CS 240 — Lecture 15

Bartosz Antczak *Instructor: Eric Schost* *March 2, 2017*

15.1 2-Dimensional Range Search

Each item in a 2-D data structure has two *aspects*: (x_i, y_i). This coordinate corresponds to a point in the Euclidean plane. There are various ways for implementing $d-$dimensional dictionaries:

- Reduce it to a one-dimensional dictionary
- Use several dictionaries: one for each dimension
- Use a *partition tree* (quadtree, kd-tree)
- Use multi-dimensional *range trees*

15.1.1 Quadtrees

We have n points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$. We build a quadtree on P using these steps:

- Find a square R that contains all the points of P. This corresponds to the root of the quadtree
- Now **partition** R into four equal subsquares (quadrants), each correspond to a child of R
- Recursively repeat this process for any node that contains more than one point
- The points that are on the split lines belong to the bottom left side. Also, any leaf not containing a point can be deleted

Example 15.1.1. *Building a sample quadtree*

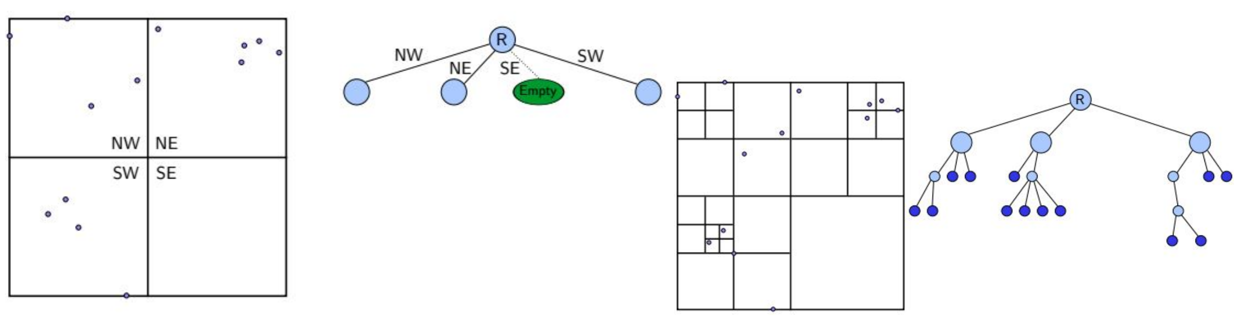


Figure 15.1: Building a quadtree. *Courtesy of the CS 240 lecture slides.*

15.1.2 Quadtree Operations

Search

It's analogous to BST search

Insert

- Search for the point
- Split the leaf if there are two points

Delete

- Search for the point
- Remove the point
- If its parent has only one child left, delete that child and continue the process toward the root

15.1.3 Quadtree Range Search

The algorithm to find a particular range in a quadtree is The complexity of range search is $\Theta(nh)$.

```
QTree-RangeSearch(T, R)
T: A quadtree node, R: Query rectangle
    if (T is a leaf) then
        if (T.point ∈ R) then
            report T.point
    for each child C of T do
        if C.region ∩ R ≠ ∅ then
            QTree-RangeSearch(C, R)
```

Figure 15.2: *Courtesy of the CS 240 lecture slides.*

15.1.4 The Height of a Quadtree

We call d_{max} the max distance between 2 points in P. The **height** of a quadtree is

$$h \in \Theta\left(\log_2\left(\frac{d_{max}}{d_{min}}\right)\right)$$

Proof of the Height of a Quadtree

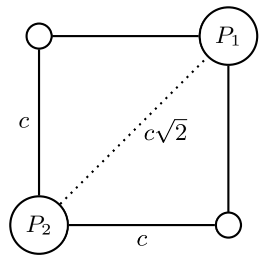
Remarks:

- Rescaling does not change the quadtree
- Rescaling does not change the ratio $\frac{d_{max}}{d_{min}}$

From these remarks, we can assume that after rescaling, the bounding square has side length 1. We rescale to simplify the proof. We call d'_{max} the **max** distance after rescaling, and we call d'_{min} the **min** distance after rescaling. From our previous remarks, observe that:

$$\frac{d'_{max}}{d'_{min}} = \frac{d_{max}}{d_{min}}$$

In any square S of side length c, the max distance between 2 points, P_1, P_2, is $c\sqrt{2}$



Note that $1 \leq d'_{max} \leq \sqrt{2}$.
After 1 level of subdivision, the squares have side length $\frac{1}{2}$. After 2 levels, it's $\frac{1}{2^2}$, and so on. In general, after h levels, the square has side length $\frac{1}{2^h}$. We stop once every square has only one point.
Now, if $\frac{\sqrt{2}}{2^i} < d'_{min}$, we must stop. If we have two points in this square, their distance is at most $\frac{\sqrt{2}}{2^i}$. This cannot happen, which is why we stop. So the height h of the tree is at most the 1st integer h such that

$$2^h \geq \frac{\sqrt{2}}{d'_{min}}$$

So the height is in $O\left(\log\left(\frac{\sqrt{2}}{d'_{min}}\right)\right)$. Observe that:

$$\begin{aligned} O\left(\log\left(\frac{\sqrt{2}}{d'_{min}}\right)\right) &= O\left(\log\left(\frac{1}{d'_{min}}\right)\right) \\ &= O\left(\log\left(\frac{d'_{max}}{d'_{min}}\right)\right) \qquad (\text{Since} 1 \leq d'_{max} \leq \sqrt{2}) \quad \square \end{aligned}$$

15.1.5 KD-trees

We have n points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$. Unlike quadtrees, which splits a square into quadrants regardless of where points actually lie, a kd-tree splits the points into two (roughly) equal subsets. We build a kd-tree using these steps:

- Split p into two equal subsets using a vertical line
- Split each of the two subsets into two equal pieces using horizontal lines
- Continue splitting, alternating vertical and horizontal lines, until every point is in a separate region
- **Notes:**
  - We initially sort the n points according to their $x-$coordinate

- The root of the tree is the point with median x coordinate (its index is $\lfloor \frac{n}{2} \rfloor$ in the sorted list)
- All other points with x coordinate to the left of the point are smaller or equal in value; the points to the right are larger (exactly how a tree should be structured)

These steps have a complexity of $\Theta(n \log n)$, and its height is $\Theta(\log n)$

Example 15.1.2. *Building a sample kd-tree on 10 points*

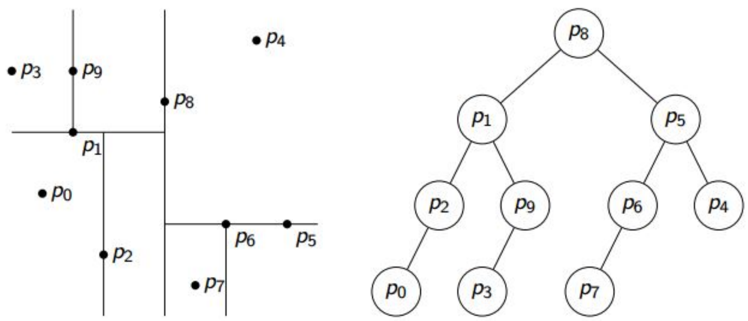


Figure 15.3: *Courtesy of the CS 240 lecture slides.*

Let $T(n)$ be the runtime for creating a kd-tree. Observe that:

$$T(n) = O(n \log n) + T'(n)$$

It's broken down into sorting $(n \log n)$ and the recursion (T').

$$T'(n) = 2T'\left(\frac{n}{2}\right) + O(n)$$

We see that $T'(n) = O(n \log n)$. So the runtime becomes:

$$T(n) = O(n \log n) + O(n \log n) \in O(n \log n)$$

15.1.6 KD Range Search

The algorithm to find a particular range in the kd-tree is: *More on this algorithm next lecture.*

```
kd-rangeSearch(T, R)
T: A kd-tree node, R: Query rectangle
    if T is empty then return
    if T.point ∈ R then
        report T.point
    for each child C of T do
        if C.region ∩ R ≠ ∅ then
            kd-rangeSearch(C, R)
```

Figure 15.4: *Courtesy of the CS 240 lecture slides.*