DCG, Wintersemester 2024/25 Lecture 17 (6 pages)

k-Nearest-Neighbor Searching with Quadtrees

Anne Driemel and Herman Haverkort updated: December 11, 2024

Given a set P of n points in $\mathbb{R}^d$, the k-nearest-neighbours (kNN) query with a point q (not necessarily in P) asks for the k points of P that are closest to q. How can we preprocess the set P to be able to answer kNN-queries efficiently? We have discussed order-k Voronoi diagrams and their representation by the set of points of level at most k in an arrangement of hyperplanes in $\mathbb{R}^{d+1}$. We could build a data structure that can report the first k planes hit by a vertical ray, and use this to answer kNN queries. However, if $d \geq 3$, the complexity of the k-th level of the arrangement of hyperplanes in $d+1$ dimensions can be quadratic in n or worse. This makes it impossible (or at least challenging) to build a data structure of acceptable size.

In the next two lectures we will discuss quadtrees as a possible solution to this problem. With quadtrees we take a different approach: they constitute a simple linear-size data structure that may be inefficient in the worst case, but supports provably efficient queries under well-defined conditions that are often reasonable. Quadtrees for $d = 3$ are also called *octrees*, but we simply use the name quadtrees also in higher dimensions.

1 Organizing points in a quadtree

A quadtree is a hierarchical decomposition, which can be represented as a tree structure, where each node corresponds to a d-dimensional cube. The root corresponds to a cube that contains all points of the set P that is to be stored. Other nodes in the tree correspond to smaller cubes. For a node u in the quadtree, we use $\text{Cube}(u)$ to denote the cube associated with u, and we use $\text{children}(u)$ to denote the children of u. The cubes associated with the nodes are properly nested according to their parent-child relations: when v is a descendant of u, we have $\text{Cube}(v) \subseteq \text{Cube}(u)$.

Definition 17.1 (Canonical cubes)**.** *We consider a cube C to be the Cartesian product of a set of equally large intervals $[x_1^-, x_1^+) \times [x_2^-, x_2^+) \times ... \times [x_d^-, x_d^+)$. So cubes are relatively closed on the low side and relatively open on the high side. The unit cube is a cube of diameter 1. The edges of the unit cube thus have length $1/\sqrt{d}$. A canonical cube is a cube with diameter $1/2^i$ for some natural number i, and is one of the cubes of the regular grid of 2^{di} cubes that fills the unit cube.*

The standard quadtree is defined as follows.

Definition 17.2 (Quadtree)**.** *Let P be a set of n distinct points in $\mathbb{R}^d$ and let C be a canonical cube that contains P. The root u of the quadtree $\mathcal{Q}(P, C)$ has $\text{Cube}(u) = C$. If $|P| = 1$, we store the single point of P with u. If and only if $|P| > 1$, that is, if P consists of more than one point, the root has children. Let D be 2^d. For a cube C of diameter z, let $C_1, C_2, ..., C_D$ be the cubes of diameter $z/2$ that form a regular grid that fills C. The children of the root are the roots of the quadtrees $\mathcal{Q}(P \cap C_i, C_i)$ for $i = 1, 2, ..., D$.*

The number of nodes of a standard quadtree cannot be bounded as a function of only n, the number of points in P. This is because it may happen that all points of P lie in the same subcube of C. In particular, the nodes at depth i in the tree correspond to cubes of diameter $1/2^i$. Therefore, if two points of P lie at distance δ from each other, they may still end up lying in the same cube at depth i as long as $1/2^i \geq \delta$, that is, $i \leq \log_2(1/\delta)$. It may thus take

$\lfloor \log_2(1/\delta) + 1 \rfloor$ levels of subdivision until the two points end up lying in different branches of the tree. As a result, a quadtree can have depth $\lfloor \log_2(1/\delta) + 1 \rfloor$, where δ is the distance between the points of P that lie closest to each other. This means that neither the depth of a quadtree, nor its total size, can be bounded by a function of only the number of points in P.

To solve the problem of unbounded storage space, we contract certain paths in the tree. In particular, any maximal path from a node u to a descendant v on which the associated set of points does not change (that is, $P \cap \text{Cube}(u) = P \cap \text{Cube}(v)$) is contracted into a single node. We may also remove leaves that do not correspond to any points of P.

The compressed quadtree is defined as follows.

Definition 17.3 (Compressed quadtree)**.** *We define the compressed quadtree $\mathcal{Q}_c(P, C)$ for a set of n distinct points P and a cube C more precisely as follows. The cube* $\text{Cube}(u)$ *associated with the root u of $\mathcal{Q}_c(P, C)$ is the infimum of canonical cubes that contain P (so if P is a single point p, then* $\text{Cube}(u) = p$*, otherwise* $\text{Cube}(u)$ *is the smallest canonical cube that contains P). If and only if $|P| > 1$, the root has children, namely the roots of the compressed quadtrees $\mathcal{Q}_c(P \cap \text{Cube}(u)_i, \text{Cube}(u)_i)$ for all $i \in \{1, 2, ..., D\}$ such that $P \cap \text{Cube}(u)_i \neq \emptyset$.*

Note that with this definition, every internal node of the compressed quadtree has at least two children. Indeed, if a node u would have only one child v whose cube $\text{Cube}(v)$ contains points of P, then $\text{Cube}(u)$ would not be the smallest canonical cube that contains P. Furthermore, each leaf v of the compressed quadtree corresponds to a single point of P, which is recorded as $\text{Cube}(v)$.

Procedure CompressedQtree(set of points P, a cube C that contains P)

Output: the root node of the quadtree $\mathcal{Q}(P, C)$

```
1  Create a node u and set:
2  Cube(u) ← C; children(u) ← ∅
3  if |P| = 1 then
4      point(u) ← the point in P
5  if |P| > 1 then
6      Sort P into the sets P_1, ..., P_{2^d} of points inside canonical cubes C_1, ..., C_{2^d}
7      if there is an i such that P = P_i then return CompressedQtree(P_i, C_i)
8      for i = 1, ... 2^d do
9          if P_i ≠ ∅ then
10             v ← CompressedQtree(P_i, C_i); insert v in children(u)
11 return u
```

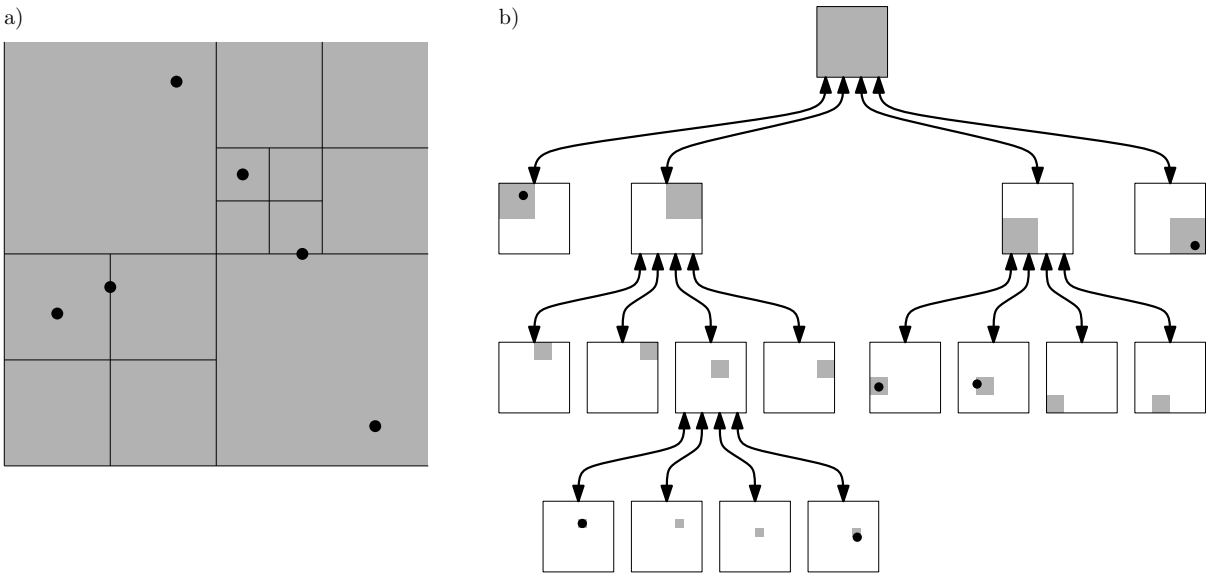


Figure 1: An example of a quadtree.

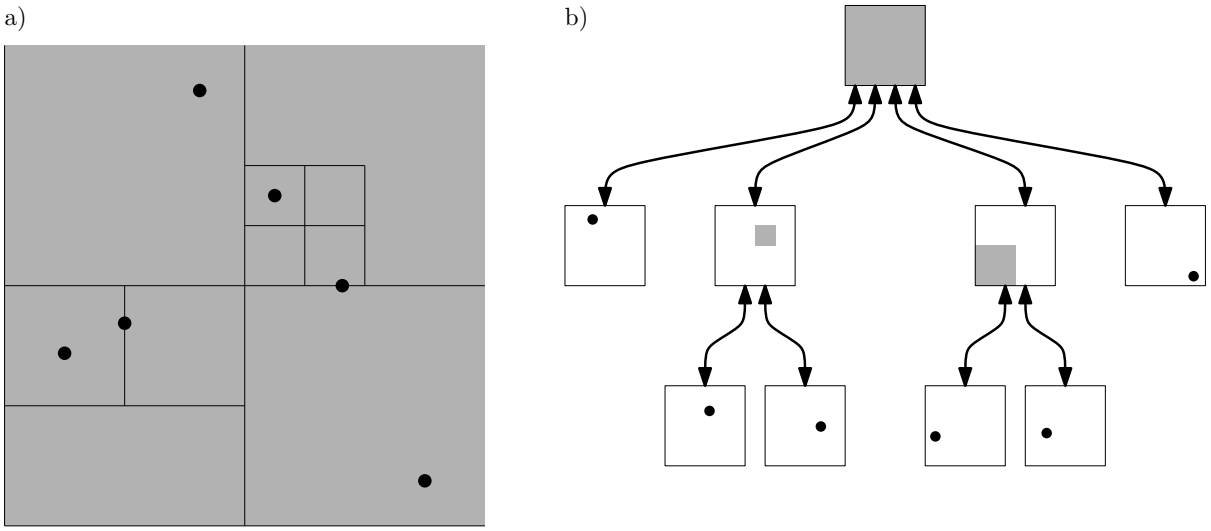


Figure 2: An example of a compressed quadtree—compare this to Figure 1.

2 Size and depth of a quadtree

Theorem 17.4. *A compressed quadtree storing $n \geq 1$ points consists of at most $2n-1$ nodes.*

Proof. The proof is by induction on the number of points stored in a subtree. We claim that if a subtree stores n points, then it consists of at most $2n-1$ nodes. The base case is $n = 1$, in this case the tree contains one node. Now assume $n > 1$. In this case the algorithm creates a root node u and with at least 2 children. Let $n_1, .., n_t$ be the number of points stored in each of the subtrees rooted at a child of u (the ones that received a non-empty set of points during construction). We have $n = \sum_{i=1}^{t} n_i$ and $t \geq 2$. By induction, the number of nodes in the ith non-empty subtree is at most $2n_i - 1$, if $n_i \geq 1$. In total we can thus bound the number of nodes stored at the root by $\sum_{i=1}^{t}(2n_i - 1) + 1 = 2\sum_{i=1}^{t} n_i - t + 1 \leq 2n - 1$ □

If the precision of the coordinates of the points in P is bounded and the numbers are encoded with fixed precision, then the height of the standard quadtree is also bounded. For example, if the coordinates are h-bit integers, then, after h levels of subdivision, distinct points will always lie in different cubes. This limits the number of nodes in the quadtree to $O(hn)$, where n is the number of points stored in tree. (Of course, a factor h overhead in storage can still be prohibitive.)

In the Real-RAM model we do not bound the precision of the coordinates (see also the discussion in Lecture 4). However, we can bound the height of the quadtree if the point set satisfies certain properties. This property is captured in the following definition.

Definition 17.5 (Spread)**.** *For a finite set of points P in $\mathbb{R}^d$, we define the spread as*

$$\phi(P) = \frac{\max_{p,q \in P} \|p - q\|}{\min_{p,q \in P} \|p - q\|}$$

Lemma 17.6. *The depth of a compressed quadtree for a set of point P is in $O(\log(\phi(P)))$.*

Proof. In each step from a node to one of its children, the diameter of the corresponding cube halves. Thus, the diameter of the cube at depth i is $\frac{\delta}{2^i}$, where δ is the diameter of the root node. The maximum distance between two points in the same cube is bounded by the diameter of the cube. Any internal node of the quadtree must contain at least two different points of P. Therefore the following holds true for any internal node at depth i

$$\frac{\delta}{2^i} \geq \min_{p,q \in P} \|p - q\|$$

Since, by definition, δ is the diameter of the smallest canonical cube that contains the set P, we have

$$\max_{p,q \in P} \|p - q\| \leq \delta \leq 2 \max_{p,q \in P} \|p - q\|$$

Here, we assume that the point set P fits into a canonical cube of appropriate size, which can be realized by applying a translation in the beginning. Putting the above together we get

$$2^i \leq 2\phi(P) \implies i \leq \log_2(\phi(P)) - 1$$

and this implies that the length of any root-to-leaf path is bounded by $O(\log(\phi(P)))$. □

3 Answering nearest-neighbour queries with quadtrees

We consider k-nearest neighbour queries for $k = 1$. Given a query point q, we want to report a point of P closest to q.

Definition 17.7. *The closest-point distance* $\mathrm{CPD}(C, q)$ *between a cube* C *and a point* q *is*

$$\mathrm{CPD}(C, q) := \min_{x \in C} ||x - q||.$$

This is the distance between q *and the point of* C *that is closest to* q*. When* q *is fixed and* u *is a node of a (compressed) quadtree, we may use* $\mathrm{CPD}(u)$ *as a shorthand for* $\mathrm{CPD}(\mathrm{Cube}(u), q)$*.*

Note that, when C contains q, then $\mathrm{CPD}(C, q) = 0$.

To find the point of P that lies closest to a query point q, we can search the cubes stored in the compressed quadtree, check all points of P (which we find in the leaves of the tree), and keep the closest point. But do we really need to check all points? Note that searching far-away cubes cannot help us exclude points in nearby cubes from consideration, but finding points in nearby cubes can help us exclude far-away cubes from consideration. In particular, thanks to the nesting structure of the cubes stored in the tree, we can use the following rule: as soon as we find a point $p \in P$ such that $||p - q|| \leq \mathrm{CPD}(u)$, we know that the entire subtree rooted at u cannot contain any points of P that are closer to q than p is. Therefore, the key to obtaining an efficient algorithm is to search the cubes of the quadtree in order of increasing distance from q. As soon as the closest unsearched cube is farther away from q than the nearest point $p \in P$ found so far, we can be sure that no better answer than p can be found, and we can report p as the point closest to q.

Recall the following definition of a priority queue.

Definition 17.8 (Priority-Queue)**.** *A priority queue stores a set of (key,object) pairs and supports the following operations.*
(i) insert an object together with its key;
(ii) extract the object that has the smallest key.
The keys do not necessarily have to be unique.

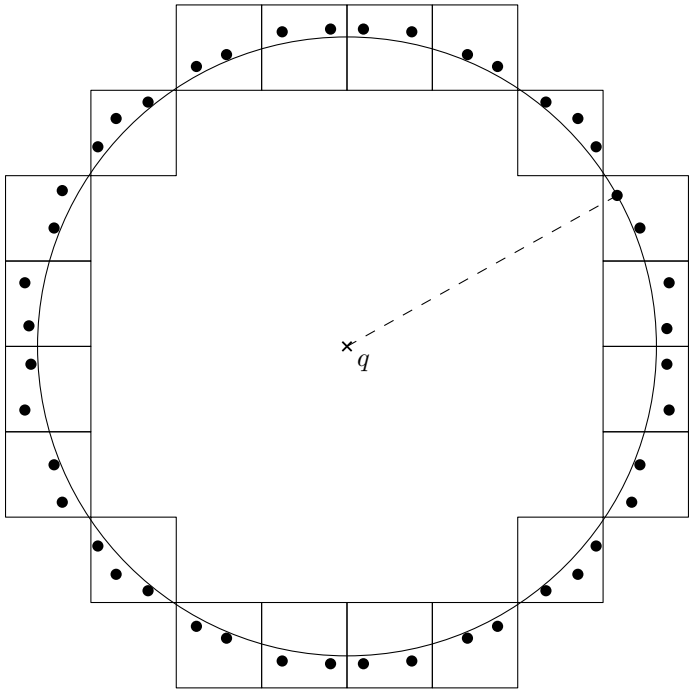
The idea is now to maintain a priority queue of subtrees to be explored, representing subtrees by their roots, and using the distances between the corresponding cubes and q as keys. As soon as a leaf (whose key is equal to the distance to the corresponding point of P) ends up at the head of the queue, we can report the corresponding point. We get the following algorithm:

Procedure FindClosestPoint(the root u of a compressed quadtree, a query point q)

Output: a point that is stored in the quadtree and is closest to q

```
1 Initialize an empty priority queue Q
2 while u is not a leaf do
3 |   foreach node v in children(u) do
4 |   |   enqueue v in Q with key CPD(v)
5 |   let u be the node with smallest key extracted from Q
6 return point(u)
```

It is hard to give a meaningful worst-case bound on the query time for nearest-neighbour queries, even if the height of the tree is bounded.



Example 17.9. *Suppose P is a set of two-dimensional points, one point p of P lies on a circle centered at q, and all other points of P lie just outside that circle. Then the squares associated with the internal nodes of the compressed quadtree all intersect the circle, and they are all inserted into the queue before we report p. The resulting query time would be linear in n and therefore not be better than a simple scan of the complete set P.*

In the next lecture we will see an alternative query strategy with provable bounds.