KDTree Theorical and experimental features

Colombia

dfsantosb@correo.udistrital.edu.co

Engineering Faculty - Systems Engineering

June 17, 2019

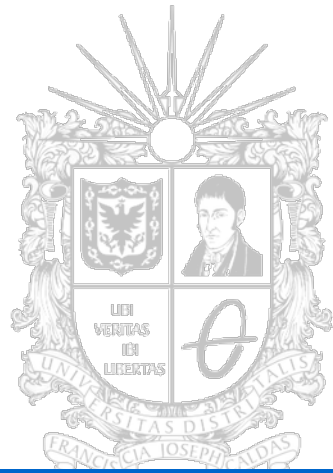


Table of Contents

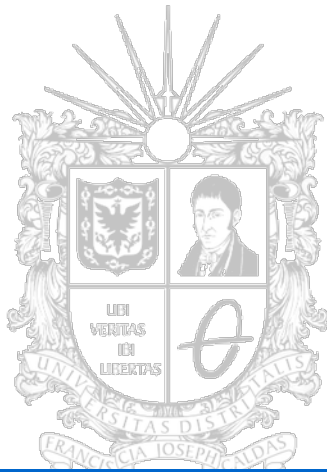
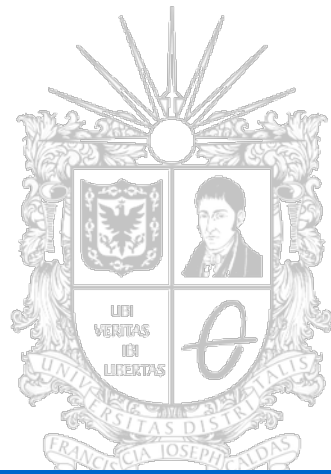


Table of Contents

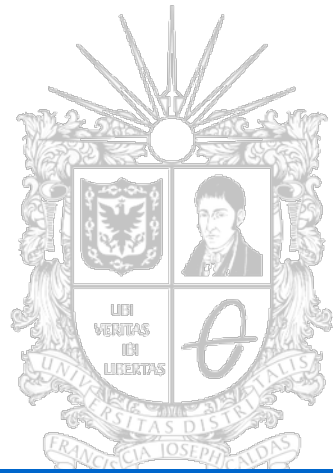


Motivation
Real Life Problems

- I want to find people with $34 <= age <= 49$ and $100k <= followers <= 150k$ to ...

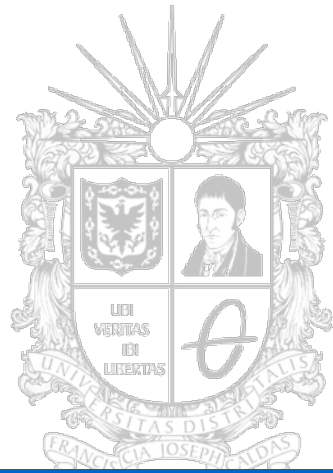
Motivation
Real Life Problems

- I want to find people with $34 <= age <= 49$ and $100k <= followers <= 150k$ to ...
- Search the nearest (Restaurant, Hospital, ...) near me, maybe with some restrictions, Orthogonal Range query



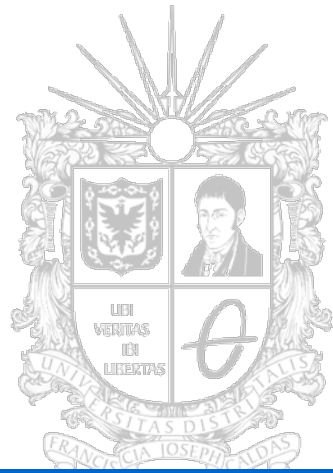
Motivation
Real Life Problems

- I want to find people with $34 <= age <= 49$ and $100k <= followers <= 150k$ to ...
- Search the nearest (Restaurant, Hospital, ...) near me, maybe with some restrictions, Orthogonal Range query
- N-Body problem



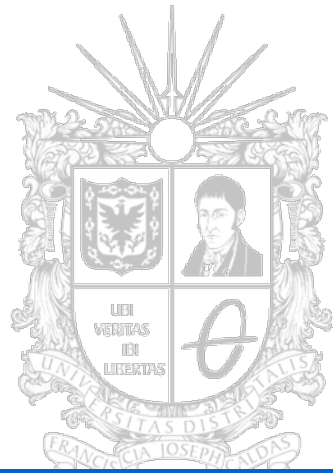
Motivation
Real Life Problems

- I want to find people with $34 <= age <= 49$ and $100k <= followers <= 150k$ to ...
- Search the nearest (Restaurant, Hospital, ...) near me, maybe with some restrictions, Orthogonal Range query
- N-Body problem
- Flocking boids



Motivation
Real Life Problems

- I want to find people with $34 <= age <= 49$ and $100k <= followers <= 150k$ to ...
- Search the nearest (Restaurant, Hospital, ...) near me, maybe with some restrictions, Orthogonal Range query
- N-Body problem
- Flocking boids
- Bounding Volume Hierarchy



Motivation
Real Life Problems

- I want to find people with $34 <= age <= 49$ and $100k <= followers <= 150k$ to ...
- Search the nearest (Restaurant, Hospital, ...) near me, maybe with some restrictions, Orthogonal Range query
- N-Body problem
- Flocking boids
- Bounding Volume Hierarchy
- ...

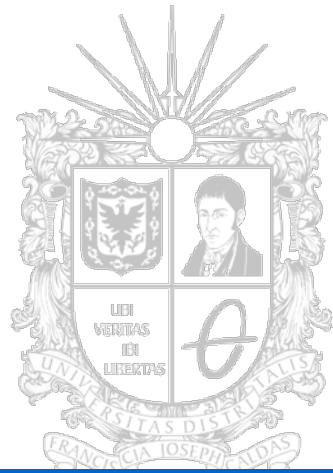
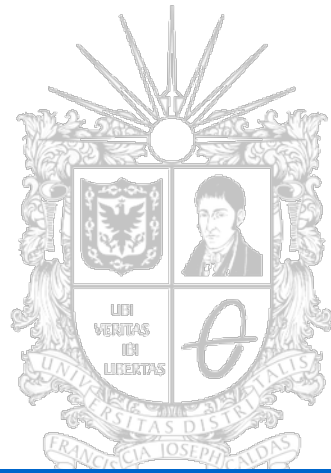
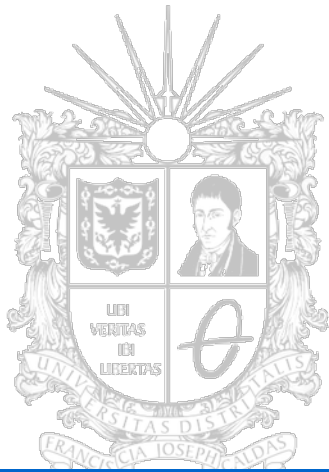


Table of Contents



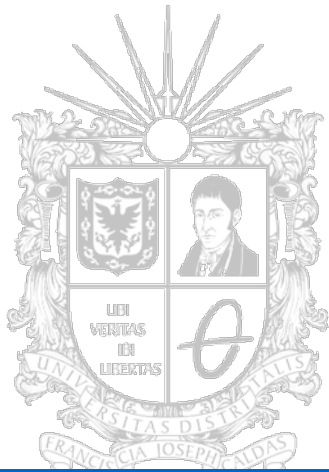
Data Structures

- KDTree



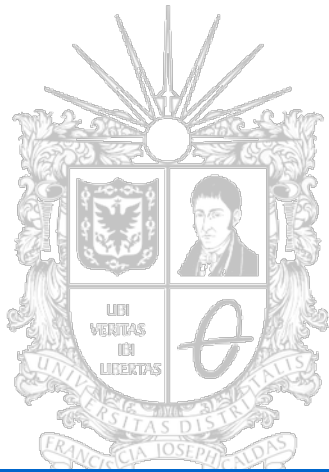
Data Structures

- KDTree
- Orchard's Algorithm (1991)



Data Structures

- KDTree
- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast



Data Structures

- KDTree
- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast
- Annulus Algorithm

Data Structures

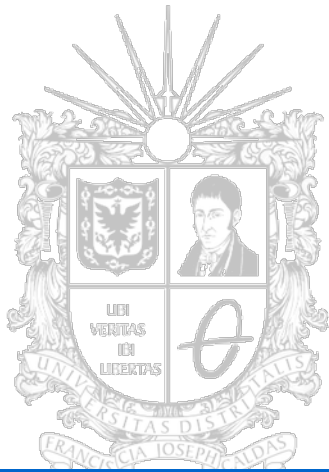
- KDTree
- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses $O(n)$ storage. Does many more distance calculations.

Data Structures

- KDTree
- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses $O(n)$ storage. Does many more distance calculations.
- Principal Component Partitioning (PCP)

Data Structures

- KDTree
- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses $O(n)$ storage. Does many more distance calculations.
- Principal Component Partitioning (PCP)
  - Zatloukal, Johnson, Ladner (1999).



Data Structures

- KDTree
- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses $O(n)$ storage. Does many more distance calculations.
- Principal Component Partitioning (PCP)
  - Zatloukal, Johnson, Ladner (1999).
  - Similar to k-d trees.

Data Structures

- KDTree
- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses $O(n)$ storage. Does many more distance calculations.
- Principal Component Partitioning (PCP)
  - Zatloukal, Johnson, Ladner (1999).
  - Similar to k-d trees.
  - Also very fast.

Data Structures

- KDTree
- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses $O(n)$ storage. Does many more distance calculations.
- Principal Component Partitioning (PCP)
  - Zatloukal, Johnson, Ladner (1999).
  - Similar to k-d trees.
  - Also very fast.
- Sweep Line

Data Structures

Comparison

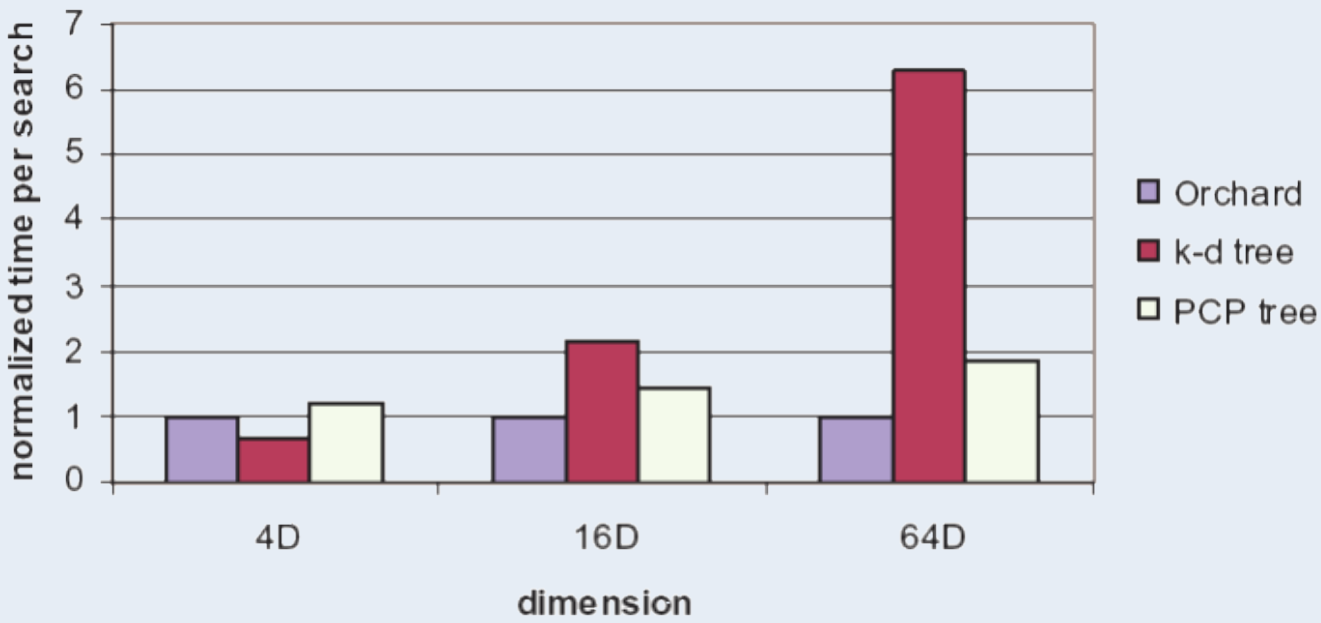
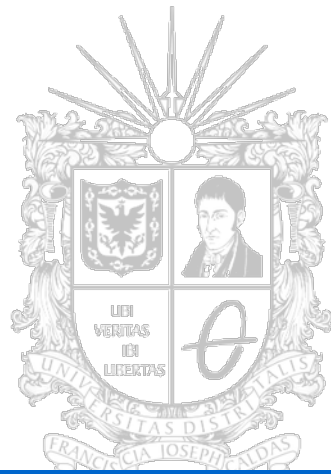


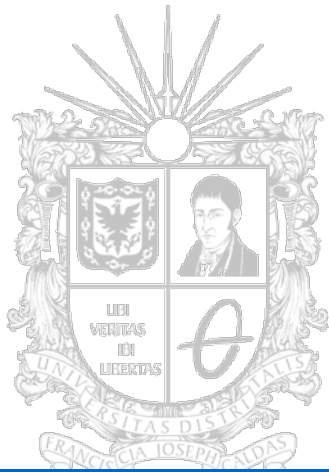
Figure: Run time algorithms. Source: [Oregon State University].

Table of Contents



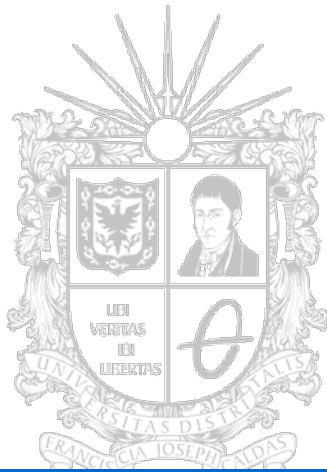
Description

- Jhon Bentley, 1975



Description

- Jhon Bentley, 1975
- Tree Used to store spacial data



Description

- Jhon Bentley, 1975
- Tree Used to store spacial data
- KDTree are guaranteed $log_2(n)$ in depth, where n is the number of points in the set (Demonstration 1.0)

Queries

Options

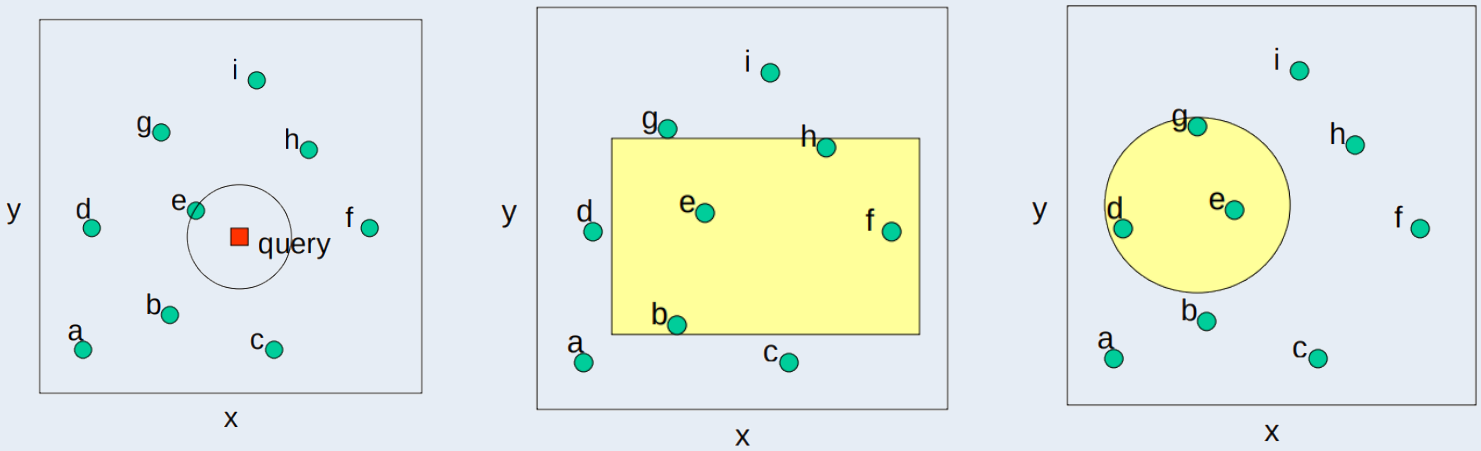
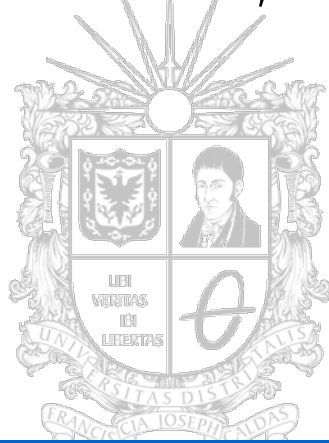


Figure: Range queries and Nearest neighbor search Source: [Washington University, 2002]

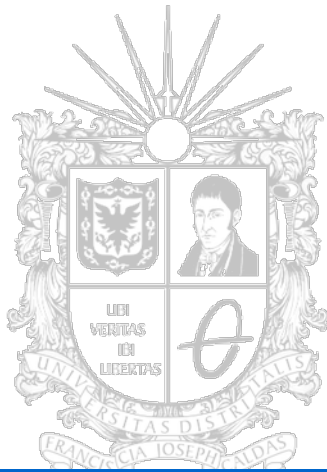
Point

```
/**
 *      author:  xdanielsb
 *      created: 04.2019
**/
struct pt{
  int x[ MAXD ];
  bool operator==( pt &c) const{
    rep(i ,0, dim) if (x[i]!=c.x[i]) return false;
    return true;
  }
};
```



Comparator

```
struct cmp{
  int d;
  cmp( int _d): d(_d){};
  bool operator()( pt &a, pt &b) const {
    return a.x[d] < b.x[d];
  }
};
```

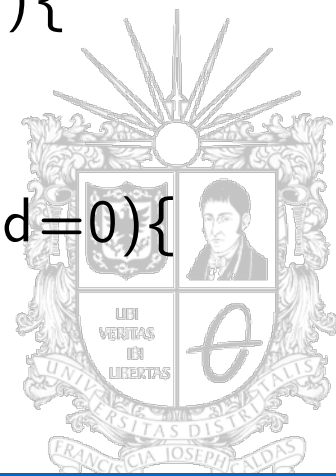


Node

```
struct node{
  int id;
  int d;
  node *l, *r;
  pt p;
  node(){r = l = nullptr;}
  node(int _id, int _d, pt &_p){
    d = _d;
    id = _id;
    p = _p;
    r = l = nullptr;
  }
};
```

KDTree

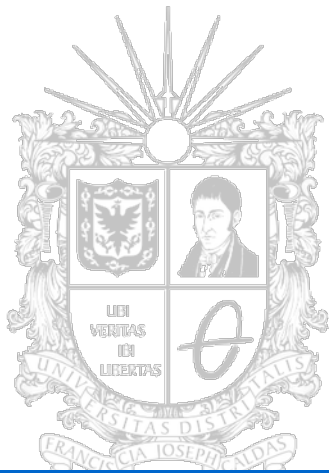
```
struct kdtree{
  vector< pt > pts;
  int n;
  kdtree(int _n ): pts( _n ), n( _n ){
    rep( i, 0, n)
      rep(j, 0, dim)
        // read  pts[ i ].x[ j ];
  }
  node* build(int id,  int l, int r, int d){
  }
  void findNearest( pt &to, node *root,
                    pt &res, lf &dis, int d=0){
  }
};
```



Build

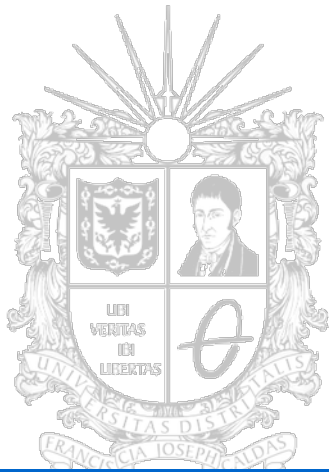
See demonstration 2.0

```
node* build(int id, int l, int r, int d){
  node *root;
  if( l == r) return new node(id, d, pts[l]);
  int nd = (d+1)%dim;
  int mid = r + ( l - r )/2;
  sort(pts.begin()+l,pts.begin()+r+1, cmp(d));
  root = new node(id, d, pts[mid]);
  root->l = build(id*2,l, mid-1, nd);
  root->r = build(id*2+1, mid+1, r, nd);
  return root;
}
```



Query Point

```
typedef long double lf;
void findNearest( pt &to, node *root,  pt &res, lf &dis, int d=0){
  if( root == nullptr) return;
  lf aux = dist( to, root->p );
  if( aux > 0 && dis > aux ) {
    res = root->p;
    dis = aux;
  }
  lf delta = pow(root->p.x[d] - to.x[d], 2.0);
  int nd = (d+1)%dim;
  if(to.x[d] > root->p.x[d] ){
    findNearest( to, root->r, res, dis, nd);
    if( dis > delta)
      findNearest( to, root->l, res, dis, nd);
  }else{
    findNearest( to, root->l, res, dis, nd);
    if( dis > delta)
      findNearest( to, root->r, res, dis, nd);
  }
}
```

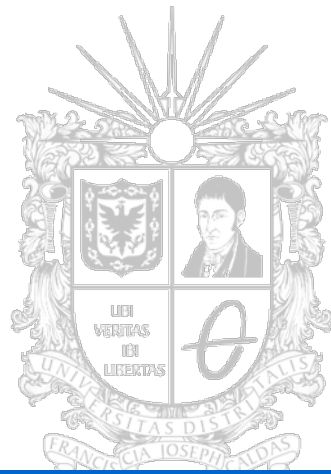


Analysis Performance
Time

Table: Analysis performance time kd-tree.

-	Best	Average	Worst
Build	$O(n)$	$O(n)$	$O(n)$
Query	$O(log_2(n))$	$O(log_2^2(n))$	$O(n)$
Memory	$O(n)$	$O(n)$	$O(n)$

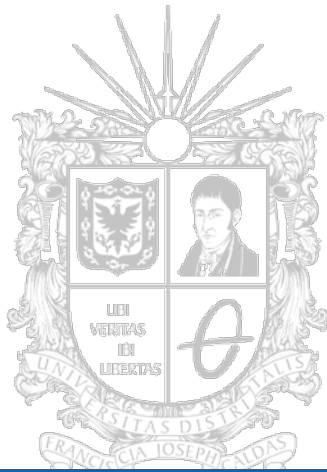
Table of Contents



code

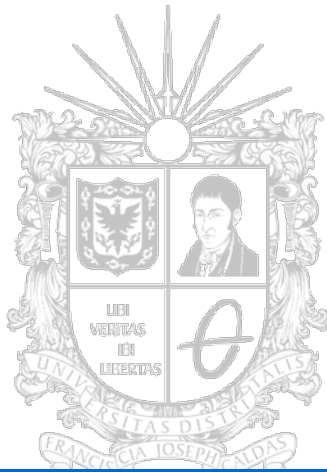
Contest VJudge

- https://vjudge.net/contest/306898



code

URL = https://github.com/xdanielsb/kdtree



References I

Oregon State University. Nearest Neighbor Search. online, accessed 20-May-2019. http://andrewd.ces.clemson.edu/courses/cpsc805/references/nearest$_s$*earch*.*pdf* .

Washington University. K-D Trees Course 373. online, accessed 20-May-2019, Jun 2002. https://courses.cs.washington.edu/courses/cse373/02au/lectures/lecture22l.pdf.

Thank you!