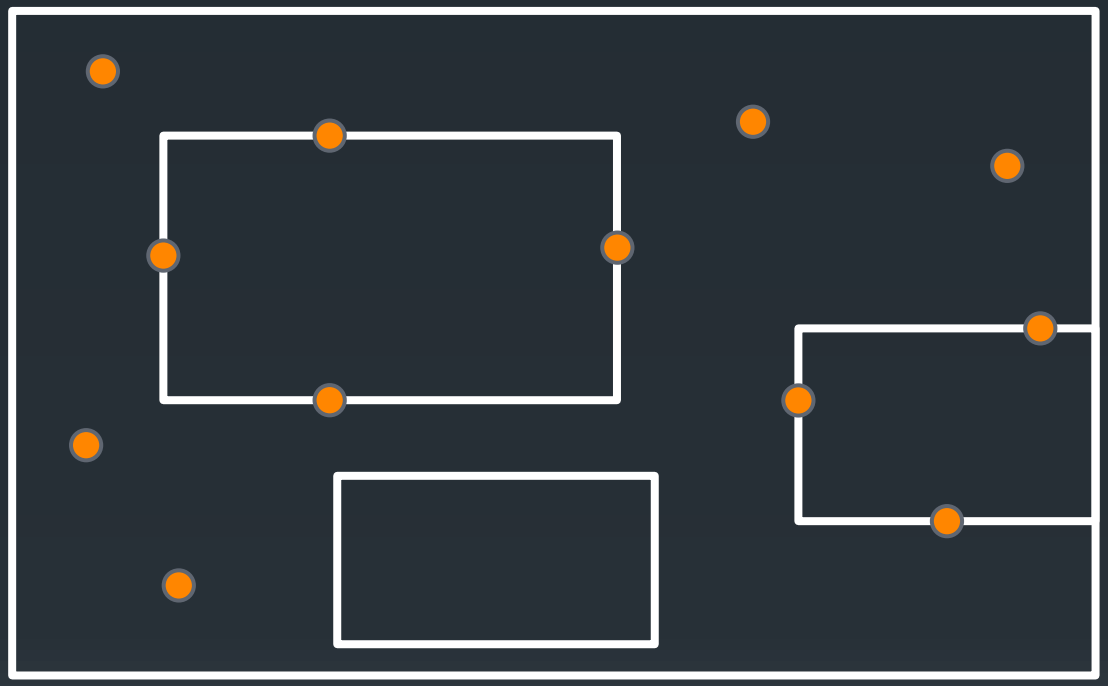
COMPUTING THE LARGEST EMPTY RECTANGLE

B.Chazelle, R.L.Drysdale and D.T.Lee

SIAM J. COMPUT
Vol.15 No.1, February 1986

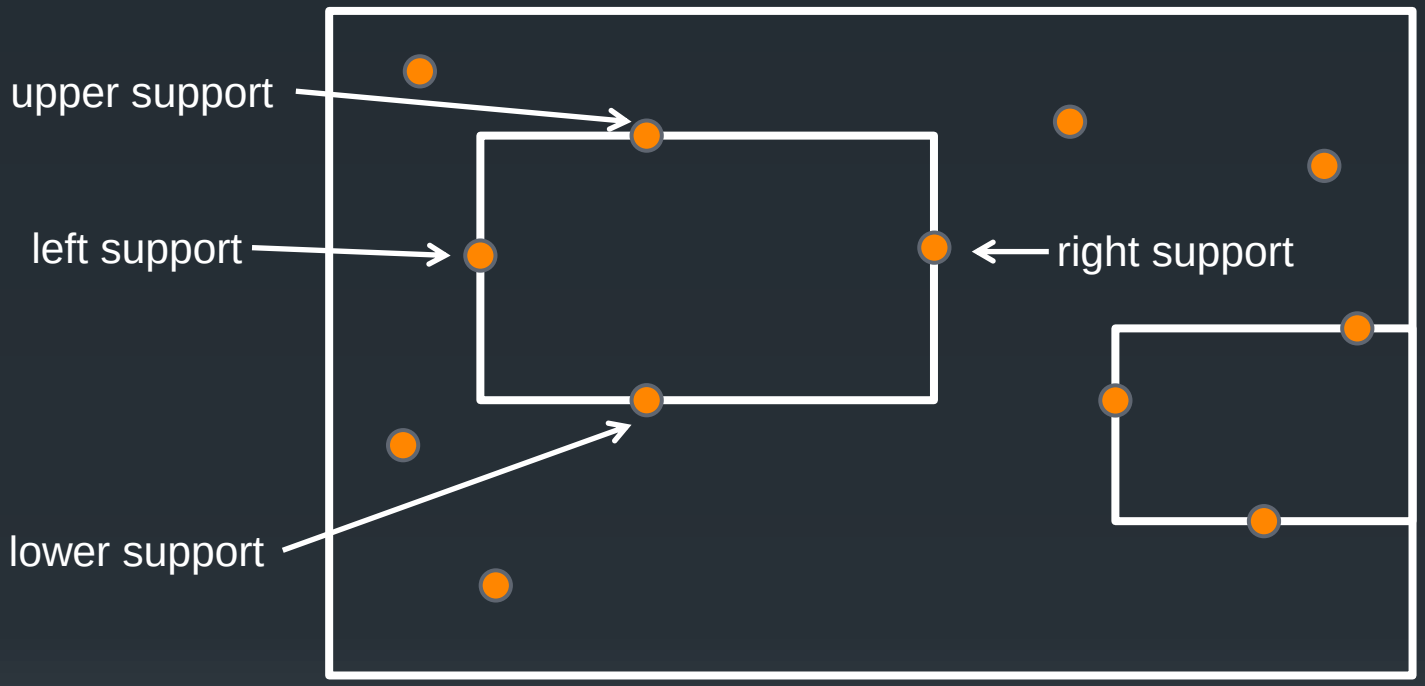
2012.7.12 TCS講究
関根　渓（情報知識ネットワーク研究室　M1）

Empty rectangle



- 内部にN個の点を含む領域長方形(bounding rectangle)が与えられたとき，その内部に形成できる長方形で，かつ領域長方形の辺に全ての辺が平行で，かつ内部に点を含まないもの．
- 辺上に点が存在したり，辺が領域長方形の辺と重なっていてもよい．

Largest empty rectangle problem

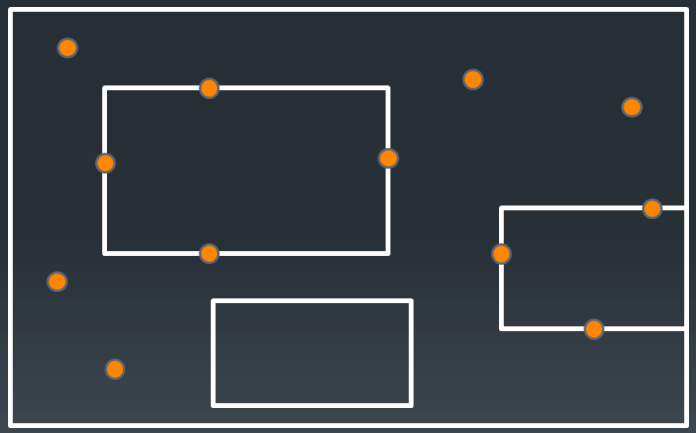


- 入力：内部にN個の点を含む bounding rectangle
- 出力：領域長方形内に形成できる最大（面積）の empty rentangle

- それぞれの辺が領域長方形の辺または最低1つの点で支えられる

Application

- 実世界への応用
  - 素材の再利用
    - 布や金属板の欠片＝領域長方形
    - 素材に付いた傷＝点集合



- **学問への応用**
  - **画像分割**
    - 画像処理
    - パターン認識

General approach and Related works

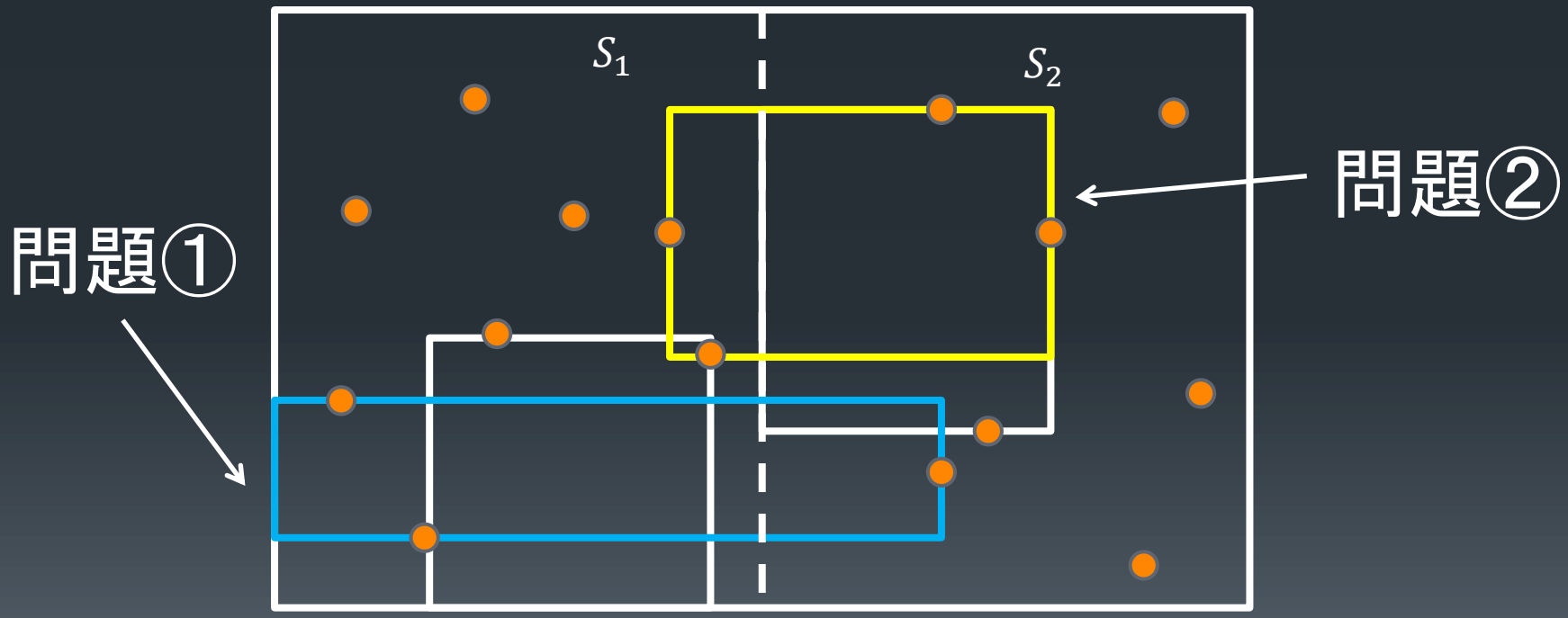
- 単純な方法では $O(N^5)$ の時間計算量
  - 4つの support の選択が $O(N^4)$ 通り
  - 各組み合わせに対して内部に点が含まれるかのチェックに $O(n)$

- A. NAAMAD, W. L. Hsu AND D. T. LEE, On maximum empty rectangle problem, Appl. Disc. Math., 8 (1984), pp. 267-277.
  - $O(N^2)$ の最悪時間計算量，$O(Nlog^2N)$ の平均時間計算量

- 本論文
  - 手法1
    - $O(Nlog^4N)$ の時間計算量，$O(NlogN)$ の領域計算量
  - 手法2（手法1を改良）
    - $O(Nlog^3N)$ の時間計算量，$O(NlogN)$ の領域計算量

Approach in this paper

- 分割統治法を用いる
  - 点集合Sを $S_1 = \{p_1, \cdots, p_{\lfloor N/2 \rfloor}\}$ と $S_2 = \{p_{\lfloor N/2 \rfloor+1}, \cdots, p_N\}$ に分割し，S_1と S_2 における部分問題を解き，統合する.



Approach in this paper

- アルゴリズムの流れ
  1. S_1とS_2，各領域に4つのサポートを持つ largest empty rectangle を計算する.
  2. S_1とS_2のどちらか片方に3つ，もう片方に1つのサポートを持つ largest empty rectangle を計算する.（問題①）
  3. S_1とS_2の両方に2つずつサポートを持つ largest empty rectangle を計算する.（問題②）
  4. これらを比べていき，最大なものをS における解とする

- 計算時間
  - アルゴリズム全体の計算時間は問題①，問題②の計算時間による

$$T(N) \leq 2T(N/2) + C(N) + D(N) \quad (1)$$

Approach in this paper

- アルゴリズムの流れ
  1. S_1とS_2，各領域に4つのサポートを持つ largest empty rectangle を計算する.
  2. S_1とS_2のどちらか片方に3つ，もう片方に1つのサポートを持つ largest empty rectangle を計算する.（問題①）
  3. S_1とS_2の両方に2つずつサポートを持つ largest empty rectangle を計算する.（問題②）
  4. これらを比べていき，最大なものをS における解とする

全体の計算量
$O(Nlog^4N)$（手法1）
$O(Nlog^3N)$（手法2）

本(

問題①
$O(N)$

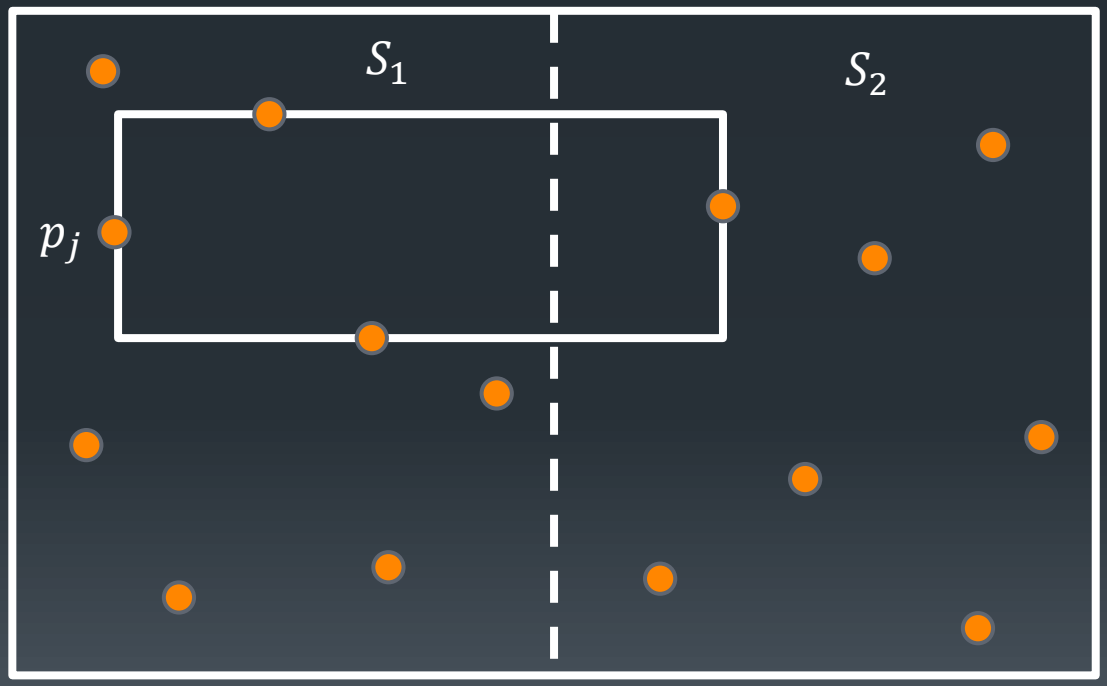
), 問

問題②
$O(Nlog^3N)$（手法1）
$O(Nlog^2N)$（手法2）

$$T(N) \leq 2T(N/2) + C(N) + D(N) \quad (1)$$

Three supports in one half, one in the other

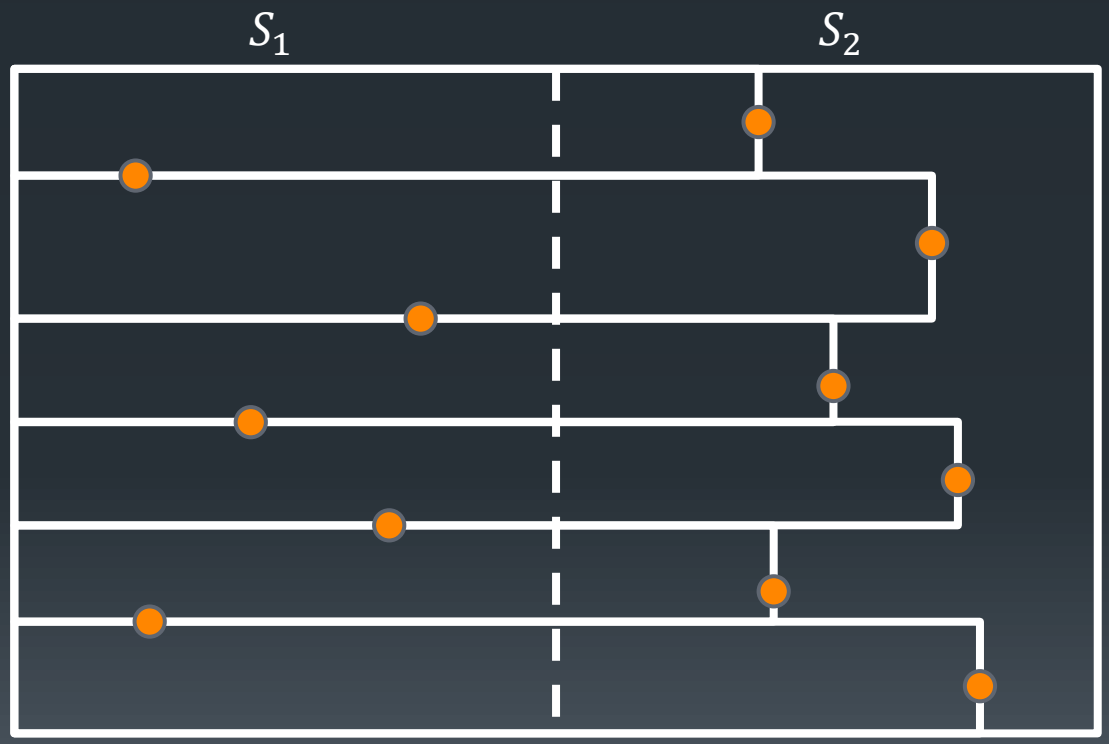
- 特徴
  - left support が与えられると長方形の形が決定する.



  - left support が点である empty rectangle は $O(N)$ 個

Three supports in one half, one in the other

- 特徴
  - left support が与えられると長方形の形が決定する.



  - left support が bounding rectangle の左辺であるものの数は $\lfloor N/2 \rfloor + 1 = O(N)$個

Three supports in one half, one in the other

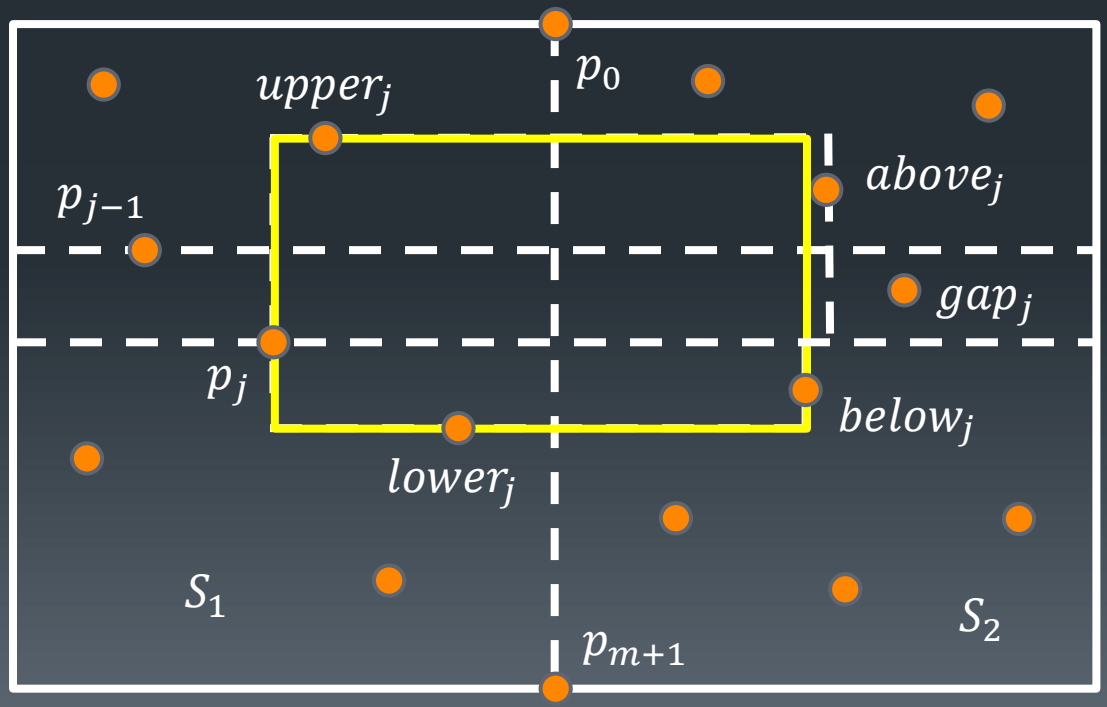
- 特徴
  - left support が与えられると長方形の形が決定する.

おおよそ$O(N)$個の長方形を考えればよい

  - left support が bounding rectangle の左辺であるものの数は $\lfloor N/2 \rfloor + 1 = O(N)$個

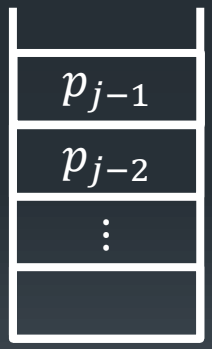
Three supports in one half, one in the other

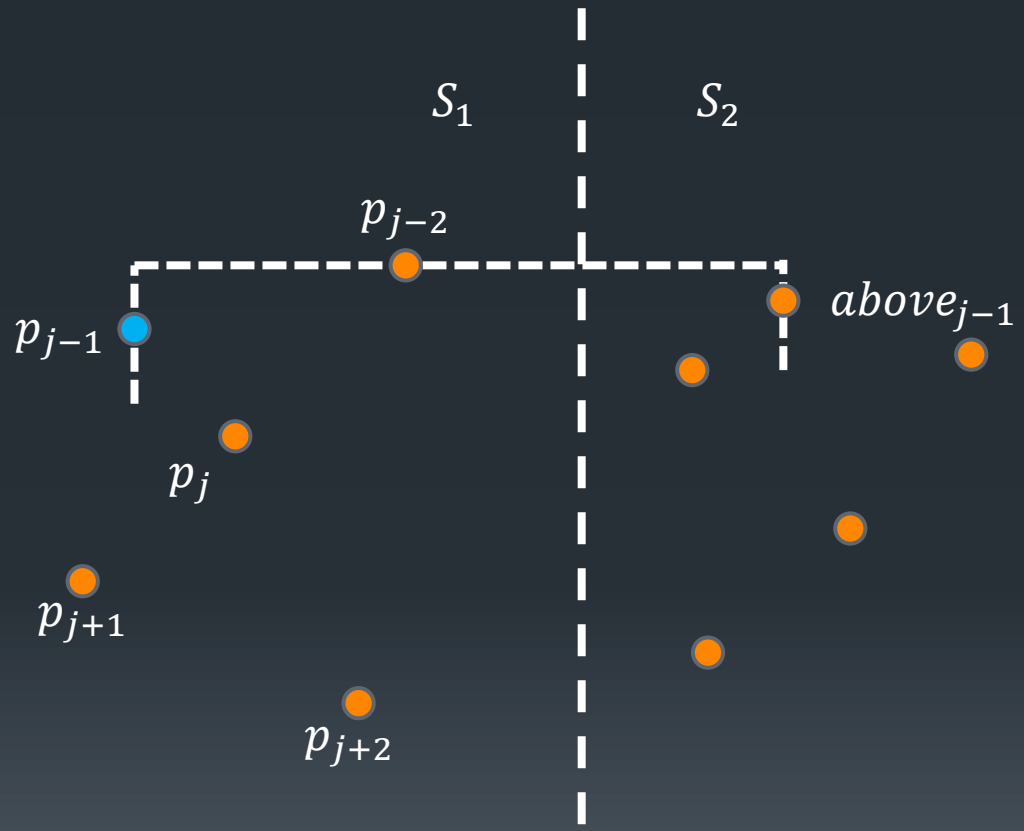
- アルゴリズム
  - 単純な方法では $O(N^2)$ の時間計算量
  - 本論文の提案アルゴリズム
    - スタックを用いて，$O(N)$ の時間計算量で empty rectangle を計算.



Three supports in one half, one in the other

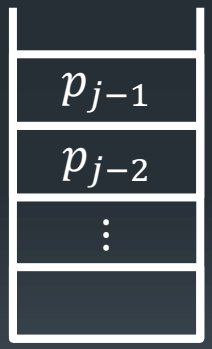
- アルゴリズム
  - 動作

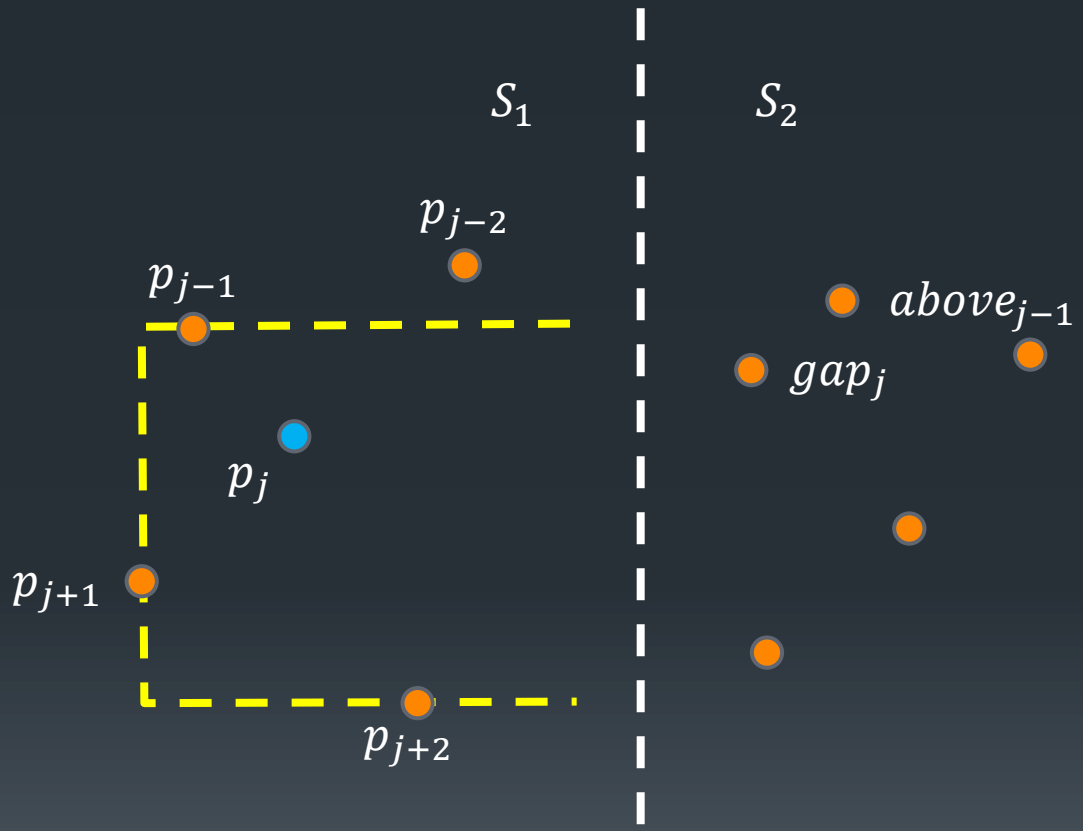




Three supports in one half, one in the other

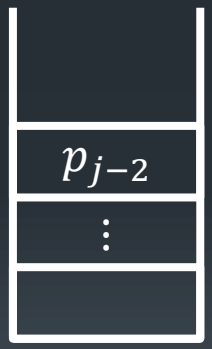
- アルゴリズム
  - 動作

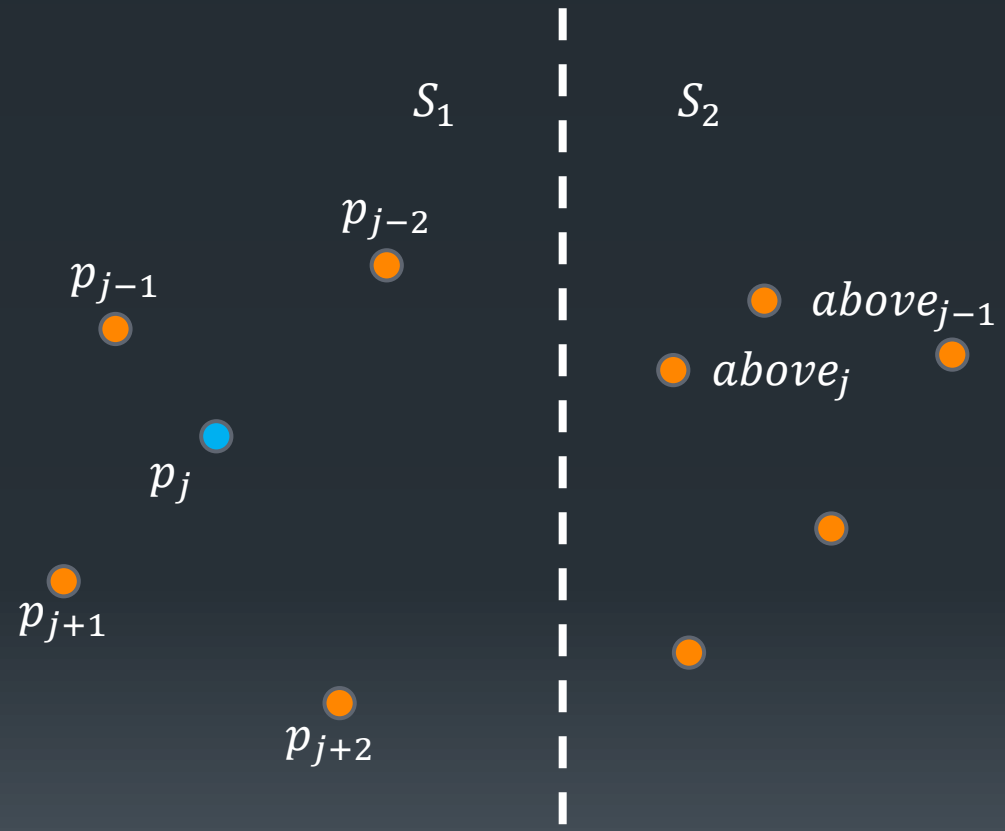




Three supports in one half, one in the other

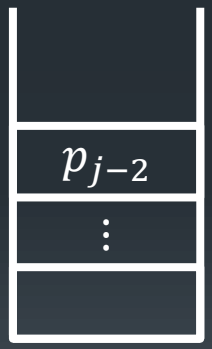
- アルゴリズム
  - 動作

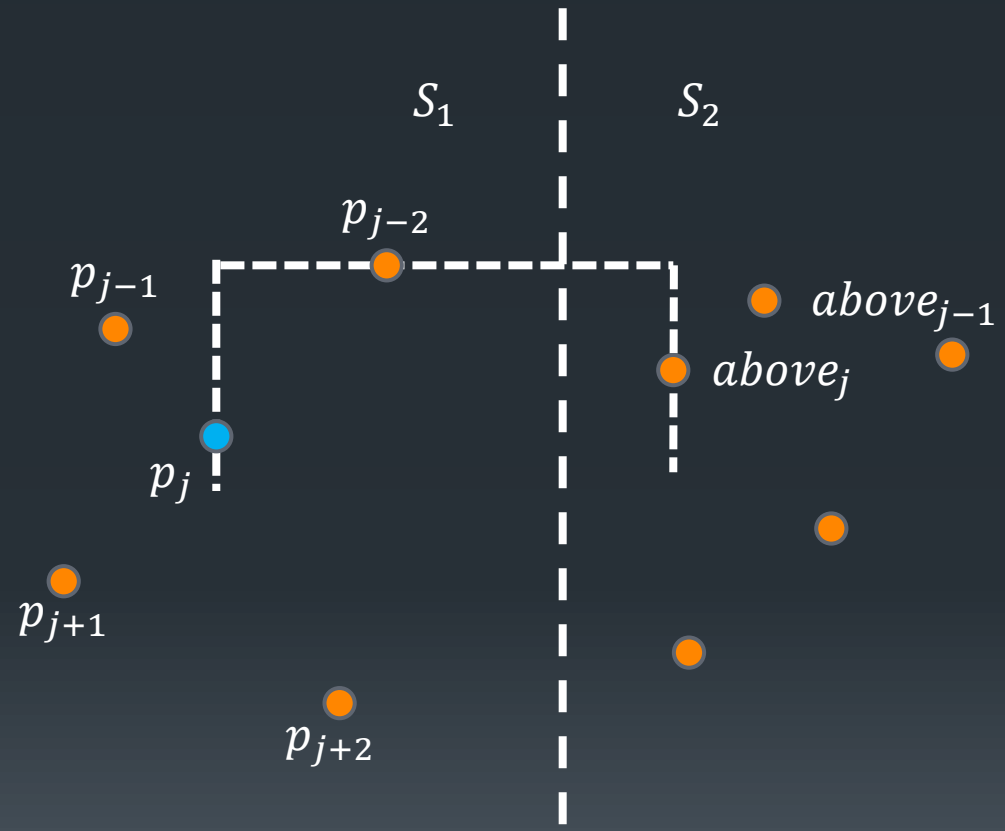




Three supports in one half, one in the other

- アルゴリズム
  - 動作





一度 POP された点はスタックに再び PUSH されることはないので,
スタックに対する操作は合計で$O(N)$回

Three supports in one half, one in the other

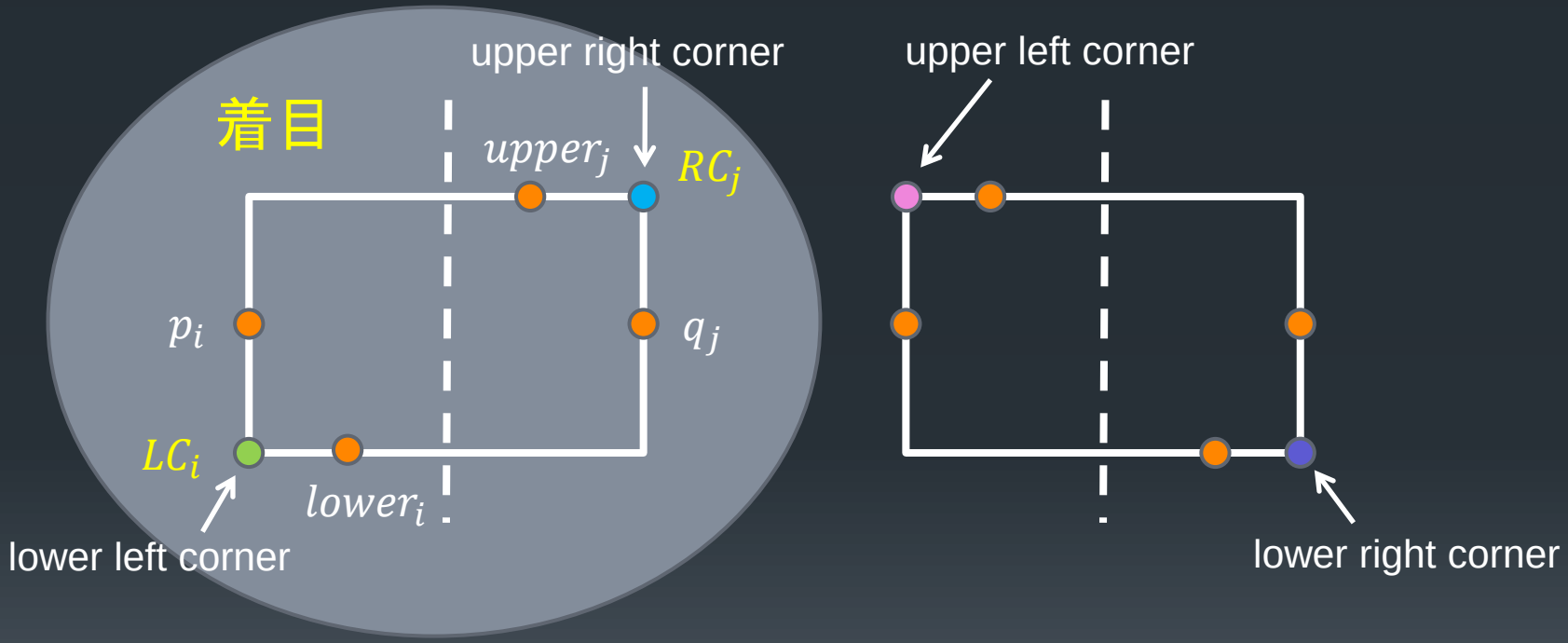
- アルゴリズム
  - 擬似コード ($upper$ と $above$ を計算する部分のみ)

1. $top \leftarrow p_0$
2. $above(top) \leftarrow q_0(= (x'_{max}, y_{max}))$
3. **for** each p_j of S
4. **if** $x(gap_j) < x(right_j)$
5. **then** $above_j \leftarrow gap_j$
6. **else** $above_j \leftarrow right_j$
7. **while** $x(top) \leq x(p_j)$ do
8. **if** $x(above_j) \geq x(above(top))$
9. **then** $above_j \leftarrow above(top)$
10. **Pop the stack.**
11. $upper_j \leftarrow top$
12. **Push** p_j **onto the stack.**

  - 補題1

分割された領域の半分に3つの support を，もう半分に 1つの support を含む，N点についての the largest empty rectangle の 計算時間 $C(N)$ は $O(N)$ である.

全てのあり得る lower left corner と upper right corner を計算し，そのペアによって形成される empty rectangle の中で最大なもの largest empty corner rectangle（LECR）を見つければよい.



$p_i = (x_i, y_i), lower_i$ によって決まる corner point を LC_i とする
$CL = \{LC_1, LC_2, \cdots, LC_s\}$ を形成 (右半分はRC_j, CR)
$\rightarrow CL, CR$は $O(N)$ 時間で構成可能

Two support in each half

- ペアリング条件 (pairing condition)

left support が p_l で，bottom support が p_b の LC_i は，top support が p_l より高く，right support が p_b より高くなるような RC_j としかペアになれない.

- 補題2

どの入力点も，the largest empty rectangle の corner point にならない．さらに，CL と CRに含まれる点は，上記のペアリング条件を満たす.

- 補題3

corner rectangle がその内部にどんな入力点も含まないなら，新しく作られたどの corner point もその内部には含まれない.

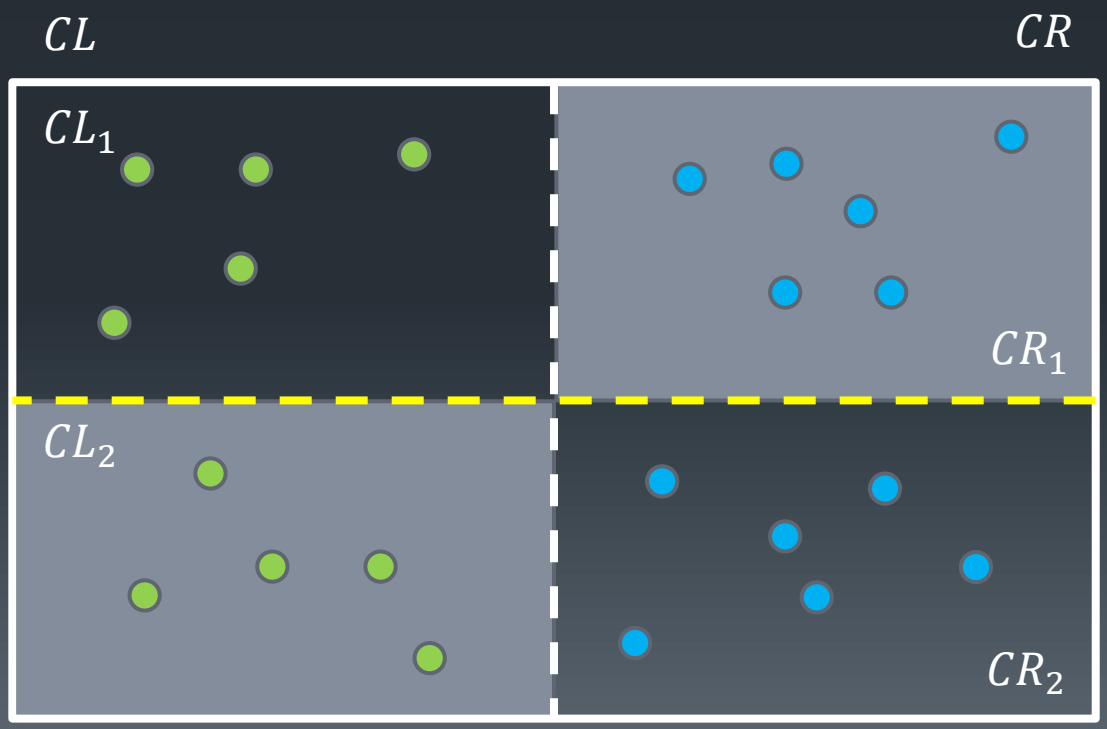
Comput

$$T(N) \leq 2T(N/2) + C(N) + D(N) \quad (1)$$

- アルゴリズム（手法1）
  - 分割統治法を用いる
    - 計算時間

$O(Nlog^2N)$

$$D(N) \leq 2D(N/2) + E(N) \quad (2)$$



Computation the largest empty rectangle

- 再帰ステップの手順（手法1）

corner point 集合CL, CR をCL_1, CL_2とCR_1, CR_2に分割 $O(N)$

↓

CL_2 の点P それぞれについて，
CR_1中のペアになり得る点の集合 S を求める $O(N)$

↓

集合 S それぞれについて LL-diagram, $LL(S)$ を計算 $O(NlogN)$

↓

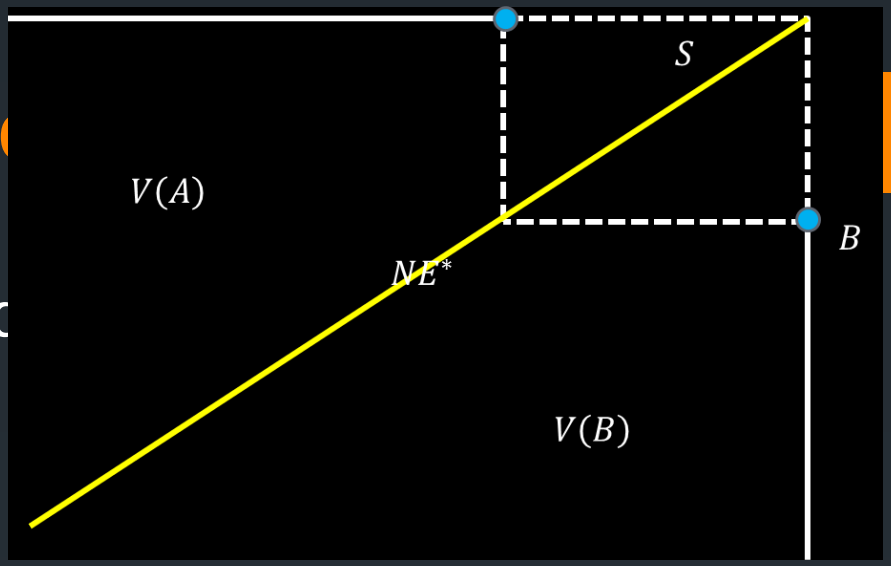
CR_1の区分木Tを構築 $O(NlogN)$

↓

CL_2の点Pそれぞれに関する$LECR$を計算 $O(Nlog^2N)$

以上から，$E(N) = O(Nlog^2N)$

Computation the large



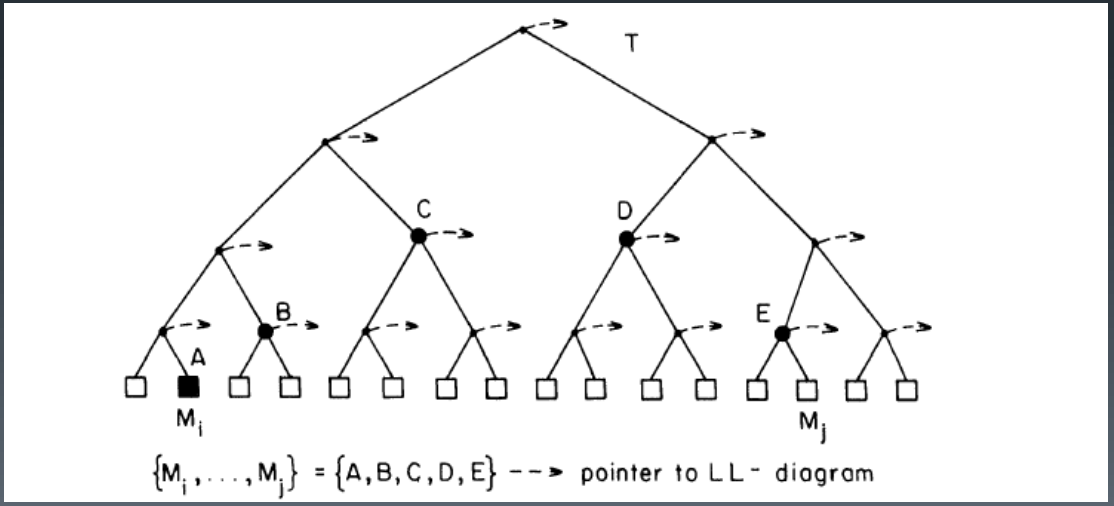
- LL-diagram (Lower-Left-d
  - Voronoi diagramに似た概念
  - 点集合 $S = \{M_1, M_2, \cdots, M_N\}$
  - 定義

$$LL(S) = \{V(M_1), V(M_2), \cdots, V(M_N)\}$$
$$V(M_i) = \{M \in NE^* | d(M, M_i) \geq d(M, M_i) \text{ for all } j = 1,2, \cdots, N\}$$

- $d(M, M_i)$ は点 M, M_i 間のd −距離といい，点M, M_i 間のユークリッド距離である.
- NE^*は $(-\infty, x_{max}) \times (-\infty, y_{max})$（ただし点集合$S$を取り囲む最小長方形のエリアを除く）で表される領域である.

Computation the largest empty rectangle

- 区分木（segment-tree）
  - LL-diagram を保持するのためのデータ構造
    - $CR_1 = \{M_1, M_2, \cdots, M_u\}$の各点をこの順に，完全二分木の左から右の葉とする.
    - 部分木の根は，その部分木の葉の連続する部分集合を表す.
      - 連続する葉の列の集合を $SL = \{\{M_1\}, \{M_2\}, \cdots, \{M_1, M_2\}, \cdots, \{M_1, M_2, \cdots, M_u\}\}$とおく
    - 任意の区分 $[M_i, M_j]$ を表す節点集合を $O(logu)$ 時間でみつける.



Computation the largest empty rectangle

- 計算量の吟味（手法1）

$$D(N) \leq 2D(N/2) + E(N) \quad (2)$$

において,

$E(N) = O(Nlog^2N)$ より,

$D(N) = O(Nlog^3N)$.

さらに,

$$T(N) \leq 2T(N/2) + C(N) + D(N) \quad (1)$$

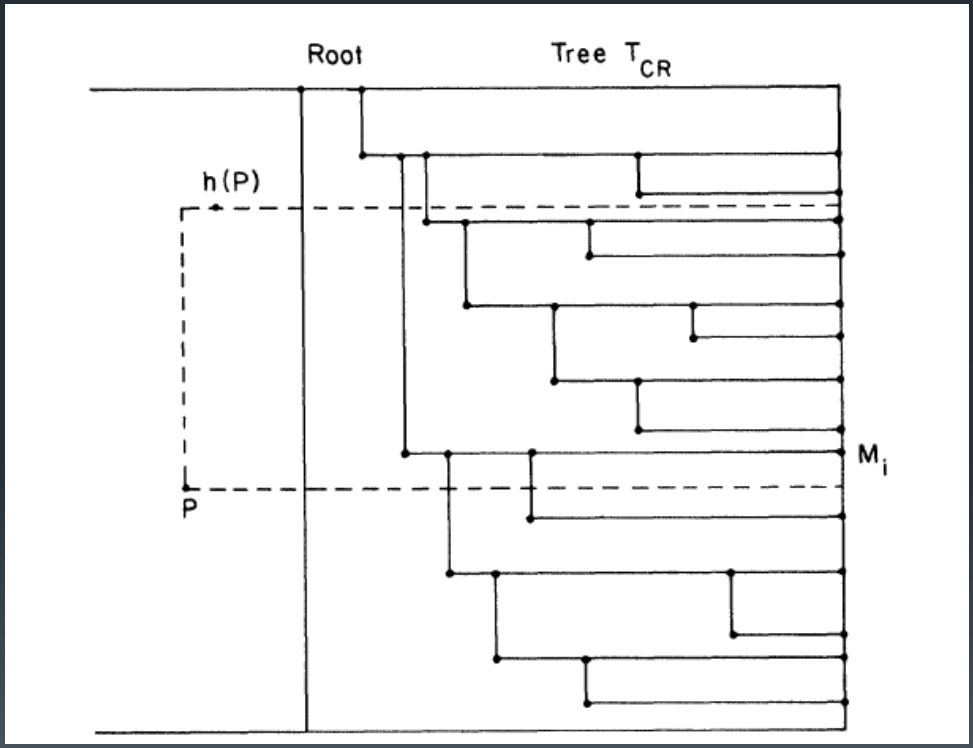
において,

$C(N) = O(N)$ より,

$T(N) = O(Nlog^4N)$.

An improved algorithm for computing LECR

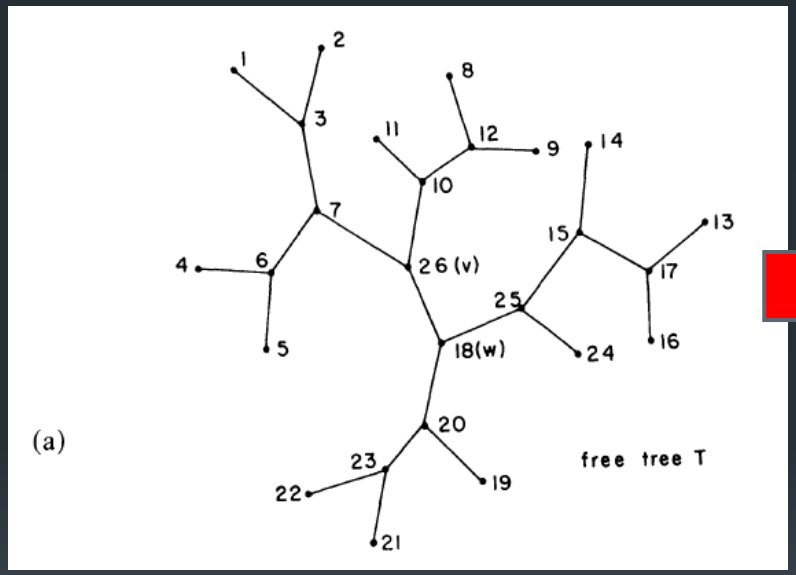
- 方針（手法2）
  - 水平方向への再帰を無くし，冗長さを減らした表現へ

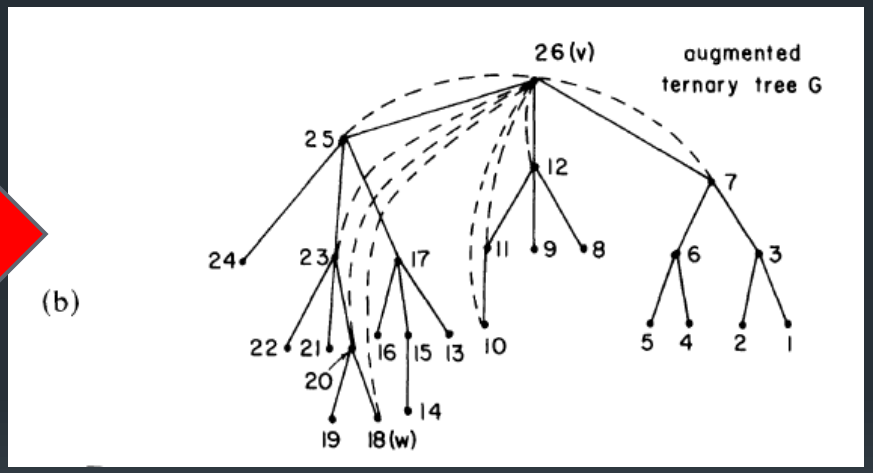


  - あるルールに基づいて，CRの点をノードとする木T_{CR}を構成する

An improved algorithm for computing LECR

- 方針（手法2）
  - 木T_{CR}を，あるルールに従い特殊な三分木Gに変換.



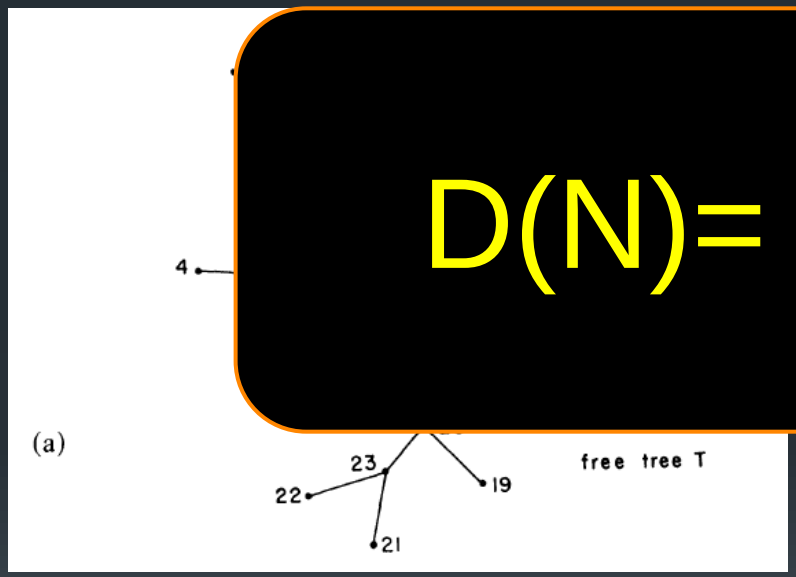


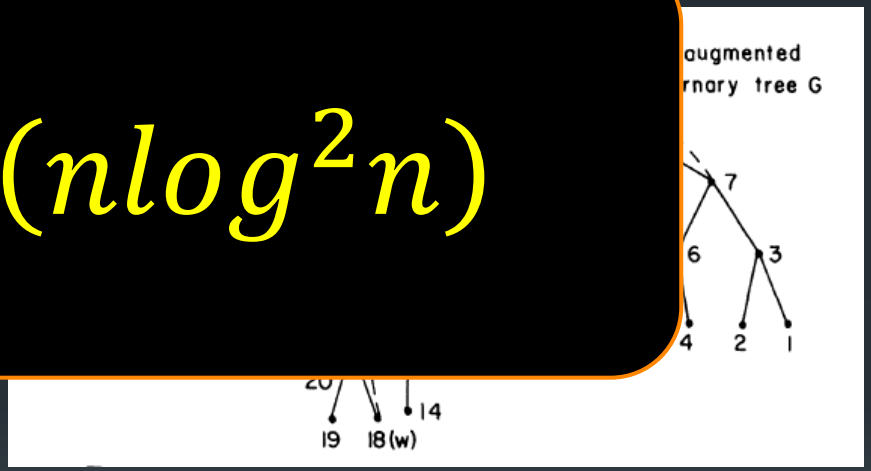
  - T_{CR}の任意の連続するパスを，Gから$O(logn)$時間で取り出せる
  - T_{CR}の構成，Gへの変換は合計で $O(nlog^2n)$ 時間.
  - CLの点Pそれぞれに対してLECRを求めるのに $O(nlog^2n)$

An improved algorithm for computing LECR

- 方針（手法2）
  - 木T_{CR}を，あるルールに従い特殊な三分木Gに変換.

$$D(N)= O(nlog^2n)$$





- T_{CR}の任意の連続するパスを，Gから$O(logn)$時間で取り出せる
- T_{CR}の構成，Gへの変換は合計で $O(nlog^2n)$ 時間.
- CLの点Pそれぞれに対してLECRを求めるのに $O(nlog^2n)$

Computation the largest empty rectangle

- 計算量の吟味（手法2）

$$D(N) = O(Nlog^2N).$$

さらに,

$$T(N) \leq 2T(N/2) + C(N) + D(N) \quad (1)$$

において,

$C(N) = O(N)$ より,

$$T(N) = O(Nlog^3N).$$

まとめ

- Largest empty rectangle probrem を解く，以下のアルゴリズムを与えた.
  - 手法1
    - $O(nlog^4n)$ の時間計算量，$O(nlogn)$ の領域計算量
  - 手法2
    - $O(nlog^3n)$ の時間計算量，$O(nlogn)$ の領域計算量