Assignment no 4

Sub : ADS

Roll : 26

Name : Rohit Rajbhar

Q. 1. What are geometric algorithms? List and describe any five applications of geometric algorithms in brief.

Ans :

These algorithms are designed to solve Geometric Problems. They require in-depth knowledge of different mathematical subjects like combinatorics, topology, algebra, differential geometry etc.

Applications of geometric algorithms:

1) computer graphics

Wide use of computer graphics has taken place where the graphics quality has taken place where the graphic quality is built using geometric algorithms. The good example is screen resolution of different screens but graphic quality remains the same.

2) robot motion planning

The calculation related to the area covered by robot vision ,distance to travel is all solved or built using geometric algorithms.

3) computer games

Huge demand for computer games which are built on geometric algorithms where the graphic quality and angles use to allocate the character. The great example of computer games using geometric algorithms is GTA.

4) simulation

In these the calculation of angles and all geometric calculations are done using geometric algorithms and then used to built the simulations models eg. Railway simulation for training purposes.

5) geographical information system

The best example we have is google maps which calculate distance from source to destination by using satellite image.

Q. 2. What is a range query? Explain the process of updating array elements using range queries.

Ans :

In data structures, a **range query** consists of preprocessing some input data into a data structure to efficiently answer any number of queries on any subset of the input. Particularly, there is a group of problems that have been extensively studied where the input is an array of unsorted numbers and a query consists of computing some function, such as the minimum, on a specific range of the array.

The process of updating array elements using range queries is as follows:

Consider an array A[] of integers and following two types of queries.

1. update(l, r, x) : Adds x to all values from A[l] to A[r] (both inclusive).
2. printArray() : Prints the current modified array.

Examples :

```
Input : A [] { 10, 5, 20, 40 }
        update(0, 1, 10)
        printArray()
        update(1, 3, 20)
        update(2, 2, 30)
        printArray()
Output : 20 15 20 40
         20 35 70 60
```

\

Q. 3. What is a segment tree? Describe the memory representation of segment trees.

Ans :

A Segment Tree is a data structure that allows answering range queries over an array effectively, while still being flexible enough to allow modifying the array. This includes finding the sum of consecutive array elements $a[l…r]$, or finding the minimum element in a such a range in $O(\log n)$ time. Between answering such queries the Segment Tree allows modifying the array by replacing one element, or even changing the elements of a whole subsegment.

The memory representation of segment tree is as follows:

The main consideration is how to store the Segment Tree. Of course we can define a $Vertex$ struct and create objects, that store the boundaries of the segment, its sum and additionally also pointers to its child vertices. However this requires storing a lot of redundant information. We will use a simple trick, to make this a lot more efficient. We only store the sums in an array. The sum of the root vertex at index 1, the sums of its two child vertices at indices 2 and 3, the sums of the children of those two vertices at indices 4 to 7, and so on. It is easy to see, that the left child of a vertex at index i is stored at index $2i$, and the right one at index $2i+1$.

Q. 4. Construct the segment tree for a given array and write an algorithm to calculate the sum of array elements in the given range.
Given Array: [1, 3, 5, 7, 9, 11]

Ans :

For diagram refer pdf

algorithm

```
int getSum(node, l, r)
{
   if the range of the node is within l and r
        return value in the node
   else if the range of the node is completely outside l and r
        return 0
   else
    return getSum(node's left child, l, r) +
           getSum(node's right child, l, r)
}
```

Q. 5. Write a note on sum queries and update queries. Also specify the time complexity of both algorithms.

Ans :

Sum queries

Given an array arr of integers of size n. We need to compute the sum of elements from index i to index j. The queries consisting of i and j index values will be executed multiple times.

Examples:

```
Input : arr[] = {1, 2, 3, 4, 5}
        i = 1, j = 3
        i = 2, j = 4
Output :  9
          12
```

A **Simple Solution** is to compute the sum for every query.

An **Efficient Solution** is to precompute prefix sum. Let pre[i] stores the sum of elements from arr[0] to arr[i]. To answer a query (i, j), we return pre[j] – pre[i-1].

Here time complexity of every range sum query is O(1) and the overall time complexity is O(n).

Update queries

Consider an array A[] of integers and following two types of queries.

1. update(l, r, x) : Adds x to all values from A[l] to A[r] (both inclusive).
2. printArray() : Prints the current modified array.

Examples :

```
Input : A [] { 10, 5, 20, 40 }
        update(0, 1, 10)
        printArray()
        update(1, 3, 20)
        update(2, 2, 30)
        printArray()
Output : 20 15 20 40
         20 35 70 60
```

A **simple solution** is to do following :

1. update(l, r, x) : Run a loop from l to r and add x to all elements from A[l] to A[r]
2. printArray() : Simply print A[].

Time complexities of both of the above operations is O(n)

Q. 6. Explain KD tree with an example.

Ans :

The ***kd tree*** is a modification to the ***BST*** that allows for efficient processing of ***multi-dimensional search keys***. The kd tree differs from the BST in that each level of the kd tree makes branching decisions based on a particular search key associated with that level, called the ***discriminator***.

Example 15.4.1 -for diagram refer to pdf

Consider searching the kd tree for a record located at P=(69,50)P=(69,50). First compare PP with the point stored at the root (record AA in Figure **15.4.1**). If PP matches the location of :math:A`, then the

search is successful. In this example the positions do not match (AA 's location (40, 45) is not the same as (69, 50)), so the search must continue. The xx value of AA is compared with that of PP to determine in which direction to branch. Because AxAx's value of 40 is less than PP's xx value of 69, we branch to the right subtree (all cities with xx value greater than or equal to 40 are in the right subtree). AyAy does not affect the decision on which way to branch at this level. At the second level, PP does not match record CC's position, so another branch must be taken. However, at this level we branch based on the relative yy values of point PP and record CC (because 1mod2=11mod2=1, which corresponds to the yy-coordinate). Because CyCy's value of 10 is less than PyPy's value of 50, we branch to the right. At this point, PP is compared against the position of DD. A match is made and the search is successful.

Q. 7. Write an algorithm to find the maximum of an array and the number of times it appears into an array using a segment tree.

Ans :

```
pair<int, int> t[4*MAXN];

pair<int, int> combine(pair<int, int> a, pair<int, int> b) {
    if (a.first > b.first)
        return a;
    if (b.first > a.first)
        return b;
    return make_pair(a.first, a.second + b.second);
}

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = make_pair(a[tl], 1);
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}

pair<int, int> get_max(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return make_pair(-INF, 0);
    if (l == tl && r == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    return combine(get_max(v*2, tl, tm, l, min(r, tm)),
                   get_max(v*2+1, tm+1, tr, max(l, tm+1), r));
```

```
}

void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = make_pair(new_val, 1);
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = combine(t[v*2], t[v*2+1]);
    }

}
```

Q. 8. Explain one-dimensional range searching.

Ans :

4.1 One-dimensional range searching

In order to extend the technique mentioned above to higher dimensions, an alternate structure for 1-D range searching is needed. This technique is more complex than a simple balanced search tree.

4.1.1 One-dimensional range searching: data structure

See Figure 4.1. A "Range Tree" is a balanced search tree T, with the following characteristics:

- leaves $\leftrightarrow$ points, sorted by x-coordinate.
- node $\mu \leftrightarrow P(\mu)$ is the subset of points at the leaves in the subtree μ.

The space needed to n points in the one-dimensional ranage tree is $O(n \log n)$

4.1.2 One-Dimensional Range Queries

In Figure 4.1.2 the bottom bold line segment represents the query range (x', x''). A query range corresponds to a set of the so-called "allocation nodes". An allocation node of μ of T for the query range (x', x'') is such that (x', x'') contains $P(\mu)$ but not $P(\text{parent}(\mu))$.

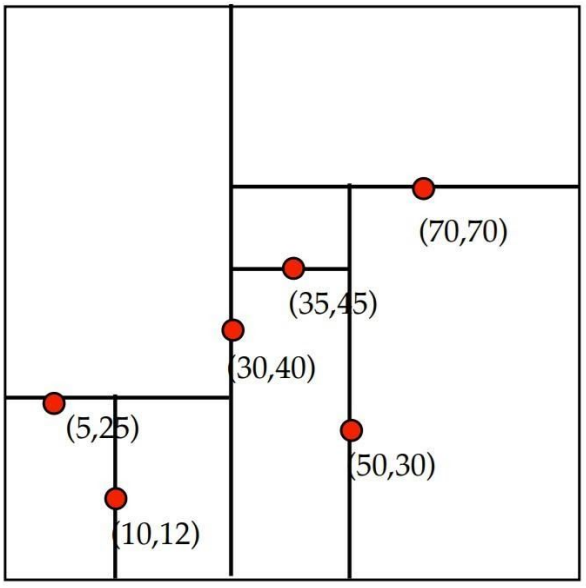
- the allocation nodes are $O(\log n)$
- they have disjoint point-sets
- the union of their point-sets is the set of oints in the range (x', x'')

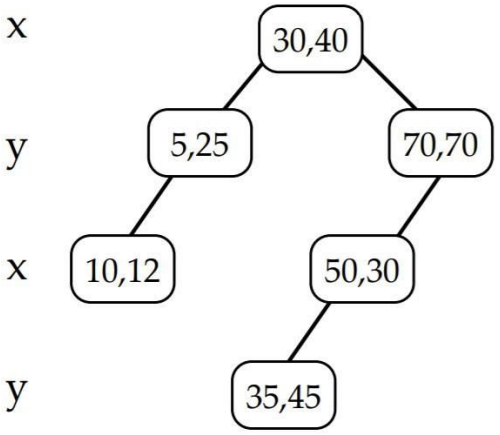
The Query Algorithm is:

1. find the alocation nodes of (x', x'')
2. *for each* allocation node μ report the points in $P(\mu)$

Q. 9. Construct the KD tree for the following geometry diagram.

insert: (30,40), (5,25), (10,12), (70,70), (50,30), (35,45)






```
insert(Point x, KDNode t, int cd) {
        if t == null
                t = new KDNode(x)
        else if (x == t.data)
        // error! duplicate
        else if (x[cd] < t.data[cd])
                t.left = insert(x, t.left, (cd+1) % DIM)
        else
                t.right = insert(x, t.right, (cd+1) % DIM)
        return t
        }
```