Updating K-d Trees

Amalia Duch
Conrado Martínez

Univ. Politècnica de Catalunya, Spain

1 Introduction

2 Updating with split and join

3 Analysis of split and join

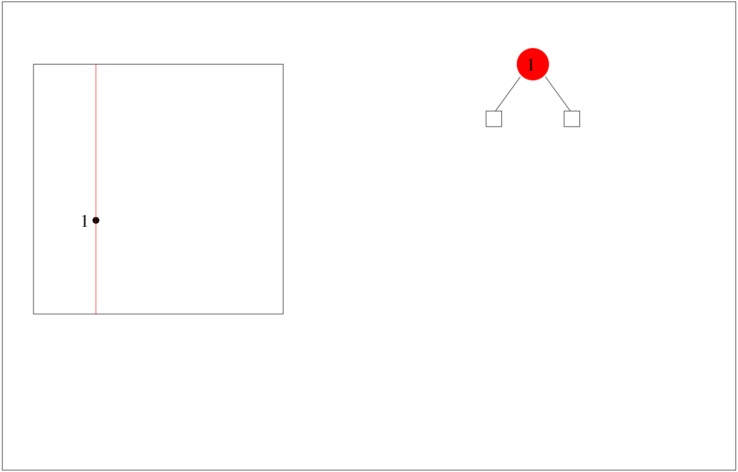
4 Copy-based updates

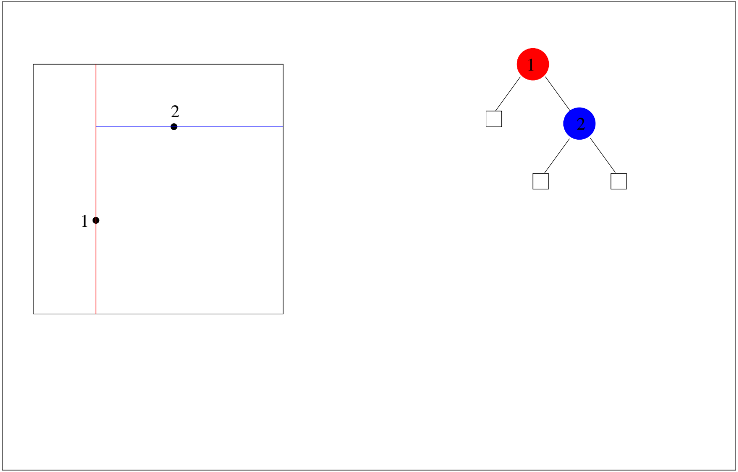
5 Analysis of copy-based updates

6 The cost of insertions and deletions

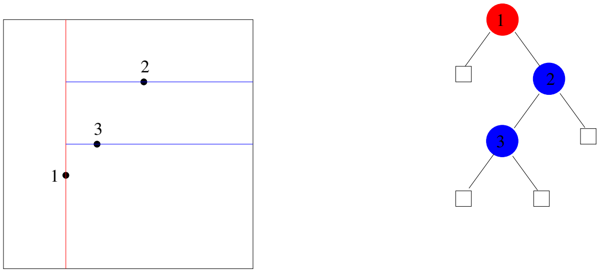
- A relaxed K-d tree is a variant of K-d trees (Bentley, 1975), where each node stores a random discriminant i, $0 \leq i < K$
- They were introduced by Duch, Estivill-castro and Martínez (1998) and subsequently analyzed by Martínez, Panholzer and Prodinger (2001), by Duch and Martínez (2002a, 2002b), and by Broutin, Dalal, Devroye and McLeish (2006)

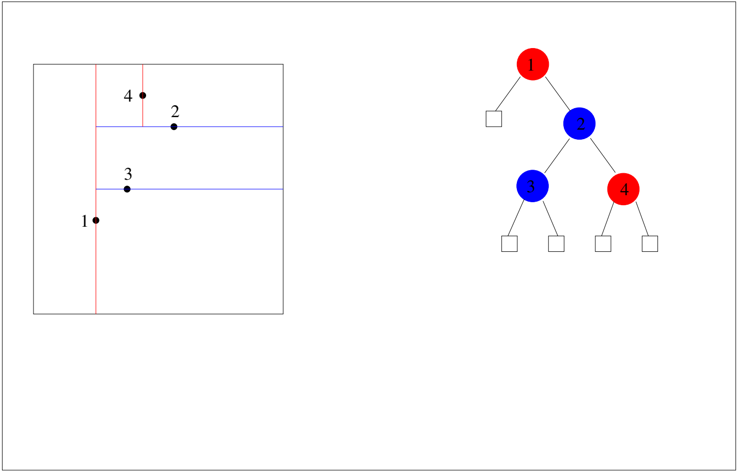
- A relaxed K-d tree is a variant of K-d trees (Bentley, 1975), where each node stores a random discriminant i, $0 \leq i < K$
- They were introduced by Duch, Estivill-castro and Martínez (1998) and subsequently analyzed by Martínez, Panholzer and Prodinger (2001), by Duch and Martínez (2002a, 2002b), and by Broutin, Dalal, Devroye and McLeish (2006)

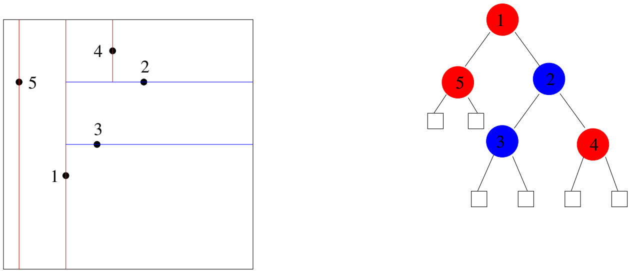

1
1



2
1
1
2


2
3
1
1
2
3


4
2
3
1
1
2
3
4



4
2
5
3
1
1
5
2
3
4

- Relaxation allows insertions at arbitrary positions
- Subtree sizes can be used to guarantee randomness under arbitrary insertions or deletions, hence we can provide guarantees on expected performance
- The average performance of associative queries (e.g., partial match, orthogonal range search, nearest neighbors) is slightly worse than standard K-d trees

- Relaxation allows insertions at arbitrary positions
- Subtree sizes can be used to guarantee randomness under arbitrary insertions or deletions, hence we can provide guarantees on expected performance
- The average performance of associative queries (e.g., partial match, orthogonal range search, nearest neighbors) is slightly worse than standard K-d trees

- Relaxation allows insertions at arbitrary positions
- Subtree sizes can be used to guarantee randomness under arbitrary insertions or deletions, hence we can provide guarantees on expected performance
- The average performance of associative queries (e.g., partial match, orthogonal range search, nearest neighbors) is slightly worse than standard K-d trees

```
struct node {
        Elem key;
        int discr, size;
        node* left, * right;
};
typedef node* rkdt;
```

Insertion in relaxed K-d trees

```
rkdt insert(rkdt t, const Elem& x) {
      int n = size(t);
      int u = random(0,n);
      if (u == n)
         return insert_at_root(t, x);
      else { // t cannot be empty
         int i = t -> discr;
         if (x[i] < t -> key[i])
            t -> left = insert(t -> left, x);
         else
            t -> right = insert(t -> right, x);
         return t;
      }
}
```

Deletion in relaxed K-d trees

```
rkdt delete(rkdt t, const Elem& x) {
     if (t == NULL) return NULL;
     if (t -> key == x)
        return delete_root(t);
     int i = t -> discr;
     if (x -> key[i] < t -> key[i])
        t -> left = delete(t -> left, x);
     else
        t -> right = delete(t -> right, x);
     return t;
}
```

1. Introduction
2. Updating with split and join
3. Analysis of split and join
4. Copy-based updates
5. Analysis of copy-based updates
6. The cost of insertions and deletions

Insertion at root

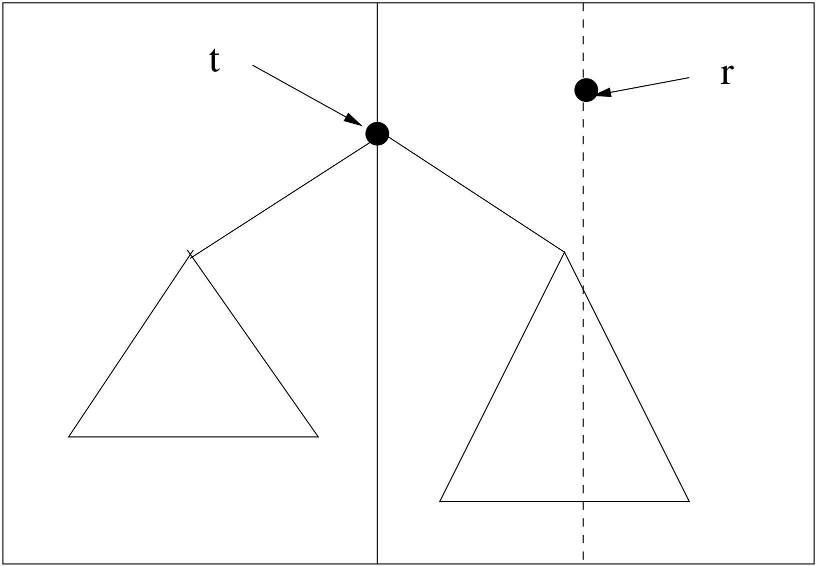
```
rkdt insert_at_root(rkdt t, const Elem& x) {
     rkdt r = new node;
     r -> info = x;
     r -> discr = random(0, K-1);
     pair<rkdt, rkdt> p = split(t, r);
     r -> left = p.first;
     r -> right = p.second;
     return r;
}
```

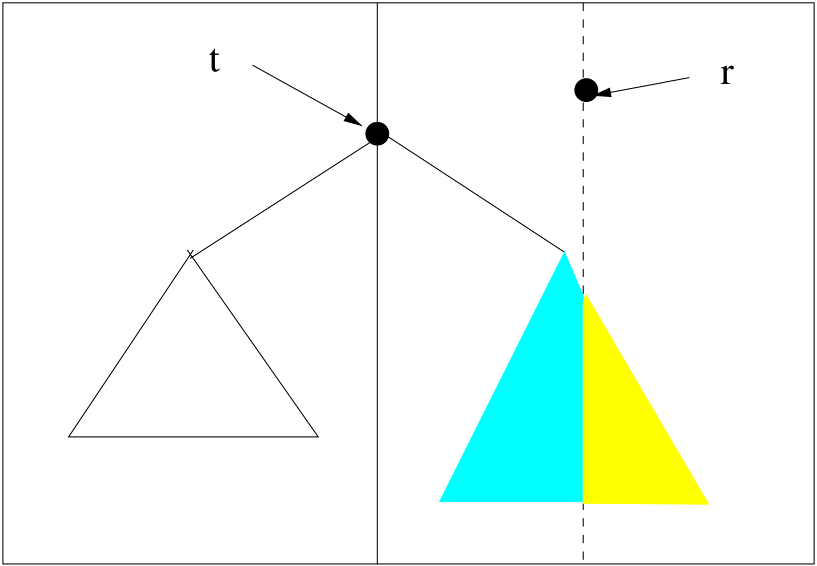
Split

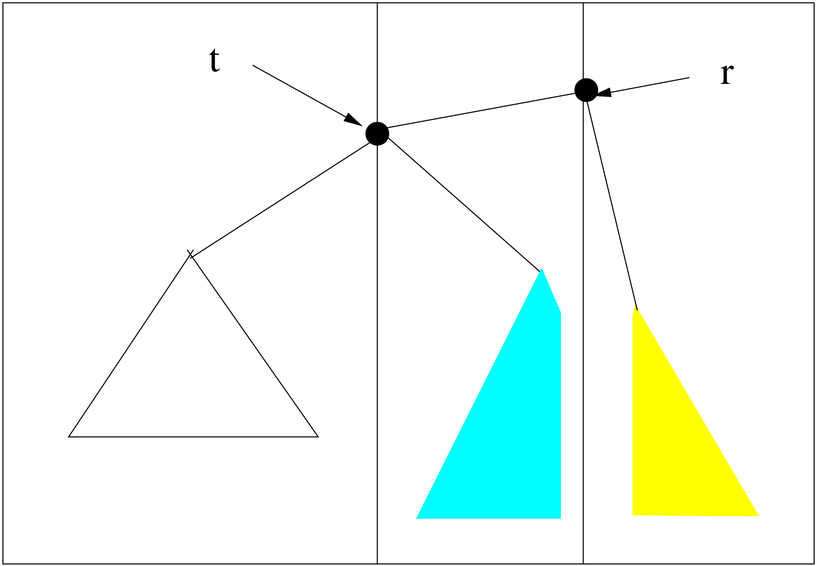
```
pair<rkdt, rkdt> split(rkdt t, rkdt r) {
    if (t == NULL) return make_pair(NULL, NULL);
    int i = r -> discr; int j = t -> discr;
    if (i == j) {
        // Case I
        ...
    } else {
        // Case II
        ...
    }
}
```

Split: Case 1

```
if (i == j) {
   if (r -> key[i] < t -> key[i]) {
      pair<rkdt,rkdt> p = split(t -> left, r);
      t -> left = p.second;
      return make_pair(p.first, t);
   } else {
      pair<rkdt, rkdt> p = split(t -> right, r);
      t -> right = p.first;
      return make_pair(t, p.second);
   }
} else { // i != j
      ...
}
```

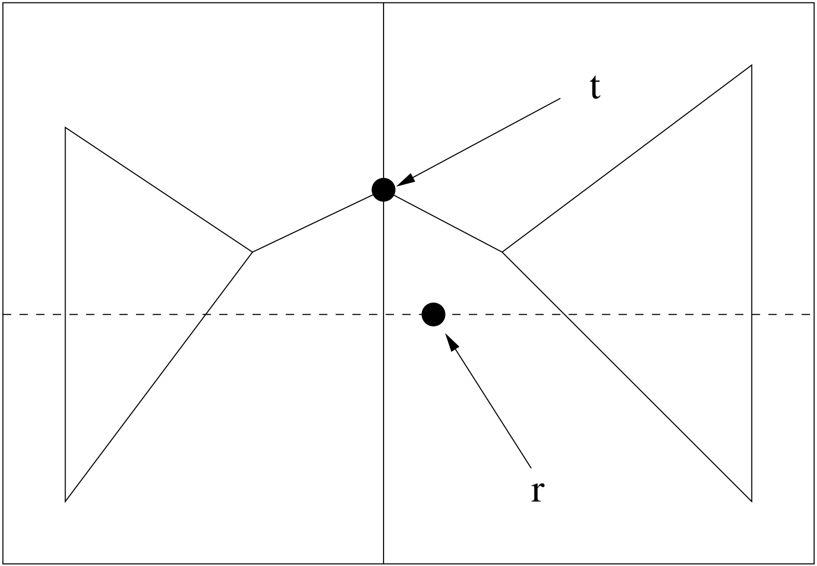

t
r

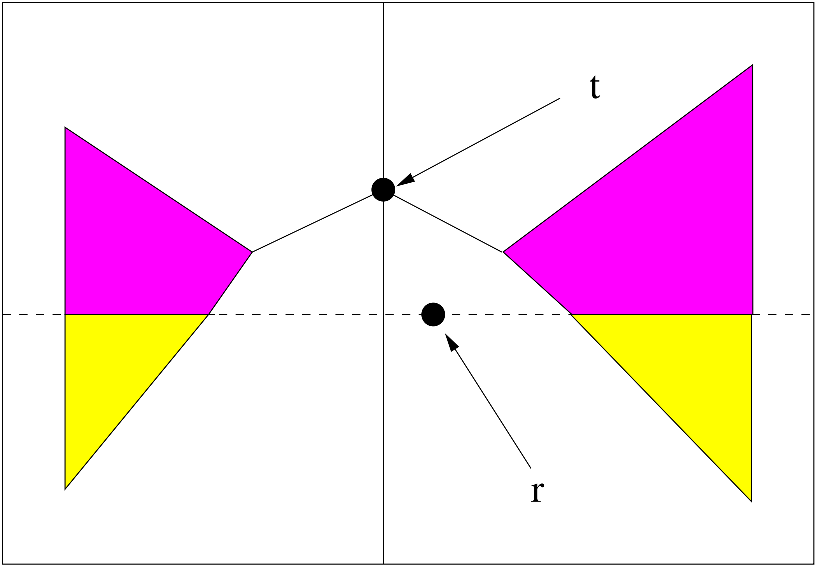

t
r

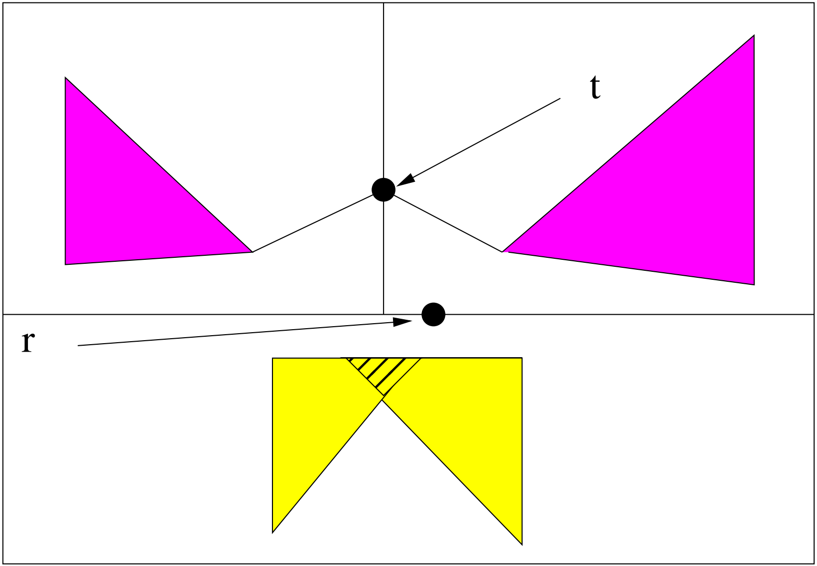

t
r

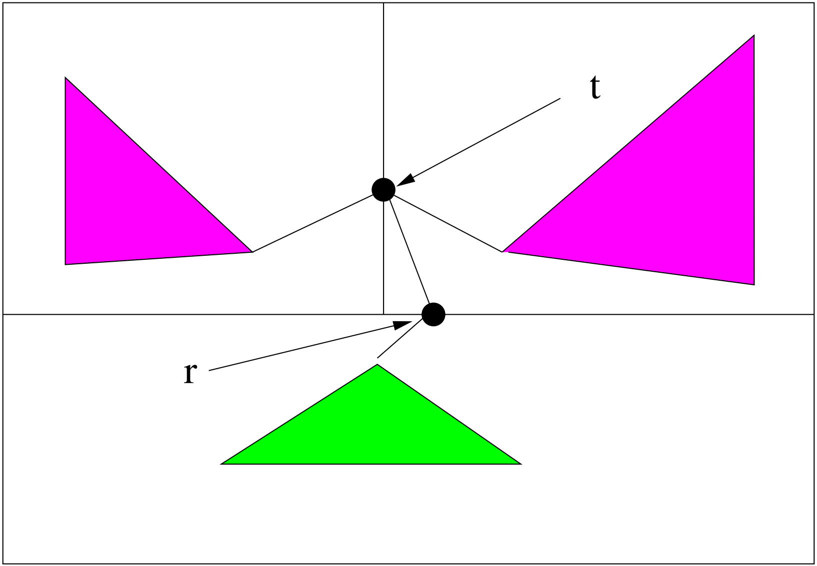
Split: Case II

```
if (i == j) {
  ...
} else { // i != j
   pair<rkdt, rkdt> L = split(t -> left, r);
   pair<rkdt, rkdt> R = split(t -> right, r);
   if (r -> key[i] < t -> key[i]) {
     t -> left = L.second;
     t -> right = R.second;
     return make_pair(join(L.first, R.first, j), t);
   } else {
     t -> left= L.first;
     t -> right = R.first;
     return make_pair(t, join(L.second, R.second, j));
   }
}
```


t
r


t
r


t
r


t
r

Deletion in relaxed K-d trees

```
rkdt delete(rkdt t, const Elem& x) {
      if (t == NULL) return NULL;
      int i = t -> discr;
      if (t -> key == x)
         return join(t -> left, t -> right, i);
      if (x -> key[i] < t -> key[i])
         t -> left = delete(t -> left, x);
      else
         t -> right = delete(t -> right, x);
      return t;
}
```

Joining two trees

```
rkdt join(rkdt L, rkdt R, int i) {
  if (L == NULL) return R;
  if (R == NULL) return L;

  // L != NULL and R != NULL
  int m = size(L); int n = size(R);
  int u = random(0, m+n-1);
  if (u < m) // with probability m / (m + n)
             // the joint root is that of L
     ...
  else       // with probability n / (m + n)
             // the joint root is that of R
}
```

1 Introduction

2 Updating with split and join

3 Analysis of split and join

4 Copy-based updates

5 Analysis of copy-based updates

6 The cost of insertions and deletions

- s_n = avg. number of visited nodes in a split
- m_n = avg. number of visited nodes in a join
- $$s_n = 1 + \frac{2}{nK} \sum_{0 \le j < n} \frac{j+1}{n+1} s_j + \frac{2(K-1)}{nK} \sum_{0 \le j < n} s_j + \frac{K-1}{K} \sum_{0 \le j < n} \pi_{n,j} m_j,$$

  where $\pi_{n,j}$ is probability of joining two trees with total size j.

- s_n = avg. number of visited nodes in a split
- m_n = avg. number of visited nodes in a join
-

$$s_n = 1 + \frac{2}{nK} \sum_{0 \le j < n} \frac{j+1}{n+1} s_j + \frac{2(K-1)}{nK} \sum_{0 \le j < n} s_j + \frac{K-1}{K} \sum_{0 \le j < n} \pi_{n,j} m_j,$$

where $\pi_{n,j}$ is probability of joining two trees with total size j.

- s_n = avg. number of visited nodes in a split
- m_n = avg. number of visited nodes in a join
- $$s_n = 1 + \frac{2}{nK} \sum_{0 \le j < n} \frac{j+1}{n+1} s_j + \frac{2(K-1)}{nK} \sum_{0 \le j < n} s_j + \frac{K-1}{K} \sum_{0 \le j < n} \pi_{n,j} m_j,$$

  where $\pi_{n,j}$ is probability of joining two trees with total size j.

- The recurrence for s_n is

$$s_n = 1 + \frac{2}{nK} \sum_{0 \le j < n} \frac{j+1}{n+1} s_j + \frac{2(K-1)}{nK} \sum_{0 \le j < n} s_j + \frac{2(K-1)}{nK} \sum_{0 \le j < n} \frac{n-j}{n+1} m_j,$$

with $s_0 = 0$.

- The recurrence for m_n has exactly the same shape with the rôles of s_n and m_n interchanged; it easily follows that $s_n = m_n$.

- The recurrence for s_n is

$$s_n = 1 + \frac{2}{nK} \sum_{0 \le j < n} \frac{j+1}{n+1} s_j + \frac{2(K-1)}{nK} \sum_{0 \le j < n} s_j + \frac{2(K-1)}{nK} \sum_{0 \le j < n} \frac{n-j}{n+1} m_j,$$

  with $s_0 = 0$.
- The recurrence for m_n has exactly the same shape with the rôles of s_n and m_n interchanged; it easily follows that $s_n = m_n$.

- Define

$$S(z) = \sum_{n \geq 0} s_n z^n$$

- The recurrence for s_n translates to

$$z \frac{d^2 S}{dz^2} + 2 \frac{1-2z}{1-z} \frac{dS}{dz} - 2\left(\frac{3K-2}{K} - z\right) \frac{S(z)}{(1-z)^2} = \frac{2}{(1-z)^3},$$

with initial conditions $S(0) = 0$ and $S'(0) = 1$.

- Define

$$S(z) = \sum_{n \geq 0} s_n z^n$$

- The recurrence for s_n translates to

$$z \frac{d^2 S}{dz^2} + 2 \frac{1-2z}{1-z} \frac{dS}{dz} - 2 \left(\frac{3K-2}{K} - z \right) \frac{S(z)}{(1-z)^2} = \frac{2}{(1-z)^3},$$

with initial conditions $S(0) = 0$ and $S'(0) = 1$.

- The homogeneous second order linear ODE is of hypergeometric type.
- An easy particular solution of the ODE is

$$-\frac{1}{2}\left(\frac{K}{K-1}\right)\frac{1}{1-z}$$

- The homogeneous second order linear ODE is of hypergeometric type.
- An easy particular solution of the ODE is

$$-\frac{1}{2}\left(\frac{K}{K-1}\right)\frac{1}{1-z}$$

Theorem

The generating function $S(z)$ of the expected cost of split is, for any $K \geq 2$,

$$S(z) = \frac{1}{2}\frac{1}{1-\frac{1}{K}}\left[(1-z)^{-\alpha}\cdot {}_2F_1\left(\begin{array}{c}1-\alpha, 2-\alpha\\ 2\end{array}\middle|\, z\right) - \frac{1}{1-z}\right],$$

where $\alpha = \alpha(K) = \frac{1}{2}\left(1+\sqrt{17-\frac{16}{K}}\right)$.

Theorem

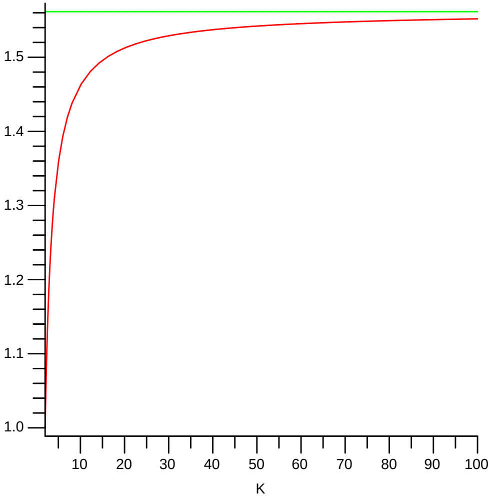
The expected cost s_n of splitting a relaxed K-d tree of size n is

$$s_n = \eta(K)\, n^{\phi(K)} + o(n),$$

with

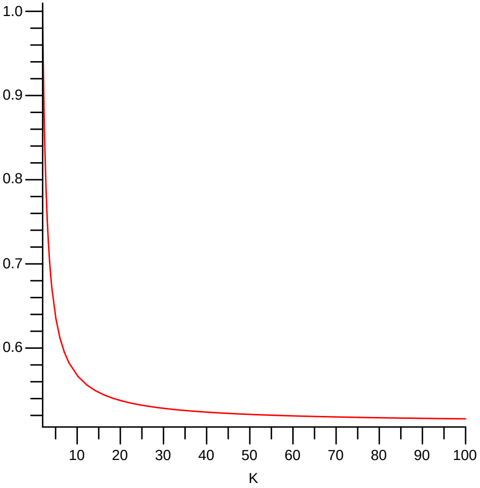
$$\eta = \frac{1}{2} \frac{1}{1 - \frac{1}{K}} \frac{\Gamma(2\alpha - 1)}{\alpha \Gamma^3(\alpha)},$$

$$\phi = \alpha - 1 = \frac{1}{2} \left(\sqrt{17 - \frac{16}{K}} - 1 \right).$$



Plot of $\phi(K)$

$$\phi(2) = 1 \leq \phi(K) \leq \phi(\infty) = (\sqrt{17} - 1)/2 \approx 1.5615, \quad K \geq 2$$



Plot of $\eta(K)$

$$\eta(2) = 1 \geq \eta(K) \geq \eta(\infty) \approx 0.5107, \quad K \geq 2$$

1. Introduction
2. Updating with split and join
3. Analysis of split and join
4. **Copy-based updates**
5. Analysis of copy-based updates
6. The cost of insertions and deletions

Modified standard insertion

```
// inserts the tree z in the appropriate leaf of T
rkdt insert_std(rkdt T, rkdt z) {
  if (T == NULL) return z;
  else {
     int i = T -> discr;
     if (z -> key[i] < T -> key[i])
        T -> left = insert(T -> left, z);
     else
        T -> right = insert(T -> right, z);
     return T;
  }
}
```

Copy-based insertion (I)

```
rkdt insert_at_root(rkdt T, const Elem& x) {
  rkdt result = new node(x, random(0, K-1));
  int i = result -> discr;
  queue<rkdt> Q;
  Q.push(T);
  while (!Q.empty()) {
    rkdt z = Q.pop(); if (z == NULL) continue;
    // insert one or both subtrees of z
    // back to Q
    result = insert_std(result, z);
  }
  return result;
}
```

Copy-based insertion (2)

```
...
if (z -> discr != i) {
        Q.push(z -> left);
        Q.push(z -> right);
        z -> left = z -> right = NULL;
} else {
   if (x[i] < z -> key[i]) {
        Q.push(z -> left);
        z -> left = NULL;
    } else {
        Q.push(z -> right);
        z -> right = NULL;
    }
}
...
```

Copy-based deletion

```
rkdt delete_root(rkdt T) {
     Elem x = T -> key;
     int i = T -> discr;
     queue<rkdt> QL, QR;
     rkdt result = NULL;
     QL.push(T -> left); QR.push(T -> right);
     while (!QL.empty() && !QR.empty()) {
       rkdt U = QL.front(); rkdt V = QR.front();
       int m = size(U); int n = size(V);
       if (random(0,m+n-1) < m) {
           QL.pop();
           // insert U (and eventually one of
           // its subtrees) into the current result;
           // insert one or two subtrees of U back into
           // QL
           result = insert_std(result, U);
       } else {
           // symmetric code with QR and V
       }
     }
     return result;
}
```

1 Introduction

2 Updating with split and join

3 Analysis of split and join

4 Copy-based updates

5 Analysis of copy-based updates

6 The cost of insertions and deletions

The cost of building T using copy-based insertion:

$$C(T) = 1 + \frac{1}{K}\left(\frac{|L|+1}{|T|+1}(P(L)+C(L))\right) + \frac{1}{K}\left(\frac{|R|+1}{|T|+1}(P(R)+C(R))\right) + \frac{K-1}{K}\left(P(L)+P(R)+C(L)+C(R)\right),$$

where $P(T)$ denotes the number of nodes visited by a partial match in a random tree T

$\Longrightarrow$

$$C(T) = P(T) + \frac{1}{K}\frac{|L|+1}{|T|+1}C(L) + \frac{1}{K}\frac{|R|+1}{|T|+1}C(R) + \frac{K-1}{K}\left(C(L)+C(R)\right),$$

The cost of making an insertion at root into a tree of size n:

$$C_n = P_n + \frac{2}{nK} \sum_{0 \le k < n} \frac{k+1}{n+1} C_k + \frac{2(K-1)}{nK} \sum_{0 \le k < n} C_k.$$

with P_n the expected cost of a partial match in a random relaxed K-d tree of size n with only one specified coordinate out of K coordinates

Theorem ((Duch et al. 1998, Martínez et al. 2001))

The expected cost P_n (measured as the number of key comparisons) of a partial match query with s out of K attributes specified, $0 < s < K$, in a randomly built relaxed K-d tree of size n is

$$P_n = \beta(s/K) \cdot n^{\rho(s/K)} + \mathcal{O}(1),$$

where

$$\rho = \rho(x) = \left(\sqrt{9-8x} - 1\right)/2,$$

$$\beta(x) = \frac{\Gamma(2\rho+1)}{(1-x)(\rho+1)\Gamma^3(\rho+1)},$$

and $\Gamma(x)$ is Euler's Gamma function.

We will use Roura's Continuous Master Theorem to solve recurrences of the form:

$$F_n = t_n + \sum_{0 \le j < n} w_{n,j} F_j, \qquad n \ge n_0,$$

where t_n is the so-called toll function and the quantities $w_{n,j} \ge 0$ are called weights

Theorem (Continuous master theorem, Roura 2001)

Let $t_n \sim Cn^a \log^b n$ for some constants C, $a \geq 0$ and $b > -1$, and let $\omega(z)$ be a real function over $[0, 1]$ such that

$$\sum_{0 \leq j < n} \left| w_{n,j} - \int_{j/n}^{(j+1)/n} \omega(z)\, dz \right| = \mathcal{O}(n^{-d})$$

for some constant $d > 0$. Let $\phi(x) = \int_0^1 z^x\, \omega(z)\, dz$, and define $\mathcal{H} = 1 - \phi(a)$. Then

1. If $\mathcal{H} > 0$ then $F_n \sim t_n \,/\, \mathcal{H}$.
2. If $\mathcal{H} = 0$ then $F_n \sim t_n \ln n \,/\, \mathcal{H}'$, where $\mathcal{H}' = -(b+1) \int_0^1 z^a \ln z\, \omega(z)\, dz$.
3. If $\mathcal{H} < 0$ then $F_n = \Theta(n^\alpha)$, where α is the unique real solution of $\phi(x) = 1$.

Applying the CMT to our recurrence we have

- $\omega(z) = \frac{2z}{K} + \frac{2(K-1)}{K}$
- $t_n = P_n \implies a = \varrho = \rho(1/K) = (\sqrt{9 - 8/K} - 1)/2$

Thus $\mathcal{H} = 0$

Applying the CMT to our recurrence we have

- $\omega(z) = \frac{2z}{K} + \frac{2(K-1)}{K}$
- $t_n = P_n \implies a = \varrho = \rho(1/K) = (\sqrt{9 - 8/K} - 1)/2$

Thus $\mathcal{H} = 0$

We have to compute $\mathcal{H}'$ with $b = 0$

$$\mathcal{H}' = -(b+1) \int_0^1 z^a \omega(z) \ln z \, dz$$

and get

$$\mathcal{H}' = 2 \frac{K\varrho^2 + (4K-2)\varrho + 4K - 3}{K(\varrho+2)^2(\varrho+1)^2}.$$

Theorem

The average cost C_n of copy-based insertion at root of a random relaxed K-d tree is

$$C_n = \gamma \cdot n^{\varrho} \ln n + o(n \ln n),$$

where

$$\varrho = \varrho(K) = \rho(1/K) = \left(\sqrt{9 - 8/K} - 1\right)/2,$$

$$\gamma = \frac{\beta(1/K)}{\mathcal{H}'} = \frac{\Gamma(2\varrho+1)K(\varrho+2)^2(\varrho+1)}{2(1-\frac{1}{K})\Gamma^3(\varrho+1)(K\varrho^2+(4K-2)\varrho+(4K-3))}.$$

The average cost C'_n of copy-based deletion of the root of a random relaxed K-d tree of size $n+1$ is C_n.

1 Introduction
2 Updating with split and join
3 Analysis of split and join
4 Copy-based updates
5 Analysis of copy-based updates
6 The cost of insertions and deletions

- The recurrence for the expected cost of an insertion is

$$I_n = \frac{\mathcal{I}_n}{n+1} + \left(1 - \frac{1}{n+1}\right)\left(1 + \frac{2}{n}\sum_{0\le j<n}\frac{j+1}{n+1}I_j\right)$$
$$= \frac{\mathcal{I}_n}{n+1} + 1 + \mathcal{O}\left(\frac{1}{n}\right) + \frac{2}{n+1}\sum_{0\le j<n}\frac{j+1}{n+1}I_j.$$

  with $\mathcal{I}_n$ the average cost of an insertion at root
- The expected cost of deletions satisfies a similar recurrence; it is asymptotically equivalent to the average cost of insertions
- We substitute $\mathcal{I}_n$ by the costs obtained previously and apply the CMT to solve

Theorem

Let I_n and D_n denote the average cost of a randomized insertion and randomized deletion in a random relaxed K-d tree of size n using split and join. Then

1. if $K = 2$ then $I_n \sim D_n = 4 \ln n + \mathcal{O}(1)$.
2. if $K > 2$ then

$$I_n \sim D_n = \eta \frac{\phi - 1}{\phi + 1} n^{\phi - 1} + \mathcal{O}(\log n),$$

where $\mathcal{I}_n = \eta\, n^{\phi} + \mathcal{O}(1)$.

Theorem

Let I_n and D_n denote the average cost of a randomized insertion and randomized deletion in a random relaxed K-d tree of size n using split and join. Then

1. if $K = 2$ then $I_n \sim D_n = 4 \ln n + \mathcal{O}(1)$.
2. if $K > 2$ then

$$I_n \sim D_n = \eta \frac{\phi - 1}{\phi + 1} n^{\phi - 1} + \mathcal{O}(\log n),$$

   where $\mathcal{I}_n = \eta\, n^{\phi} + \mathcal{O}(1)$.

Note that for $K > 2$, $\phi(K) > 1$!

Theorem

For any fixed dimension $K \geq 2$, the average cost of a randomized insertion or deletion in random relaxed K-d tree of size n using copy-based updates is

$$I_n \sim D_n = 2 \ln n + \Theta(1).$$

Theorem

For any fixed dimension $K \geq 2$, the average cost of a randomized insertion or deletion in random relaxed K-d tree of size n using copy-based updates is

$$I_n \sim D_n = 2 \ln n + \Theta(1).$$

The "reconstruction" phase has constant cost on the average!

Summary:

- Updating with split and join is only practical for $K = 2$ despite the algorithms are elegant and simple; but their use induces expected cost $\Theta(n^{\phi})$ with $\phi > 1$ for insertions and deletions in higher dimensions
- Copy-based updates are also simple and practical, yielding expected logarithmic cost of insertions and deletions for any fixed dimension K
- The optimization of copy-based updates does only apply to relaxed K-d trees; without the optimization it yields insertions and deletions with expect cost $\Theta(\log^2 n)$
- Logarithmic time for insertions and deletions had only been achieved before using rather complex schemes (e.g. pseudo K-d trees, divided K-d trees)

Summary:

- Updating with split and join is only practical for $K = 2$ despite the algorithms are elegant and simple; but their use induces expected cost $\Theta(n^{\phi})$ with $\phi > 1$ for insertions and deletions in higher dimensions
- Copy-based updates are also simple and practical, yielding expected logarithmic cost of insertions and deletions for any fixed dimension K
- The optimization of copy-based updates does only apply to relaxed K-d trees; without the optimization it yields insertions and deletions with expect cost $\Theta(\log^2 n)$
- Logarithmic time for insertions and deletions had only been achieved before using rather complex schemes (e.g. pseudo K-d trees, divided K-d trees)

Summary:

- Updating with split and join is only practical for $K = 2$ despite the algorithms are elegant and simple; but their use induces expected cost $\Theta(n^{\phi})$ with $\phi > 1$ for insertions and deletions in higher dimensions
- Copy-based updates are also simple and practical, yielding expected logarithmic cost of insertions and deletions for any fixed dimension K
- The optimization of copy-based updates does only apply to relaxed K-d trees; without the optimization it yields insertions and deletions with expect cost $\Theta(\log^2 n)$
- Logarithmic time for insertions and deletions had only been achieved before using rather complex schemes (e.g. pseudo K-d trees, divided K-d trees)

Summary:

- Updating with split and join is only practical for $K = 2$ despite the algorithms are elegant and simple; but their use induces expected cost $\Theta(n^\phi)$ with $\phi > 1$ for insertions and deletions in higher dimensions
- Copy-based updates are also simple and practical, yielding expected logarithmic cost of insertions and deletions for any fixed dimension K
- The optimization of copy-based updates does only apply to relaxed K-d trees; without the optimization it yields insertions and deletions with expect cost $\Theta(\log^2 n)$
- Logarithmic time for insertions and deletions had only been achieved before using rather complex schemes (e.g. pseudo K-d trees, divided K-d trees)