FTP_Alg
Kd–Trees (second part) and Range Trees

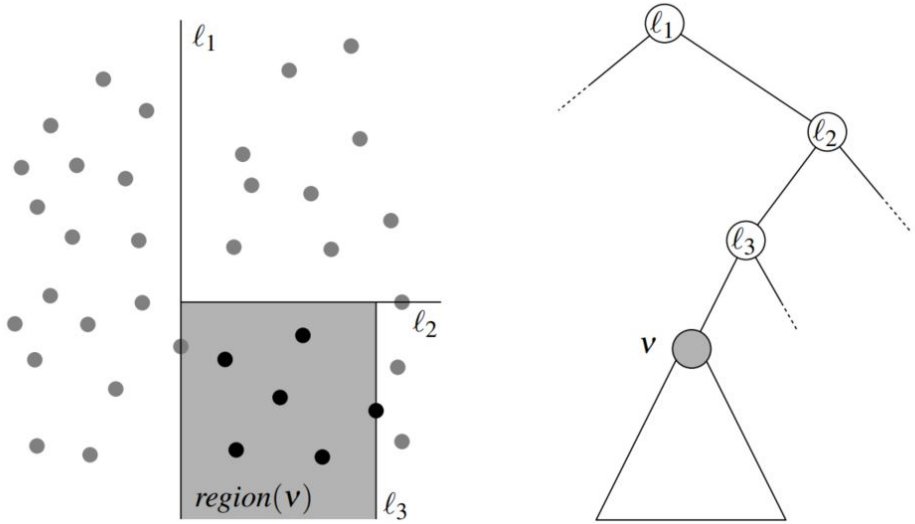
jungkyu.canci@hslu.ch

14. October 2024

Query on KD–Trees

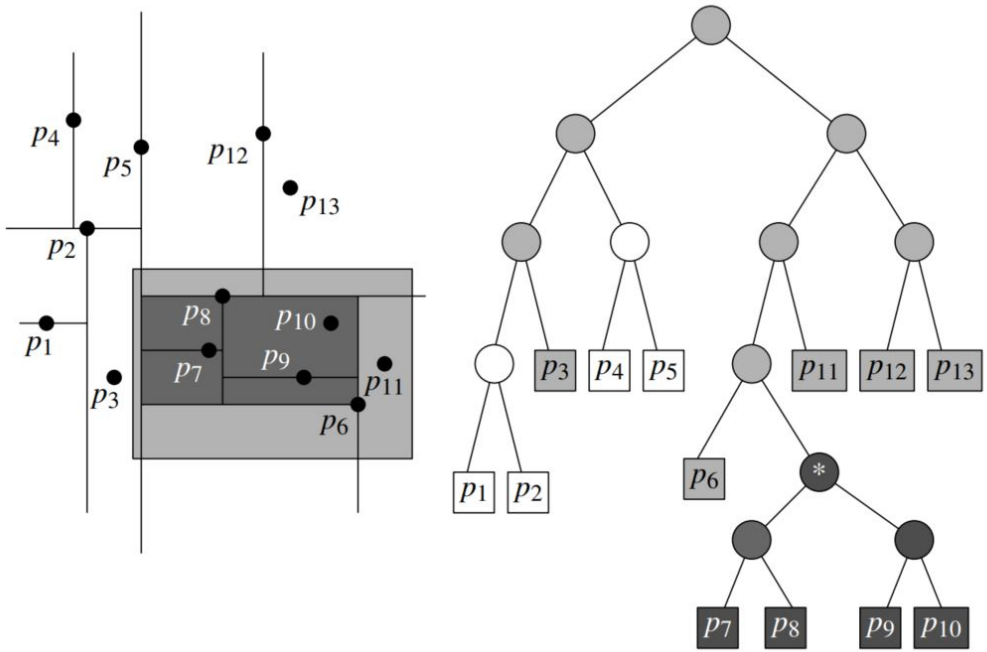
Query on a KD-Tree works similarly to the one in 1-dimensional case, where the intermediate nodes are represented by the splitting lines.

The points in the left (right) half plane are stored in the left (right) subtree. Points on the lines belongs to the left subtree. In the picture the "left" part of an horizontal line is the below part.



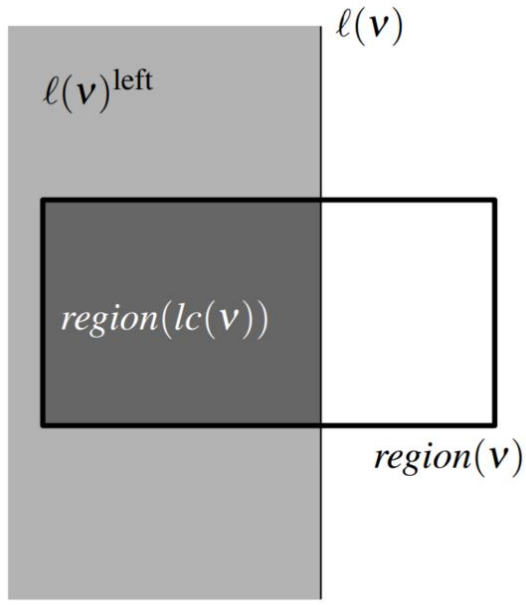
The **region(v)** (in grey in the plane) is the set of all leafs in the subtree rooted in v. We have to consider the subtree rooted at v only if the query rectangle intersects the region v.

- In a query on a KD–Tree we visit only nodes whose region is intersected by the query rectangle (in the picture below the grey nodes).
- When a region is fully contained in the query rectangle we report all leaves in the corresponding subtree (dark grey nodes).
- In the picture, nodes p_3, p_{12} and p_{13} are visited (they are grey) but not reported. All other grey points are visited and reported.



Algorithm SEARCHKDTREE(v, R)
Input. The root of (a subtree of) a kd-tree, and a range R.
Output. All points at leaves below v that lie in the range.

```
1.  if v is a leaf
2.     then Report the point stored at v if it lies in R.
3.     else if region(lc(v)) is fully contained in R
4.             then REPORTSUBTREE(lc(v))
5.             else if region(lc(v)) intersects R
6.                     then SEARCHKDTREE(lc(v), R)
7.          if region(rc(v)) is fully contained in R
8.             then REPORTSUBTREE(rc(v))
9.             else if region(rc(v)) intersects R
10.                    then SEARCHKDTREE(rc(v), R)
```



SEARCHKDTREE uses REPORTSUBTREE(v), which traverses the whole tree rooted in the node and report all the leaves. In the pseudocode "$region(lc(v))$ intersects R", means that the intersection of the set of the points in the leaves of the left subtree of v has non empty intersection with R. In this case we have an iteration of SEARCHKDTREE. Similarly with the right subtree.

Storage and Running for a KD–Tree and Search on it

Theorem. A KD–tree for a set P of n points uses $O(n)$ storage and can be built in $O(n \log n)$ time. A rectuangular range query on the KD—tree takes $O(\sqrt{n} + k)$ time, where k is the number of reported points.

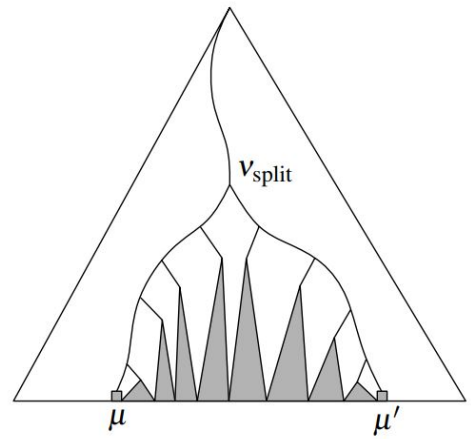
Range Tree (Optional part. It will be not examinated)

- In previous slides we have seen KD–Trees and a query on it, which have $O(\sqrt{n}+k)$ query running time. When k is small compares to n, the running time is not so optimal; it takes long time to give a small report.
- Here we introduce a new data structure, i.e. **Range Trees**, whose query time is $O(k+\log^2 n)$, but they need more storage, which is $O(n \log n)$.
- We are still considering 2–dimensional range query, characterized by a x–coordinate and a y–coordinate.
- We build KD-Tree by splitting the set of points by considering alternatively x–coordinates and y–coordinates.
- In a Range Tree we consider in another manner these two coordinates for the queries.

Construction of a Range Tree

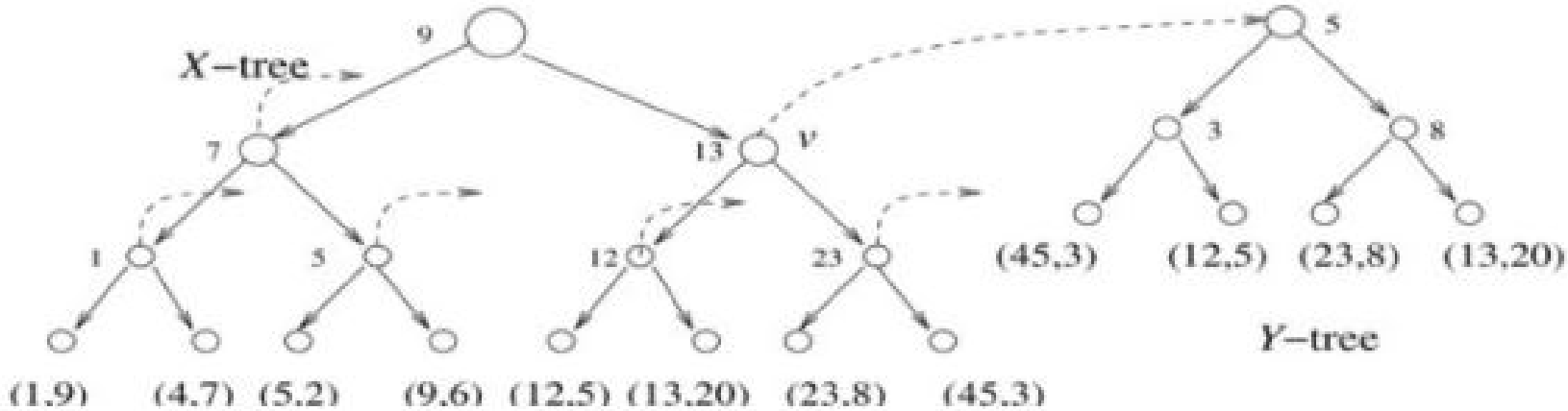
Let P be a set of n points and $[x, x'] \times [y, y']$ be a query range. We assume that the x–coordinates are pairwise distinct and the same holds with the y–coordinates.

We consider only the x–coordinates of the points in P and we build a binary search tree, whose leafs are the points with the above x coordinates. Thus we consider the 1–dimensional range query presented in Lecture 8. Let us denoted by $\mathscr{T}$ this tree.



At each node v of $\mathscr{T}$, the key in v split the set P in two subsets P_{left} (leaves in the left subtree) and P_{right} (leaves in the right subtree).

On $lc(v)$ we attach a binary search three with respect the y–values of the points in P_{left}, similarly for $rc(v)$.
See for example



From this example you can understand that in this way we need more storage than the one for KD–Trees. The figure above present the ideas of Range Tree.

Definition of Range Tree

- Let v be a node on a tree, we denote by $P(v)$ the points on the leaves of the subtree rooted at v. This set is called *canonical subset of* v. For example, the canonical set of the root of a tree is the set of all points stored on its leaves.
- Now we can define a Range Tree for rectangular range query on a set P of n points in the plane:
  - We build a balanced binary search three $\mathscr{T}$, where to create the tree we only consider the x coordinates of the point in P.
  - For each node v in $\mathscr{T}$, the canonical subset $P(v)$ is stored in a balanced binary search three $\mathscr{T}_{assoc}(v)$ with respect the y–coordinates. The pointer to the tree $\mathscr{T}_{assoc}(v)$, stored in v, is called *associate structure* of v.
  - $\mathscr{T}$ is called the *first–level tree* and the associated structures are *second–level trees*

Build Range Tree

Algorithm BUILD2DRANGETREE(P)
Input. A set P of points in the plane.
Output. The root of a 2-dimensional range tree.

1. Construct the associated structure: Build a binary search tree $\mathcal{T}_{\text{assoc}}$ on the set P_y of y-coordinates of the points in P. Store at the leaves of $\mathcal{T}_{\text{assoc}}$ not just the y-coordinate of the points in P_y, but the points themselves.
2. **if** P contains only one point
3. **then** Create a leaf v storing this point, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of v.
4. **else** Split P into two subsets; one subset P_{left} contains the points with x-coordinate less than or equal to x_{mid}, the median x-coordinate, and the other subset P_{right} contains the points with x-coordinate larger than x_{mid}.
5. $v_{\text{left}} \leftarrow$ BUILD2DRANGETREE(P_{left})
6. $v_{\text{right}} \leftarrow$ BUILD2DRANGETREE(P_{right})
7. Create a node v storing x_{mid}, make v_{left} the left child of v, make v_{right} the right child of v, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of v.
8. **return** v

One can prove that a range tree on a set of n points in the plane requires $O(n \log n)$ storage.

Queries on Range Trees

A 2–dimensional range query on a range tree is the iteration of 1–dimensional range query with respect x and y. So we consider a Range Three $\mathscr{T}$ and a rectangle range $[x, x'] \times [y, y']$

- The first step is to determine the subsets, whose union contains the points whose x coordinate is in $[x, x']$. This can be with 1-dimensional query algorithm, which uses FindSplitNode.
- For each above subset we report the points with y–coordinate in $[y, y']$, where we use a 1-dimensional query algorithm again, but now with respect y.

Algorithm 2DRANGEQUERY($\mathcal{T}, [x : x'] \times [y : y']$)
Input. A 2-dimensional range tree $\mathcal{T}$ and a range $[x : x'] \times [y : y']$.
Output. All points in $\mathcal{T}$ that lie in the range.

1. $v_{\text{split}} \leftarrow$FINDSPLITNODE($\mathcal{T}, x, x'$)
2. **if** v_{split} is a leaf
3. **then** Check if the point stored at v_{split} must be reported.
4. **else** ($*$ Follow the path to x and call 1DRANGEQUERY on the subtrees right of the path. $*$)
5. $v \leftarrow lc(v_{\text{split}})$
6. **while** v is not a leaf
7. **do if** $x \leqslant x_v$
8. **then** 1DRANGEQUERY($\mathcal{T}_{\text{assoc}}(rc(v)), [y : y']$)
9. $v \leftarrow lc(v)$
10. **else** $v \leftarrow rc(v)$
11. Check if the point stored at v must be reported.
12. Similarly, follow the path from $rc(v_{\text{split}})$ to x', call 1DRANGEQUERY with the range $[y : y']$ on the associated structures of subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.

Theorem. Let P be a set of n points in a (x, y)–plane. A range tree for P uses $O(n \log n)$ storage and the building time is $O(n \log n)$. The above query algorithm on a rectangular range query has $O(k + \log^2 n)$ running time, where k is the number of reported points in the query.

Composite numbers

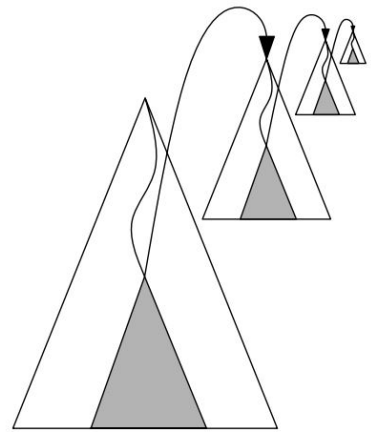
In the previous algorithms we used the fact that we can order (in a total order) the x coordinates of the points in P and the y coordinates of the points in P. Therefore we have a problem, when there are two (distinct) points with same x–coordinates or the same y–coordinates. We solve the problem by using the so called **composite numbers**, that are ordered pairs of (real) numbers written in the form $(a \mid b)$. We use a lexicographic order on them. For real numbers a, a', b, b', we have

$$(a \mid b) < (a' \mid b') \;\Leftrightarrow\; a < a' \text{ or } (a = a' \text{ and } b < b').$$

Now for a point $p = (x, y)$, we consider its representation $\hat{p} = ((x \mid y), (y \mid x))$ in composite numbers. In this way two distinct points in the plane have always distinct first coordinates and distinct second coordinates. Now in the previous data sets we can use the points written with composite numbers.

High–Dimensional Range Trees

The construction of a d–dimensional range three is a straightforward generalization of the one in the 2–dimensional case. Let P be a set of n points in a d –dimensional space. We build a balanced binary search three by using the first coordinates of the points in P.



At each node v we define a pointer to an associate structure, which is a $(d-1)$ dimensional range tree using the canonical subset $P(v)$, and so on...
One needs $O(n \log^{d-1} n)$ storage constructed in $O(n \log^{d-1} n)$ time. The query runs in $O(k + \log^{d})$ (where k is the number of reported points)

Reference

The material of these slides is taken form the book "Computational Geometry" by de Berg et al., Section 5.2, Section 5.3 and Section 5.4