ISSN: 2350-1022
International Journal of Recent Research in Mathematics Computer Science and Information Technology
Vol. 1, Issue 2, pp: (20-26), Month: October – December 2014, Available at: www.paperpublications.org

DATA STRUCTURE- THE BASIC STRUCTURE FOR PROGRAMMING

Shubhangi Johri, Siddhi Garg, Sonali Rawat

Abstract: **Every program whether in c, java or any other language consists of a set of commands which are based on the logic behind the program as well as the syntax of the language and does the task of either fetching or storing the data to the computer, now here comes the role of the word known as "data structure". In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently. Data structures provide a means to manage large amounts of data efficiently, such as large databases and internet indexing services. Usually, efficient data structures are a key in designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Storing and retrieving can be carried out on data stored in both main memory and in secondary memory. Now as different data structures are having their different usage and benefits, hence selection of the same is a task of importance. "Therefore the paper consists of the basic terms and information regarding data structures in detail later on will be followed by the practical usage of different data structures that will be helpful for the programmer for selection of a perfect data structure that would make the programme much more easy and flexible.**

Keywords: **Data structures, Arrays, Lists, Trees.**

I. INTRODUCTION

All the programs consists of some or other functions related to data for accomplishing the main aim of the Software. The duty of handling the data is of data structures, functions it has to perform are either Storage, fetching or transmitting a relation between the data present in the memory. In-fact advance knowledge about the relationship between data items allows designing of efficient algorithms for the manipulation of data therefore data Structures are playing a great role in many ways.

II. LITERATURE SURVEY

Data structures basically evolved for surveying the purpose of storing, fetching and arranging the data in a specific logical manner. The simplest one is arrays but as programming goes on getting specific for better performance of program we started using different data structures.

III. DATA STRUCTURES

A data structure is a specialized format for organizing and storing data. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.

Now here are some of the practical applications of data structures-

- Compiler Design,
- Operating System,

- Database Management System,
- Statistical analysis package,
- Numerical Analysis,
- Graphics,
- Artificial Intelligence
- Simulation

Till now we have made a clear description of the term data structures moving on further now its time to divide the paper into two levels that is the basic level which is the level of basic data structures and then further in we will go to the data structures which are being used specially for several purposes.

Starting with the very basic

3.1 Linear Data Structures

(a) Array

An array is a number of elements in a specific order. They are accessed using an integer to specify which element is required (although the elements may be of almost any type). Typical implementations allocate contiguous memory words for the elements of arrays (but this is not always a necessity). Arrays may be fixed-length or expandable. At the time when we are already aware of the memory needed at run time it is initially of fixed length.

For example, an array of 10 32-bit integer variables, with indices 0 through 9, may be stored as 10 words at memory addresses 2000, 2004, 2008, … 2036, so that the element with index i has the address 2000 + 4 × i.

The first digital computers used machine-language programming to set up and access array structures for data tables, vector and matrix computations, and for many other purposes. Von Neumann wrote the first array-sorting program (merge sort) in 1945, during the building of the first stored-program computer. 159 Array indexing was originally done by self-modifying code, and later using index registers and indirect addressing.

Applications

Arrays are used to implement mathematical vectors and matrices, as well as other kinds of rectangular tables. Many databases, small and large, consist of (or include) one-dimensional arrays whose elements are records. Arrays are used to implement other data structures, such as heaps, hash tables, deques, queues, stacks, strings, and VLists.
Arrays can also be used to determine partial or complete control flow in programs,

(b) Lists

In computer science, a linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data and a reference (in other words, a link) to the next node in the sequence; more complex variants add additional links. This structure allows for efficient insertion or removal of elements from any position in the sequence.

Advantages:

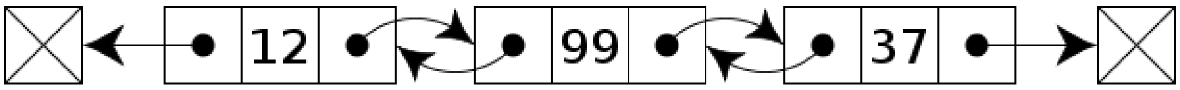
- Linked lists are a dynamic data structure, allocating the needed memory when the program is initiated.
- Insertion and deletion node operations are easily implemented in a linked list.

Disadvantages:

- They have a tendency to waste memory due to pointers requiring extra storage space.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.

(i) Doubly Linked Lists

In computer science, a **doubly-linked list** is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' **previous** and **next** links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.



A doubly-linked list whose nodes contain three fields: an integer value, the link to the next node, and the link to the previous node.

Advance Research

An asymmetric doubly-linked list is somewhere between the singly-linked list and the regular doubly-linked list. It shares some features with the singly linked list (single-direction traversal) and others from the doubly-linked list (ease of modification).It is a list where each node's previous link points not to the previous node, but to the link to itself. While this makes little difference between nodes (it just points to an offset within the previous node), it changes the head of the list: It allows the first node to modify the firstNode link easily.

(ii) Self Organizing List

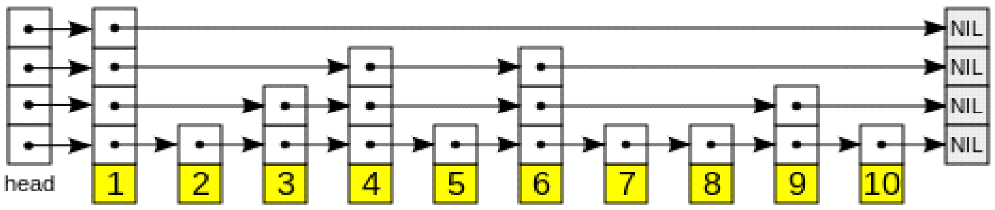
A self-organizing list is a list that reorders its elements based on some self-organizing heuristic to improve average access time. The aim of a self-organizing list is to improve efficiency of linear search by moving more frequently accessed items towards the head of the list. A self-organizing list achieves near constant time for element access in the best case. A self-organizing list uses a reorganizing algorithm to adapt to various query distributions at runtime.

Applications

Language translators like compilers and interpreters use self-organizing lists to maintain symbol tables during compilation or interpretation of program source code. Currently research is underway to incorporate the self-organizing list data structure in embedded systems to reduce bus transition activity which leads to power dissipation in those circuits. These lists are also used in artificial intelligence and neural networks as well as self-adjusting programs. The algorithms used in self-organizing lists are also used ascaching algorithms as in the case of LFU algorithm.

(iii) Skip List

In computer science, a **skip list** is a data structure that allows fast search within an ordered sequence of elements. Fast search is made possible by maintaining a linked hierarchy of subsequences, each skipping over fewer elements. Searching starts in the sparsest subsequence until two consecutive elements have been found, one smaller and one larger than the element searched for. Via the linked hierarchy these two elements link to elements of the next sparsest subsequence where searching is continued until finally we are searching in the full sequence. The elements that are skipped over may be chosen probabilistically.[2][3]



The elements used for a skip list can contain more than one pointer since they can participate in more than one list. Insertions and deletions are implemented much like the corresponding linked-list operations, except that "tall" elements must be inserted into or deleted from more than one linked list.

Applications

List of applications and frameworks that use skip lists:

- Cyrus IMAP server offers a "skiplist" backend DB implementation (source file)
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.
- QMap (up to Qt 4) template class of Qt that provides a dictionary.

(iv) Vlist

In computer science, the VList is a persistent data structure designed by Phil Bagwell in 2002 that combines the fast indexing of arrays with the easy extension of cons-based (or singly linked) linked lists.Like arrays, VLists have constant-time lookup on average and are highly compact, requiring only O(log n) storage for pointers, allowing them to take advantage of locality of reference. Like singly linked or cons-based lists, they are persistent, and elements can be added to or removed from the front in constant time. Length can also be found in O(log n) time.The primary operations of a VList are:
Locate the kth element (O(1) average, O(log n) worst-case)
Add an element to the front of the VList (O(1) average, with an occasional allocation)

(iv) Zipper

A zipper is a technique of representing an aggregate data structure so that it is convenient for writing programs that traverse the structure arbitrarily and update its contents, especially in purely functional programming languages. The zipper was described by Gérard Huet in 1997. It includes and generalizes the gap buffer technique sometimes used with arrays.

Applications

The zipper is often used where there is some concept of 'focus' or of moving around in some set of data, since its semantics reflect that of moving around but in a functional non-destructive manner.The zipper has been used in

- Xmonad, to manage focus and placement of windows
- Huet's papers cover a structural editor based on zippers and a theorem prover.

3.2 Binary Trees

In computer science, a binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. A recursive definition using just set theory notions is that a (non-empty) binary tree is a triple (L, S, R), where L and R are binary trees or the empty set and S is a singleton set.
A binary tree is a rooted tree that is also an ordered tree (aka plane tree) in which every node has at most two children. A rooted tree naturally imparts a notion of levels (distance from the root), thus for every node a notion of children may be defined as the nodes connected to it a level below. Ordering of these children (e.g. by drawing them on a plane) makes possible to distinguish left child from right child.But this still doesn't distinguish between a node with left but not a right child from a one with right but no left child.

(i) Red-Black Tree

A red–black tree is a data structure which is a type of self-balancing binary search tree.
Balance is preserved by painting each node of the tree with one of two colors (typically called 'red' and 'black') in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

Application

Red–black trees offer worst-case guarantees for insertion time, deletion time, and search time. Not only does this make them valuable in time-sensitive applications such as real-time applications, but it makes them valuable building blocks in other data structures which provide worst-case guarantees; for example, many data structures used in computational geometry can be based on red–black trees, and the Completely Fair Scheduler used in current Linux kernels uses red–black trees.

(ii) Splay Tree

A splay tree is a self-adjusting binary search tree with the additional property that recently accessed elements are quick to access again. It performs basic operations such as insertion, look-up and removal in O(log n) amortized time. For many sequences of non-random operations, splay trees perform better than other search trees, even when the specific pattern of the sequence is unknown. The splay tree was invented by Daniel Dominic Sleator and Robert Endre Tarjan in 1985.

Advantage

- Simple implementation—simpler than self-balancing binary search trees, such as red-black trees or AVL trees.[citation needed]
- Comparable performance—average-case performance is as efficient as other trees.[citation needed]
- Small memory footprint—splay trees do not need to store any bookkeeping data.

(iii) Threaded Binary Tree

"A binary tree is threaded by making all right child pointers that would normally be null point to the inorder successor of the node (if it exists) , and all left child pointers that would normally be null point to the inorder predecessor of the node.A threaded binary tree makes it possible to traverse the values in the binary tree via a linear traversal that is more rapid than a recursive in-order traversal. It is also possible to discover the parent of a node from a threaded binary tree, without explicit use of parent pointers or a stack, albeit slowly. This can be useful where stack space is limited, or where a stack of parent pointers is unavailable (for finding the parent pointer via DFS).

(v) Treap

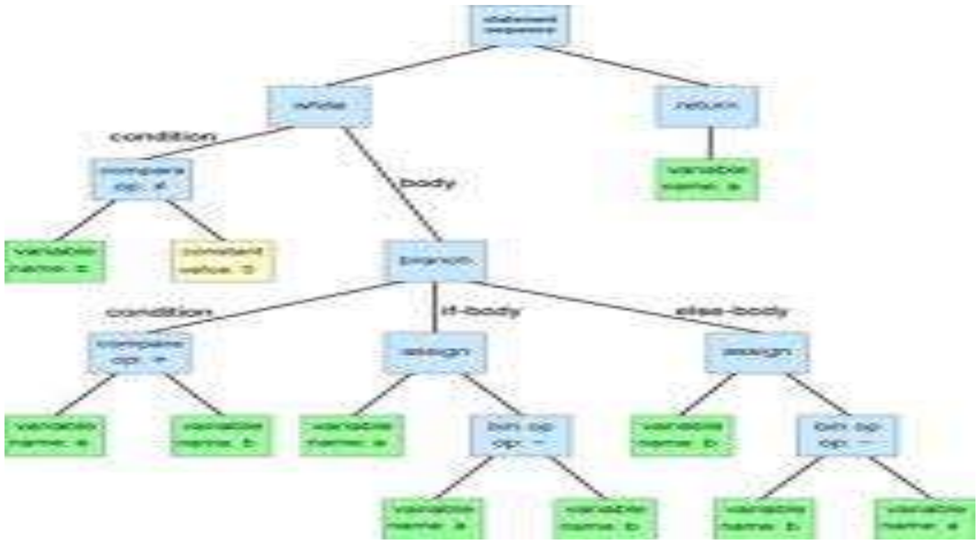
In computer science, the treap and the randomized binary search tree are two closely related forms of binary search tree data structures that maintain a dynamic set of ordered keys and allow binary searches among the keys. After any sequence of insertions and deletions of keys, the shape of the tree is a random variable with the same probability distribution as a random binary tree; in particular, with high probability its height is proportional to the logarithm of the number of keys, so that each search, insertion, or deletion operation takes logarithmic time to perform.

3.3 Application Binary Trees

(i) Abstract Syntax Tree

In computer science, an abstract syntax tree (AST), or just syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is "abstract" in not representing every detail appearing in the real syntax.

This distinguishes abstract syntax trees from concrete syntax trees, traditionally designated parse trees, which are often built by a parser during the source code translation and compiling process. Once built, additional information is added to the AST by means of subsequent processing, e.g., contextual analysis.



Applications

Abstract syntax trees are data structures widely used in compilers, due to their property of representing the structure of program code. An AST is usually the result of the syntax analysis phase of a compiler. It often serves as an intermediate representation of the program through several stages that the compiler requires, and has a strong impact on the final output of the compiler.

(ii) Parse tree

A concrete syntax tree or parse tree or parsing tree[1] or derivation tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. Parse trees are usually constructed according to one of two competing relations, either in terms of the constituency relation of constituency grammars (= phrase structure grammars) or in terms of the dependency relation of dependency grammars. Parse trees are distinct from abstract syntax trees (also known simply as syntax trees), in that their structure and elements more concretely reflect the syntax of the input language. Parse trees may be generated for sentences in natural languages (see natural language processing), as well as during processing of computer languages, such as programming languages.

(iii) Decision tree

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm.
Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal.

Advantages

- Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.
- Have value even with little hard data. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes.

(iv) Finger Tree

A finger tree is a purely functional data structure used in efficiently implementing other functional data structures. A finger tree gives amortized constant time access to the "fingers" (leaves) of the tree, where data is stored, and also stores in each internal node the result of applying some associative operation to its descendants. This "summary" data stored in the internal nodes can be used to provide the functionality of data structures other than trees

IV. CONCLUSION

The use of data structures is highly specific and it plays a great role in either making a program successful or for making it slow or to create an error in it. “The difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships”. To conclude data structures are playing a very great role in the programming aspect”. So we should go through a detail knowledge about the actual requirement of the type of data structure required for storing, sorting or arrangement of the data.

Glossary

- Data structures
- Linear data structures
- Array
- Lists
- Doubly linked lists
- Self-organizing list
- Zipper
- Skip list
- Vlist
- Red-Black tree
- Splay tree
- Threaded binary tree
- Treap
- Abstract syntax tree
- Parse tree

- Decision tree
- Finger tree

ACKNOWLEDGEMENT

I would like to give a great note of gratitude to my institution Dronacharya College of Engineering, for giving me such opportunity and support for writing a research paper. I would like to thank my parents also for their help and Ms.Pratibha Sharma my guide for giving me such great support and knowledge.

REFERENCES

[1] en.wikipedia.org/wiki/**Data_structure**

[2] careerride.com/**Data-structure**-defined.aspx

[3] en.wikipedia.org/wiki/List_of_**data_structures**

[4] www.indiabix.com › Technical Interview

[5] www.cplusplus.com/doc/tutorial/**structures**/

[6] interactivepython.org/runestone/static/pythonds/BasicDS/basic.html

[7] https://www.stat.auckland.ac.nz/~paul/ItDT/HTML/node64.html