Data Structure: Theoretical Approach

Durgesh Raghuvanshi
B-Tech Department of Computer Science,
IILM Academy of Higher Learning, Greater Noida, Uttar Pradesh, India

Abstract

Run with accordance with significance. The first if these this paper explains about the basic terminologies used in this paper in data structure. Better running times will be other constraints, such as memory use which will be paramount. The most appropriate data structures and algorithms rather than through hacking removing a few statements by some clever coding. Data structures serve as the basis for abstract data types (ADT). "The ADT defines the logical form of the data type. The data structure implements the physical form of the data type."Different types of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, relational databases commonly use B-tree indexes for data retrieval, while compiler implementations usually use hash tables to look up identifiers.

1. Introduction

Data structures serve as the basis for abstract data types (ADT). "The ADT defines the logical form of the data type. The data structure implements the physical form of the data type."Different types of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, relational databases commonly use B-tree indexes for data retrieval, while compiler implementations usually use hash tables to look up identifiers. Data structures provide a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services. Usually, efficient data structures are key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Data structures can be used to organize the storage and retrieval of information stored in both main memory and secondary memory. Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer—a bit string, representing a memory address, that can be itself stored in memory and manipulated by the program. Thus, the array and record data structures are based on computing the addresses of data items with arithmetic operations, while the linked data structures are based on storing addresses of data items within the structure itself. Many data structures use both principles, sometimes combined in non-trivial ways (as in XOR linking).[citation needed] The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analyzed separately from those operations. This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost).[citation needed]An array is a number of elements in a specific order, typically all of the same type (depending on the language, individual elements may either all be forced to be the same type, or may be of almost any type). Elements are accessed using an integer index to specify which element is required. Typical implementations allocate contiguous memory words for the elements of arrays (but this is not necessity). Arrays may be fixed-length or resizable. A linked list (also just called list) is a linear collection of data elements of any type, called nodes, where each node has itself a value, and points to the next node in the linked list. The principal advantage of a linked list over an array, is that values can always be efficiently inserted and removed without relocating the rest of the list. Certain other operations, such as random access to a certain element, are however slower on lists than on arrays. Most assembly languages and some low level languages.

2. Sequential search

computer to fetch and store data at any place in its memory, All printed material, including text, illustrations, and charts, must be kept within a print area of 6-1/2 inches (16.51 cm) wide by 8-7/8 inches (22.51 cm) high. Do not write or print anything outside the print area. All text must be in a two-column format. Columns are to be 3 inches (7.85 cm) wide, with a 5.1/16 inch (0.81 cm) space between them. Text must be fully justified.
This formatting guideline provides the margins, placement, and print areas. If you hold it and your printed page up to the light, you can easily check your margins to see if your print area fits within the space allowed.

3. Algorithm Complexity

Algorithm	Best case	Expected
Selection sort	O(N2)	O(N2)
Merge sort	O(NlogN)	O(NlogN)
Linear search	O(1)	O(N)
Binary search	O(1)	O(logN)

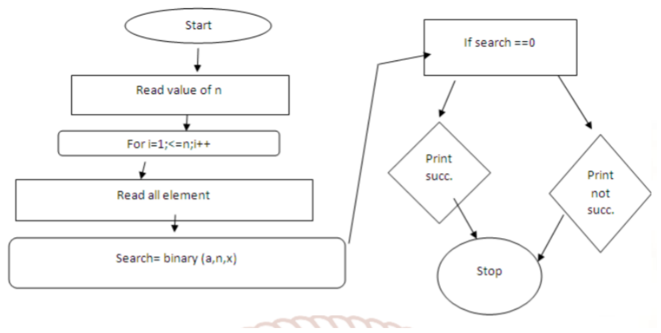
4. Depth of node

The depth of node is the length of the path from the root to the node. A rooted tree with only one node has a depth of zero.

5. Binary search

Binary search is a fast search algorithm with run-time complexity of (log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form. Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the sub array reduces to zero. B-trees are generalizations of binary search trees in that they can have a variable number of sub trees at each node. While child-nodes have a pre defined range, they will not necessarily be filled with data, meaning B-trees can potentially waste some space. The advantage is that B-trees do not need to be re-balanced as frequently as other self-balancing trees. Due to the variable range of their node length, B-trees are optimized for systems that read large blocks of data. They are also commonly used in databases. A ternary search tree is a type of tree that can have 3 nodes: a lo kid, an equal kid, and a hi kid. Each node stores a single character and the tree itself is ordered the same way a binary search tree is, with the exception of a possible third node. Searching a ternary search tree involves passing in a string to test whether any path contains it. The time complexity for searching a balanced ternary search tree is O(log n).

6. Graphics/Images



- Resolution: 600 dpi
- Color Images: Bicubic Downsampling at 300dpi
- Compression for Color Images: JPEG/Medium Quality
- Grayscale Images: Bicubic Downsampling at 300dpi
- Compression for Grayscale Images: JPEG/Medium Quality
- Monochrome Images: Bicubic Downsampling at 600dpi
- Compression for Monochrome Images: CCITT Group 4

The left subtree of a node contains only nodes with keys lesser than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. The left and right subtree each must also be a binary search tree. There must be no duplicate nodes. delete operation is discussed. When we delete a node, three possibilities arise. 1) Node to be deleted is leaf: Simply remove from the tree.3) Node to be deleted has two children: Find in order successor of the node. Copy contents of the in order successor to the node and delete the in order successor. Note that in order predecessor can also be used.2) Node to be deleted has only one child: Copy the child to the node and delete the child The important thing to note is, in order successor is needed only when right child is not empty. In this particular case, in order successor can be obtained by finding the minimum value in right child of the node. Time Complexity: The worst case time complexity of delete operation is O(h) where h is height of Binary Search Tree. In worst case, we may have to travel from root to the deepest leaf Now when we want to search for an item, we simply use the hash function to compute the slot name for the item and then check the hash table to see if it is present. This searching operation is $O(1)$, since a constant amount of time is required to compute the hash value and then index the hash table at that location. If everything is where it should be, we have found a constant time search algorithm.

7. Some common internal sorting algorithms include

Bubble Sort Insertion Sort Quick Sort Heap Sort Radix Sort Selection sort Consider a Bubblesort, where adjacent records are swapped in order to get them into the right order, so that records appear to “bubble” up and down through the dataspace. If this has to be done in chunks, then when we have sorted all the records in chunk 1, we move on to chunk 2, but we find that some of the records in chunk 1 need to “bubble through” chunk 2, and vice versa (i.e., there are records in chunk 2 that belong in chunk 1, and records in chunk 1 that belong in chunk 2 or later chunks). This will cause the chunks to be read and written back to disk many times as records cross over the boundaries between them, resulting in a considerable degradation of performance. If the data can all be held in memory as one large chunk, then this performance hit is avoided. On the other hand, some algorithms handle external sorting rather better. A Merge sort breaks the data up into chunks, sorts the chunks by some other algorithm (maybe bubblesort or Quick sort) and then recombines the chunks two by two so that each recombined chunk is in order. This approach minimises the number or reads and writes of data-chunks from disk, and is a popular external sort method. It is useful to understand how storage is managed in different programming languages and for different kinds of data. Three important cases are: static storage allocation stack-based storage allocation heap-based storage allocation Static Storage Allocation Static storage allocation is appropriate when the storage requirements are known at compile time. For a compiled, linked language, the compiler can include the specific memory address for the variable or constant in the code it generates.

8. Comparison between linear search and binary search

Basis for comparison	Linear search	Binary search
Time complexity	O(N)	O(log2N)
Best case time	First element 0(1)	Centre element 0(1)
Usefulness	Easy to use	Tricky algorithms
Lines of code	less	More

9. Some mathematical equation

$$E = mc^2 \tag{1}$$

$$g(x) = \frac{1}{x} \tag{2}$$

$$F(x) = \int_b^a \frac{1}{3}x^3 \tag{3}$$

10. Conclusion

This paper covered the basics of data structures. With this we have only scratched the surface. Although we have built a good foundation to move ahead. Data Structures is not just limited to Stack, Queues, and Linked Lists but is quite a vast area. There are many more data structures which include Maps, Hash Tables, Graphs, Trees, etc. Each data structure has its own advantages and disadvantages and must be used according to the needs of the application. A computer science student at least know the basic data structures along with the operations associated with them. Many high level and object oriented programming languages like C, Java, Python come built in with many of these data structures. Therefore, it is important to know how things work under the hood. Dynamic data structures require dynamic storage allocation and reclamation. This may be accomplished by the programmer or may be done implicitly by a high-level language. It is important to understand the fundamentals of storage management because these techniques have significant impact on the behavior of programs. The basic idea is to keep a pool of memory elements that may be used to store components of dynamic data structures when needed. Allocated storage may be returned to the pool when no longer needed. In this way, it may be used and reused. This contrasts sharply with static allocation, in which storage is dedicated for the use of static data structures.

11. References

1.Book of Data structures through C G. S Baluja.
2. Pieren Garry Department of computer science New York University.
3.Paul Xavier department of algorithms in c Amsterdam.
4.Surendrakumar Ahuja IItdelhi department of computer science delhi .
5. Nick jones department of data mining Australia.
6. Wikipedia sequential search.