Unit 1 Data Structures Basics

Structure:

1.1 Introduction

We will begin this unit with the discussion about the structuring of problem and how the task of formatting a solution to a problem is made simpler if the problem can be analyzed in terms of sub problems. We will also introduce the subject data structures along with a discussion of the different operations which are applied to this data structures. Last, you will learn about the algorithm complexity and time –space tradeoff that may occur in choosing the algorithm and data structure for a given problem.

Objectives:

After studying this unit, you should be able to:

- define the structure and problem solving technique for given any problem
- describe the different types of data structure
- state different operations occurring in data structure
- explain the algorithm complexity and time- space tradeoff

1.2 Structure and Problem Solving

Earlier the computer programmers have to implement large programs dealing with large volumes of data and information. Since the problems solved on digital computers have become progressively larger and complex, the computer programs to solve such problem and to provide solution have grown larger.

To simplify the task of writing the large programs, the concept of modularity was introduced. Modularity is dividing the larger problems to sub problems. So, a complex program is divided into number of sub modules and the solution obtained by solving the sub modules are combined to get the actual solution of that problem. The modularity of most systems can be represented by hierarchical structure as shown in figure 1.1.

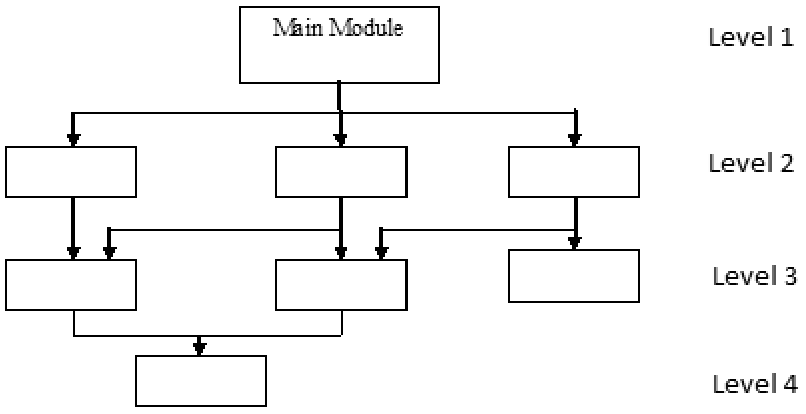

Figure 1.1: Hierarchical Structure

The hierarchical structure in figure 1.1 has a single main module at level 1 that gives a brief general description of the problem. The main module is sub- divided into number of sub modules at level 2 that gives more detailed description of the problem than the main module. The modules at level 2 are further sub divided as modules at level 3, and so on. The concept of hierarchical structuring of a problem in this fashion is a fundamental one in problem solving.

In organizing a solution to a problem which is to be solved with the aid of a computer, we are confronted with four interrelated sub problems.

The first sub problem is to understand the logical relationship between the data elements in the problem. For which first we have to understand the data itself. Data consist of a set of elementary items or atoms. An atom usually consists of single element such as integers, bits, character or set of such items. The possible ways in which the data items or atoms are logically related define different data structures.

The second sub problem is that, to decide on operations which must be performed on the data structures that are to be used. A number of operations can be performed on data structures like operation to create and destroy operation to insert and delete, so on. These operations have various functionally for different data structures.

The third problem is the representation of a data structure in the memory. The representation of data structures in memory is called a storage structure. We have many storage structures for a data structure such as an array and it is possible for two data structures to be represented by the same storage structure. The data structure can be stored both in main and the auxiliary memory of the computer. A storage structure representation in auxiliary memory is often called as file structure.

The fourth sub problem which is the selection of a programming language to be used in the solution of the problem. The programming language chosen for the implementation of an algorithm should posses the particular representation chosen for the data structures in the problem being solved.

The data structures, their storage structure and the operations on it are interrelated to problem solving using a computer.

1.3 Data Structures

As we have seen previously the data structure represents the logical relationship of the particular data sets. In this section you are going to learn about some of the data structures which we will discuss in detail in further units.

Data structures can be divided in to two types

- Linear data structure
- Non Linear data structure

Linear data structure

When the data is stored in the memory in linear or sequential form is called linear data structure. Examples of linear data structures include Array, Stack, Queue, Linked list.

Non Linear data structure

When the data is stored in the memory in dispersed or non sequential order is called non linear data structure. Example of non linear data structures includes trees, graphs, files etc.

Arrays

Linear array (one dimensional array) is the simplest type of data structures. Linear array means a list of a finite number n of similar data type referenced respectively by a set of n consecutive numbers 1, 2, 3… n. For example, if the name of the array is A then the elements of array A are denoted either by subscript notation, or by the parenthesis notation, or by the bracket notation.

$$a_1, a_2, a_3 \ldots a_n$$

or

A (1), A (2), A (3)… A (N)

or

A [1], A [2], A [3]… A [N]

Where the number N in A [N] is called a subscript and A [N] is called a subscripted variable.

Linked Lists

Linked list is a way to store data in memory. A linked list consists of a serious of nodes, which are not necessarily adjacent in memory. Each node contains the data field which has the element and the next field which has the link to the node containing its successor. The figure 1.2 shows a node with data and link fields.

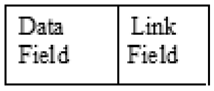

Figure 1.2: Node of a linked list

Generally in linked list the elements are connected by the link field which contains the address of the next node. The link field of last node is marked by null pointer which indicates the end of the list and a START variable which contains the address of the first node in the list. The figure 1.3 shows the linked list.

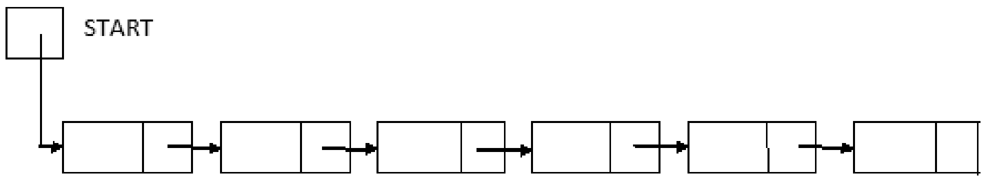

Figure 1.3: Linked list

Stack

It is a linear list in which insertions and deletions are restricted at one end, called the top. The following figure 1.4 shows a stack with 5 elements and the top pointer where both insertion and deletion are performed.

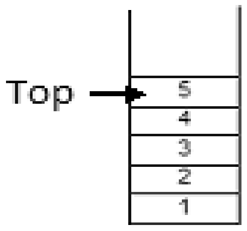

Figure 1.4: Stack

Stack contains two operation push and pop. Push is used to insert an element into stack and pop is used to remove an element from stack. The following figure 1.5 shows the two operations how it is performed in stack. It is also called as last- in first- out (LIFO), because the element pushed last need to be popped out first.

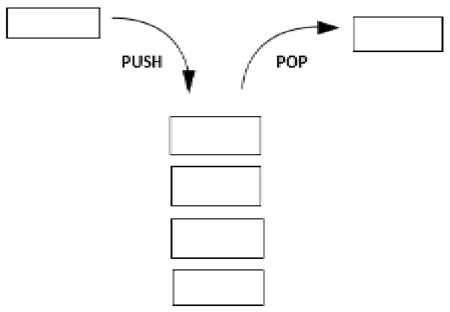

Figure 1.5: Push & Pop

Queue

It is a linear list in which insertion and deletion will take place in different end. As shown in figure 1.6 the end where we insert an element is called the “rear” end and deletion at “front” end. The element entered first need to be removed first, so it is also called as first- in first- out (FIFO).

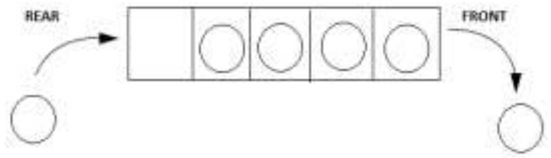

Figure 1.6: Queue

Trees

Tree is a nonlinear data structure which contains the hierarchical relationship between various elements is called a tree. One node is distinguished as a root, every other node is connected by a directed edge from exactly one other node, with the direction of parent -> children as referred in the figure 1.7.

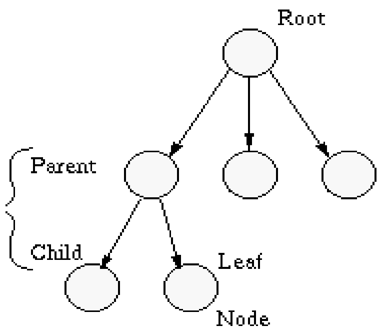

Figure 1.7: Tree Structure

Graph

Data sometimes contain a relationship between pairs of elements which is necessarily hierarchical in nature. The data structure referred in figure 1.8 reflects this type of relationship is called a graph.

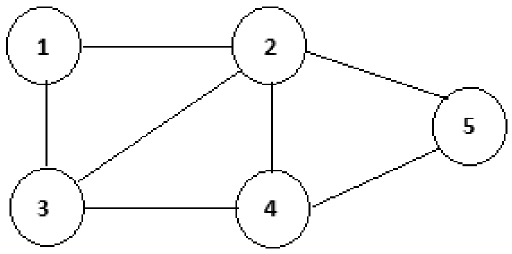

Figure 1.8: Graph Structure

Self Assessment Questions

1. _____________ is used in dividing the larger problems to sub problems.
2. The data structure which contains the hierarchical relationship between various elements is called a ____________.
3. Insertion and deletion of element in queue are done in ________ and ______ ends respectively.
4. _____ is called as a list of finite number of elements of similar data types.

1.4 Data Structure Operation

Operations are used to process the data appearing in the data structure. In fact, the data structure is chosen based on the frequency with which specific operations are performed. The following are the some of the operations used frequently.

- ***Traversing:*** Accessing each record exactly once so that certain items in the record may be processed.
- ***Searching:*** Finding the location of the record with the given key value.
- ***Inserting:*** Adding a new record.
- ***Deleting:*** Removing a record.
- ***Sorting:*** Arranging the records in some logical order.
- ***Merging:*** Combining the records in two different file into a single file.

1.5 Algorithms: Complexity and Time- Space Tradeoff

An algorithm is a step by step procedure for solving a particular problem. One major purpose of this text is to develop efficient algorithms for the processing of our data. The efficiency of the algorithm is depending on the time and space it uses. The complexity of an algorithm is the function which gives the running time and/ or space in terms of the input size.

Each of our algorithms will involve a particular data structure. The data structure which we choose will depend on many things, including the data and the frequency with which various data operation are applied. Sometimes the choice of data structure involves a time- space tradeoff by increasing the amount of space for storing the data, one may be able to reduce the time needed for processing the data, or vice versa. So, we may not always be able to use the most efficient algorithm. We illustrate the idea with two examples.

Searching Algorithms

Consider employee details file in which each record contains, among other data, the name and telephone number of its employee. Suppose we are given with the name of the employee and we want to find his or her telephone number. One way to do this is to linearly search through the file.

Linear search: It searches each record one at a time, until finding the given name and hence the corresponding telephone number. The time required to execute the algorithm is proportional to the number of comparison. So, the average number of comparison for a file with n records is equal to n/2 i.e. the complexity of linear search is C (n)= n/2. This algorithm is impossible if we are searching through a list consisting of thousands of names, as in telephone books. If we have the records as sorted file, then we can use an efficient algorithm called binary search.

Binary Search: Compare the given Name with the name in the middle of the list; this tells which half of the list contains Name. Then compare Name with the name in the middle of the correct half to determine which quarter of the list contains Name. Continue the process until finding Name in the list. The complexity of binary search is C (n) = $\log_2 n$. Although the binary search algorithm is a very efficient algorithm, it has some major drawbacks. Specially, the algorithm assumes that one has direct access to the middle name in the list or a sublist. This means that the list must be stored in some type of array. Unfortunately, inserting an element in an array requires elements to be moved down the list, and deleting an element from an array requires element to be moved up the list.

An Example of Time- Space Tradeoff

Suppose a file of records contains names, social security numbers and much additional information among its fields. Sorting the file alphabetically and rising a binary search is a very efficient way to find the record for a given name. On the other hand, suppose we are given only with the social security number of a person. Then we have to do a linear search for the record, which is extremely time- cost consuming for a very large number of records. One way to solve that problem is to have another file which is sorted numerically according to social security number. This however would double the space required for storing the data. Another way is to have the main file sorted numerically by social security number and to have a auxiliary array with only two columns, the first column containing an alphabetized list of the names and the second column containing pointers which give the locations of the corresponding records in the main file. This is one way of solving the problem that is used frequently, since additional space, containing only two columns, is minimal for the amount of extra information it provides.

Self Assessment Questions

5. ___________ is used in finding the location of the record with the given key value.
6. The ______ of the algorithm is depending on the time and space it uses.
7. ________ is used in combining the records in two different file into a single file.

1.6 Summary

In this unit we have discussed how modularity is used in simplifying the problem solving and we have also discussed the different type of data structures and its operations. Last, we have discussed on notation of algorithm and its complexity, and we have discussed on the time-space tradeoff occurs during the selection of the algorithm and data structures for a problem.

1.7 Terminal Questions

1. Explain in detail about modularity.
2. What is a data structure? Discuss briefly on types of data structures.
3. Write a short note of operations of data structures.
4. Discuss on algorithm complexity and time space tradeoff.

1.8 Answers

Self Assessment Questions

1. Modularity
2. Tree
3. Rear and Front
4. Array
5. Searching
6. Efficiency
7. Merging

Terminal Questions

1. Modularity is dividing the larger problem into sub problem. (Refer section 1.2 for detail)
2. Data structures is the logical relationship between the data items. (Refer section 1.3 for detail)
3. Operations are used to process the data appearing the data structure. In fact, the data structure is chosen based on the frequency with which specific operations are performed. (Refer section 1.4 for detail)
4. Efficiency of algorithm is calculated with its time and space complexity (Refer section 1.5 for detail)