A Reseach Project Report

Reporting and counting maximal points in a query orthogonal rectangle

By

Ananda Swarup Das: *IBM India Research Lab, New Delhi, India*
Prosenjit Gupta: *Heritage Institute of Technology, Kolkata, India*
Kishore Kothapali: *International Institute of Technology, Hyderabad, India*
Kannan Srinathan: *International Institute of Technology, Hyderabad, India*

PUBLISHER: Journal of Discrete Algorithms

ACCEPTED: 4th December 2014

Under the supervision of

TATHAGATA RAY

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

HYDERABAD CAMPUS

ANAMIKA DAS

2014H313059H

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof Tathagarte Ray who gave me the opportunity to do this Research project on the topic of Counting the maximal points in an orthogonal rectangle, which helped me doing the Research and learn a new dimension from it.

I would express my gratitude to BITS Pilani Hyderabad Campus for providing me with the required opportunity and infrastructure.

ABSTRACT

In the given Research Paper, a proposal of a solution with sub-logarithmic query time for counting the number of maximal points in an axis parallel query rectangle.. The problem has been previously studied, to the authors' best knowledge, this is the first sub-logarithmic query time solution for the problem. The model of computation is the word RAM with word size of log(n) bits.

The main purpose of the project is to analyse,study and implement the algorithm to solve open problems.

This algorithm is useful in the context of database where enormous data are executed per second. There are customer inputs as to retrieve the results. Time complexity increases as the size of the data increases. To solve this problem, the author has given an efficient algorithm.

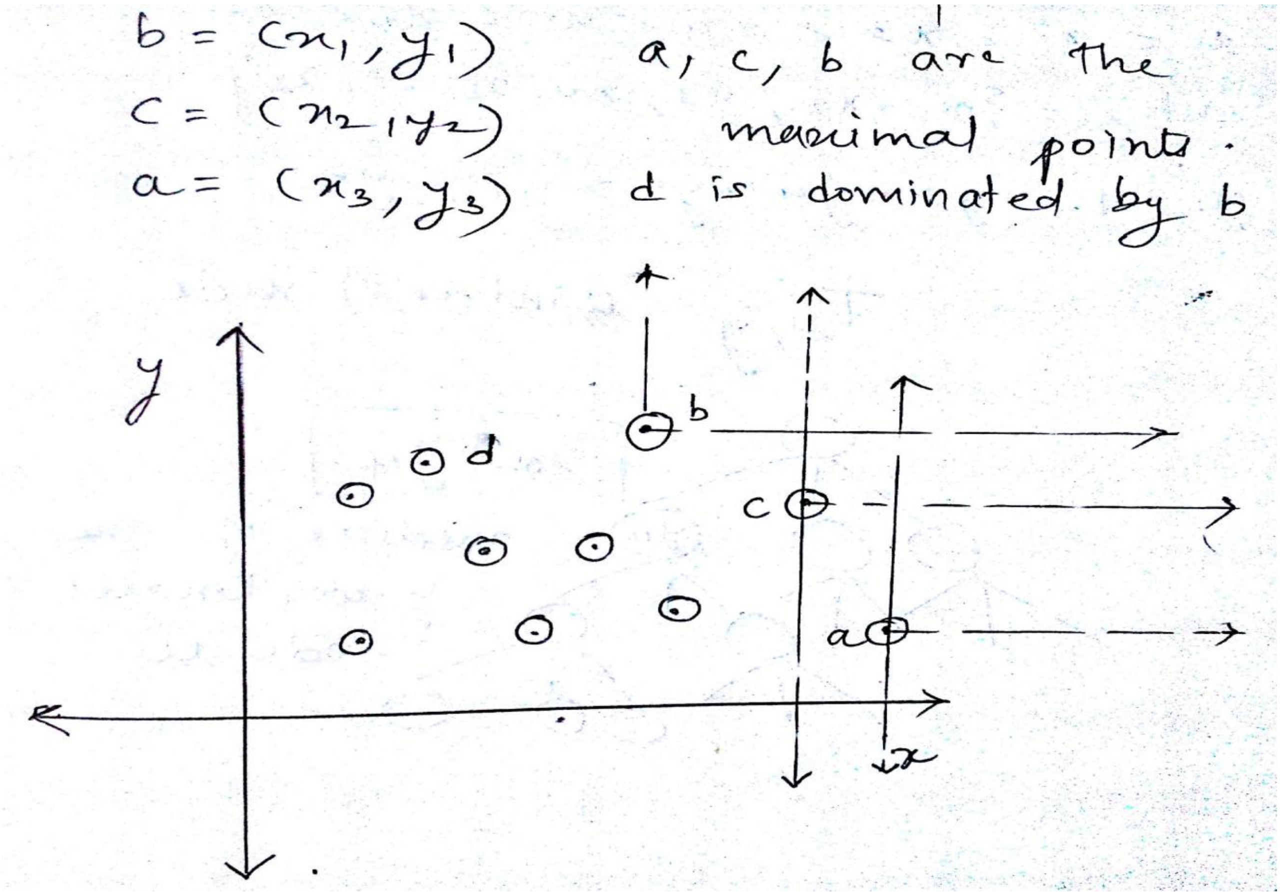
INTRODUCTION

The given reseacrch paper deals with the skyline points or maximal points.

Accoring to the definition, we have two coordinates x and y.

Let us consider set of points in the 2 D space i.e., x and y coordinates.

According to the definition of maximal points, if point q(x1, y1) is dominated by point p(x2, y2) iff x1<=x2 and y1<=y2. This point p is called as maximal point.



From the above diagram, we can observe that point d is dominated point b. similarly, a, c are dominating points. a, c, b are the maximal points.

Our main context, we need to compute the count of maximal points from a given set of points S and a query rectangle R in O(logn/loglogn).

PROBLEM DEFINITION

PROBLEM 1:

Given a set R of n points on an $n \times n$grid, that is $R = \{(xi, yi) | xi \in \{1, n\}, yi \in \{1, n\}\}$, preprocess R into a data structure such that given an orthogonal query rectangle q, the maximal points in $q \cap R$ can be reported efficiently.

Assume all the points to have distinct x and y co-ordinates.This problem requires storage of size $\boldsymbol{O(n\log n/\log\log n)}$ and query time $\boldsymbol{O((\log n/\log\log n)+k)}$ time.

SOLUTION:

1. Consider two level binary search Tree Tx.

2. Given a query rectangle q=[a, b] x [c, d]. the segment [a, b] can be allocated to at most O(log n) canonical nodes of the tree Tx. Ordered position: right to left.

3. To achieve the desired complexity, increase the internal degree of each internal node of Tx from two to $\boldsymbol{O(\sqrt{\log n})}$**, there**by decreasing the height of the tree to $\boldsymbol{O(\log n/\log\log n)}$

4. Given a query rectangle [a, b] x [c, d], allocate the segment [a, b] to a node **$\mu \in Tx$, if $int(\mu) \subset [a, b]$ but $int(parent(\mu)) \neq [a, b]$. The node μ is known as a *canonical node*.**

5. The set of all such canonical nodes can be grouped into $\boldsymbol{O(\log n/\log\log n)}$ groups with each group containing children v1, …, vk for some node v.

6. Node v is called as group leader for the group g(i). Denoted by GL(i).

7. **Let $G = \{GL(1), ..., GL(O(\log n/\log\log n))\}$ be the set of *group leaders* stored in order of their positions from right to left in the tree *Tx*.**

8. If we visit each group leader node GL(i) from right to left and the process count in O(1) time the number of maximal points in q from the nodes of the group g(i) where GL(i) is the group leader, we can count the maximal points in q in $\boldsymbol{O(\log n/\log\log n)}$**.**

In order to solve the problem 1, we need to solve the subproblems.

PROBLEM 2:

Given a set S of n points from a universe of [1, log(^ρ)n] ×[1, n] for 0 <ρ≤1/2, preprocess the points into a data structure such that given an axis parallel rectangle

q =[a, b] ×[c, d] for (a, b) belongs to [1, log(^ρ)n] ×[1, log(^ρ)n] and (c, d) belongs to [1, n] ×[1, n], we can efficiently count the number of maximal points in $S∩q$.

SOLUTION:

There are 4 points: (x1, y1), (x2, y2), (x3, y3), (x4, y4)

1. Array A={y1, y2, y3, y4} where y1 > y2 > y3 > y4.

2.Construct a height-balanced binary search tree *Ty*.

3. Leaf nodes store the values in the array A , the level is same for every leaf node. At each internal node *m* belongs to *Ty*, store a key value *ym* which is the median of the points stored in the subtree rooted at *m*.

For example: ym = (y1+y2)/2 where y1 and y2 are the leaf nodes of subtree m.

3. For each pair of possible points (*a*,b) belongs to [1, logρ*n*] ×[1, logρ*n*], form an interval [*a*, *b*].

4. For each value *y*2 belongs to *A* and for each possible interval [*a*, *b*], form 3-sided anchored rectangles [*a*, *b*] ×[*y*2, ∞)and [*a*, *b*] ×(−∞, *y2*]. Next, for the interval [*a*, *b*] and the value *y*2.

(a)Visit the ancestors of *y*2 in the tree *Ty*. At each ancestor *m*, find(*ym)*.
Median of the root= (y2+y3)/2

(b)If *ym*<*y*2, form an axis-parallel rectangle *R*1=[*a*, *b*] ×[*ym*, *y*2].

i.For the points in *S*∩*R*1, compute the subset of points that are not dominated by any other point. This subset is called as maximal chain.
pymax= the topmost point of the maximal chain
pxmax = the bottommost point of the maximal chain

ii.Count the number of maximal points in the chain *pxmax* to *pymax*. Denote the value as *countm*(*y*2)= |*pxmax*, *pymax*|.

iii.Find the topmost point *pnodom* in $[a, b] \times (-\infty, ym]$ such that *pnodom* is not dominated by *pxmax*.

iv.Create a tuple (*countm*(*y*2), *pnodom*)and store it with reference to the rectangle *R*1in a lookup table.
Here the suffix *m* denotes the index for the node *m* belong to *Ty* that is an ancestor of the leaf node storing the value *y*2.

v.**Special cases**
A.If no points are present in the rectangle *R*1, store (0, *NULL*).
B.If the point *pnodom* does not exist, store (*countm*(*y*2), *NULL*).

(c) If $ym>y2$

Rectangle $R2=[a, b] \times [y2, ym]$

Find(*p'ymax* , *p'xmax)*

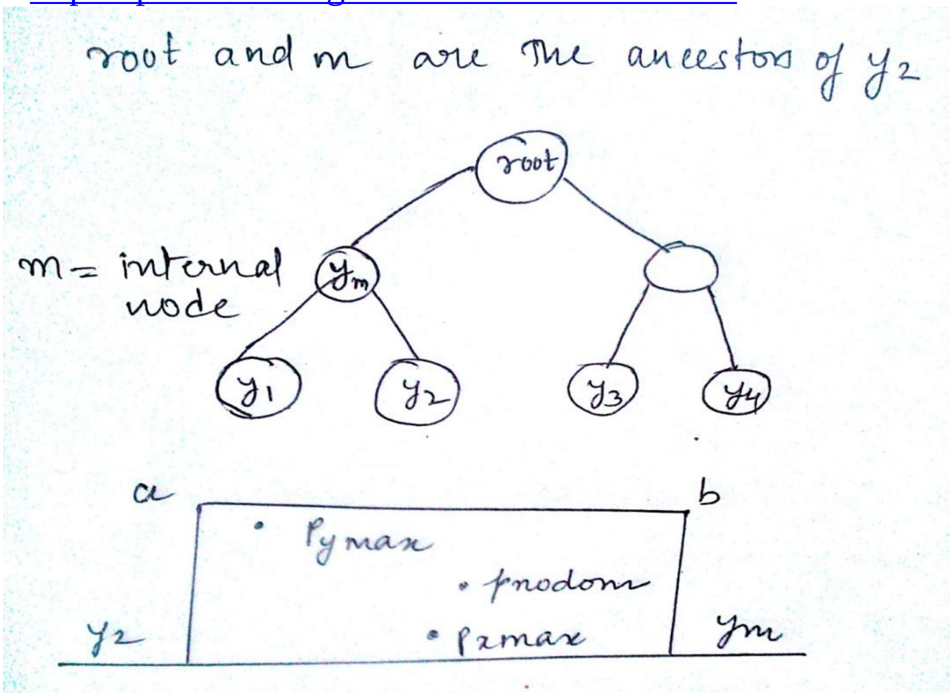
count'*m*(*y*1)=|*p*'*xmax*, *p*'*ymax*|.

Store it in a tuple (*count*'*m*(*y*1), *p*'*xmax*).

(d) Special Case: If there are no points in R2, store(0, NULL).

5. Maintain the data structure (*) inorder to find the least common ancestor for two given leaf of the tree Ty in O(1) time.

* http://epubs.siam.org/doi/abs/10.1137/0213024



LEMMA 1:

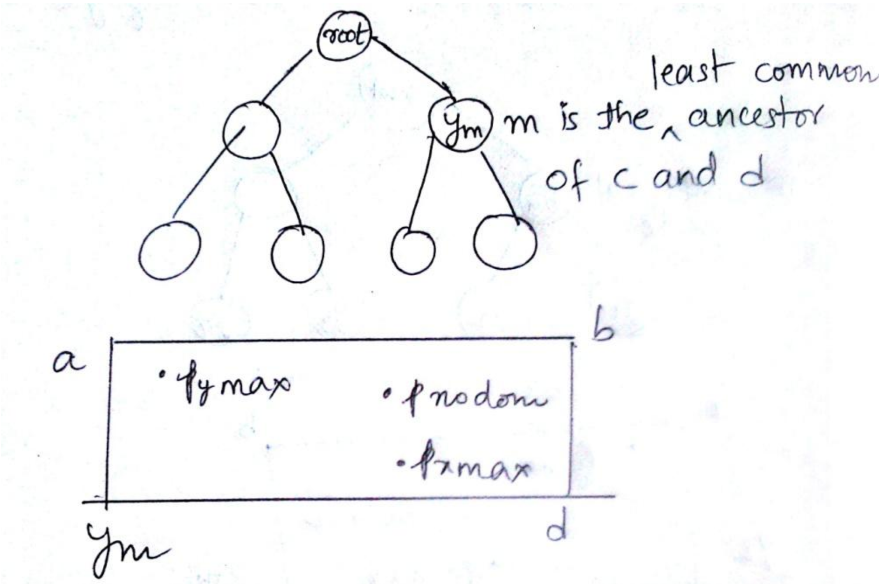
The total storage space needed by the data structure is O(n log(1+2ρ)n) words.

QUERY ALGORITHM:

1. Given a query rectangle [*a*, *b*] ×[*c*, *d*] such that (*a*, *b*) belongs to [1, logρ*n*] ×[1, logρ*n*] for 0 <ρ≤1/2 and (*c*, *d*) belong to [1, *n*] ×[1, *n*].

Find the least common ancestor *m* for the values *d*, *c* in the tree *Ty*. Let the key value stored at *m* be *ym*.

2. Consider the rectangle [*a*, *b*] ×[*ym*, *d*]and the corresponding tuple (*countm*(*d*), *pnodom*). Where countm(d) has the same processor as countm(y2).



3. <u>Pseudo code</u>

get_maximal_chain(Rectangle)
find pxmax and pymax and return the count of points between pxmax and pymax.

If ($pnodom \neq NULL$)
$pnodom(y)$ => y-coordinate of the point *pnodom*.

If ($c \leq pnodom(y)$)
Rectangle => $[a, b] \times [pnodom(y), ym]$.
Rectangle=> $[a, b] \times [c, ym]$.
$count'm(pnodom(y))$ => get_maximal_chain($[a, b] \times [pnodom(y), ym]$).
$count'm$(c)=> get_maximal_chain ($[a, b] \times [c, ym]$)
return $count'm(c) - count'm(pnodom(y)) + 1 + countm(d)$.

else if($pnodom(y) < c$)
return $countm(d)$.

else if ($pnodom=NULL$)
if ($countm(d) \neq 0$)
return $countm(d)$.
else
return $count'm(c)$.

// Special Cases

if(count'm(c)=0) return countm(d).

if (countm(d)=0) return count'm(c).

if (count'm(c) $\neq$ 0 and *pnodom*= NULL) return countm(d).

PROBLEM 3:

Given an unsorted array $A$of n integers from the range of $[0, \sqrt{\log n}-1]$, preprocess A into a linear space data structure such that given two indices i, j and two values, a, b :(a, b) belongs to $[0, \sqrt{\log n}-1] \times [0, \sqrt{\log n}-1]$, we can efficiently report the smallest value t belongs to $A[i, j]$ for $a \le t \le b$.

This is clearly a variant of range successor problem and we can solve the problem as follows:

PREPROCESSING:

1.For A[i]=x , we create a 2D point (x, i). Let S' be the set of such points.

2.We preprocess S into a data structure denoted by RS.

3. RS is an instance of the data structure of Theorem 4.

4. From [*], RS queries of the form $[x1, \infty)\times[y1, y2]$:

$x1$ belongs to $[1, \log n/\log\log n]$ and ($y1$, $y2$) belongs to $[1, n] \times[1, n]$ queried in $O(1)$ time using a linear space data structure for a set of n 2D points with coordinates from a rectangle of $[1, \log n/\log\log n] \times[1, n]$.

In the given query, for any point (x, i) belongs to S', x belongs to $[0, \sqrt{\log n}-1]$ and i belongs to $[1, n]$.

Theorem 4:

If the coordinates of the points are on an integer grid of $[1, \log n/\log\log n] \times[1, n]$, *then there exists a linear space data structure that can be queried in* $O(1)$ *time to report the point with smallest x-coordinate in* $[a, \infty)\times[c, d]$ *where* a belongs to

$[1, \log n/\log\log n] \times[1, \log n/\log\log n]$ *and* (c, d) belongs to $[1, n] \times[1, n]$.

RS: Range Successor

[*]http://www.sciencedirect.com/science/article/pii/S0925772110000696

QUERY ALGORITHM:

1.Given the two indices i, j :(i, j) belongs to $[1, n] \times [1, n]$ and two values a, b :(a, b) belongs to $[0, \sqrt{\log n}-1] \times [0, \sqrt{\log n}-1]$, we construct a three sided query rectangle of the form $[a, \infty) \times [i, j]$.

2.We run the range successor query on the data structure *RS* and find the smallest value t in $[a, \infty) \times [i, j]$.

If $(t \leq b)$ return t

else return *null*.

PROBLEM 1 SOLUTION

R={(x1, y1), (x2, y2),………..(xn, yn)}

1. Construct a tree Tx, the leaf nodes are at the same height.

2. The leaf nodes store the x-coordinates of the points in the set R in non-descreasing order.

3. Each internal node μ belong to *Tx* has O(√logn) children. Left most child numbered as √logn -1 and right most child numbered as 0.

4. μ is an internal node. int(μ) = union of x1, x2, x3, …. The values at the leaf node subrooted at μ.

5. For the internal node except root:

(a) create an auxillary array A(μ) which stores the y coordinates of the x coordinates subrooted at μ in decreasing order.

A(μ)= {y1, y2, y3,..} where y1>y2>y3> … and A(μ) = U(A(i)): i=0, 1, … √logn -1 where U is the union and vi is a child of node μ.

(b) Each element of A(i), i= 0, 1, … √logn -1 will point to its corresponding position in the array A(μ).

(c) A data structure D(μ) for A(μ): A(μ)[j]= yj.

Create a 2D point (vi, yj) where yj belongs to A(i) , A(i)= an auxillary array of the child vi of the node μ and vi belongs to [0, √logn -1]. These points belong to [0, √logn -1] x [1, n].

(d) Each child vi of of μ maintains a binary string lookup of size |A(μ)| where A(μ)[z]=1 if belongs to A(i).

(e) lookup supports rank() and select() queries

Rank query: A rank query denoted by rankc(S, i) reports the number of c in S from position 1 to position i. Let S=222111331. The query rank2(S,4) =3

Select query:A select query selectc(S, j) returns the position of the jth occurrence of the character c in S. Let S=222111331 The query select2(S, 2) =2.

(f) RMA(μ), a range maxima data structure: return the maximum x coordinates whose y coordinates are between A(μ)[i] to A(μ)[j].

(g) For the values of A(μ):

VT(μ) => van Emde Boas tree and T(μ, y)=> height balanced binary tree for any node n of T(μ, y) stores the median of the values stored in the leaf nodes.

4. For the root :

(a) Each index *i* of the array at the root node *A(root)* will have 2 $\sqrt{\log n}$ pointers of which $\sqrt{\log n}$ pointers will be pointing to the smallest elements greater than *A*(*root*[*i*]) in each of the arrays *A[j]* for *j* being a child of the root node.

(b) The other $\sqrt{\log n}$ pointers will be pointing to the largest elements greater than *A*(*root*[*i*]) in the arrays *A[j]* for *j* being a child of the root node.

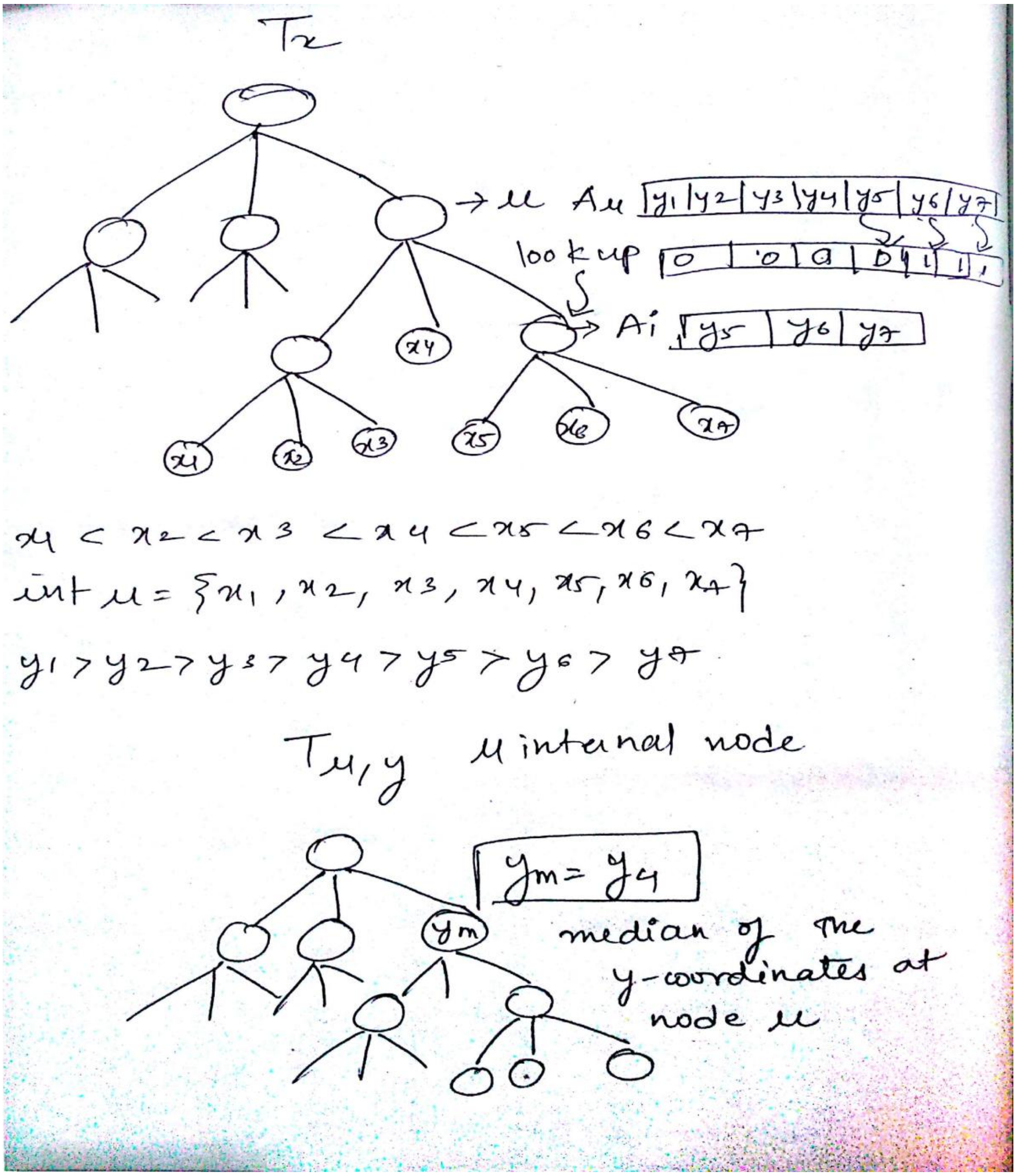
(c) Construct a range maximum data structure *RMA(root)* :return the maximum x coordinates whose y coordinates are between *A*(*root*[*i*]) to *A(root*[*j*]).

5**.** At each internal node μ belongs to *Tx:*
S' be the set of children of the node μ where children= vi, …, vj.

(a)R1= [i, j] x [y, ym] , ym>y or R1=[i, j] x [ym, y], ym<y where ym is stored in tree T(μ, y) for each element y belong to A(μ).

(b)Find_maximal_chain(Rectangle R1, set S') denote it ptop and pbottom and count the maximal points.

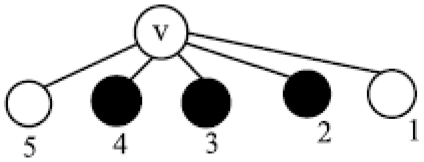
(c) Tuple(R1) =(ptop, pbottom, count) for Rectangle R1



Lemma2 : *The total storage space needed by the data structure for the counting problem will be O*(*n*log3*n/*loglog*n*)*.*

DECOMPOSITION OF RECTANGLE

1. Given a rectangle [a,b] x [c,d] , the segment [a, b] is allocated to a node μ in Tx. If int(μ) subset of [a,b] but int(parent(μ)) doesnot belong to [a, b]. There will be O(log(^3/2)/loglogn) such points. The set of such nodes be V.



. The segment $[a, b]$ is allocated to the nodes colored black.

2. These are canonical points and group it as GL(1), ..GL(logn/loglogn) and GL is group leader.

3. To count the maximal point, we decompose the rectangle into smaller rectangle. The number of rectangles will be O(logn/loglogn).
Rightmost Rectangle: R1 and Leftmost Rectangle: Rz.
All the rectangles have same height.

For width:
Let vi be the rightmost point in Tx among all the nodes in V then

(a) Consider GL(1) for which the node vi is a child and g(1) conists of the children of GL(1).
R1 width interval = U(int(vj)) where U is the union. Iff vj is a child of GL(1) and int(vj) is a subset of [a, b] or vj belongs to g(1).

(b) V'=V-g(1) for this vk be the rightmost node. Consider GL(2) and set g(2) for the node vk. Width of R2= U(intervals of the nodes in g(2))where U is the union.

(c) Thus we can calculate the width of all the rectangle.

COUNTING ALGORITHM

Tx tree
lca= Find__Least_Common_Ancestor(a, b)
Get(A(lca)) // array of lca
A(lca)[i] (smallest value)>= c and A(lca)[j](largest value) <=d
//use van Emde Boas tree at the node lca.
GL(1)=> leader node subtree rooted at the child node vm for node lca.
t= number_of_ones_lookup_till_(i-1)th msb.
select(1)(A(vm), (t+1)) = position z //use select operation
The largest value <d = position z
find y'2<=d and y'1>=c in A(GL(1)).
The children of GL(1) are in the set g(1) //vi to v√logn -1.

Search(m in D(GL(1))) m is the smallest index such that the node vm has the rightmost maximal point in the rect [a, b] x [y'1, y'2].run RS query to find m.

Total1=Count_maximal_chain in the rectangle [m, √logn -1] x [y'1, y'2]
pymax=(pymax(x), pymax(y))
GL(2)=>leader node and g(2) = vw, .., vq
Total2=Count_maximal_chain in the rectangle [w, q] x [pymax(y), d]

GL(2) satisfy one of the following:

(a) GL(2) is in the path from GL(1) to lca.
A(GL(1)) has a pointer to the parent of GL(1) array.
y'2 <=d in A(GL(2)) is easy to find.

(b) GL(2) is in the path from lca to GL(1).
First move to the node lca by following pointers from GL(1)
Then proceed with (a)

(c) GL(2) is the lca
Move to the node lca by following pointers from GL(1).

Repeat for all the group leader nodes

Return Total=$\sum i\, Total$: i=1, .., O(logn/loglogn).

QUERY TIME ANALYSIS AND CORRECTION PROOF

The query algorithm correctly counts the number of maximal points inside the query rectangle.

Divide the problem of reporting maximal points for a query rectangle in a general setting to $O(\log n/\log\log n)$ instances of the problem of counting maximal points for a query rectangle (as per problem 2).

By Lemma1, the correct count of maximal points can be calculated.

Thus, the query algorithm takes O(logn/loglogn) time to count the number of maximal points in the query rectangle.

The storage space can be improved without sacrificing the query time efficiency by reducing the degree of each internal node of the tree to log(^ρ) *n* for 0 <ρ<12.

CONCLUSION

According to the theorem and analysis of the algorithm, we can implement this algorithm in many practical areas of database management system for speeding the querry time. We have also learnt that we can reduce the storage capacity without compromising the query time.
For future work, we can increase the dimension i.e experiment with 3D coordinates and analyze the time complexity of the algorithm.

Here is the link of the ongoing code:
https://drive.google.com/file/d/0B6O5O3fDqH3Ld1JWX21YNmF1SzQ/view?usp=sharing

REFERENCES

- http://epubs.siam.org/doi/abs/10.1137/0213024
- https://en.wikipedia.org/wiki/Van_Emde_Boas_tree
- Books
- http://www.sciencedirect.com/science/article/pii/S0925772110000696
- http://refhub.elsevier.com/S1570-8667(14)00095-1/bib7975s1