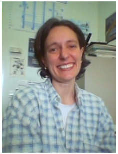
Rank Selection in Multidimensional Data

Conrado Martínez
Univ. Politècnica Catalunya
Journées ALÉA, CIRM, Marseille-Luminy, March 2010

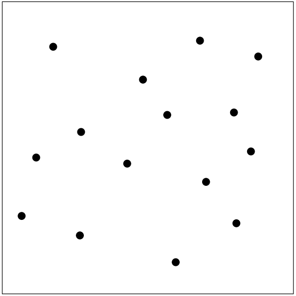
Joint work with:


A. Duch


R.M. Jiménez

Introduction

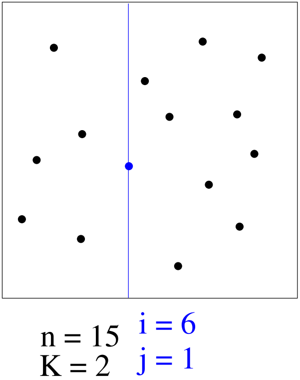
The problem: Given a collection of n multidimensional records, each with K coordinates, and values i, $1 \leqslant i \leqslant n$, and j, $1 \leqslant j \leqslant K$, find the i-th record along the j-th coordinate



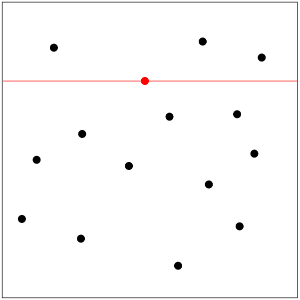
$n = 15$
$K = 2$

Introduction

The problem: Given a collection of n multidimensional records, each with K coordinates, and values i, $1 \leqslant i \leqslant n$, and j, $1 \leqslant j \leqslant K$, find the i-th record along the j-th coordinate

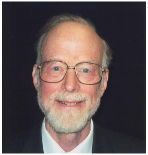


The problem: Given a collection of n multidimensional records, each with K coordinates, and values i, $1 \leqslant i \leqslant n$, and j, $1 \leqslant j \leqslant K$, find the i-th record along the j-th coordinate



n = 15
K = 2
i = 12
j = 2

Introduction



C.A.R. Hoare

R. Floyd

R. Rivest

Easy solution: use an efficient selection algorithm, with (expected) linear cost, e.g., using Hoare's Quickselect or Floyd and Rivest's algorithm for selection

Introduction

- What if the collection is organized in some multidimensional index? (e.g., a K-d tree, a quadtree, ...)
- If $K = 1$ and the collection of n records is stored in some kind of binary search tree $\Rightarrow$ (expected) time $\Theta(\log n)$, using some little extra space
- We look for an algorithm that uses space $\Theta(n)$, independent of K
- The data structure for the n records should efficiently support usual spatial queries, e.g., orthogonal range search
- We assume w.l.o.g. the n records are points from $[0, 1]^K$

Introduction

- What if the collection is organized in some multidimensional index? (e.g., a K-d tree, a quadtree, ...)
- If $K = 1$ and the collection of n records is stored in some kind of binary search tree $\Rightarrow$ (expected) time $\Theta(\log n)$, using some little extra space
- We look for an algorithm that uses space $\Theta(n)$, independent of K
- The data structure for the n records should efficiently support usual spatial queries, e.g., orthogonal range search
- We assume w.l.o.g. the n records are points from $[0, 1]^K$

Introduction

- What if the collection is organized in some multidimensional index? (e.g., a K-d tree, a quadtree, ...)
- If $K = 1$ and the collection of n records is stored in some kind of binary search tree $\Rightarrow$ (expected) time $\Theta(\log n)$, using some little extra space
- We look for an algorithm that uses space $\Theta(n)$, independent of K
- The data structure for the n records should efficiently support usual spatial queries, e.g., orthogonal range search
- We assume w.l.o.g. the n records are points from $[0, 1]^K$

Introduction

- What if the collection is organized in some multidimensional index? (e.g., a K-d tree, a quadtree, ...)
- If $K = 1$ and the collection of n records is stored in some kind of binary search tree $\Rightarrow$ (expected) time $\Theta(\log n)$, using some little extra space
- We look for an algorithm that uses space $\Theta(n)$, independent of K
- The data structure for the n records should efficiently support usual spatial queries, e.g., orthogonal range search
- We assume w.l.o.g. the n records are points from $[0, 1]^K$

Introduction

- What if the collection is organized in some multidimensional index? (e.g., a K-d tree, a quadtree, ...)
- If $K = 1$ and the collection of n records is stored in some kind of binary search tree $\Rightarrow$ (expected) time $\Theta(\log n)$, using some little extra space
- We look for an algorithm that uses space $\Theta(n)$, independent of K
- The data structure for the n records should efficiently support usual spatial queries, e.g., orthogonal range search
- We assume w.l.o.g. the n records are points from $[0, 1]^K$

K-d trees

J.L. Bentley

Definition

A K-d tree for a set $X \subset [0,1]^K$ is either the empty tree if $X = \emptyset$ or a binary tree where:

- the root contains $y \in X$ and some value j, $1 \leqslant j \leqslant K$
- the left subtree is a K-d tree for $X^- = \{x \in X \,|\, x_j < y_j\}$
- the right subtree is a K-d tree for $X^+ = \{x \in X \,|\, y_j < x_j\}$

K-d trees

- In standard K-d trees, discriminants (the values j) of the nodes are cyclically assigned by level: the root has $j = 1$, the nodes in next level have $j = 2, \ldots$, nodes at level K have $j = K$, then at level $K + 1$ all nodes have $j = 1$, etc.
- In relaxed K-d trees discriminants are assigned uniformly at random
- In squarish K-d trees discriminants are assigned to divide the region corresponding to each node as evenly as possible

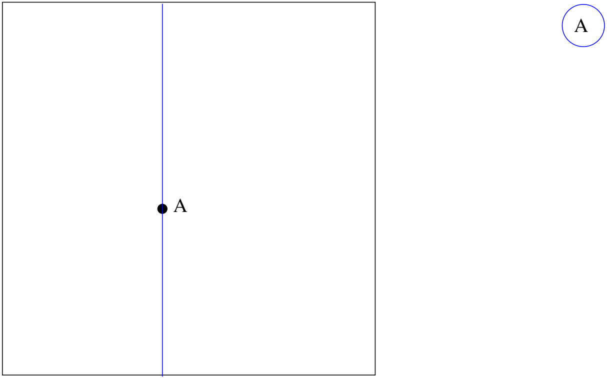
K-d trees

- In standard K-d trees, discriminants (the values j) of the nodes are cyclically assigned by level: the root has $j = 1$, the nodes in next level have $j = 2, \ldots$, nodes at level K have $j = K$, then at level $K + 1$ all nodes have $j = 1$, etc.
- In relaxed K-d trees discriminants are assigned uniformly at random
- In squarish K-d trees discriminants are assigned to divide the region corresponding to each node as evenly as possible

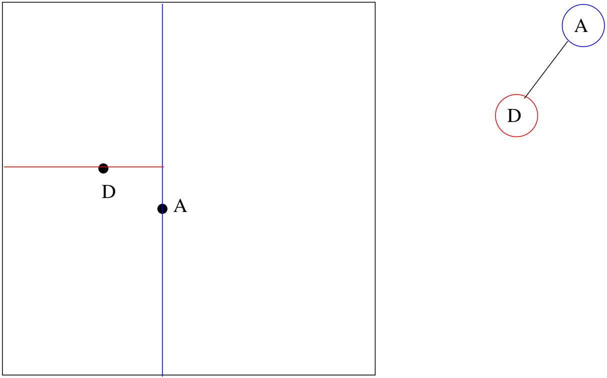
K-d trees

- In standard K-d trees, discriminants (the values j) of the nodes are cyclically assigned by level: the root has $j = 1$, the nodes in next level have $j = 2, \ldots,$ nodes at level K have $j = K$, then at level $K + 1$ all nodes have $j = 1$, etc.
- In relaxed K-d trees discriminants are assigned uniformly at random
- In squarish K-d trees discriminants are assigned to divide the region corresponding to each node as evenly as possible

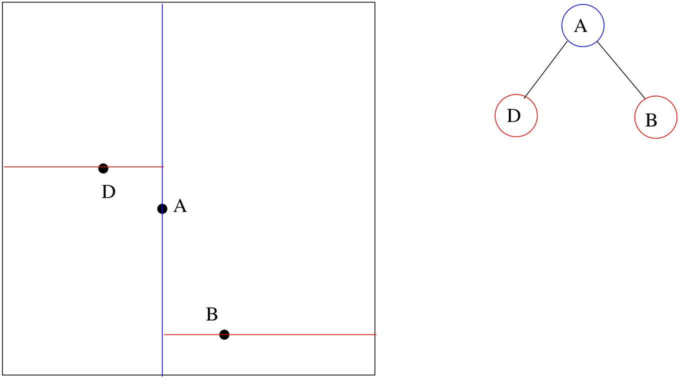
K-d trees



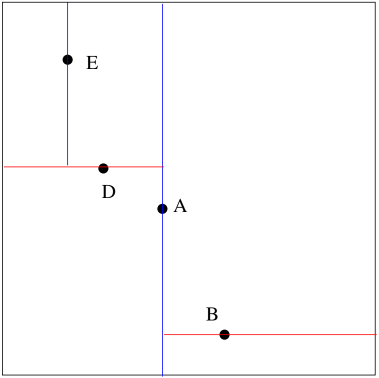
K-d trees

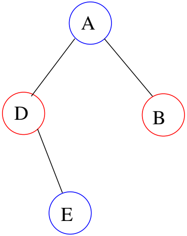


K-d trees

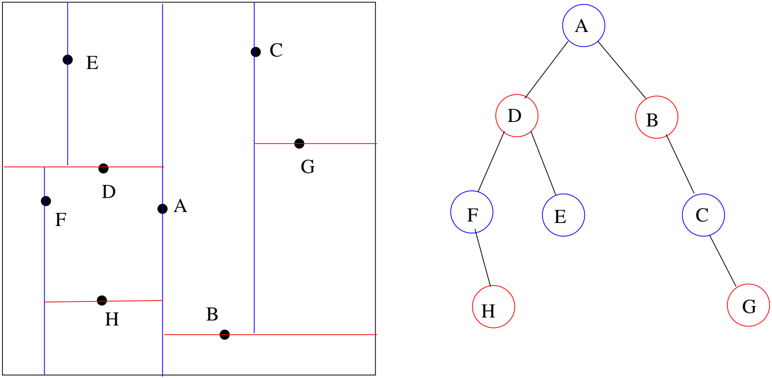


K-d trees





K-d trees

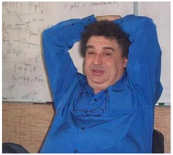


K-d trees

- In a partial match query we are given a query $q = (q_1, \ldots, q_K)$ where s coordinates are specified and $K - s$ are “don’t cares”
- The goal is to find all records in a collection that satisfy the query
- Flajolet and Puech (1986) showed that a partial match in a random standard K-d tree of size n has expected cost $\Theta(n^{\alpha(s/K)})$, where $\alpha(x) = 1 - x + \phi(x)$, $0 \leqslant \phi(x) < 0.07$
- Similar results have been proved for other variants of K-d trees, quadtrees, etc.

- In a partial match query we are given a query $q = (q_1, \ldots, q_K)$ where s coordinates are specified and $K - s$ are "don't cares"
- The goal is to find all records in a collection that satisfy the query
- Flajolet and Puech (1986) showed that a partial match in a random standard K-d tree of size n has expected cost $\Theta(n^{\alpha(s/K)})$, where $\alpha(x) = 1 - x + \phi(x)$, $0 \leqslant \phi(x) < 0.07$
- Similar results have been proved for other variants of K-d trees, quadtrees, etc.

K-d trees

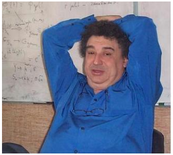


Ph. Flajolet

C. Puech

- In a partial match query we are given a query $q = (q_1, \ldots, q_K)$ where s coordinates are specified and $K - s$ are “don’t cares”
- The goal is to find all records in a collection that satisfy the query
- Flajolet and Puech (1986) showed that a partial match in a random standard K-d tree of size n has expected cost $\Theta(n^{\alpha(s/K)})$, where $\alpha(x) = 1 - x + \phi(x)$, $0 \leqslant \phi(x) < 0.07$
- Similar results have been proved for other variants of K-d trees, quadtrees, etc.

K-d trees



Ph. Flajolet

C. Puech

- In a partial match query we are given a query $q = (q_1, \dots, q_K)$ where s coordinates are specified and $K - s$ are “don’t cares”
- The goal is to find all records in a collection that satisfy the query
- Flajolet and Puech (1986) showed that a partial match in a random standard K-d tree of size n has expected cost $\Theta(n^{\alpha(s/K)})$, where $\alpha(x) = 1 - x + \phi(x)$, $0 \leqslant \phi(x) < 0.07$
- Similar results have been proved for other variants of K-d trees, quadtrees, etc.

K-d trees

L. Devroye

- Orthogonal range queries ask for all records falling inside an hyperrectangle (with sides parallel to the axis); their expected cost has been analyzed by Chanzy, Devroye and Zamora-Cura (2001) and Duch and Martínez (2002):

$$n \cdot \text{volume of query} + n^{\alpha(1/K)} \cdot \text{perimeter of query} + \text{l.o.t.}$$

The algorithm

Our algorithm has three main steps

- The main loop starts with a strip $x_j \in [low, high] = [0, 1]$ and explores the K-d tree, reducing the strip in such a way that it always contains the i-th record along coordinate j
- When the main loop finishes, it has found the sought element (if it is stored in a node that discriminates w.r.t. j) or the strip does only contain nodes discriminating w.r.t. a coordinate $\neq j$; if needed, the second step performs an orthogonal range search to locate all records within the strip
- A conventional selection algorithm is used to find the sought element among the elements reported in the previous step

The algorithm

Our algorithm has three main steps

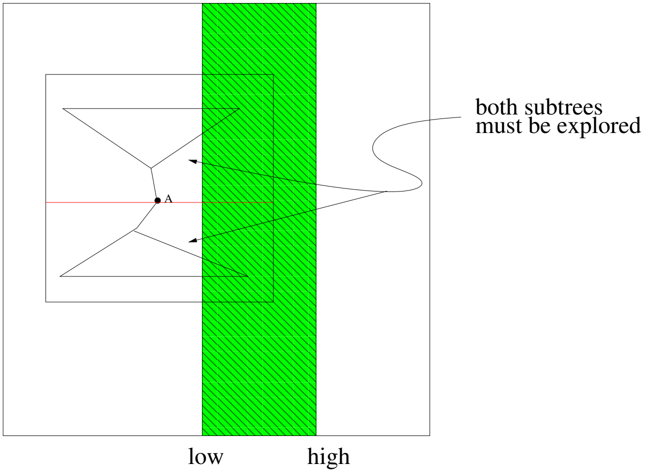
- The main loop starts with a strip $x_j \in [low, high] = [0, 1]$ and explores the K-d tree, reducing the strip in such a way that it always contains the i-th record along coordinate j
- When the main loop finishes, it has found the sought element (if it is stored in a node that discriminates w.r.t. j) or the strip does only contain nodes discriminating w.r.t. a coordinate $\neq j$; if needed, the second step performs an orthogonal range search to locate all records within the strip
- A conventional selection algorithm is used to find the sought element among the elements reported in the previous step

The algorithm

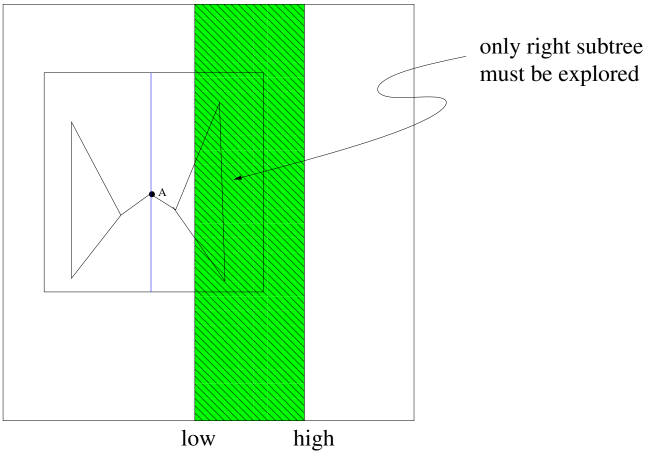
Our algorithm has three main steps

- The main loop starts with a strip $x_j \in [low, high] = [0, 1]$ and explores the K-d tree, reducing the strip in such a way that it always contains the i-th record along coordinate j
- When the main loop finishes, it has found the sought element (if it is stored in a node that discriminates w.r.t. j) or the strip does only contain nodes discriminating w.r.t. a coordinate $\neq j$; if needed, the second step performs an orthogonal range search to locate all records within the strip
- A conventional selection algorithm is used to find the sought element among the elements reported in the previous step

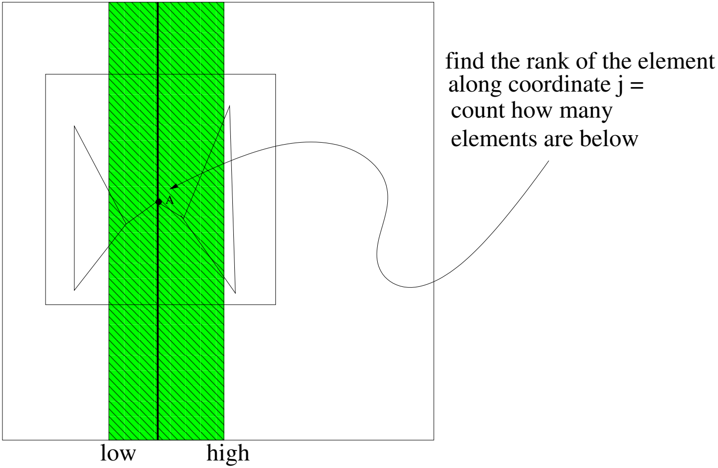
The algorithm: main loop



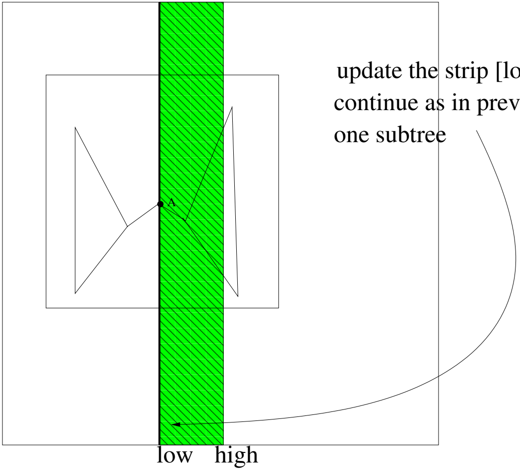
The algorithm: main loop



The algorithm: main loop



The algorithm: main loop



The algorithm

```
procedure KD-SELECT(T, i, j)
    Q.PUSH(T)
    low ← 0; high ← 1
    found ← false
    while ¬Q.EMPTY() ∧ ¬found do
        t ← Q.POP()
        if t.discr ≠ j then
            Q.PUSH(t.left); Q.PUSH(t.right)
        else
            . . . next slide . . .
    ▷ found = true or the “strip” [low, high] contains
    ▷ the i-th record along coordinate j
    . . .
```

The algorithm

while $\neg$Q.EMPTY() $\wedge \neg \mathrm{found}$ **do**
    $t \leftarrow$ Q.POP()
    if $t.\mathrm{discr} \neq j$ **then** ...
    else ▷ $t.\mathrm{discr} = j$
        $z \leftarrow t.\mathrm{key}[j]$
        if $z \in [\mathrm{low}, \mathrm{high}]$ **then**
            ▷ BELOW returns the number of points x in T such that $x_j \leqslant z$
            $r \leftarrow$ BELOW(T, j, z)
            if $i < r$ **then** $\mathrm{high} \leftarrow z$
            else if $i > r$ **then** $\mathrm{low} \leftarrow z$
            else $\mathrm{found} \leftarrow$ **true**
        if $z \leqslant \mathrm{low}$ **then** Q.PUSH($t.\mathrm{right}$)
        if $\mathrm{high} \leqslant z$ **then** Q.PUSH($t.\mathrm{left}$)

Analysis

Hypothesis for the analysis:

- The n records are independently drawn from a continuous distribution in $[0, 1]^K$ (standard probability model for random K-d tree)
- The sought rank i is random, with uniform probability in $[1..n]$
- The given coordinate j is also random, with uniform probability in $[1..K]$

Analysis

Hypothesis for the analysis:

- The n records are independently drawn from a continuous distribution in $[0,1]^K$ (standard probability model for random K-d tree)
- The sought rank i is random, with uniform probability in $[1..n]$
- The given coordinate j is also random, with uniform probability in $[1..K]$

Analysis

Hypothesis for the analysis:

- The n records are independently drawn from a continuous distribution in $[0, 1]^K$ (standard probability model for random K-d tree)
- The sought rank i is random, with uniform probability in $[1..n]$
- The given coordinate j is also random, with uniform probability in $[1..K]$

Analysis

Five key observations

1. The number of visited nodes in the main loop is at most the number of nodes visited by an orthogonal range search with the strip $[low, high]$
2. The cost of a call to BELOW is that of a partial match with a single specified coordinate
3. The expected number of calls to BELOW is $\Theta(\log n)$
4. The main loop finds the sought point when the node discriminates along j-th coordinate or the strip $[low, high]$ contains it and no point that discriminates with respect to j
5. The strip contains $\Theta(1)$ points on average

Analysis

Five key observations

1. The number of visited nodes in the main loop is at most the number of nodes visited by an orthogonal range search with the strip $[low, high]$
2. The cost of a call to BELOW is that of a partial match with a single specified coordinate
3. The expected number of calls to BELOW is $\Theta(\log n)$
4. The main loop finds the sought point when the node discriminates along j-th coordinate or the strip $[low, high]$ contains it and no point that discriminates with respect to j
5. The strip contains $\Theta(1)$ points on average

Analysis

Five key observations

1. The number of visited nodes in the main loop is at most the number of nodes visited by an orthogonal range search with the strip $[low, high]$
2. The cost of a call to BELOW is that of a partial match with a single specified coordinate
3. The expected number of calls to BELOW is $\Theta(\log n)$
4. The main loop finds the sought point when the node discriminates along j-th coordinate or the strip $[low, high]$ contains it and no point that discriminates with respect to j
5. The strip contains $\Theta(1)$ points on average

Analysis

Five key observations

1. The number of visited nodes in the main loop is at most the number of nodes visited by an orthogonal range search with the strip $[\mathrm{low}, \mathrm{high}]$
2. The cost of a call to BELOW is that of a partial match with a single specified coordinate
3. The expected number of calls to BELOW is $\Theta(\log n)$
4. The main loop finds the sought point when the node discriminates along j-th coordinate or the strip $[\mathrm{low}, \mathrm{high}]$ contains it and no point that discriminates with respect to j
5. The strip contains $\Theta(1)$ points on average

Analysis

Five key observations

1. The number of visited nodes in the main loop is at most the number of nodes visited by an orthogonal range search with the strip $[low, high]$
2. The cost of a call to BELOW is that of a partial match with a single specified coordinate
3. The expected number of calls to BELOW is $\Theta(\log n)$
4. The main loop finds the sought point when the node discriminates along j-th coordinate or the strip $[low, high]$ contains it and no point that discriminates with respect to j
5. The strip contains $\Theta(1)$ points on average

Analysis

To achieve a good expected performance for a call to BELOW, it is necessary that each node contains the size of the subtree rooted at that tree

```
procedure BELOW(T, j, z)
    if T = □ then return 0
    if T.discr ≠ j then
        c ← ⟦T.key[j] ⩽ z⟧
        return BELOW(z, j, T.left) + BELOW(z, j, T.right) + c
    else
        if z < T.key[j] then return BELOW(z, j, T.left)
        else return T.left.size + BELOW(z, j, T.right)
```

Analysis

- The expected cost of the second and third phases (if needed) is $\Theta(1)$ (Observation #5)
- The expected cost of the main loop, without counting the cost of calls to BELOW is $\Theta(n^\alpha)$ (Observation #1), where $\alpha = \alpha(K)$ depends on the type of K-d tree; for any K and any variant of K-d trees

$$1 - \frac{1}{K} \leqslant \alpha(K) < 1$$

  For instance $\alpha(2) \approx 0.56$ for standard K-d trees
- The expected cost of a call to BELOW is $\Theta(n^\alpha)$ (Observation #2)
- The expected cost of the algorithm is $\Theta(n^\alpha \log n)$ (Observations #1 – #3)

Analysis

- The expected cost of the second and third phases (if needed) is $\Theta(1)$ (Observation #5)
- The expected cost of the main loop, without counting the cost of calls to BELOW is $\Theta(n^{\alpha})$ (Observation #1), where $\alpha = \alpha(K)$ depends on the type of K-d tree; for any K and any variant of K-d trees

$$1 - \frac{1}{K} \leqslant \alpha(K) < 1$$

For instance $\alpha(2) \approx 0.56$ for standard K-d trees

- The expected cost of a call to BELOW is $\Theta(n^{\alpha})$ (Observation #2)
- The expected cost of the algorithm is $\Theta(n^{\alpha} \log n)$ (Observations #1 – #3)

Analysis

- The expected cost of the second and third phases (if needed) is $\Theta(1)$ (Observation #5)
- The expected cost of the main loop, without counting the cost of calls to BELOW is $\Theta(n^\alpha)$ (Observation #1), where $\alpha = \alpha(K)$ depends on the type of K-d tree; for any K and any variant of K-d trees

$$1 - \frac{1}{K} \leqslant \alpha(K) < 1$$

  For instance $\alpha(2) \approx 0.56$ for standard K-d trees
- The expected cost of a call to BELOW is $\Theta(n^\alpha)$ (Observation #2)
- The expected cost of the algorithm is $\Theta(n^\alpha \log n)$ (Observations #1 – #3)

Analysis

- The expected cost of the second and third phases (if needed) is $\Theta(1)$ (Observation #5)
- The expected cost of the main loop, without counting the cost of calls to BELOW is $\Theta(n^\alpha)$ (Observation #1), where $\alpha = \alpha(K)$ depends on the type of K-d tree; for any K and any variant of K-d trees

$$1 - \frac{1}{K} \leqslant \alpha(K) < 1$$

  For instance $\alpha(2) \approx 0.56$ for standard K-d trees
- The expected cost of a call to BELOW is $\Theta(n^\alpha)$ (Observation #2)
- The expected cost of the algorithm is $\Theta(n^\alpha \log n)$ (Observations #1 – #3)

Final remarks

- A simple algorithm with sublinear expected cost
- It can easily be extended to many other multidimensional data structures
- Very little overhead: storing the size of each subtree is not very space consuming and it can also be sucessfully used for balancing (e.g., randomized relaxed K-d trees)
- Experiments show that it is competitive in practice compared to alternative solutions, for reasonably low dimensions (when K grows, $\alpha(K) \to 1$)

Final remarks

- A simple algorithm with sublinear expected cost
- It can easily be extended to many other multidimensional data structures
- Very little overhead: storing the size of each subtree is not very space consuming and it can also be sucessfully used for balancing (e.g., randomized relaxed K-d trees)
- Experiments show that it is competitive in practice compared to alternative solutions, for reasonably low dimensions (when K grows, $\alpha(K) \to 1$)

Final remarks

- A simple algorithm with sublinear expected cost
- It can easily be extended to many other multidimensional data structures
- Very little overhead: storing the size of each subtree is not very space consuming and it can also be sucessfully used for balancing (e.g., randomized relaxed K-d trees)
- Experiments show that it is competitive in practice compared to alternative solutions, for reasonably low dimensions (when K grows, $\alpha(K) \to 1$)

Final remarks

- A simple algorithm with sublinear expected cost
- It can easily be extended to many other multidimensional data structures
- Very little overhead: storing the size of each subtree is not very space consuming and it can also be sucessfully used for balancing (e.g., randomized relaxed K-d trees)
- Experiments show that it is competitive in practice compared to alternative solutions, for reasonably low dimensions (when K grows, $\alpha(K) \to 1$)

Merci beaucoup!