iJETRM
International Journal of Engineering Technology Research & Management

DESIGN OF EFFICIENT DATA STRUCTURES FOR GEOMETRIC DESIGN

[*1]Bonu Sri Rama chandra Murthy,
*** [2] Dr. Balakrishna Ph.D. ***
[1] PG Student from Department of Civil Engineering, MVR Engineering College,
[2] Head of the department & Associate professor from Civil Engineering. MVR Engineering College, Vijayawada.

ABSTRACT
Objects in 2d geometry have varied shapes such as lines, polygons; circles etc. and are also of many dimensions. Most of the applications such as geographic information systems, computer graphics and robotics involve data such as cities, scenes, robots that are represented as geometric objects. Computational geometry offers efficient solutions to geometric problems that solve many real-world problems such as locating cities within a particular region of a map and collision detection in traffic control. This is possible by means of geometric data structures, algorithms and techniques from computational geometry. Range search is a standard search operation that is performed in database applications. Specific data structures for range search problems that exist require high preprocessing and storage time. In general, range search addresses different types of queries such as exact match, partial match and range query. For instance, a stabbing query on axis-parallel segments, finds all segments that contain a given point. A range query finds all segments that overlap a query segment. Therefore, range search problems depend on output size rather than the input size, and hence the algorithms are output sensitive. Thus, the size of output, namely objects reported are crucial to find the query time complexity. This thesis explores and addresses range search problems on points and lines, both in the static as well as in the kinetic context. Moreover, efficient data structures that are asymptotically fast in addressing stabbing query as well as range search for window query are proposed. This thesis presents four contributions namely, the BITS-tree, Extension of BITS-trees, BITS KD-tree AND KI-graph respectively for segments, rectangles, points as well as kinetic segments. The BITS-tree is proposed to perform stabbing query by visiting a fewer number of nodes as compared to the static and dynamic segment trees.

Keywords:
Geometric data structures Stabbing, DST, BITS KD-tree and KI-graph.

1. INTRODUCTION

Computational Geometry is a field of study that was introduced by Shamos (1978). It mainly deals with the design of data structures and algorithms for geometrical problems. The efficiency of algorithms designed for these geometric applications are measured based on storage, preprocessing and update time of the algorithms. Generally, objects in computational geometry are points, lines, polygons and other varied shapes. Moreover, computational geometry offers solutions to many real world applications found in Geographic Information System (GIS), computer graphics, robotics, multimedia databases, text indexing, string processing, network analysis and many others (deBerg et al.2008; Nekrich 2008: Preparata Shamos 1985).

OVERVIEW

Many geometric data structures with optimal or near optimal solutions are available for various problems in computational geometry Agar al et al. 2012; Arya et al, 1998; d'Amore Franciosa 1992). Most theoretical results are either based on certain assumptions or involve degenerate cases that may or may not be handled. However, there are some standard algorithms that are used to solve other complex, related problems. For instance, one of the simplest geometric problems is to find the convex hull of a given set of points. Delaunay triangulation and Voronoi diagrams (Arge 1995: Edelsbrunner 1987; Fortune 1992) are other efficient data structures used to handle point location problems and finding nearest neighbors of a given point. Yet another interesting problem is to find line segment intersections for which the well known plane-sweep technique is used to solve this problem in O(n log n + I log where n is the number of segments and I is the number of intersection points (Bentley Ottmann 1979; deBerg et al. 2008). it is important to note here that, the problem of

iJETRM
International Journal of Engineering Technology Research & Management

line segment intersection can be extended to higher dimensions such as rectangle intersections but the storage and running time tends to increase. Hence, there is a need for introducing new techniques, in order to solve the same problem for higher dimensions. Therefore, in this thesis, the primary focus is to design a novel data structure to store one-dimensional line segments and later to make it suitable for storing2 other objects such as rectangles as well.

STABBING QUERY

The problem of reporting all intervals in which a point lies is a kind of stabbing query (Hajos 1957). For a stabbing query, data tends to be the geometric objects and query is a point. For instance, consider a set of intervals (segments) a, b and c given in Figure 1.1. Here, all intervals that contain the point x = 11, reports intervals a and b. A database application for this problem is as follows: Given a set of company contracts wherein, each contract is for a specified number of years, a stabbing query retrieves all contracts available in a particular year.

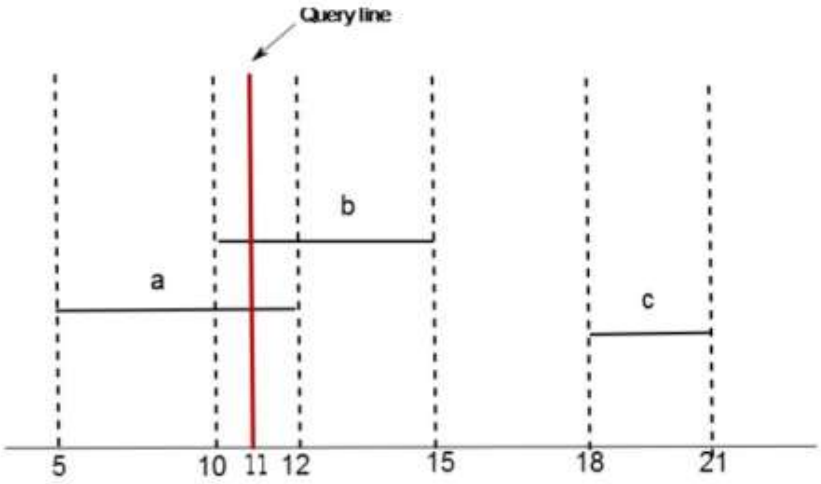

Figure 1.1: A stabbing query on given set of segments

From the above stabbing query on given set of segments, another interesting application of this type of query is found in Pattern matching (Oiling 1984). It is also used in windowing, wherein, given a geometric structure, one needs to display that part of the structure within the window. In such a case, one may want to find line segments going through the window. The very first solution to a stabbing query problem is an 0(n) time algorithm using a linear list, to find intervals which include the query point.

RANGE SEARCH

Range Search is one of the standard search operations which is an integral part of computational geometry, and some specific data structures for range search problems are also available in the literature gar al & Erickson 1996; Bentley & Friedman 1979). In general, range search addresses different types of queries such as exact match, partial match and range query. A typical example of an exact match query is to locate a particular city in a map with latitude and longitude denoted by a two-dimensional point say fix, y). However partial match query finds cities coinciding on latitude while longitudes differ, say (x,*). A range query on points finds all cities within a range of latitude and longitudes. Similarly, by considering a database query, where one has to find all employees who have 5-10 years of experience and whose age lies in the range 40-55 years, the abscissa represents years of experience and the ordinate represents their age. Points represent other related employee information. When this is translated into a geometric problem, the goal is to find all points that lie within a rectangular range to answer this query. Formally, it is termed as orthogonal range search that commonly makes use of a multidimensional binary search tree called KD-trees which performs d-dimensional range query in 0,011-k) time, where k is the number of points reported (Bentley 1979b).For this type of query, data tends to be a set of points and query to be an object, such as rectangle, cube etc. Range trees introduced by Bentley (1979a) are used to solve this problem faster than ICD-trees and takes 0(log +k) time, where lc is the number of points reported, in the case of one dimension. They can be extended to two dimensions using two such one-dimensional range trees and in general, it takes 0 (log d If + k) time for d-dimensional range search.

iJETRM
International Journal of Engineering Technology Research & Management

However, they require high preprocessing and storage time. Generally, a range query on two dimensions is as follows: Given a rectangle R, finding all rectangles that intersect R is referred as range query. Normally, this type of query is also called window query.

R-trees introduced by Guttman (1984) are data structures used to solve this problem. R-trees are particularly used in spatial data bases for storing rectangles and perform range search in two dimensions, Hilbert R-tree (Kamel Faloutsos 1994), a variant of R-trees uses Hilbert curves to order the rectangles efficiently. Since a rectangle can be visualized as line segments, one horizontally and the other vertically in an orthogonal case, the main goal is to design a data structure for segments that suits to extend to higher dimensions. Another important query operation is the nearest neighbor query. called NN-query (Lin et al. 1979; Roussopoulous et al.1995). This is yet another type of range query where R-trees are used. Recent work on NN-query shows that there is a performance degradation using R-trees, especially when the dimension increases, for d > 4.

QUERY IN HIGHER DIMENSIONS

As far as range searching problems are concerned, geometric data structures are normally designed to support 3d-dimensional queries also however, the flexibility of the data structure to Nupport direct extension to higher dimensions is cumbersome. For instance, interval trees (Edelsbrunner 1980:Edelsbrunner 1983a Edelsbrunner I 983b) and priority search trees (McCreight 1985) cannot be directly extended, although they can be combined with segment trees and generalized to d-dimensions.

OUTPUT SENSITIVE ALGORITHMS

An efficient solution to a range search problem such as those mentioned above, much depends on the time taken by the data structure to perform a query in such a may that storage size and construction time of the data structure is reasonably less (deBerg et al. 2008). Also, the time taken to insert and delete elements becomes important for the analysis. Another factor that is crucial to any type of query is the output size. For instance, in order to find all employees who have 5-10 years of experience and whose age lies in the range 40-55 years, the number of employees reported refers to output size and time taken to report them is also included in the overall query time for the search. In the case of a stabbing query, k segments intersect the vertical line, and here l c also refers to the output size. The work in this thesis mainly addresses such kind of problems, specifically those that are output sensitive and an effort has been made to reduce the search time by reporting the isometric objects in the various data-structures proposed, while maintaining the storage and update time reasonably less..

POINTER MACHINE MODEL

A pointer machine model was first introduced by Tarjan (1979).Normally, a pointer machine model of a data structure stores data in each node or a pointer to the actual data. Linked lists and binary search trees (BST) are examples of such data structures (Connell et al. 2001). Specifically, range search data structures make use of this model of computation (Agarwal & Erickson 1996). Some of the common geometric problems addressed by these data structures are orthogonal range reporting problems as well as rectangle stabbing queries. Segment trees and lid-trees belong to this category. Yet another model is the Random Access Machine (RAM) model otherwise known as Input/ Output WO) model. In this model, the memory system comprises of a main memory of limited size alow2 with some external memory devices. Algorithms designed for the I/0 model are hence called external menzoly algorithms.

Arge (1996) proposed segment trees for the RAM model. Also, rectangle-trees (R-trees) are good examples of data structures based on this model. Many standard R-tree variants are found in the literature (Manolopoulos et al. 2006), with different order in2 as well as grouping techniques to minimize the search time. But there are a few drawbacks in using R-trees as the datasets 2row in size. In the case of large datasets, the fanout is more and hence the query time also increases. Also, dynamic insertion and deletion of rectangles requires reconstruction of the tree.Nevertheless, a good algorithm should be fast when all the memory is internal memory. Since many range search problems deal with such internal memory structures based on pointer machine model, this study focuses on pointer model data structures for comparisons with the proposed data structure.

KINETIC DATA STRUCTURES

Many objects in the real world are moving continuously and it is important to store the data on moving objects with time. Such objects in motion find applications in various fields such as motion planning, computer graphics, animation etc. Also, many geometrical problems such as convex hull, Voronoi diagram, Delaunay triangulation, line segment intersections and those related to moving points are addressed using a framework

iJETRM
International Journal of Engineering Technology Research & Management

called Kinetic data structures (KDS). KDS first introduced by Basch et al. (1997a) primarily maintains a set of attributes of continuously moving data some of the common applications of ADS are in physical simulation of has molecules that involve position over time. Yet another well known application is a convex hull of moving points. When the points start moving, obviously some points on the hull are updated and may contribute in making the hull.

A simple solution is to take small time steps to recalculate the convex hull for each configuration of points that takes 0 (n log n) times at each time step. However, an alternate solution is to create a data structure that associates the convex hull directly with the vertices that form it I Next, one can use the trajectories of the nodes to determine when they enter or leave the convex hull list. This best explains a KDS. The important terminology used in a KDS is as follows: Configuration fiction (CF) is a collection of continuous or discrete attributes of moving data. For instance, the ordered list of points in a convex hull defines the CF (Figure 1.2(a)). Certificate is defined as a set of conditions that prove the configuration function. A certificate changes when the position of one of the objects of interest changes in the ordered list due to movement. CF invalidates whenever a certificate changes and the proof is updated.

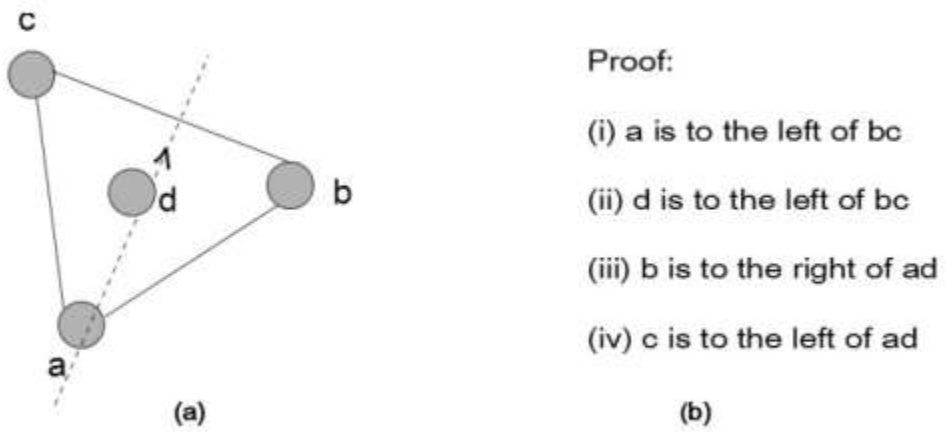

Figure 1.2: (a) Convex hull for a set of points (b) Proof containing four certificates (Bash 1999)

When points start moving, after a time instance, say t, some new data from the above information

(i) a is to the left of lac
(ii) d is to the left of be
(iii) b is to the right of ad
(iv) c is to the left of ad
(a) (b)

Figure 1.2: (a) Convex hull for a set of points (b) Proof containing four certificates (Bash 1999)points are associated with the hull. Proof: Certificates that are valid or hold good for the structure is a proof. A proof justifies that the description of attribute is correct. A sample proof for a convex hull is given in Figure 1.2(b). Event: An event may be either internal or external. External events result in a certificate failure and CF is changed as a whole. An event that does not affect the attribute is called internal events. Before describing the standard measures for a good KDS, it is important to understand the function of the event loop which forms the KDS framework. Whenever a certificate fails, the proof of correctness for that certificate will be altered and hence it is changed. As a result, a new proof is created. Since a proof consists of related attributes of interest for the certificates, attributes are also updated accordingly. This is a cyclic process and takes place every time a critical event happens and in turn a certificate fails. This is otherwise called the event loop shown by the visual representation in Figure 1.1 Basch et al. (1997a) describe the following criteria that are essential to measure the quality of a KDS. i) Responsiveness: The responsiveness of a kinetic data structure is the time taken in the worst case to update the structure after a certificate changes. A structure is said to be responsive, if the time needed to

iJETRM
International Journal of Engineering Technology Research & Management

process a certificate failure is small.

In general, a d-dimensional KD-tree for n points occupies 0(n) storage, 0(n log n) construction time (assuming d to be a constant) and time taken for range query is 0(71-11d +k), where k is the number of points reported by the query.

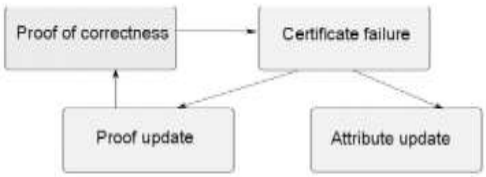

Figure 1.3: Event Loop for a *KDS* (Bash 1999)

From the above figure 1.3: Event Loop for a KDS (Bash 1999) Locality: The locality of a kinetic data structure is the maximum number of certificates in which an object is present. A structure is said to be local if an object has fewer certificates. Moreover, if a structure is local which also means it is compact.

ii) Compactness: The compactness of a KM' is the maximum number of certificates ever involved in a proof. A structure is said to be compact if the maximum number of certificates is linear.

iii) Efficiency: The efficiency of a KDS is determined by the number of events it processes in the worst case. A structure is said to be efficient if the ratio of total events compared to external events is small. Our aim is to design an efficient KDS for moving axis parallel segments, and the aforementioned terminologies and measures are explained in detail with respect to the segments in Chapter 6.

Applications

Computational geometry finds application in various fields. For instance, in GIS, the most common application is to store map data, where maps are first classified into layers, depending on their boundaries. From this map, one may need to find where the roads and rivers cross each other. These are called map overlay problems, and can be seen from a geometric perspective as the problem of finding line intersections of map boundaries. The hereditary segment trees (Chazelle 1988) are the earliest data structure used to solve this problem. Later plane sweep technique is used to solve such line segment intersection problems. Yet another application of GIS is to locate cities in a rectangular region called orthogonal range search.

MOTIVATION AND OBJECTIVES

Generally, geometrical problems are difficult to solve mainly due to the complexity involved in handling objects with different dimensions and shapes. Several algorithms are available in the literature (deBerg et al. 2008; Preparata & Shamos 1985) for storing and retrieving geometrical objects. Among these, segment trees are one of the earliest data structures in computational geometry, for staring one dimensional segments, where the extreme ends are fixed and belong to a closed interval [—Gc,+Go]. Layered segment tree (Vaishnavi & Wood 1982), distributed segment tree (Shen et al. 2007) are some of the segment tree variants available in the literature. Recently, unified segment trees (Wagner 2013) have been used to address two dimensional rangy search problem. Here again, many nodes have to be visited while addressing a stabbing query. Moreover, to extend them to d-dimensions, the only possibility is a multi-level segment tree which may require more nodes to be visited in the process, were each node is augmented with another segment tree. This makes the search also tedious. Standard segment trees such as Static Segment Tree (SST) and Dynamic Segment Tree (DST) are useful in storing intervals. But the limitations are two- fold: The first limitation is that, a static segment tree has afrced abscissae and can be built in O(n log 10 time and occupy 007 log n) space. Also it takes logarithmic query time to address stabbing query. The second limitation is that, during a binary tree traversal, since union of nodes provide answer to a stabbing query, many nodes have to be visited in the process. The dynamic counterpart of static segment trees (DST) practically require more storage and construction time. In DST, though it is possible to insert or delete a segment, it has a limitation of restricted ranges of intervals.

iJETRM
International Journal of Engineering Technology Research & Management

SIGNIFICANT CONTRIBUTIONS

The major contributions addressed in this thesis are listed below:

1) A novel data structure, namely BITS-tree, to dynamically store segments and to process stabbing and range queries efficiently. In the static version of segment trees (SST), it is required that the segments must be contained within a predefined range as it is constructed for a given set of segments. Thus, further insertion and deletion of segments is not possible, unless otherwise the tree is constructed again. However, the dynamic version of the segment tree (DST) allows dynamic insertion and deletion of segments, but must be within a fixed range. As a result, insertion of a segment beyond this range is impossible. Moreover, both the SST and DST are suitable for addressing stabbing queries, but it cannot address range queries, such as findin2 set of overlapping segments within a specific range.

To overcome these issues, a new data structure, called as BITS-tree is proposed in this thesis that exhibits the following benefits:

a) Overall range of segments is not limited to a predefined or a fixed interval.

b) Freedom to insert a new segment beyond the current left and right extremes of the tree.

c) Unlike in DST ., insertion or removal of a segment does not require visiting many nodes, as the number of nodes in BITS-tree is much less compared to DST.

d) Unlike in segment trees, answer to any stabbing query lies in only one node, thus practically reducing the search time.

e) Can perform range queries in 0(log rt +p) time, where p is the number of output intervals, which the SST and DST do not support within this complexity.

f) Number of nodes is much less, as compared to SST and DST , and thus reduces the storage space.

ii) A novel data structure, which is an extension of BITS-trees, that stores objects of higher dimensions to process range search for window queries. In the existing literature on range search for d-dimensions, the best known data structure is the d-dimensional range tree with a query time of 0(logd ii + k) time where k is the number of d-dimensional objects reported. In this thesis, we extend BITS-tree to store higher dimension objects, and answer a range search on a d-dimensional space with an improved time complexity of 0(d(log it + k)), as compared to the known 0(logd 11 +k) query time on range trees, where k is the number of objects reported.

iii) A novel data structure, BITS kd-tree, to store point in d-dimensional space, and that supports answering range query in constant time. To address range search on points of d-dimensional space, an efficient method is proposed in this thesis, by incorporating a new data structure, called BITS kd-tree. Moreover, this structure answers range queries Nvith an optimal time of 0(ic) time, where k is the number of points reported, and to the best of our knowledge, the best known complexity for this problem is 000241-1 +10, where d is the dimension and k the number of points reported (Willard 1978).

iv) A graph-based data structure, namely KI-Graph, which efficiently answers range queries on continuously moving segments in one-dimension. In any KDS, the quality of the structure is based on criteria such as local, responsive, compact and efficient. Thus to address range queries on moving segments, a novel graph-based data structure, called KI-Graph is proposed and the same satisfies all the four criteria described above. To the best of our knowledge, various KD► existing in the literature for moving segments do not possess all these four criteria.

RELATED WORK METHODOLOGY

This chapter deals with the study of existing data structures, which are related to stabbing query problem, rectangle intersection problems, and orthogonal range search on points as well as kinetic range search problems. Also, theoretical results with respect to storage, preprocessing, update and query time for best known data structures involved in each of the aforementioned problems are discussed

STABBING QUERY

Usually, the most common query operations performed on segment trees are stabbing queries, where one can find all the intervals that contain a given point, called the query point. In this section, prior work on segment treesare discussed. A static segment tree is one where it is not possible to insert or delete the abscissa. It requires 001 log space, and stabbing query processing time is O(k +log II), where n is the total number of segments and k is the number of segments reported. Though it looks reasonable in space and query processing, it does not

iJETRM
International Journal of Engineering Technology Research & Management

support dynamic insertion and deletion of segments. A segment tree based on loci approach definitely addresses stabbing query efficiently. Here again, there is a restriction on the overall range. However, in the case of dynamic segment trees, one can insert or delete a segment in the existing segment tree, but storage space requirements are likely to get increased as the range of the tree is not based on the segment end points. The range of the tree is predefined, and one can insert or delete the segments whose interval is within this range. In such cases where the range of the segment falls beyond the range of the segment tree, reconstruction of the tree is the only possibility, which is not advisable. Also, this may practically increase the increases. A segment tree based on locus approach makes the stabbing query much easier. This is because intervals on the real line are partitioned such that the range of a segment as well as each endpoint is also considered as an elementary interval. Hence, a query point will definitely lie within any one of the partitioned ranges only.

Nevertheless, segment ranges are to be known in advance. Variants of segment trees offer solutions to a wide variety of problems, right from the earliest rectilinear line segment intersection problem, contour problems for union of rectangles, to contour of restricted polygons to the recent packet classification problem (Feldman 8z Muthukrishnan 2000). Segment trees have been used to solve the problem of determining the contour of the union of a set of axis parallel rectangles (van Leeuwen & Wood 1981) as well as line segment intersections (Gifting 1984) in areas such as pattern recognition and image processing too. As regards axis-parallel line segments, binary space partition trees and its variants are available in the literature (d'Amore Franciosa 1992; Dumitrescu et al. 2004). A detailed survey on binary space partition techniques for storing axis-aligned segments as well as segments and rectangles in higher dimensions are found in (Toth 2005). For our discussion, we consider the relevant literature on stabbing query using pointer-based structures only, although quite a lot of I/0 efficient algorithms for external memory segment trees are available in the literature, most of which are based on B-Trees (Ble'loch 2008; Degermark et al. 1997).

In networks, they offer solutions to the IF look up problem in routers. Especially, in packet classification, a dynamic segment tree that stores a set of ranges is proposed in the query time complexity for traversing a DST is 0(log n), where n is the number of arbitrary ranges for a given IP address. Thiswas later improved by combining the features of a DST as well as a Grid of Tries (GoT) (Chang et al. 2010). The results show an improvement in terms of fewer nodes traversed during the search while maintaining a height of tree as 00log n).

Switch pointers make the traversal much faster and the memory requirement is also less when compared to the Go T Given a set of horizontal segments, finding line intersections with a vertical query segment has been addressed by a layered segment tree (Vaishnavi & Wood 1982) with a query time of 011og n. + k), were k is the number of intersections. A refined version of layered segment tree is given in (Racherial et al. 2002). The solution adheres to both real-time and online-applications in pattern matching as well as image processing. This segment tree

improves the amortized time to insert an arbitrary horizontal segment to 0(log ii) time, by reducing the sorting time using a binary search tree. Hence, the space requirement remains to be 0(n log ti). Also, the query time complexity remains to be the same of 011og n k) time, where k segments are reported by the query. Another well known solution to the union-find problem is a dynamic version of segment trees (Sosna 1997;van Kreveld Overrnars 1993). Union find problem is one where, given a collection of sets, one has to find the union as well as a copy of the sets and finally find those sets containing a given element. The data structure proposed therein

requires 00 log n) space and stabbing queries can be performed in 0(log n k) time. The main advantage of this structure is that it is independent of the site of input. However, the storage and preprocessing time is 00 log n).As mentioned earlier, in the one dimensional packet classification problem. each interval corresponds to a prefix of an IP address and query is an IP address. This is geometrically interpreted as a segment overlap problem for which Fat Inverted Segment (FIS)-trees (Feldman & Muthulcrishnan 2000) are used. The space requirements of FIS-trees are 0011+1) and they time taken for insertion and deletion is n), where is the height of the tree. Here, the tree is compressed (made 'fat') by increasing the degree to more than two in orderto decrease the height, and a in-way search tree is thus formed. However, this has an upper bound on the total number of insertions and deletions allowed. The storage was later reduced to 007") while maintaining the same update time (Thorup 2003). TWO other multiday segment trees used for IP look up are proposed in (Haibin Sahni 2005).Also, Agarwal et al. (2012) proposed a linear-size data structure for the stabbing semi-group queries by combining the features of interval and segment tree, with the assumption that the end points of all intervals belong to a fixed set of points. Interval skip list (IS-list) (Hanson & Johnson 1991) is yet another efficient data structure to store dynamic segments and addresses stabbing query in logarithmic time. However, there are some limitations as it works well only for limited overlaps among segments. Moreover, the update time is Wog' n). Other than the aforementioned data structures, some internal memory algorithms are also found in

iJETRM
International Journal of Engineering Technology Research & Management

the literature that address ray stabbing query for a given set of axis parallel segments, where one can find all segments intersecting a vertical ray. One such dynamic data structure is proposed in (ChengJanardan 1990) that requires O(n) size andthat solves stabbing query in 0(log2 n + lc) time, where lc is the number of segments reported. Here, the time to insert and delete a segment is 0(log n). Mehlhorn Naher (1990) proposed a pointer model structure of size 0(ii log with query time of 0(log n loglog n +k) and a 0(log 11 loglog n +k) amortizedtime to update.

The importance of segment trees in database applications has been already mentioned in the previous chapter. Therein, time period of a contract may be represented as a segment for each company. Therefore, when a new company enters the contract, from a geometric view it is required to dynamically add a new segment. Likewise, when a contract expires, that particular segment has to be removed. Such dynamic updates are tedious in the traditional segment trees as they require the overall range to be known in advance.

Hence, the only possibility is to rebuild the tree as a whole for ranges beyond the given range. Table 2.1 lists the related segment trees in the literature and their storage, construction, update and query time complexities. From the above table, one can observe that when the stabbing query time is improved to logarithmic query time, the update time increases. It is from the Table 2.1: Theoretical comparison of various segment tree

Description	Storage	Update	Stabbing Query
Segment Trees (Bentley 197'7)	Ow*	0(logn;	0(logn-i-k)*
Fat In Segment Trees (HS-trees) (Feldman 8z. Muthukrishnan 2000)	Or!! i	] Q(, log a) I	000gloB n +I)
Dynamic Fat Inverted Segment Tree.; (Thorup 2003)	3 101(n) 1	0(n] log n)	OW
Unified segment trees (Wagner 2013)	On 1og4 it)	Woe n)	0(logi II + k)

Number of segments, 1-height of rue ire-e, d-dimension. k- number of segments reported, *denotes for two dimensions. important to note here that while permitting dynamic insertion and deletion of segments, it is quite important to manage the update time as well. As far as degenerate cases are concerned, normally specific assumptions on the input are made. In certain cases, they are efficiently handled by the algorithms or may be omitted in certain cases. In general, for segment trees one may come across the following special cases: 0) All the segments pr axis-parallel and have equal length. (ii) One large segment spans across the rest of the segments. (iii)All the given segments overlap each other.

RECTANGLE INTERSECTION PROBLEM

Some of the applications where geometric intersections happen include map overlay in GIS to find where the bridges cross a river. In multimedia databases (Subrahmanian 1998), images and other objects of varied shapes are generally bounded by rectangular boxes, called bounding boxes. Two objects are said to intersect only if their corresponding bounding boxes intersect. R-trees (Guttman 1984) are known to be particularly useful in storing a set of rectangles. Therein, every leaf & Wood 1980; Leutenegger et al. 1997; Manolopoulos et al. 2006; Selliset al. 1993) and many R-tree variants such as Hilbert R-tree in two and higher dimensions perform window query (Beckmann et al. 1990; Garc et al. 1998:, Haverkort Walderween 1983; Kanael Faloutsos 1994). However. in such bulk loading algorithms, it is difficult to prove a good worst case running time complexity (Kao 2007). Some of the well known techniques for rectangle intersection problem are found in (d'Amore Franciosa 1992; Dumitreseu et al. 2004; Samet. 1990). Most of the common geometric data structures in the pointer machine model for range queries are the layered segment trees (Vaishnavi & Wood 1982), interval trees (Edelsbrunner 1983), priority search tree (McCreight 1985) and layered range tree. But they augment a secondary tree structureto represent d dimensional data (d > 2). As a result, the search time is 0(log-d n + k), in a d-dimensional space, where k rectangles are reported. Thus, Chazelle (1988) showed that a compressed range tree can be used to answer a &dimensional rectangle-intersection query in time 0(log(d-i) n k) using 0(17 log(d-1) n/ log log ti) space, where k is the number of rectangles reported. But this data structure is quite complex for practical purposes in two- dimensional space.

It is a well known fact that rectangle trees (R-trees) are the best known structures that support efficient storage of rectangle objects. They are similar to B- trees and the regions are stored as minimum bounding rectangles at

iJETRM
International Journal of Engineering Technology Research & Management

each level. The data objects appear at the leaf nodes. They are grouped based on block size and the tree is normally constructed bottom up. Among R-tree variants, Box-tree (Agarwal et al, 2002) answers a window query in 0((,)//B)1-11d + TlogBN) Ws. However, they depend on nature of input. Priority R-tree (Arge et al. 2008) is the only best known R-tree variant that is asymptotically optimal for window query and answers a window query in 0((N/B)1-lid+TIB) 110s, where N is the number of el-dimensional objects, is the disk block size, and T is the output size. Although they outperform other R- tree variants such as Hilbert R-tree (Darnel Faloutsos 1994) practically I as far as updates are concerned, it is not proven to be theoretically optimalln general, R-trees are efficient, but the query performance depends on how the data rectangles are ordered in the tree. Moreover, when the fan-out is large, which means that the dataset is huge, as well as for dimensions (d > 4), R-tree performance deteriorates (Manolopoulos et al. 2006). This is one strongmotivation of this thesis to revisit the rectangle intersection problem from a new perspective. Although a large variety of R-trees have been discussed. it is known that they belong to a class of RAM model. Note that this thesis is focused on data structures that belong to the pointer machine model. and hence our comparisons directly relate to only those data structures that fall under pointer machine model of computation and not with respect to external memory algorithms,

ORTHOGONAL RANGE SEARCH

Given n points in Rd, where d is the dimension. finding all points that lie within a particular range, is formally defined as orthogonal range search, also called as rectangular range query. These problems belong to a class of range reporting problems, widely used to locate cities within a latitude and longitudinal range in GIS. kd-trees are commonly used to solve this problem. As the name suggests, k-d trees are k-dimensional trees introduced by Bentley (1 980) and are the first multidimensional BST that address many types of queries such as exact match, partial match, range and nearest neighbour queries etc. They are mainly used to store point data. BITS kd-tree, the new data structure proposed in this paper performs orthogonal range search faster than the existing counterparts. Given n points, the time taken to find the median is O(n). This results in two sets of na points. Therefore the time taken to construct the 2d-tree is O(n log n). It is possible to extend a 2d-tree to a 3d-tree in a similar manner as mentioned above. In the case of 3d-trees, it is required to alternate between three axis x, y, z at levels 0, 1, 2 respectively, followed by 3, 4, 5 and so on. In thiscase, it is not possible to distineuish levels as odd or even, and hence one has totake mod k, where I is the level value and k is the number of dimensions, in this case k=3. In general, it is called the kd-tree where k stands for the dimension.

It explores the limitations of kd-trees in four main aspects listed below:
i) Data points are usually known in advance.
ii) When there are collinear points, deletion of a point requires swapping of sub trees.
iii) Depth of the tree is 0(n), as the tree is not balanced.
iv) During deletion, finding a candidate for replacement takes longer time.

Recent work on orthogonal range reporting in the pointer machine model describes a data structure of 0(n(log 11/ log log ti)d) space and answers d- dimensional orthogonal range reporting queries in (log n(log ni log log n)d-4+11W-2} k), where k is the number of points (Afshani et al. 2012). Range queries are mostly used in GIS applications to locate cities within a certain region in a map. Similarly, in the geometrical view of a database, one can use orthogonal range search to perform a query. PR k-d tree, Bucket PR k-d tree (Orenstein 1982), PMR lcd-trees (Nelson Samet 1986) and Path level compressed PR kd-trees (Nilsson Tikkanen 2002) are some of the trie-based kd-trees used to store point data. However, these trees are not always balanced, especially when the data is clustered. One of the dynamic versions of kd-tree is the pseudo lad-trey (Over mars & van Leuven 1982) where arbitrary points are used for partitioning the regions.
the main drawback of pseudo k d-tree is that it cannot store any two points having same co-ordinate in a given dimension and worst case range search time of n + k), where k is the number of points reported. A d-dimensional range query on died k d-trees (van K reveled Over mars 1991) requires0 (111-1-idlogl/d n k) time, which is better than the pseudo k d- trees. In the case of divided k d-trees, the lower level trees are split on first coordinate and upper level trees are based on second coordinates, such that leaves of upper tree connect to corresponding lower tree. These trees are hence balanced and perform exact match query in logarithmic time. But during updates, it is required to reconstruct the trees which is quite difficult and takes O(n log n) time. Yet another variant of pseudo k d-trees is the k-d-B-trees, the very first external memory based kd-trees (Robinson 1981), that performs range search in 0(Ar I B + KM) Ills, where B is the number of points in a disk block and K Is the number of points reported, but the insertion process is quite complex due to the choice of partitioning line.

iJETRM
International Journal of Engineering Technology Research & Management

Later the Bkd-trees (Procopius et al. 2003) improved the insertion time to 0(log m/B(N/B)log2(N/A1)), where M is the memory buffer with 100% space utilization and maintaining the same query time of k-d-B trees. An alternative to k- d-B trees are the k-d-B-tries which stored-dimensional data and are more flexible but the trees are not always balanced. Buddy-Trees (Seeger cg.: Kriegel 1990) overcome the drawbacks of k-d-B trees especially in terms of nature of split, when they cut the rectangles in intersecting regions. Thus, by means of a special technique called B- partitions, buddy trees outperform R-tree variants. The best known dynamically balanced free uses bitwise interlaced data (Tropf & Herzog 1981) over k d-trees mapping k dimensions to one dimension. Although their search time is 0(d(log n k)) for reporting k points, bitwise interlacing leads to discarded areas during range search. This means that the data structure is more suitable for compact ranges only.

KINETIC SEARCH QUERIES

In this section. We discuss the existing work on kinetic range search problems. Segment tree based kinetic BSP structures are used to find segments that overlap a query segment in kinetic range search on segments. A detailed introduction to KDS as well as kinetization of geometric data structures such as convex hull, Delaunay triangulation, segments etc. is given in (Bash 1999; Guttman 1984).

It is indeed difficult to maintain a set of moving segments ina data structure, be it axis- parallel or arbitrary. The main reason for this is that, it requires the KDS to be updated often and hence it is time consuming. In spatio-temporal database applications (Salzberg & T sotras 1999), stabbing query on moving segments as well as finding points that lie within a rectangle.

BITS-TREES

In this BITS-TREES a novel data structure named Binary In order Threaded Segment Tree (BITS-tree), is proposed for storing segments and to address both stabbing and range queries. The main objective of this new structure is mainly to overcome the issues in the segment trees. The comparative studies of this structure with those of segment trees are also addressed in the later part of this chapter.

BACKGROUND

Trees are the fundamental objects in computer science (Horowitz & S'ihni 1983), and specifically binary trees are important data structures used in several applications. In binary trees, no node can have more than two children. The main distinction between trees and a binary tree is that binary trees may have zero nodes. Whereas trees have at-least one node. Thus, a binary tree is a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called left sub-tree and right sub-tree. The left and right sub-trees of a binary tree is also a binary tree. A binary tree is said to be balanced, if the difference between the height of left and right sub-trees differ by utmost

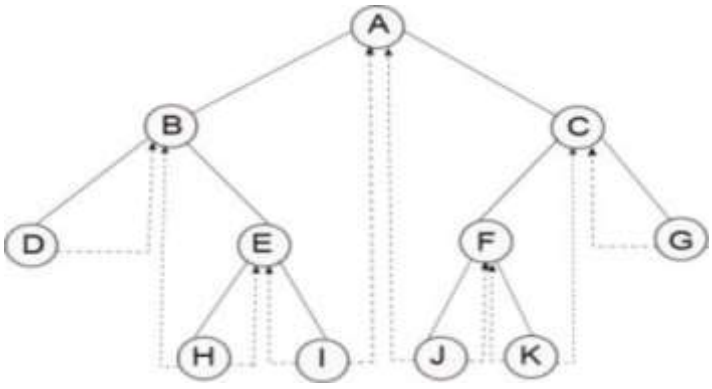

Figure 3.1: A threaded binary tree

1. A binary tree can be implemented either with an array or with a linked list. In the linked representation of binary trees, each node has two fields, one for the left child and the other for the right child. Hence, a binary tree with n nodes, require 2n links, out of which n+1 links are null link values. This means that more than half the number of links is not useful tree pointers. Perlis & Thronton (1960) suggested to replace these null link values by a valid pointer, called thread. That is, the null left and right child pointers are respectively replaced by an address of the node that immediately precedes and succeeds them in the in order traversal. Thus, the respective threaded tree is called in-order threaded tree. The in-order threaded representation of a sample binary tree is

iJETRM
International Journal of Engineering Technology Research & Management

given in Figure 3.1. Observe that in Figure 3.1, arrows denote threads, while lines denote normal links. Here, the in-order sequence of the binary tree is DBHEIAJFKCG. Hence, the left thread of H points to B and the right thread of H points to E which are respectively the in-order predecessor and successor of H in the in-order traversal. In a similar manner, other threads are drawn. Note that two threads are dangling (nodes D and G). In order to avoid these loose threads, a head node is used in some cases at level 0 to which the left thread of the first node in in-order sequence (node D) and right thread of last node in the in order sequence (node G) are linked. When the binary trees are dealt with keys, the following properties will be used. That is, for every node in the binary tree, its key value is larger than the key values present in its left sub-tree and is also lesser than the key values present in its right sub tree. This structure is termed as Binary Search Tree (BST). This supports searching a key value by pruning either the left or right sub tree, after comparing the key value of a node. But, it has a disadvantage that sometimes the height of the tree is linear to the number of nodes in the tree. Thus, to perform the search in logarithmic time, the tree needs to be balanced using some technique and the one is due to Adelson-Velskii & Landis (1962).

A PRIMER ON SEGMENT TREES

Segment trees are known to be one of the earliest non-conventional data structures with a wide range of applications. Originally, unit segment trees were used to represent a single line segment by dividing the range [() : N] in units. It is a complete binary tree in which consecutive unit intervals represent a node. Obviously, the drawback of unit segment trees is that, it stores a single segment only, whereas both SST and DST store a given set of segments.

STATIC SEGMENT TREE

As the name suggests, the abscissa are static, and is designed to handle intervals on the real line, whose extremes belong to a fixed set of N abscissa. Moreover, insertion or deletion of abscissa is not possible in SST. The abscissa can be normalized by replacing them by their ranks in the left-right order, assuming the abscissa are integers in the range [1].M. In general, a segment tree is a rooted binary tree where the interval corresponding to an internal node is the union of the ranges of its children, and the intervals of each leaf is one of the partitions obtained by partitioning the range [— 00, + 00]using the distinct extreme points of given set of segments. For instance, consider the segments a, b and c as in Figure 3.2. Thus the set of segments S = {[5, 12], [10, 15], [18, 21]}. Now, the construction of SST is as follows: Initially, the segment endpoints are sorted in increasing order by considering the real line ranging from — co to + co. The segments are then partitioned into the following fragments: [— 00, 5], [5, 10]. [10, 12]. [12, 15]. [15, 18], [18, 21]. [21, + co] as shown in Figure 3.3, and that form ranges to leaf nodes.

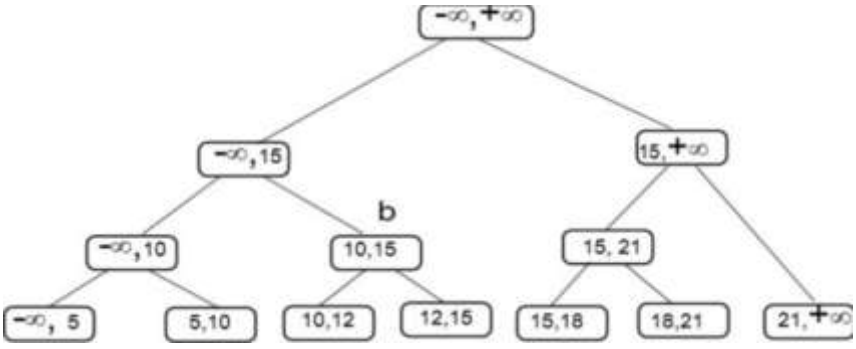

Figure 3.3: A static segment tree

Now, internal nodes at next level are constructed by making two consecutive leaf nodes as its children, as shown in Figure 3.3 and its range is the union of ranges of those two children. This process is to be repeated until forming the root node. Here, the root contains the entire range [— co, + 00], Note here that the construction of segment tree requires prior knowledge of all segments that contribute to the segment tree, and that is a severe limitation of SST.

DYNAMIC SEGMENT TREE

A dynamic segment tree (DST) is quite similar to the SST discussed above, as both are balanced rooted binary trees and their root node stores the overall range. However, unlike the SST insertion and deletion of an interval

iJETRM
International Journal of Engineering Technology Research & Management

range is possible in DST, but it is not possible to insert an interval beyond the overall range for which the tree is specified. The formal definition of DST is as follows:

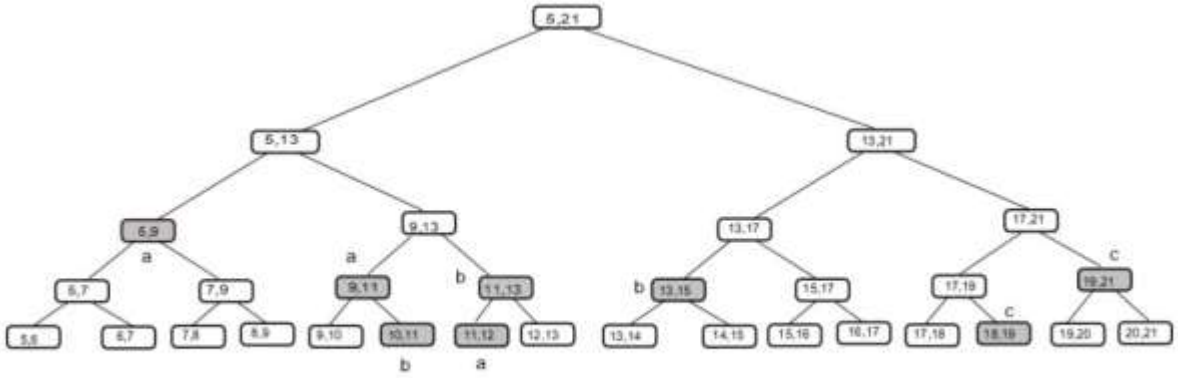

Figure 3.4: Dynamic segment tree for the interval [5 : 21]

From the above figure 3.4: Dynamic segment tree for the interval [5: 21] Definition : (Preparata & Shamos 1985) With given integers 1 and h, 1 < h, the segment tree T (1, 11) contains the root v with maximum range [1, 11] and if h-1 > 1, then it has a left subtree T (1, [(1 +10/2,) and a right sub h tree T ([(1 + h)/2;(5.21 ba).For instance, let us consider the same set of segments given in Figure 3.2, the corresponding DST T (5, 21), is shown in Figure 3.4. Observe that leaf nodes contain the elementary or atomic intervals of size one. Also, parts of each segment with label a, b, c appear in several nodes as indicated by the shaded (gray) nodes. In general, a segment tree T (1, 11) stores intervals whose extremes belong to the set {/, 1 + 1 . . . h}. This supports insertions and deletions in a dynamic fashion. A stabbing query on DST reports all segments intersecting a vertical segment, say Sq, for which we consider only xq, the x-coordinate of segment Sq. At every node of T , starting from the root, one has to check if xq is contained in the interval of that node, or required to check further on left or right child depends on whether .,to falls left or right of the interval range of the node. This approach is applicable to SST too, but the number of nodes visited will probably be less compared to DST practically. Also, note here that required answer lies in several nodes, both in SST and DST. One of the driving reasons to revisit the segment tree is that, therefore mentioned properties of segment trees are not useful for range search operation. In addition to this, the following thoughts are the real motivation behind the study of BITS-Tree. Reduce the number of nodes to be visited during stabbing query and range query. Extending the structure to higher dimensions. The theoretical results reported in the literature for complexity measures with respect to the segment trees are as follows: Preprocessing time: 0(11 log n) Storage time: 0(n log n) Query time: O(log n + k), where k is the number of segments reported.

CONSTRUCTION OF BITS-TREES

The key idea here is to fragment the segments at their end points, instead of finding the median as in the DST. Other than that, unlike in SST a particular range cannot be contained in many nodes. This greatly benefits the overall reduction in the number of nodes of the tree as compared to both SST and DST. Now, the formal definition of BITS-Tree is as follows: Definition 3.2: A BITS-tree is a height balanced two-way in order-threaded binary tree T that satisfies the following properties. Each node v of T is represented as r([a, b], L), where [a, b] is the range associated with the node v, and L is the list of segments containing the range [a, b].
i.e., if [c,E L] then [a, b] [c, d].Given vi([ai, b1], Li) v2([a2, b2], L2), then , [bi] if a2 = bi [ai, bid n [a2, b2] =[b2] if al = b2 cp otherwise i.e., ranges can either overlap only at end points or do not overlap at all. [1 0, 1 2] . [1 2, 15] and [18, 21]. Observe that the ranges are sorted in ascending order, as this satisfies the binary search tree properties on ranges. So, searching for a particular range could be done as in the case of BST . The dotted lines in Figure 3.5 denote threads. The hanging threads point to the dummy node, which is omitted in the figure. The threads are normally used in binary tree to speed up the traversal as the null pointers are replaced by some valid pointers. In other words, a null pointer of a node points to either the in-order successor or in-order predecessor, which depends on whether the null pointer is the left or right one. Note that these threads are usually unidirectional. However, in this work, bidirectional threads are used in order to have access to both the predecessor and successor in constant time. Now, when any two segments overlap, it may belong to any one of the possibilities such as left overlap, right overlap, contained in, or covered up. In each of these cases, they are

iJETRM
International Journal of Engineering Technology Research & Management

partitioned into three fragments and inserted into the tree. Definition 3.3: Given two segments S = [p, q] and T = [in, S < T implies q 111, that is the segment S falls left of T . Similarly, S > T implies n < p, and this means V falls right of T. Definition 3.4: Given two segments S = [p, q] and T = Em, n], where p q and 111 i = 11, p < in < q < n denotes S left overlaps with T. Similarly, m < p < n < q denotes S right overlaps with T. Otherwise, S is said to be contained in T when m <p<qizorm5_p<q< n, or covered up T when p<m<n q or p m <n < q. All the four cases stated in Definition 3.4 are commonly called as S and T overlaps. Note that, S left overlaps with T also implies T right overlaps with S. Similarly. S contained in T also implies S covered up T.

RESULTS

For our comparison, the same set of segments (Figure 3.2) is used to construct the SST , DST and BITS-tree in order to make it is easier to understand the difference in their representation and number of nodes visited during a stabbing query, and this comparison is listed in Table 3.1. From Table 3.1, we observe that the height as well as number of nodes of the BITS-tree are much lesser than the other two structures. in addition to this, other factors are also better than the other two structures. From the Table 3.1: Comparison of trees given in Figures 3.3, 3.4 and 3.5. SST denotes Static Segment Tree, DST denotes Dynamic Segment Tree and BITS denotes BITS-tree.

Table 3.1: Comparison of trees given in Figures 3.3, 3.4 and 3.5. *SST* denotes Static Segment Tree, *DST* denotes Dynamic Segment Tree and *BITS* denotes *BITS*-tree

Description	SST	DST	BITS
Number of nodes	13	31	4
Cumulative list size (for segment storage)	4	8	5
Height of the tree	3	4	2
Maximum stabbing query time	4	4	2
Maximum number of nodes stabbing query answer lies on	4	4	1

However, it is impossible to use in order sequence in standard segment trees due to the following two reasons.

i) The SST or BST does not have in order thread to efficiently access the next node in the in order sequence. ii) The range of the nodes overlaps with its ancestors. Thus, to perform a range search on SST or DST, one has to unseen known binary tree traversal method to answer the query, and that takes the time of 0(m), where in is the number of nodes in the tree. This is due to the fact that the properties of segment trees are not useful, in this case, to find the range search.

Thus, the BITS-trees outperform both SST and DST in case of answering the range query. From the Table 3.2: Theoretical comparison of static segment trees, dynamic segment trees and the proposed BITS-trees. SST denotes Static Segment Trees, DST denotes Dynamic Segment Trees and BITS denotes BITS-trees.

Table 3.2: Theoretical comparison of static segment trees, dynamicsegment trees and the proposed *BITS*-trees. *SST* denotes Static Segment Trees, *DST* denotes Dynamic Segment Trees and *BITS* denotes *BITS*-trees

Description	SST	DST	BITS
Maximum number of nodes	$4n+1$	$\geq 2(n_2 - n_1) - 1$	$2n-1$
Maximum cumulative list size (for segment storage)	$> n^2$	$> n^2$	n^2
Maximum height of the tree	$\lceil \log(2n+1) \rceil$	$\lceil \log(n_2 - n_1) \rceil$	$1.441\lceil \log n \rceil$
Maximum stabbing query time	$2\lceil \log(2n+1) \rceil - 1 + k$	$2\lceil \log(n_2 - n_1) \rceil - 1 + k$	$1.441\lceil \log n \rceil + 1 + k$

iJETRM
International Journal of Engineering Technology Research & Management

Other than the analysis given above, in BITS-tree the answer for a Stabbing query lies in only one node, but in SST or DST it may spread across several nodes. Due to the reduction in height and number of nodes in the tree, BITS-trees are definitely much better compared to the standard segment trees.

SUMMARY

A novel geometric data structure BITS-trees, to store one- dimensional segments is presented. The tree is a balanced two-way in order threaded segment tree, which is useful for Aiming geometric data objects available in one dimension. Our results give a logarithmic worst case upper bound for query time of the range search algorithm and fewer nodes required constructing the tree than SST and DST Also, since a line segment degenerates to a point, the BITS-tree can also be used to store points. In the forthcoming we discuss how to make use of the BITS-trees, one at each dimension to store multidimensional points and perform rectangular range search. It is quite easy to extend this data structure to store rectangle data also as in the case of R-tries. Therefore, our next contribution is based on this work, to find rectangle- rectangle intersections and address range search for window query as an alternate solution for those using R-trees. To summarize this chapter, we conclude that BITS-tree exhibits good dynamic behavior and meets the required conditions for an efficient solution to range search problems on segments.

REFERENCES

1. Abarn, MA gl de Berg, M 2005, 'Kinetic sorting and kinetic convex hulls', `Proceedings of SCG '05', pp. 190-197.
2. Abam, MA, de Berg, M, iSzr Speckrnann, B 2007, 'Kinetic kd-trees and longest side kd-trees', Proceedings of the twenty-third Annual Symposium On Computational Geometry, SCG 07', pp. 364-372.
3. Adelson-Velskii, GM & Landis, EM 1962, 'Algorithm for the organization of information', Soviet Mathematics, vol. 3, pp. 1259-1263.
4. Afghani, P, Arge, L, & Larsen, KG 2012, 'Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model', Proceedings of the twenty-eighth Annual Symposium on Computational Geometry', ACM, pp. 323-332.
5. Agarwal, PK, Arge, L, Erickson, J, & Yu, 1-12004, 'Efficient trade off schemes in data structures for moving objects', 'Twelfth Annual European Symposium on Algorithms (ESA), Norway', pp. 364-372.
6. Agarwal, PK & Erickson, I 1996, 'Geometric range searching and its relatives', Proceedings of the AWN-MS-SIAM Advances in Discrete and Computational Geometry', pp. 1-56.
7. Agarwal, PK, Erickson, I, Guibas, LJ 1998, 'Kinetic binary space partitions for intersecting segments and disjoint triangles', 'Proceedings of the ninth ACM-SIAM Symposium on Discrete Algorithms, Dallas', pp. 107-116.
8. Agarwal, PK, Guibas, LJ, Murali, TM, Vitter, JS 2000, 'Cylindrical static and kinetic binary space partitions', Computational Geometry-Theory and Applications, vol. 16, no. 2, pp. 103-127.
9. Agarwal, PK, Kaplan, H, Sharir, M 2008, 'Kinetic and dynamic data structures for closest pair and all nearest neighbors', ACM Transactions on Algorithms, vol. 5, no. 1, pp. 4-37.
10. Agarwal, PK et al. 2002, 'Box-trees and R-trees with near optimal query time', Discrete & Computational Geometry, vol. 28, no. 3, pp. 291-312.
11. Agarwal, PK et al. 2012, 'An optimal dynamic data structure for stabbing- sernigroup queries', SIAM Journal of Computing, vol. 41, no. 1, pp.104-127.
12. Allen, 1983, 'Maintaining knowledge about temporal intervals', Communications of the ACM, vol. 26, no. 22, pp. 832-843.
13. urge, L 1995, The buffer tree: A new technique for optimal 1/0 algorithms', `Proceedings of the fourth Workshop on Algorithms and Data Structures', pp.334-345.
14. Arge, L 1996, Efficient External-Memory Data Structures and Applications, PhD Thesis, University of Aarhus, Denmark,151 Arge, L, cleBerg, ivi, Haverkort, H, gf. K.Yi 2008, 'The priority R-tree: practically efficient and worst-case optimal R-tree', ACM Transactions on Algorithms [TALC], vol. 4, no. 1, pp. 9-34.

iJETRM
International Journal of Engineering Technology Research & Management

16. Arya, S et al. 1998, An optimal algorithm for approximate nearest neighbor searching fixed dimensions', Journal of the ACM, vol. 45, no. 6, pp. 891-923.
17. Basch, J, Guibas, LJ, Hershberger, J 1997a, 'Data structures for mobile data', `Proceedings of the eighth ACM-SIAM Symposium on Discrete Algorithms',pp. 747-756.
18. Basch, j, Guibas, Shang, L 1997b, Proximity problems on moving points.', 'Proceedings of the thirteenth Annual ACM Symposium onComputational Geometry', pp. 344-351.
19. Bash, 1999, Kinetic Data Structures, Stanford University, Department of Computer Science, Ph.D Thesis.
20. Beckmann, N, Kriegel, HP, Schneider, R, Seeger, B 1990, 'The R*tree-an efficient and robust access method for points and rectangles', Proceedings of the ACM SIGMOD nineteenth International Conference on Management of Data. (SIG OD 1990)', pp. 322-331.
21. Bentley, JL 1975, 'Multidimensional binary search trees used for associative searching', Communications of AC.M, vol. 18, no. 9, pp. 509-516.
22. Bentley,11,1977, 'Solutions to klee's rectangle problems', Tech. rep., Carnegie Mellon University.
23. Bentley, 11,1979a, Teconnposable search problems', Information Processing Letters, vol. 8, no. 5, pp. 5-9.
24. Bentley, IL 1979b, 'Multidimensional binary search trees in database applications', IEEE Transactions on Software Engineering, vol. SE-5, no.4, pp. 333-340.
25. Bentley, 1980, 'Efficient worst-case data structures for range searching', Acta Informatics, vol. 13, pp. 155-168.
26. Bentley, JL &. Friedman, JH 1979, 'Data structures for range searching', ACM Computing Surveys (Cain vol. 11, no. 4, pp. 399-422.