ISSN 2278 - 3091
**International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)**, Vol. 5 , No.1 Pages : 06 - 08 (2016)
*Special Issue of ICCEIT 2016 - Held on February 16, 2016 in The Solitaire Hotel, Bangalore, India*
***http://warse.org/IJATCSE/static/pdf/Issue/icceit2016sp02.pdf***

# Comparative Study for Searching in Data Structures

**Kumudavalli M.V, Suraj A Jain, Nikith N Shetty**
Department of Computer Applications (BU), DSI, Bangalore, India
nikithnshetty@gmail.com, surajasjain@gmail.com

**Abstract***:* Generally there are a lot of difficulties in practical areas of Data Mining, Database Management System, Computer Science, Networks, and Artificial intelligence. Searching is a common fundamental operation used in solving searching problem in a dissimilar arrangements of these fields. In this paper we present the basic type of searching algorithms of data structures. An effort is made to cover some scientific features to these searching algorithms.

**Key Words:** ADT, Segment Tree, Intcache-Oblivious Model, Interval Tree.

## INTRODUCTION

In computer science, data structure is a particular way of organizing data in a computer so that it can be used efficiently. Data structures can implement one or more particular abstract data types (ADT), which are the means of specifying the contract of operations and their complexity. In comparison, a data structure is a concrete implementation of the contract provided by an ADT.

Different kinds of data structures are suitable for different kinds of applications, and some are highly specialized for specific tasks. For example, relational databases most 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 the key for designing efficient algorithms. Some formal designing methods and programming languages emphasize on data structures, rather than algorithms, as the key organizing factor in software design. Storing and retrieving operations can be carried out on data stored in both main memory and in secondary memory.

## LIST OF DATA STRUCTURES

The numerous types of data structures, generally built upon simpler primitive data types are:

- An array is a number of elements in an explicit order; characteristically all of them are of same type. Elements are accessed using an integer directory to specify which element is required (the elements can be of any data type). Typical implementations assign contiguous memory location for the elements of arrays (but this is not always necessary). Arrays may be fixed-length or resizable.
- An associative array (also called dictionary or map) is a more bendable disparity on an array, in which name-value pairs can be added and deleted freely. A hash table is a common implementation of an associative array.
- A record (also called tuple or struct) is an aggregate data structure. A record is a value that contains other values, typically in fixed number and sequence and typically indexed by names. The elements of records are usually called fields or members.
- A union is a data structure that specifies which number of permitted primitive types may be stored in its instances, e.g. float or long integer. Contrast with a record, which could be defined to contain a float and an integer; whereas in a union, there is only one value at a time. Enough space is allocated to contain the widest member data type.
- A tagged union (also called variant, variant record, discriminated union, or disjoint union) contains an additional field indicating its current type, for enhanced type safety.
- A set is an abstract data structure that can store specific values, in no particular order and with no duplicate values.
- A graph and a tree are linked abstract data structures composed of nodes. Each node contains a value and one or more pointers to other nodes arranged in a hierarchy. Graphs can be used to represent networks, while variants of trees can be used for sorting and searching, having their nodes arranged in some relative order based on their values.

A class is a data structure that mainly contains data fields like records, with various methods which operate on the contents of the record. In the perspective of object-oriented programming, records are known as plain old data structures to distinguish them from classes.

## OPERATIONS ON DATA STRUCTURE

1. Traversing
2. Searching
3. Inserting

ISSN 2278 - 3091
**International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)**, Vol. 5 , No.1 Pages : 06 - 08 (2016)
*Special Issue of ICCEIT 2016 - Held on February 16, 2016 in The Solitaire Hotel, Bangalore, India*
***http://warse.org/IJATCSE/static/pdf/Issue/icceit2016sp02.pdf***

4. Deleting
5. Sorting
6. Merging

1. Traversing- It is used to access each data item exactly once so that it can be processed.

2. Searching- It is used to find out the location of the data item if it exists in the given collection of data items.

3. Inserting- It is used to add a new data item in the given collection of data items.

4. Deleting- It is used to delete an existing data item from the given collection of data items.

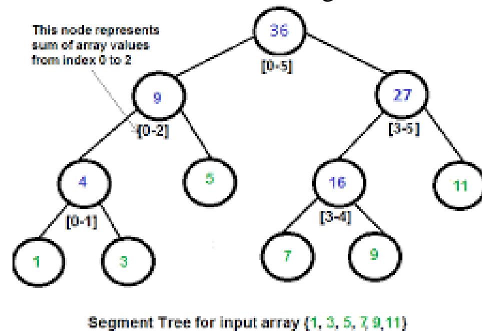
5. Sorting- It is used to arrange the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

6. Merging- It is used to combine the data items of two sorted files into single file in the sorted form.

**SEGMENT TREE**

In computer science, segment tree is a tree data structure for storing intervals, or segments. It allows querying on the stored segments which contain a given point. By principle, it is a static structure; that is, its structure cannot be customized once it is made. A similar data structure is the interval tree.

A segment tree for a set I of n intervals uses O(n log n) storage and can be built in O(n log n) time. Segment trees support searching for all the intervals that contain a query point in O(log n + k), where k is the number of retrieved intervals or segments.



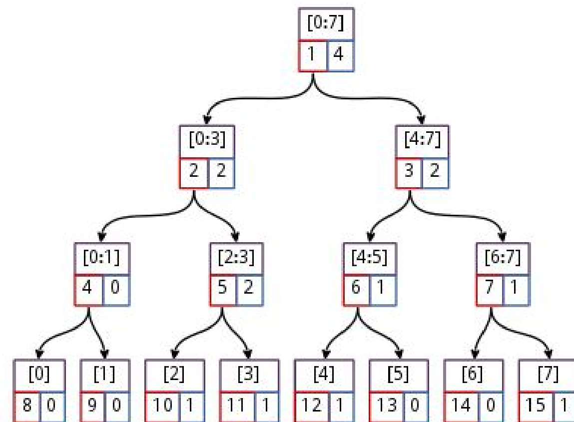

Segment Tree for input array {1, 3, 5, 7, 9, 11}

Segment trees are applied in the areas of computational geometry, and geographic information systems. They can be generalized to higher dimension spaces as well.

**INTERVAL TREE**

In computer science, an interval tree is a tree data structure which is used to hold intervals. Particularly, it allows one to efficiently find all intervals that surpass with any given interval or point. It is frequently used for windowing queries, for example, to find all roads on a computerized map inside a rectangular viewport, or to find all visible elements inside a three-dimensional scene. A similar data structure is the segment tree.

The insignificant solution is to visit each interval and check whether it intersects the given point or interval, which requires O(n) time, where n is the number of intervals in the set. Given that a query may return all intervals, for example if the query is a large interval intersecting all intervals in the collection, this is asymptotically optimal; On the other hand, we can do better by considering output-sensitive algorithms, where the runtime is expressed in terms of m, the number of intervals produced by the query. They have a query time of O(log n + m) and an original creation time of O(n log n), while limiting memory consumption to O(n). After creation, interval trees may be dynamic, allowing efficient insertion and deletion of an interval in O(log n). If the endpoints of intervals are within a smaller integer range (e.g., in the range [1,...,O(n)]), faster data structures exist with preprocessing time O(n) and query time O(1+m) for reporting m intervals containing a given query point.




**CACHE-OBLIVIOUS MODEL**

The logic behind the cache-oblivious model is to design external memory algorithms without the interruption of memory parameters. The cache-oblivious algorithm avoids the limitations imposed to external memory algorithms i.e., they are not architecture independent and it is very difficult to adapt them to multiple levels of memory. A cache-oblivious algorithm as described by Prokop is the one in which problem variable is independent of the memory parameters like cache size or block size; and is tuned to reduce the number of cache misses. Taking into account this definition the RAM model algorithms can also be measured as cache-oblivious. These algorithms are evaluated in an ideal-cache model.

The ideal-cache model consists of two-level of memory hierarchy i.e., a small cache and a very large main memory. The cache in ideal-cache model is of M words and cache line of size B. But this simple idea has surprisingly several powerful consequences.

ISSN 2278 - 3091
**International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)**, Vol. 5 , No.1 Pages : 06 - 08 (2016)
*Special Issue of ICCEIT 2016 - Held on February 16, 2016 in The Solitaire Hotel, Bangalore, India*
***http://warse.org/IJATCSE/static/pdf/Issue/icceit2016sp02.pdf***

One of the consequence is that, if a cache-oblivious algorithm performs well between two levels of the memory hierarchy (nominally called cache and disk), then it must automatically work well between any two adjacent levels of the memory hierarchy. An additional consequence is that, if the number of memory transfers is optimal up to a constant factor between any two adjacent memory levels, then any weighted combination of these counts (with weights corresponding to the relative speeds of the memory levels) is also within a steady factor of optimality.

**Lazy Evaluation using Buffers:** Another approach, for achieving sequential data processing is by using buffers. This technique is described by Arge, and used to develop the cache-aware buffer-tree data structure. The idea can, however, be reused in a cache-oblivious context, if the sizes of the buffers are not fixed to the sizes of the actual memory hierarchy, but instead grow, beginning with a small constant size. This way, the buffers will at some point fit into a cache level, and processing a buffer of this size will not incur more cache misses on that particular level.

The advantage of this technique is that the work is buffered in a lazy fashion. Only when the buffer is full, the content of a buffer is moved to the next larger level, and the elements stored in the buffer are all processed in one operation. This way, the main data processing occurs on the smallest level, and elements are moved up or down in the structure in a lazy or batched fashion.

## CACHE-OBLIVIOUS ALGORITHM

In computing, a cache-oblivious algorithm (or cache-transcendent algorithm) is an algorithm designed to take advantage of a CPU cache without having the size of the cache (or the length of the cache lines, etc.) as an explicit parameter. An optimal cache-oblivious algorithm is a cache-oblivious algorithm that uses the cache optimally (in an asymptotic sense, ignoring constant factors). Thus, a cache oblivious algorithm is designed to perform well, without modification, on multiple machines with different cache sizes, or for a memory hierarchy with different levels of cache having different sizes. Cache-oblivious algorithms are contrasted with explicit blocking, as in loop nest optimization, which explicitly breaks a problem into blocks that are optimally sized for a given cache.

Further machine-specific tuning may be required to obtain nearly optimal performance in an absolute sense. The goal of cache-oblivious algorithms is to reduce the amount of such tuning that is required.

Characteristically, a cache-oblivious algorithm works by a recursive divide and conquer algorithm, where the problem is divided into smaller and smaller sub problems. Eventually, one reaches a sub problem size that fits into cache, regardless of the cache size. For example, an optimal cache-oblivious matrix multiplication is obtained by recursively dividing each matrix into four sub-matrices to be multiplied, multiplying the sub matrices in a depth-first fashion. In tuning for a specific machine, one may use a hybrid algorithm which uses blocking tuned for the specific cache sizes at the bottom level, but otherwise uses the cache-oblivious algorithm.

## CONCLUSION

We are trying to cover some scientific features to these searching algorithms, in practical areas of Data Mining, Database Management System, Computer Science, Networks, and Artificial intelligence. Searching is a common fundamental operation used in solving searching problem in a dissimilar arrangements of these fields. In this paper we are presenting the basic types of searching algorithms in data structures and a conceptual discussion about the same.

## REFERENCES

[1] AGGARWAL, A., ALPERN, B., CHANDRA, A. K., AND SNIR, M. A model for hierarchical memory. In Proceedings of the 19th Annual ACM Symposium on Theory of Computing (May 1987), pp. 305–314.

[2] AGGARWAL, A., CHANDRA, A. K., AND SNIR, M. Hierarchical memory with block transfer. In 28th Annual Symposium on Foundations of Computer Science (Los Angeles, California, 12–14 Oct. 1987), IEEE, pp. 204–216.

[3] AGGARWAL, A., AND VITTER, J. S. The input/output complexity of sorting and related problems. Communications of the ACM 31, 9 (Sept. 1988), 1116–1127.

[4] AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. The Design and Analysis of Computer Algorithms. Addison-Wesley Publishing Company, 1974.

[5] ALPERN, B., CARTER, L., AND FEIG, E. Uniform memory hierarchies. In Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (Oct. 1990), pp. 600–608.

[6] SLEATOR, D. D., AND TARJAN, R. E. Amortized efficiency of list update and paging rules. Communications of the ACM 28, 2 (Feb. 1985), 202–208.

[7] STRASSEN, V. Gaussian elimination is not optimal. Numerische Mathematik 13 (1969), 354–356.

[8] SUPERCOMPUTING TECHNOLOGIES GROUP, MIT LABORATORY FOR COMPUTER SCIENCE. Cilk-5.2 (Beta 1) Reference Manual. Cambridge, MA, 1998. Available on the Internet from http://supertech.lcs.mit.edu/cilk.

[9] TOLEDO, S. Locality of reference in LU decomposition with partial pivoting. SIAM Journal on Matrix Analysis and Applications 18, 4 (Oct. 1997), 1065–1081.

[10] Dulal Chandra Samanta,Debabrata Samanta,"The Magic Square from Myth to Mystery",International Journal of Engineering Research and Development
ISSN: 2278-067X, Volume2, Issue 1(July 2012), PP.30-38.

[11] VITTER, J. S. External memory algorithms and data structures. In External Memory Algorithms and Visualization, J. Abello and J. S. Vitter, Eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society Press, Providence, RI, 1999.