounts of data.

[***Challenge!***] In the above tutorial problem, determine the ***diameter*** of the network. The diameter is given by: **Dia** = **max**{ dist(i,j) | i,j in V}. ***Think!*** *What is the use of this quantity?*

[*BONUS PROBLEM* - DIY] (*If you have time, try on your own! Not for your exam, but if you want some challenge, implement and enjoy! If you are attempting, measure the running time too!*)

1. Simple Bubble Sort Algorithm:

Algorithm to sort, say N integer numbers. Define an array of integers to store N numbers; Use the following logic:

Compare Element i and Element i+1;

If element i > element i+1 then swap elements i and i+1 and repeat the procedure for element i; Bubble the element to the position where it fits; Repeat for all the elements of the array until all are sorted;

Step-by-step example

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort algorithm. In each step, elements written in **bold** are being compared.

First Pass:
(**5 1** 4 2 8) →(**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps them.
(1 **5 4** 2 8) →(1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) →(1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) →(1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.
Second Pass:
(**1 4** 2 5 8) →(**1 4** 2 5 8)
(1 **4 2** 5 8) →(1 **2 4** 5 8)
(1 2 **4 5** 8) →(1 2 **4 5** 8)
(1 2 4 **5 8**) →(1 2 4 **5 8**)
Now, the array is already sorted, but our algorithm does not know if it is completed. Algorithm needs one **whole** pass without **any** swap to know it is sorted.
Third Pass:
(**1 2** 4 5 8) →(**1 2** 4 5 8)
(1 **2 4** 5 8) →(1 **2 4** 5 8)

(1 2 **4** **5** 8) →(1 2 **4** **5** 8)
(1 2 4 **5** **8**) →(1 2 4 **5** **8**)

==

Pseudo Code is as follows:

```
do
  swapped := false
  for each i in 0 to length(A) - 1
    if A[i] > A[i+1] then
      swap( A[i], A[i+1] )
      swapped := true
    end if
  endfor
while swapped
```

==