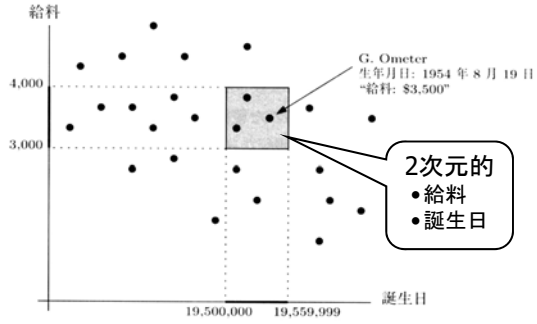
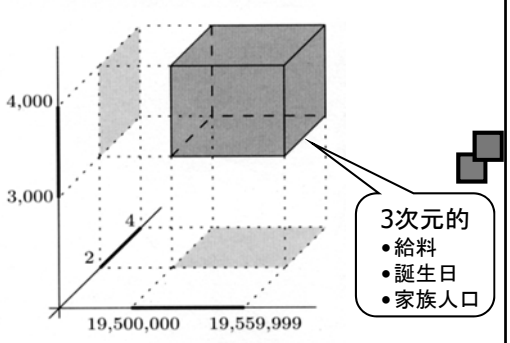
計算幾何学
Computational Geometry

第六章　幾何的領域探索
Geometric range search

データベースの問合せ－1



データベースの問合せ－2



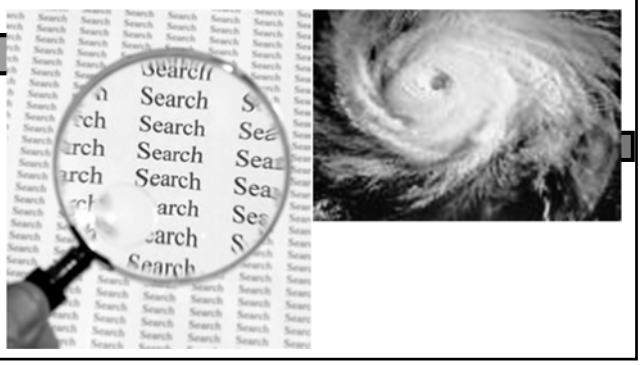
幾何的領域探索(range search)

- データベースの中から対応するフィールドが指定された範囲内にあるレコードをすべて報告せよという問合せは、
  ① レコードのd個のフィールドをd次元空間に変換
  ② レコードをd次元空間の点に変換
  ③ 範囲内にある点集合をすべて求める
  という幾何学的な問題になる
- 幾何学的領域探索
  - 半平面領域探索
  - 長方形領域探索(直交領域探索)
  - 多角形領域探索
  - 円形領域探索
  - 扇形領域探索

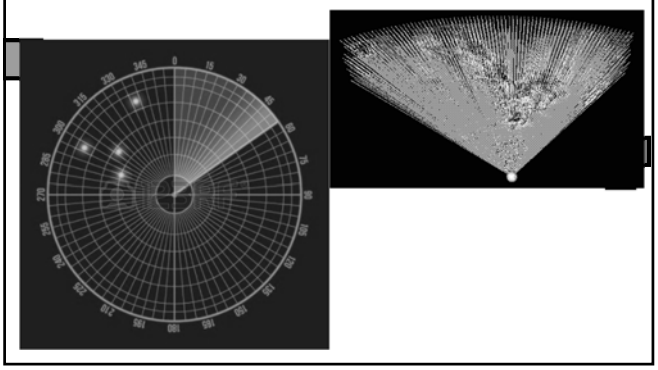
半平面領域

直交領域と多角形領域

円形領域

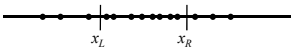


扇形領域



1次元領域探索

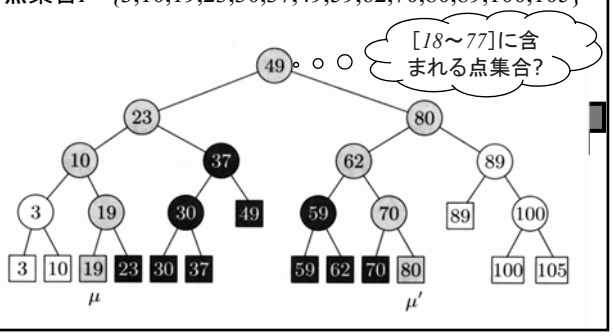
- 既知：直線上の点集合$P=\{p_1,p_2,...,p_n\}$
- 設問：区間[$x_L : x_R$]に含まれる点集合を探せ



- 2分探索木Tに点集合を保存すると、高速な探索が可能
- Tの構築方法
  1. Tの外点← P
  2. Tの内点←探索を助けるための分割値
  3. 内点vの分割値をx_vとする
  4. 左部分木節点の値＜＝ x_v
  5. 右部分木節点の値＞ x_v

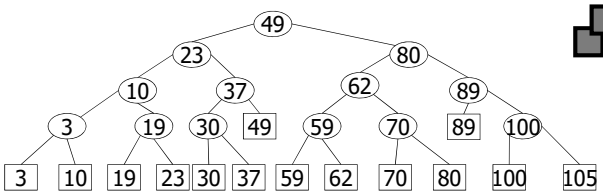
点集合と2分探索木

- 点集合$P=\{3,10,19,23,30,37,49,59,62,70,80,89,100,105\}$



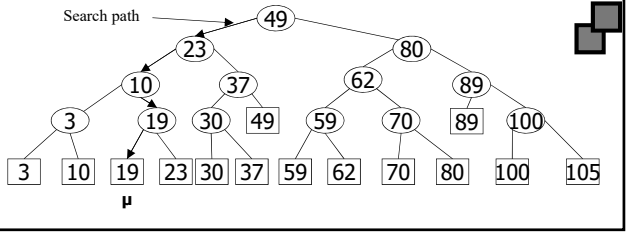
Query [18:77]

- 分岐節点V_{split}を探索する
- Report all values in left subtree on search path for μ
- Report all values in right subtree on search path for μ'
- 左右経路の先端にある外点を報告すべきか判定する



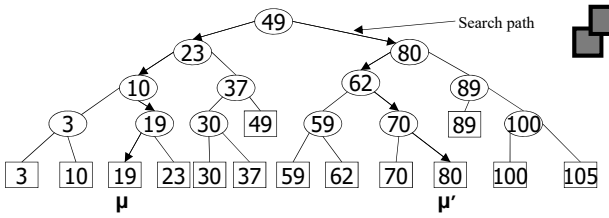
Query [18:77]

- 分岐節点V_{split}を探索する
- Report all leaf values in left subtree on search path for μ
- Report all leaf values in right subtree on search path for μ'
- 左右経路の先端にある外点を報告すべきか判定する



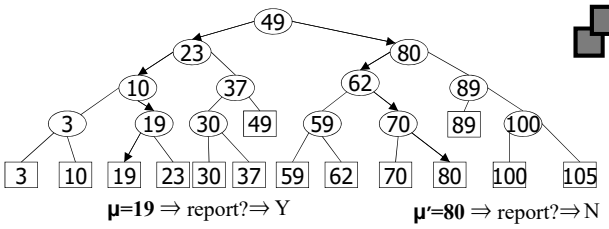
Query [18:77]

- 分岐節点V_{split}を探索する
- Report all leaf values in left subtree on search path for μ
- Report all leaf values in right subtree on search path for μ'
- 左右経路の先端にある外点を報告すべきか判定する



Query [18:77]

- 分岐節点V_{split}を探索する
- Report all values in left subtree on search path for μ
- Report all values in right subtree on search path for μ'
- 左右経路の先端にある外点を報告すべきか判定する



μ=19 ⇒ report?⇒ Y　　**μ'=80** ⇒ report?⇒ N

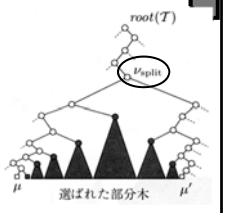
分岐節点を探索する

FindSplitNode(T, x_L, x_R)

- 入力：木Tと2つの値x_Lとx_R、ただし、$x_L \le x_R$
- 出力：①x_Lへの探索経路とx_Rへの探索経路が分岐する内点v、又は、②両方の経路が共に終了する外点v

1. $v \leftarrow$root(T)
2. while ($v \neq$leaf) and ($x_v \ge x_R$ or $x_v < x_L$)
3. do if $x_v \ge x_R$
4. then $v \leftarrow lc(v)$
5. else $v \leftarrow rc(v)$
6. return v

lc = left child
rc = right child



アルゴリズム

1DRangeQuery(T, x_L, x_R)

- 入力：木Tと2つの範囲値x_Lとx_R、ただし、$x_L \le x_R$
- 出力：指定された範囲内にある点集合

1. $v_{split} \leftarrow$FindSplitNode(T, x_L, x_R)
2. if v_{split}=leaf
3. then v_{split}に蓄えられる点を報告すべきか判定する
4. else
5. ① x_Lへの経路をたどり、経路の右側にある部分木の外点を報告し、経路の先端にある外点を報告すべきか判定する
6. ② x_Rへの経路の左側について①と類似な処理

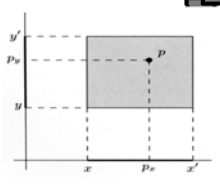
計算量

- 構築時間
  - 2分探索木→$O(n\log n)$
- 探索時間
  - 区間内にある点の数＝k
  - ReportSubTreeの計算時間＝ $O(k)$
  - FindSplitNodeの計算時間＝木の高さ＝ $O(\log n)$
  - 全体の時間＝$O(\log n + k)$

2次元領域探索

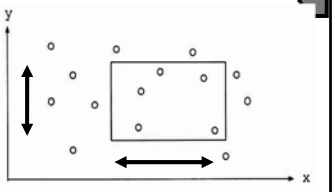
- 平面上の点集合

  $P=\{p_1, p_2, \ldots, p_n\}$、 $p_i=\{p_{ix}, p_{iy}\}$
- 長方形領域[$x : x'$]×[$y : y'$]に含まれる点集合を探索する
- 必要十分条件 $p_x \in$[$x : x'$]
  and $p_y \in$[$y : y'$]



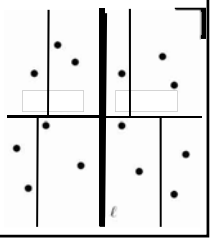
基本的な考え方

- $p_x \in [x : x']$ & $p_y \in [y : y']$→必要十分条件
- 点のx座標値と点のy座標値、それぞれについて、2つの1次元問題にすれば・・・？
- 2分探索木を使う
- But, how to …?
- 具体的に言えば…



基本的な考え方続き

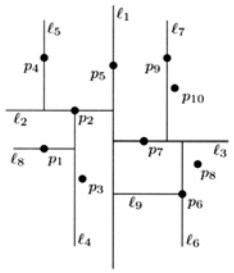
1. Pを2分割する垂直線l→根
2. P_{left}→左部分木、P_{right}→右部分木
3. P_{left}を水平線で2分割
   1. 上側→右部分木
   2. 下側→左部分木
4. P_{right}も同様
5. 根の孫節点では、再び垂直線で分割する
6. 垂直・水平分割を交互に繰返す

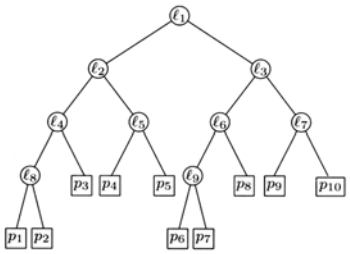


kd-木と対応する2分木

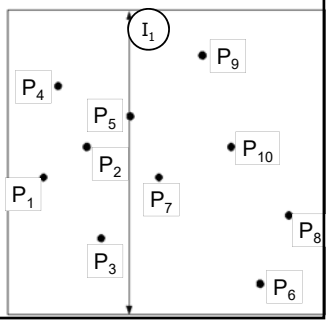
- kd＝k-dimensional

≤: left
>: right

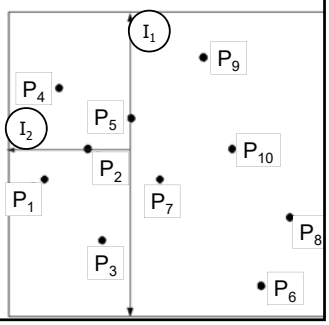




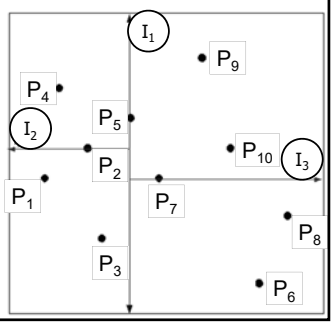
Build KD-Tree – I_1

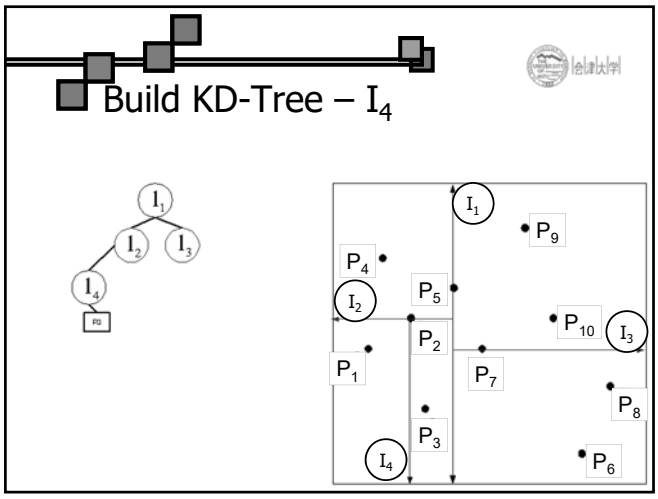


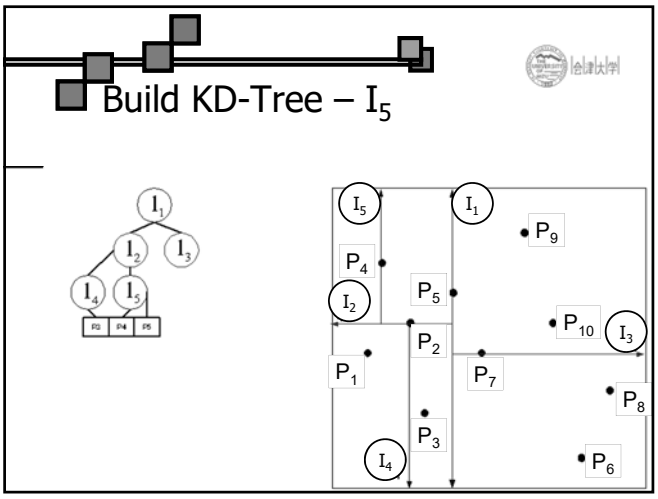
Build KD-Tree – I_2

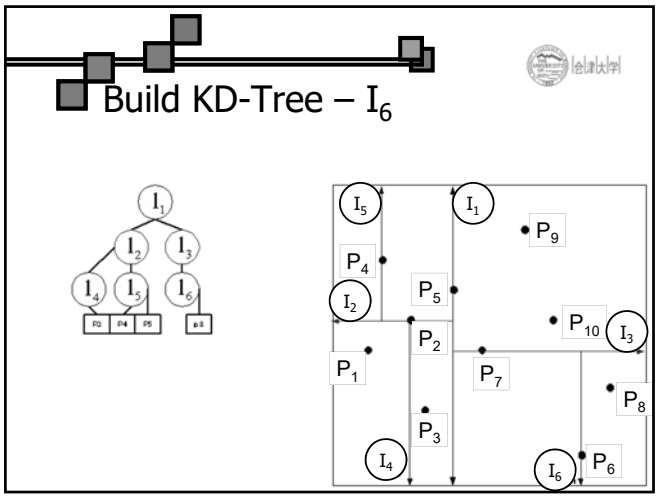


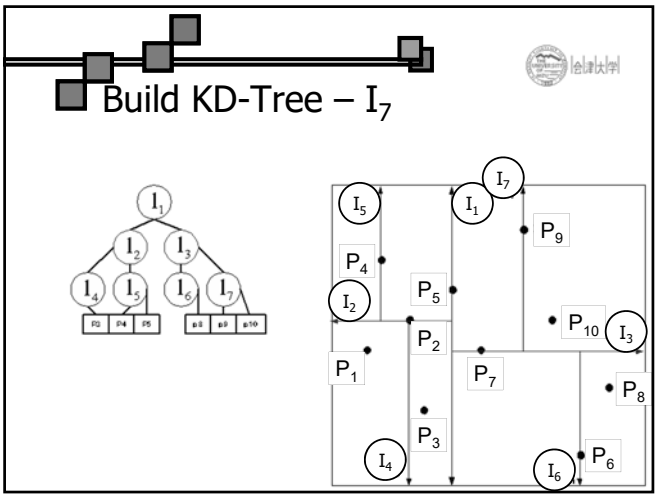
Build KD-Tree – I_3

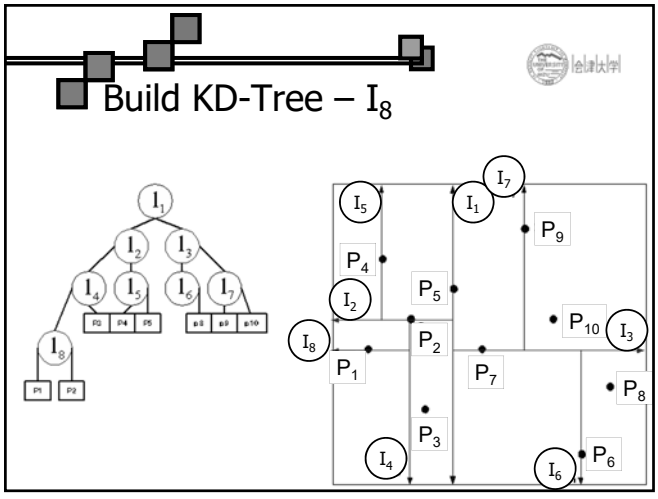


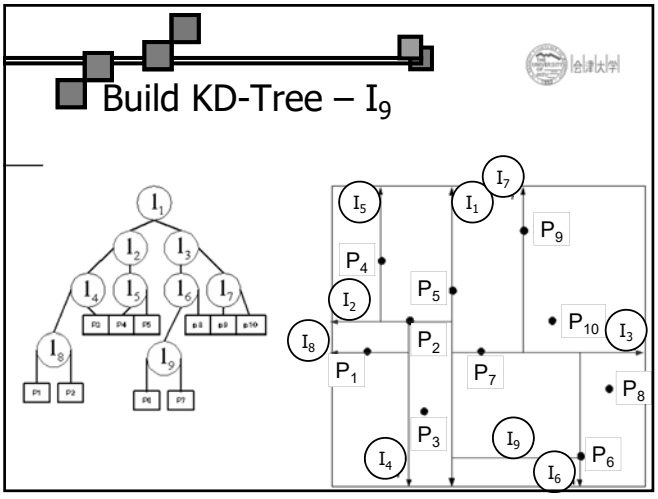

Build KD-Tree – I_4
I_1 I_2 I_3 I_4
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10}


Build KD-Tree – I_5
I_1 I_2 I_3 I_4 I_5
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10}

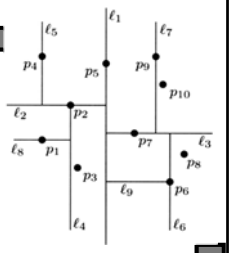

Build KD-Tree – I_6
I_1 I_2 I_3 I_4 I_5 I_6
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10}


Build KD-Tree – I_7
I_1 I_2 I_3 I_4 I_5 I_6 I_7
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10}


Build KD-Tree – I_8
I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10}


Build KD-Tree – I_9
I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8 I_9
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10}

アルゴリズム



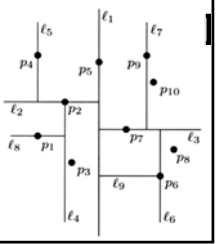
BuildKdTree(P, $depth$)

- 入力:点集合P、木の深さ$depth$
- 出力: Pを蓄えるkd木の根

1. if Pの点数=1 // 一点のみの場合
2. then return その点を蓄えている外点
3. else
4. if $depth$ = even // count from 0
5. then // split with a vertical line (垂直分割)
6. Pのx座標値の中央値x_{median}を通る垂直線lでPを2分割
   →左側P_1点集合のx座標値$\leqq x_{median}$
   →右側P_2点集合のx座標値$> x_{median}$

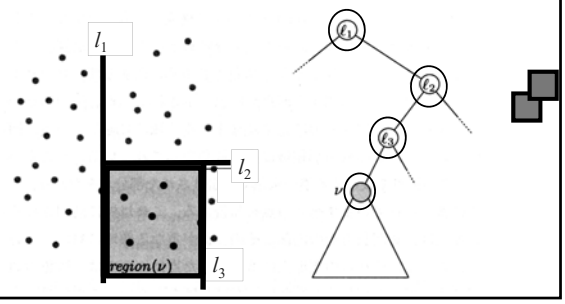
アルゴリズム続き

7. else // split with a horizontal line (水平分割)
8. Pのy座標値の中央値y_{median}を通る水平線lでPを2分割
   →下側P_1点集合のy座標値$\leqq y_{median}$
   →上側P_2点集合のy座標値$> y_{median}$
9. v_{left}←BuildKdTree(P_1, $depth$+1)
10. v_{right}←BuildKdTree(P_2, $depth$+1)
11. lを蓄えている節点vを作り、
    vの左子＝v_{left}、vの右子＝v_{right}
12. return v



kd-木と平面領域の対応関係

- 根l_1⇔全平面
- 木の節点v⇔平面領域$region(v)$



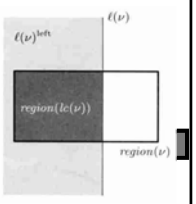
節点vと質問領域R

- 主な判定は、質問領域Rがある節点vに対応する領域と重なりを持つかどうかの判定。
- 垂直分割（深さが偶数）である節点vの左子に対応する領域は、region(v)を用いると、次のような関係が成立。

$$region(lc(v)) = region(v) \cap l(v)^{left}$$

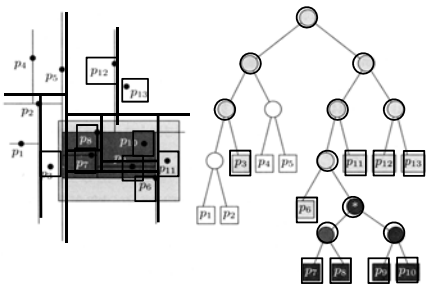
- 水平分割（深さが奇数）である節点vの左子に対応する領域は、region(v)を用いると、次のような関係が成立。

$$region(lc(v)) = region(v) \cap l(v)^{lower}$$

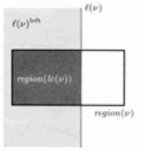


kd-木での平面領域探索

■=query range, ●=visited nodes
●=completely contained nodes in the query rectangle



Basic Steps in Query



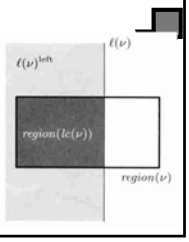
- Visit only nodes whose region is intersected by the query rectangle
- When a region is fully contained in the query rectangle, report all the points stored in its subtree
- When a region is partially contained in the query rectangle, check if the points stored in its subtree should be reported or not
- When the traversal reaches a leaf, check if the point stored at the leaf is contained in the query rectangle and, if so, report it

アルゴリズム

SearchKdTree(v, R)

- 入力:kd木の根（部分木の節点）vと質問領域R
- 出力:この範囲に入っているvの下にある全ての外点

1. if v = leaf
2. then v に蓄えられた点を調べ、Rの内部にあれば、報告
   else
3. if *region*(lc(v))がRに完全包含
4. then ReportSubtree(lc(v))
5. else if *region*(lc(v))がRと部分包含
6. then SearchKdTree(lc(v), R)
7. if *region*(rc(v))がRに完全包含
8. then ReportSubtree(rc(v))
9. else if *region*(rc(v))がRと部分包含
10. then SearchKdTree(rc(v), R)



計算量

- n =点集合の総点数
- k =報告すべき領域内の点数
- 記憶Storage = $O(n)$
- 構築Construction = $O(n\log n)$
  - 分割線探索→xとyについて2分探索木を用意し、交互に中央値を再帰取り出す
  - 計算量$T(n)$の漸化式

$$T(n)=\begin{cases}O(1), & n=1\\ O(n)+2T\left(\left\lceil\frac{n}{2}\right\rceil\right), & n>1\end{cases}$$

中央値を求める

- 探索Query = $O(\mathrm{sqrt}(n)+k)$
  - 水平と垂直分割線をそれぞれ用いて再帰探索→一つ領域に高々「$n/4$」個の点をしか含まない
  - 計算量$T(n)$の漸化式

$$T(n)=\begin{cases}O(1), & n=1\\ 2+2T\left(\left\lceil\frac{n}{4}\right\rceil\right), & n>1\end{cases}$$

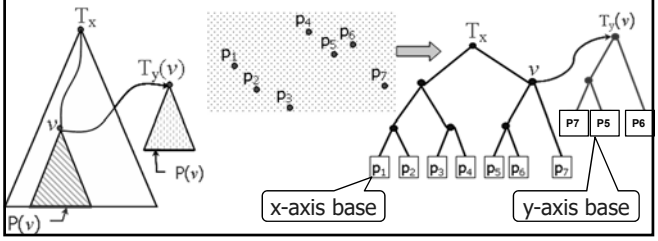
左右上下探索を含む

kd-木と領域木(Range tree)

- kd-木: x、y座標交互に
  - ① 2つ1次元領域探索
  - ② x、y座標値について交互に分割
- 領域木: x、y座標順番に
  - ① x座標値について→1次元領域探索
  - ② y座標値について→1次元領域探索
- 共通点＝2次元領域探索→2つ1次元問題

領域木の構築

- x座標に基づき構築した木　Tx
- 節点vを根とする部分木の葉の点集合　P(v)
- y座標に基づき点集合P(v)を構築した木　Ty(v)
- Pointer from v to Ty(v)



アルゴリズム(Build)

Build2DRangeTree(P)

- 入力:点集合P
- 出力:2次元領域木(根)

1. Construct T_y for P (store entire points at leaves)
2. if number of (P)=1 // 一点のみの場合
3. then この点を蓄えている外点vを生成、Ty(v):= Ty
   else
4. x座標の中央値x_{mid}を基準にPをP_{left}とP_{right} に2分割
5. v_{left}← Build2DRangeTree(P_{left})
6. v_{right}←Build2DRangeTree(P_{right})
7. 新節点vを生成し、 x_v←x_{mid}を蓄える
8. Leftchild(v)←v_{left}
9. Rightchild(v)←v_{right}
10. Ty(v):=Ty
11. return v

アルゴリズム(Query)

Query2DRangeTree(T, [x : x'] × [y : y'])

- 入力:2次元領域木Tと探索領域[x : x'] × [y : y']
- 出力:この領域内にある点集合

1. v_{split}←FindSplitNode(T, x, x')
2. if v_{split}=leaf
3. then
   v_{split}に蓄えられる点を報告すべきか判定する
4. else //xへの経路をたどり、経路の右側にある部分木Tyに関してQuery1DRangeTreeを呼び出す
5. v=LeftChild(v_{split})

アルゴリズム(Query)続き

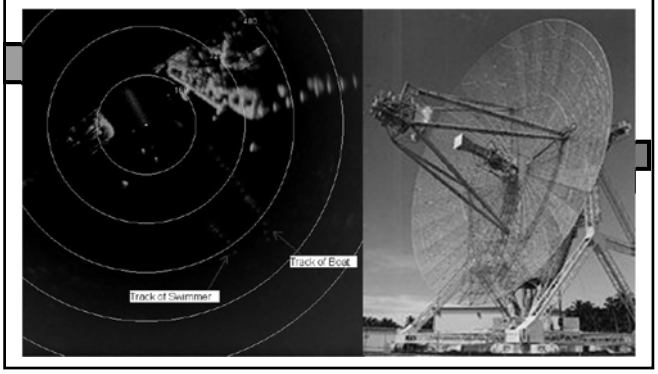
```
6.   while  v != Leaf  do
7.     if  x_v≧x
8.       then  Query1DRangeTree(Ty(RightChild(v)), [y : y'])
9.             v=LeftChild(v)
10.      else  v=RightChild(v)
11.  vに蓄えられる点を報告すべきか判定する
     //RightChild(v_split)からx'への経路の左側にある部分木Tyに
      関してQuery1DRangeTreeを呼び出す
12.  v=RightChild(v_split)
13.  while  v != Leaf  do
14.    if x_v≦x'
15.      then Query1DRangeTree(Ty(LeftChild(v)), [y : y'])
16.           v=RightChild(v)
17.      else  v=LeftChild(v)
18.  vに蓄えられる点を報告すべきか判定する
```

計算量

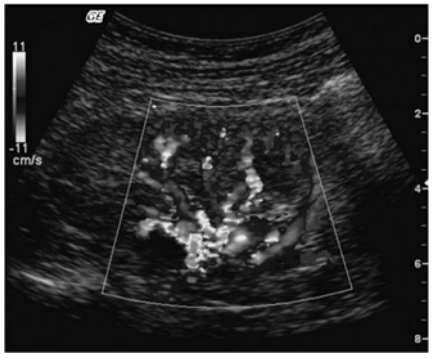
- n =点集合の総点数
- k =報告すべき領域内の点数
- 記憶Storage = $O(n\log n)$
- 構築Construction = $O(n\log n)$
  - x座標に関する2分探索木の構築$O(n\log n)$
  - 各$T_{assoc}(v)$でy座標に関する2分探索木の構築→
    一つの深さ=$O(n)$、すべての深さ=$O(n\log n)$
- 探索Query = $O((\log n)^2 + k)$
  - ノードvに対し1DRangeQuery(rc(v), [$y : y'$]) を呼び出すときの計算量 $O\left(\log \frac{n}{2^{depth}} + k_v\right)$
  - 総計算量 $\sum_{i=0}^{\log n}\left(\log \frac{n}{2^i} + k_v\right) = \sum_{i=0}^{\log n}\log \frac{n}{2^i} + \sum_{i=0}^{\log n} k_v = \sum_{i=0}^{\log n}\log n - \sum_{i=0}^{\log n} i + O(k)$
    $= (\log n)^2 - \frac{\log n(\log n+1)}{2} + O(k) = O\left((\log n)^2 + k\right)$

応用－GIS

応用－軍事



応用－医学



応用－天気予報