ADS UNIT-5

Range trees:

* The simplest form of multidimensional data are d-dimension-al points.
* They can be represented by a sequence of $(x_0, x_1, \cdots, x_{d-1})$
* A range-search query is a request to retrieve all points in a multi-dimensional collection whose coordinates fall within given ranges.

1-D Range Searching:

* Given an ordered dictionary D, the following query operations is performed:

> findAllinRange(k_1, k_2): Return all the elements in dictionary D with key k such that $k_1 \leq k \leq k_2$

* It is implemented using a balanced binary search tree T.
* A recursive method 1DTreeRangeSearch is used that takes as arguments the range parameters k_1 and k_2 and a node v in T. If node 'v' is external, the search ends at 1st instance. If node 'v' is internal:

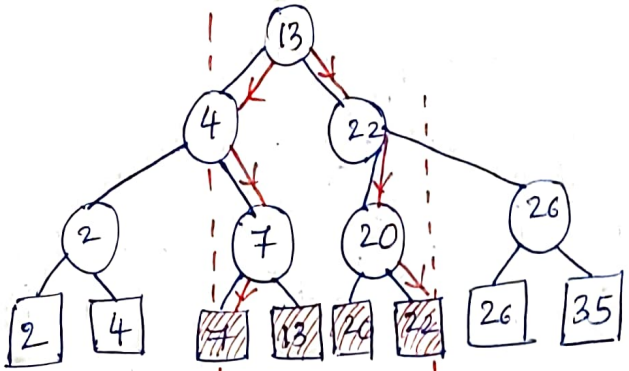
> ① key(v) $\leq k_1$: The right child of 'v' is recursed.
> ② $k_1 \leq$ key(v) $\leq k_2$: The element 'v' is reported and both children of 'v' are recursed.
> ③ key(v) $> k_2$: The right child of v is recursed.

Algorithm:

Algorithm: 1DTreeRangeSearch (k_1, k_2, v):
Input: Search keys k_1, k_2 and a node v of a BST T.
Output: The elements stored in subtree of T rooted at v, whose keys are greater than or equal to k_1 & less than or equal to k_2.

```
if T.isExternal(v) then:
    return ϕ
if k1 ≤ key(v) ≤ k2 then:
    L ← 1DTreeRangeSearch(k1, k2, T.leftChild(v))
    R ← 1DTreeRangeSearch(k1, k2, T.rightChild(v))
    return L ∪ {element(v)} ∪ R
else if key(v) < k1:
    return 1DTreeRangeSearch(k1, k2, T.rightChild(v))
else if k2 < key(v):
    return 1DTreeRangeSearch(k1, k2, T.leftChild(v))
```

NOTE: The values to be retrieved are stored in the range tree's leaf nodes.



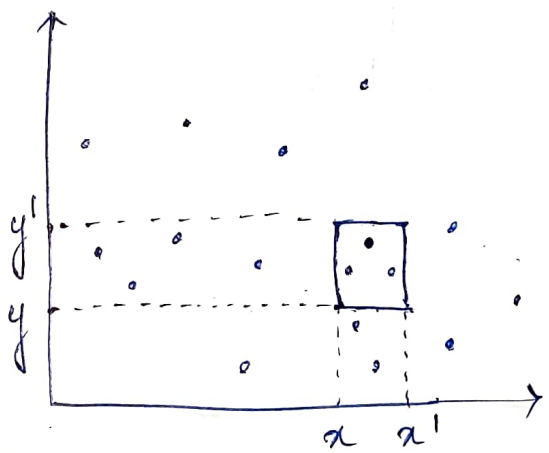
Search Range [6,25] , Report [7, 13, 20, 22]

PERFORMANCE:

* A balanced BST that supports 1-D range searching in an ordered dictionary with n items:
  - The space used is $O(n)$
  - findAllinRange takes $O(\log n + s)$ time. 's' is no. of elements.
  - insertItem & removeElement each take $O(\log n)$ time.

2-D Range Searching:

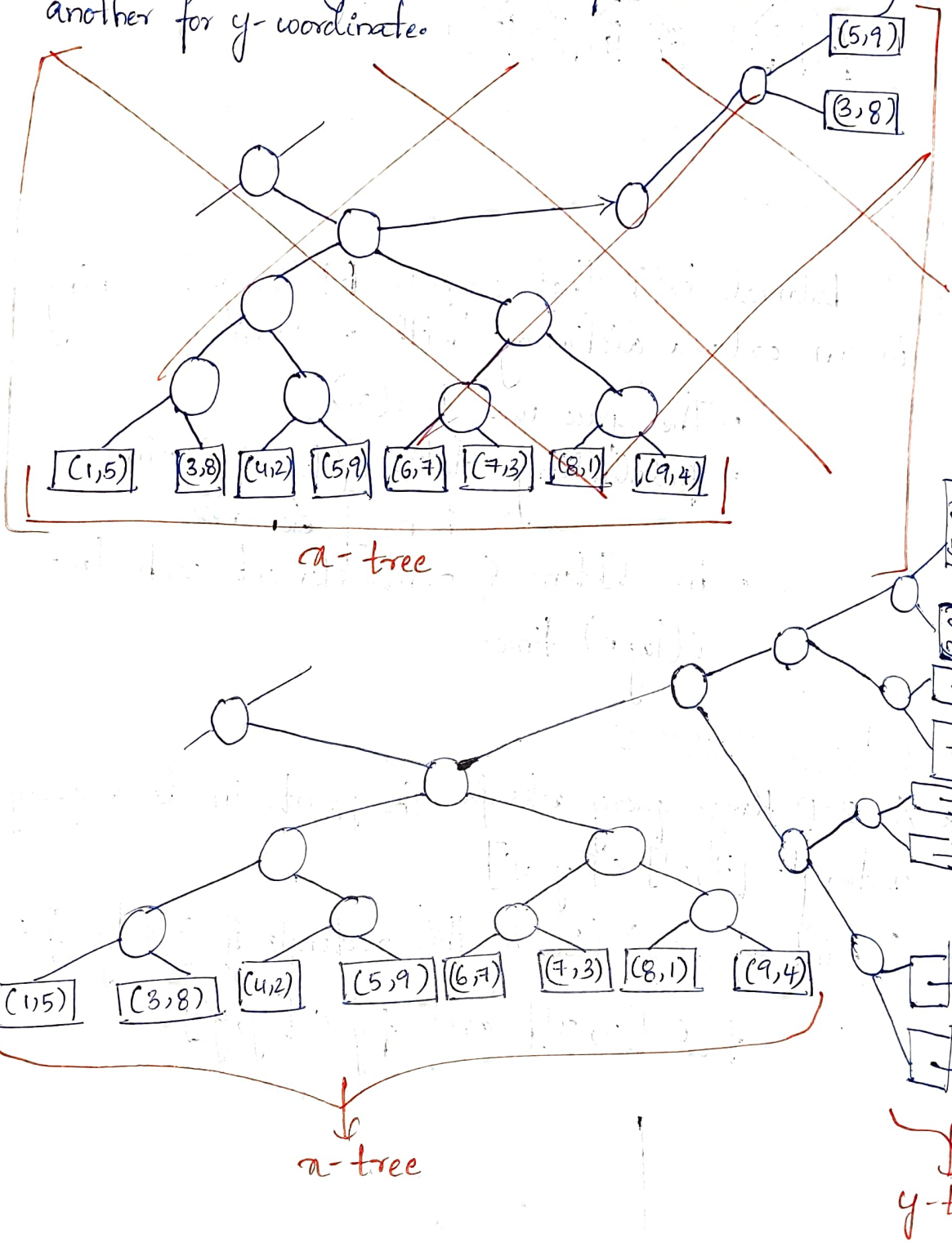
* A 2-D Range query asks for the points inside a query rectangle $[x, x'] \times [y, y']$
* A point (p_x, p_y) lies in this rectangle if & only if: $p_x \in [x, x']$ and $p_y \in [y, y']$



* It is divided into two 1-D range queries.

- One on x-coordinate of points
- Another on y-coordinate of points.

* It has two range trees. One for x-coordinate & another for y-coordinate.



Algorithm: 2DTreeRangeSearch $(x_1, x_2, y_1, y_2, v, t)$:

Input: The co-ordinate points x_1, x_2, y_1, y_2, Node v and the type t of node v.

Output: Items that lie in region enclosed by x_1, x_2, y_1, y_2.

```
if item is out of range tree:
    return ϕ
if x1 ≤ x(v) ≤ x2 then:
    if y1 ≤ y(v) ≤ y2 then:
        M ← {element(v)}
    else
        M ← ϕ
    if t = "left" then:
        L ← 2DTreeRangeSearch(x1, x2, y1, y2, T.leftChild(v),
            "left")
        R ← 1DTreeRangeSearch(y1, y2, T.rightChild(v))
    else if t = "right" then:
        L ← 1DTreeRangeSearch(y1, y2, T.leftChild(v))
        R ← 2DTreeRangeSearch(x1, x2, y1, y2, T.rightChil-
            -d(v), "right")
    else
        {t = "middle"}
        L ← 2DTreeRangeSearch(x1, x2, y1, y2, T.leftChild(v),
            "left")
        R ← 2DTreeRangeSearch(x1, x2, y1, y2, T.rightChild(v),
            "right")
else:
    M ← ϕ
```

if $x(v) < x_1$ then:

$L \leftarrow \phi$

$R \leftarrow$ 2DTreeRangeSearch(x_1, x_2, y_1, y_2, T.rightChild(v), t)

else:

{$x(v) > x_2$}

$L \leftarrow$ 2DTreeRangeSearch(x_1, x_2, y_1, y_2, T.leftChild(v), t)

$R \leftarrow \phi$

return LUMUR

Performance:

Space $\rightarrow O(n \log n)$

Construction $\rightarrow O(n \log n)$

2D Search $\rightarrow O(\log^2 n + (s))$

s = no. of elements to be reported.