

# アルゴリズムとデータ構造⑦

## ～ 探索問題（二分探索木）～

鹿島久嗣

# 探索問題：

データ集合から所望の要素を見つける

---

- 探索問題は、データ集合から特定のデータを見つける問題
  - データは「キー」と「内容」からなる
  - 与えられたキーに一致するキーをもったデータを見つけ、その内容を返す
- これは、**ハッシュ**や**二分探索木**等によって実現可能

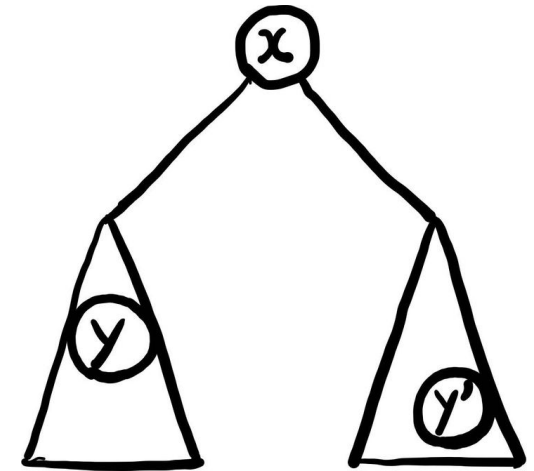
今回の話題

# 二分探索木

# 二分探索木：

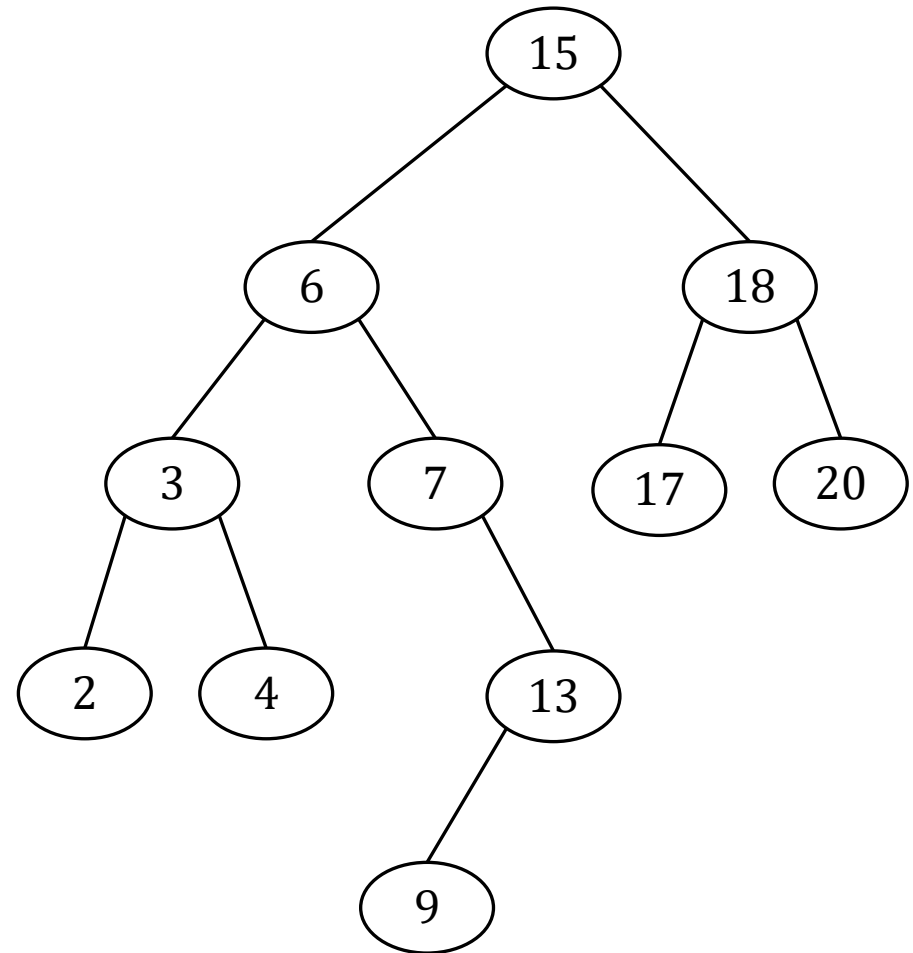
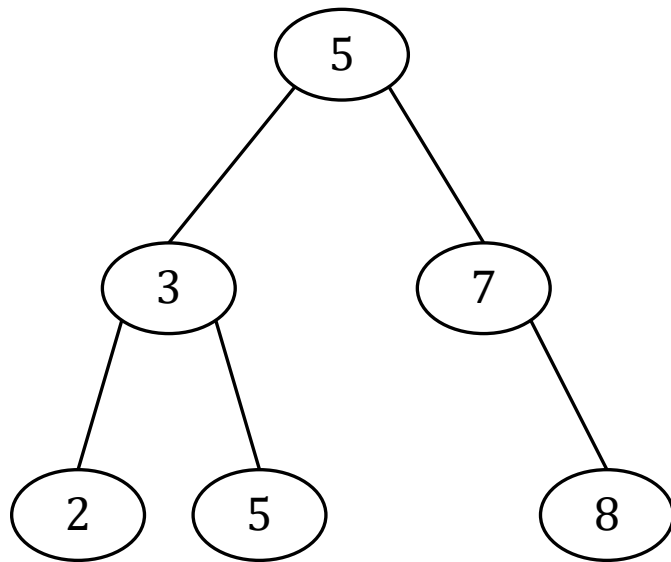
## データ集合から所望の要素を見つけるデータ構造

- 各節点が  $\text{key}$ ,  $\text{left}$  (左の子),  $\text{right}$  (右の子),  $p$  (親) をそれぞれ最大 1 つもつ二分木
- キーには順序がつけられる ; 2 つの節点  $x, y$  に対して  $\text{key}(x) = \text{key}(y)$ ,  $\text{key}(x) > \text{key}(y)$ ,  $\text{key}(x) < \text{key}(y)$  のいずれかが成り立つ
- キーは以下の条件を満たす :
  - $y \in x$  の左の子を根とする部分木
  - $y' \in x$  の右の子を根とする部分木とすると  $\text{key}(y) \leq \text{key}(x) \leq \text{key}(y')$  が成立



# 二分探索木の例：

## 二分探索木の条件を満たすことを確認



## 二分探索木を用いた探索： 木の高さに比例する時間で可能

- キーの満たす条件を用いて  $O(h)$  で発見（ $h$  は木の高さ）
- $\text{SEARCH}(x, k)$  : これを「 $x = \text{根}$ 」で呼ぶ;  $k$  は探したいkey
  - if  $x = \text{NULL}$  または  $k = \text{key}(x)$  then  $x$  を返す
  - if  $k < \text{key}(x)$  then  $\text{SEARCH}(\text{left}(x), k)$  : 左にあるはず
  - if  $k > \text{key}(x)$  then  $\text{SEARCH}(\text{right}(x), k)$  : 右にあるはず
- $\text{SEARCH}(x, k)$  の再帰を用いない表現
  - while  $x \neq \text{NULL}$  または  $k \neq \text{key}(x)$ 
    - if  $k < \text{key}(x)$  then  $x \leftarrow \text{left}(x)$  else  $x \leftarrow \text{right}(x)$
  - end while;  $x$  を返す

# 二分探索木からソート済み配列を取り出す： 中順での巡回による要素列挙

- 二分探索木から、全てのキーを整列された順で出力できる

INORDER( $x$ ) : 中順での巡回 (これを  $x = \text{根}$  で呼ぶ)

if  $x$ が葉 then  $\text{key}(x)$ を出力

else

この順序が重要

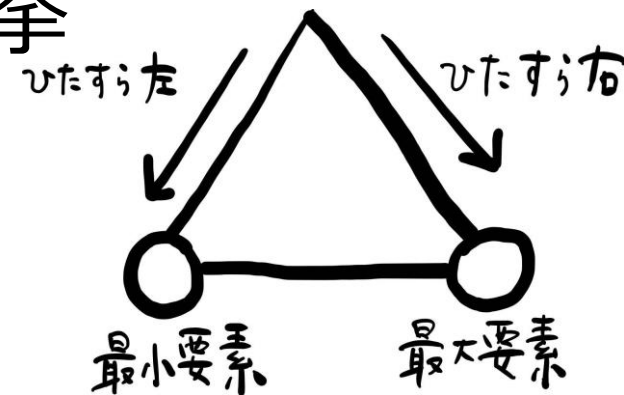
① INORDER(left( $x$ )) :  $x$ 以下の要素が列挙される

②  $\text{key}(x)$ を出力

③ INORDER(right( $x$ )) :  $x$ 以上が列挙

end if

- なお、最小 (最大) の要素の発見は  
left (right) をたどることで  $O(h)$  で可能



## 前順・後順での巡回：

要素出力のタイミングによって異なる巡回順になる

### ■ PREORDER( $x$ )：前順での巡回

②  $\text{key}(x)$  を出力

要素を出力する  
タイミングに注意

① PREORDER(left( $x$ ))

③ PREORDER(right( $x$ ))

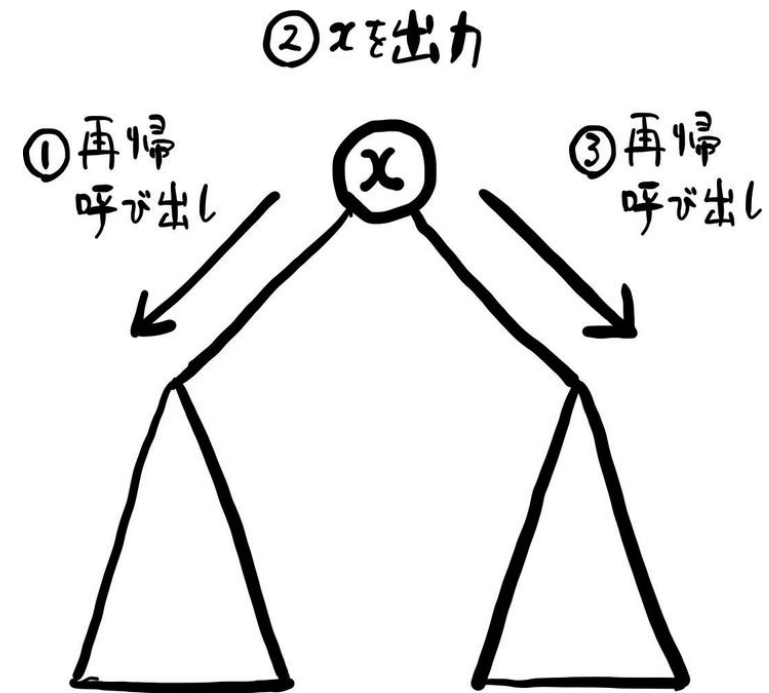
### ■ POSTORDER( $x$ )：後順での巡回

① POSTORDER(left( $x$ ))

③ POSTORDER(right( $x$ ))

②  $\text{key}(x)$  を出力

### ■ 出力の位置に注意（中順は①→②→③）



## 次節点・前節点：

次に小さい（大きい）要素を取り出す

- 次節点(successor)：中順で次の節点（=次に小さい）
- 前節点(predecessor)：中順でひとつ前の節点
- SUCCESSOR( $x$ )：次節点の発見

if right( $x$ )  $\neq$  NULL then MINIMUM(right( $x$ ))

$y \leftarrow \text{parent}(x)$

右の子がいるなら  
その右部分木の最小要素が次接点

while  $y \neq \text{NULL}$  かつ  $x = \text{right}(y)$

$x \leftarrow y; y \leftarrow \text{parent}(x)$

自分が親の右の子である限り  
上にあがっていく

end while

$x$ を返す

自分が親の左の子なら  
親が次節点のはず

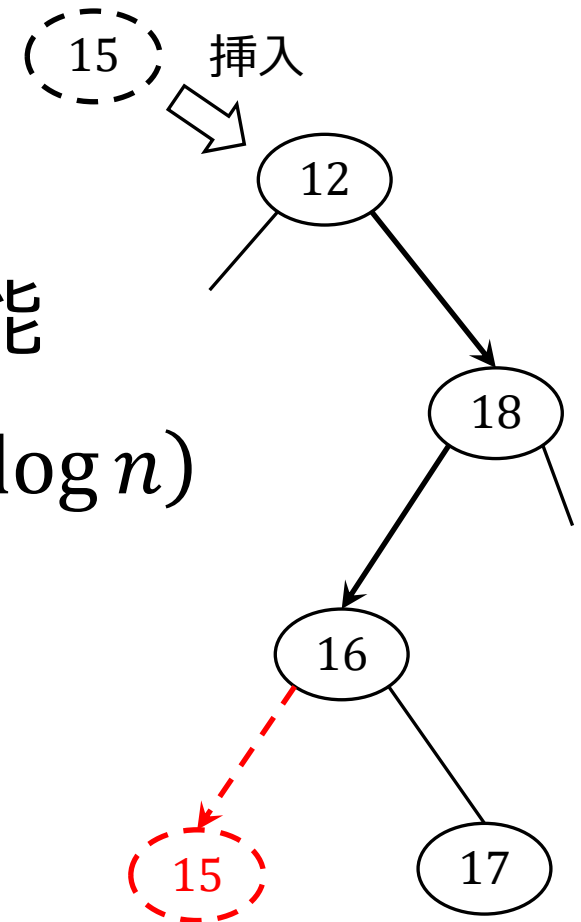
(親は自分より先に挙げられるはずなので  
親は次節点ではない)

(中順では、親は自分の次に挙げられる)

# 二分探索木への新たな要素の挿入：

挿入は、実際に探索してみることで $O(h)$ で実行可能

- 探索と同様にkeyの比較で辿っていき、該当する節点が無くなった時にそこに入れる
- 高さ $h$ の木では $O(h)$ 時間かかる
- これを繰り返して二分探索木を構成可能
  - ランダムな順で挿入すれば平均高さ $O(\log n)$ 
    - $\approx 1.39 \log n$
  - 最悪の場合には、高さ $n$



# 二分探索木からの要素削除： 次節点（か前節点）で置き換える

## ■ 3つの場合に分けて考える

1. 削除する節点 $z$ が葉のときは単に削除
2.  $z$ の子が1つの場合：  $z$ を削除して子をその位置に移動
3.  $z$ の子が2つの場合：

SUCCESSORの最初の場合

  - I.  $z$ の次節点 $y$ を見つける（ $y$ は $z$ の右の子孫の最小要素）  
（前節点でもいい）
  - II.  $y$ を削除して、 $z$ の位置に $y$ を入れる

上記1 or 2

    - $y$ の子は高々 1 個(右の子)なので $y$ の削除は容易
    - 子孫との大小関係が保たれていることに注意

$\text{key}(z) \leq \text{key}(y)$ なので、 $z$ の左の子孫と $y$ との大小関係はOK  
 $z$ の右の子孫のなかで $y$ は最小なので、こちらもOK

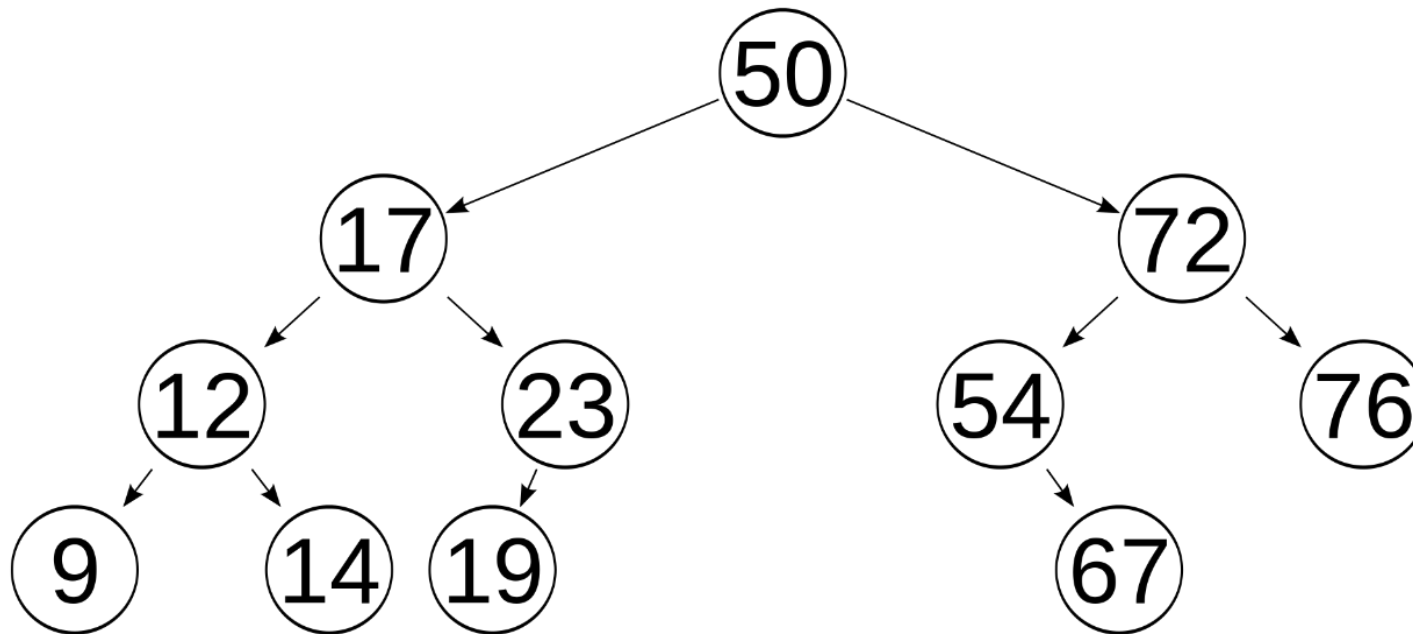
# 平衡木

# 平衡木： バランスのとれた二分探索木

- 二分探索木をもちいた探索のコストは、根から所望の節点までの道のりの長さ  
(探索対象が見つからない場合には葉までの長さ)
- 二分探索木が完全二分木に近い場合には $O(\log n)$   
しかし、バランスが悪いとコストがかかる場合がある
- 平衡木：木の高さ（根から葉までの道のりの長さ）が常に $O(\log n)$ であるような探索木
  - AVL木、赤黒木、スプレー木、B木、...

# AVL木： バランスのとれた二分探索木

- どの節点についても、右の部分木と左の部分木の高さの差が最大1であるような二分探索木



<https://ja.wikipedia.org/wiki/AVL%E6%9C%A8#/media/File:AVLtreef.svg>

# AVL木の性能： 最悪ケースで $O(\log n)$

- 2分木のなかで最も低いものは完全二分木 ( $\log n$ )
- いっぽう、もっとも高いものが最悪ケース

– 頂点数 $n$ をもつ二分木のなかで最も高いもの

⇔ 高さ $h$ の二分木のうち、もっとも頂点数が少ないもの

– 高さ $h$ のAVL木の最小の頂点数を $N_h$ とすると

$$N_h = \frac{1}{\sqrt{5}} \left( \left( \frac{1+\sqrt{5}}{2} \right)^{h+3} - \left( \frac{1-\sqrt{5}}{2} \right)^{h+3} \right) - 1 \approx \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^{h+3} \quad (h \text{が大のとき})$$

$$\left| \frac{1-\sqrt{5}}{2} \right| < 1$$

– つまり  $h = \frac{\log N_h}{\log \left( \frac{1+\sqrt{5}}{2} \right)} - 3 \approx 1.44 \log N_h$   
(最良ケースの1.44倍)

$$\approx \frac{1}{1.44}$$

補足：

なるべくバランスの悪いAVL木をつくる

---

– 高さ $h$ のAVL木の最小の頂点数を $N_h$ とすると

$$N_h = N_{h-1} + N_{h-2} + 1$$

–  $f_h = N_h + 1$ とすれば  $f_h = f_{h-1} + f_{h-2}$   
(フィボナッチ数列)

– フィボナッチ数列の解：

$$f_h = \frac{1}{\sqrt{5}} \left( \left( \frac{1 + \sqrt{5}}{2} \right)^{h+3} - \left( \frac{1 - \sqrt{5}}{2} \right)^{h+3} \right)$$

# 近傍探索

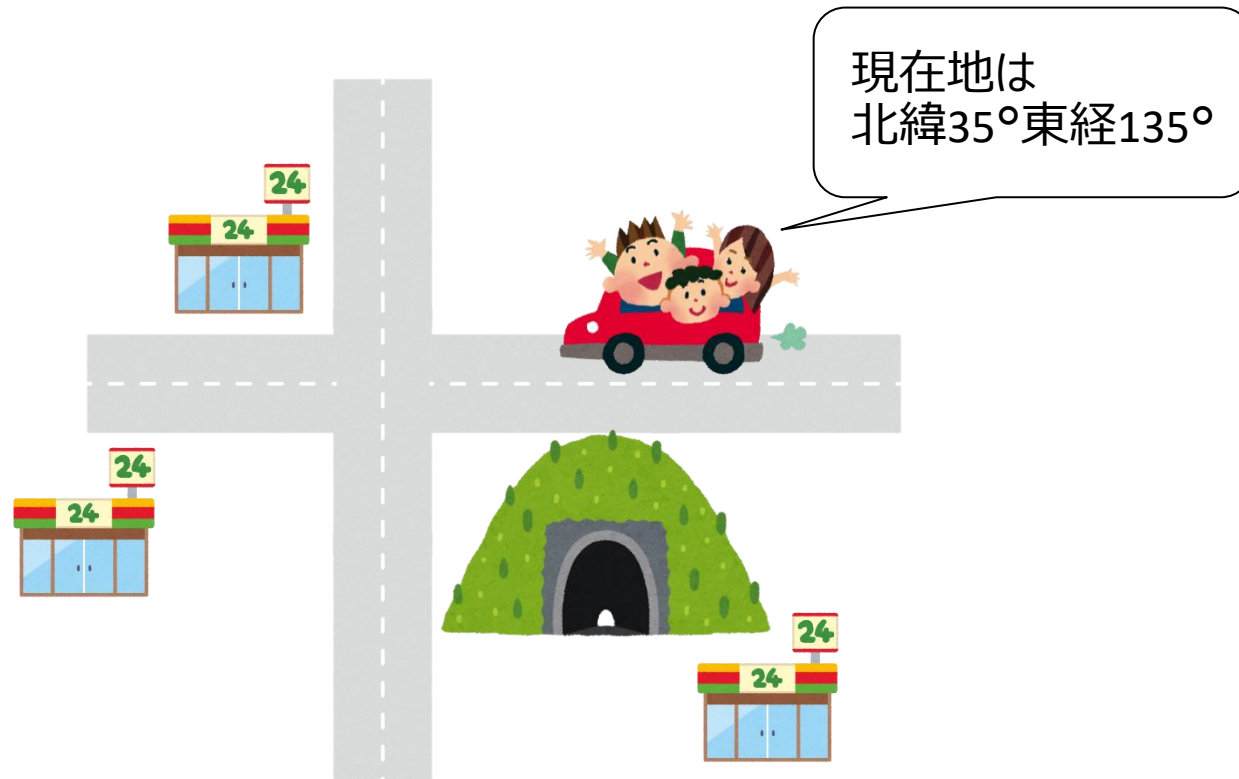
# 近傍探索： 類似データを探す問題

---

- これまで考えてきた探索問題は、質問と同一のキーをもつデータを探す問題
- 近傍探索：質問に類似したデータを探す
  - 最近傍探索：最も類似したデータを探す
  - もしくは一定度以上類似したデータを探す
- たとえば：
  - 近隣施設の検索（地理情報システム）
  - 文字認識（パターン認識）

# 最近傍探索の例： 近隣施設の検索

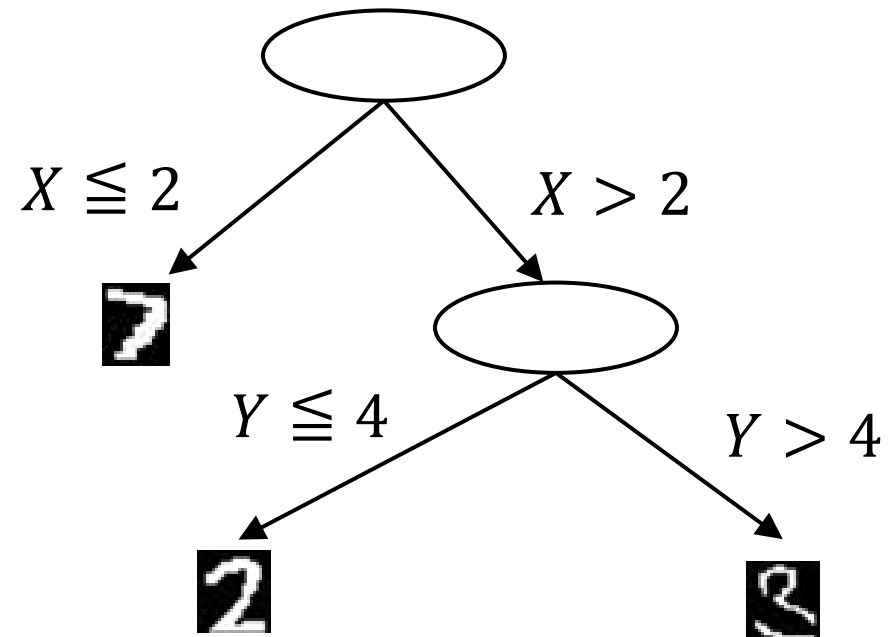
- 「OK、G<sup>o</sup>o<sup>g</sup>le。最寄りのコンビニはどこ？」



# $k$ -d木：

## 空間探索のための二分探索木

- 空間を探索するための二分探索木：
  - ある次元における二分割を繰り返すことで空間を分割
- $k$ -d木：  $k$ 次元の空間を探索する2分探索木
  - 分割はデータ点で行う



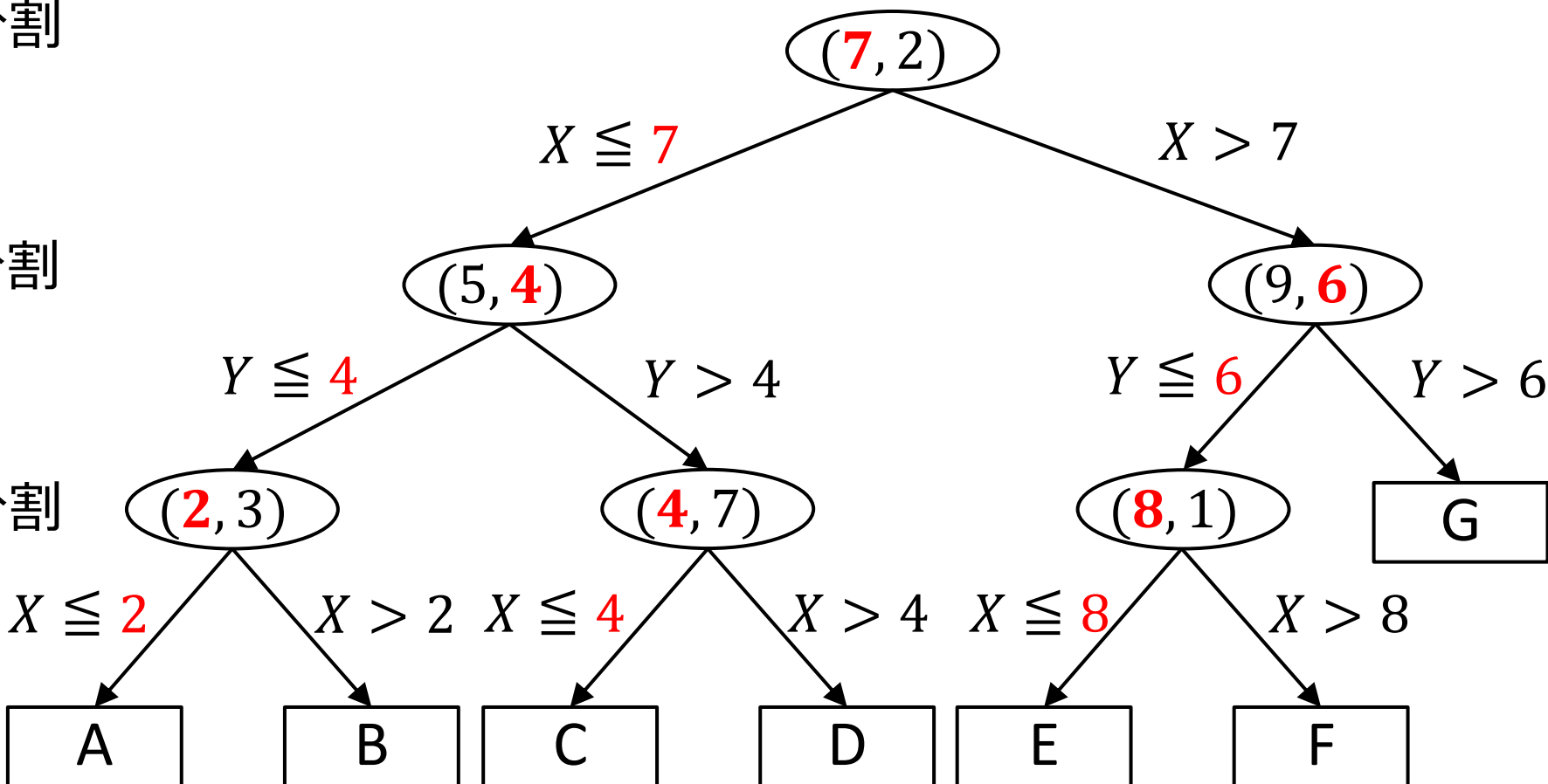
# $k$ -d木の例： 空間をデータ点で領域に分割

- 例：2次元データ  $(2,3)$ ,  $(5,4)$ ,  $(9,6)$ ,  $(4,7)$ ,  $(8,1)$ ,  $(7,2)$

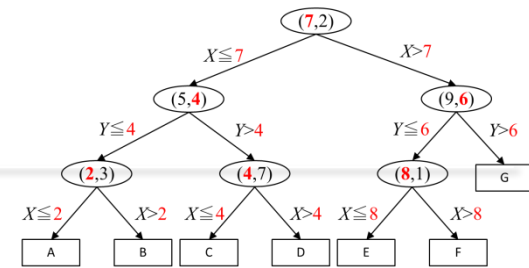
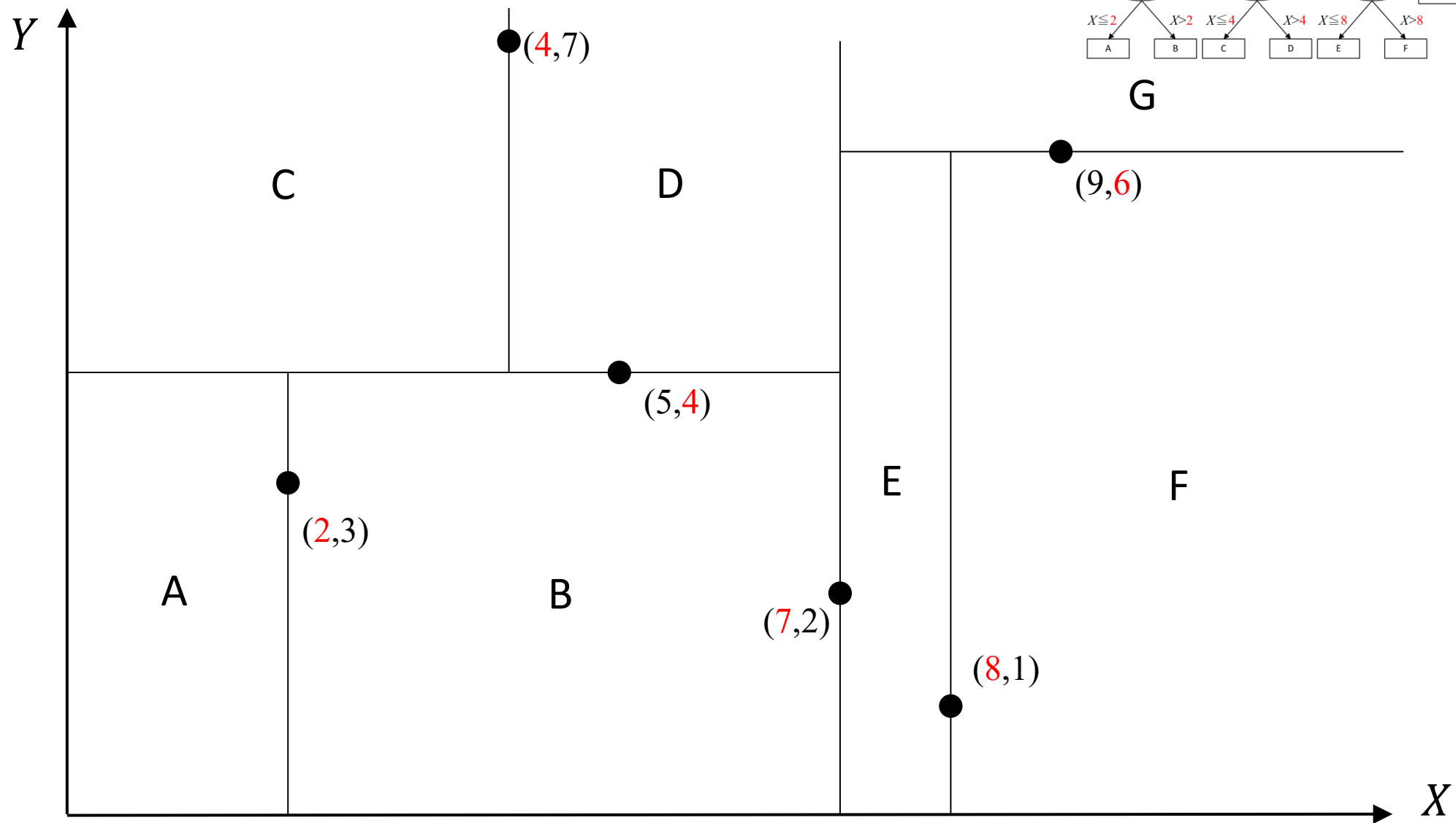
X軸で分割

Y軸で分割

X軸で分割



# $k$ -d木の例： 空間をデータ点で領域に分割



G

(9,6)

E

F

(7,2)

(8,1)

(5,4)

(2,3)

(4,7)

C

D

A

B

X

Y

## $k$ -d木の構成：

分割軸を決めて中央値で分割する

---

■ 領域分割を繰り返すことで構成：

1. 分割する軸を選ぶ

（例えば領域内データの分散が最大の軸）

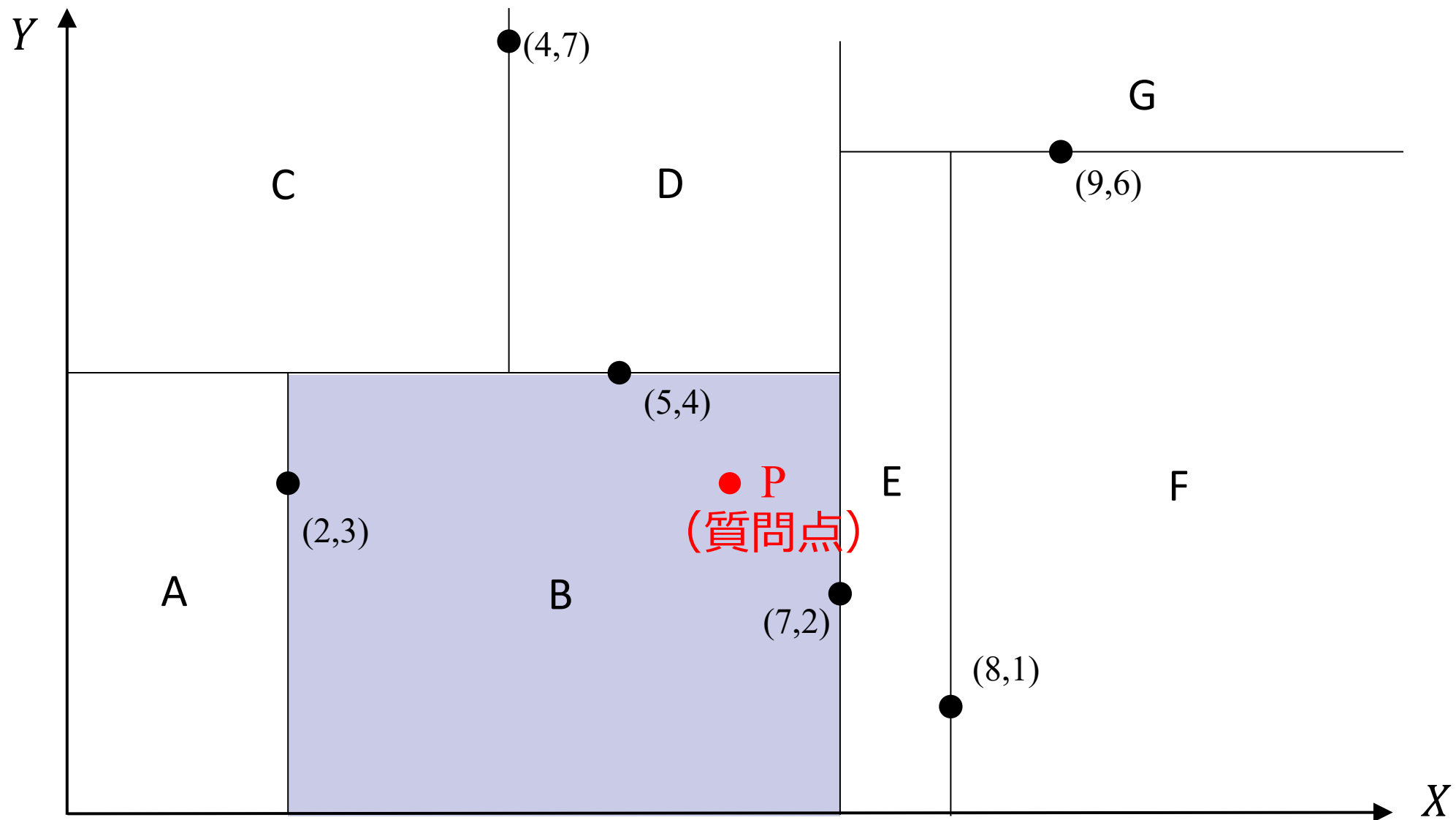
2. 選んだ軸について、領域に含まれるデータの中央値となるデータで分割する

3. 再帰的に分割を繰り返す

これ以上分割できなくなったときに終了

■ 探索木は平衡木が望ましい：中央値で分割を繰り返すので、高さはおおむね  $\log n$  を達成

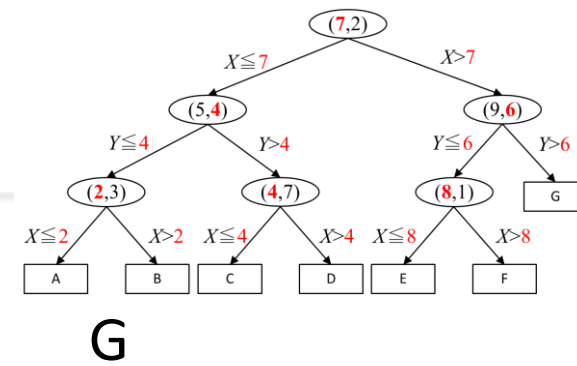
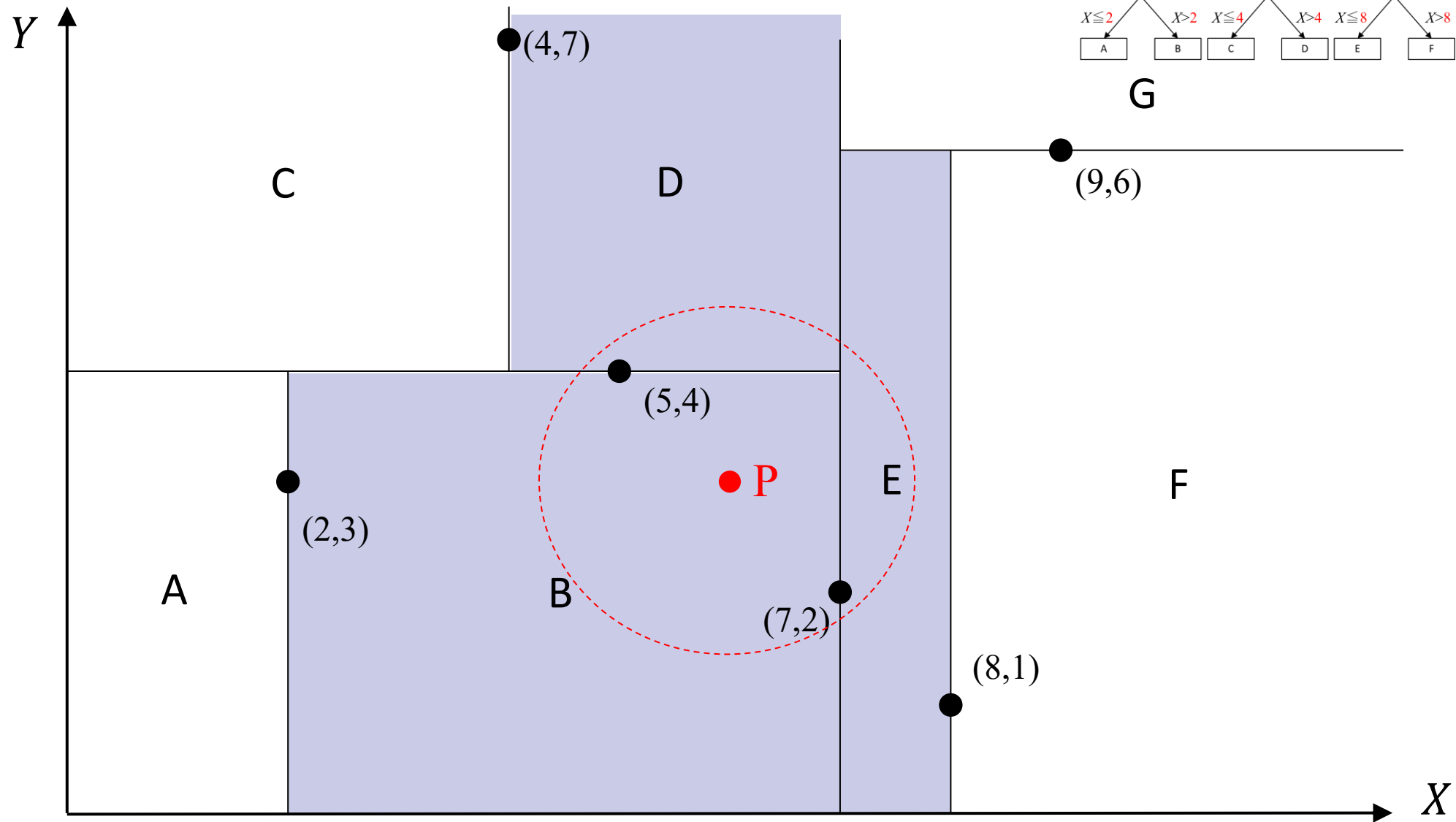
# $k$ -d木による質問点の探索： 探索木を下ることで質問点を含む領域を $O(\log n)$ で発見



## $k$ -d木による近傍点の探索： 枝刈りによって効率的に探索を行う

- 質問点 $P$ を含む半径 $r$ の領域にデータがあるかを調べたい
- 基本方針：質問点 $P$ を含む半径 $r$ の領域と、分割された各領域に重なりがあるかをチェックしながら $k$ -d木を下る
- $k$ -d木の各分岐において：質問点 $P$ を含む半径 $r$ の領域に...
  - 分岐点が含まれれば解として記録しておく
  - 分岐先の領域が重なるなら、その領域の探索を続行する
    - 両方の分岐先の領域と重なるなら、両方探索する
    - 重ならない領域については、以降の分岐の探索は打ち切れる
- 最悪の場合、すべての分岐をチェックする必要がある

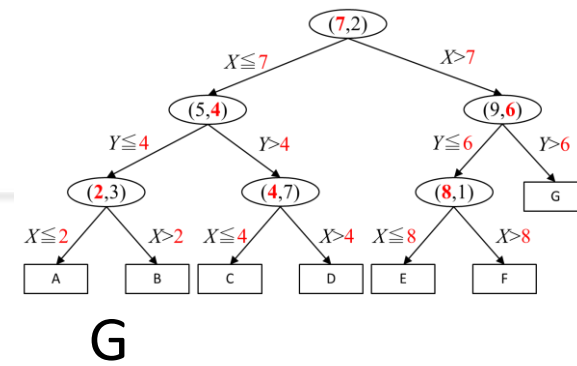
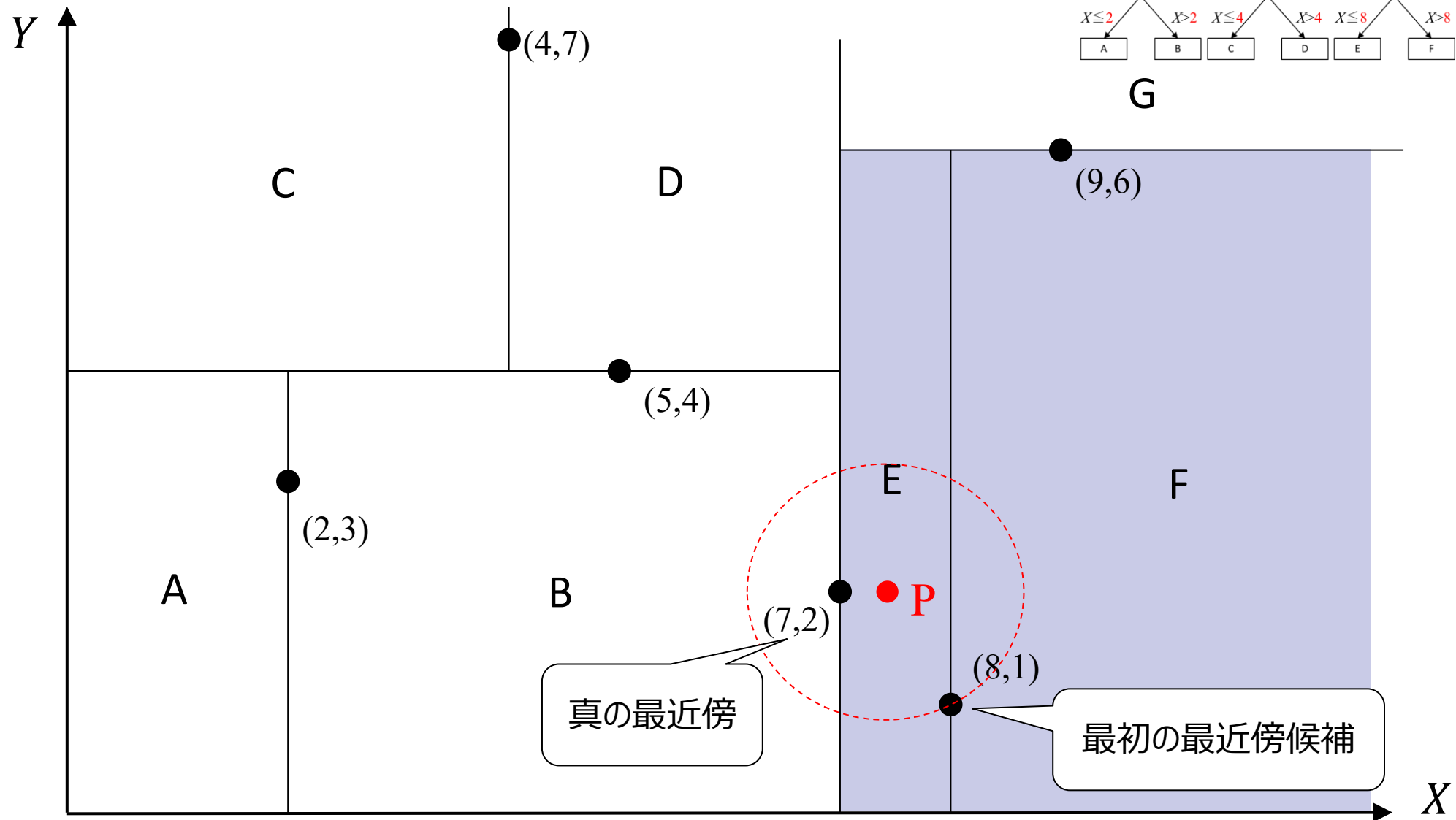
# $k$ -d木による近傍点の探索： 枝刈りによって効率的に探索を行う



## $k$ -d木による最近傍点の探索： 近傍点探索を少し変更

- $P$ に近い点が見つかる度に、暫定的な最近傍点 $P'$ と距離 $r$ を更新
  - 初めは $P$ を含む領域を見つけ最も葉に近い分割点を初期 $P'$ とする
  - $P$ から半径 $r$ 以内により近い点がなければ $P'$ が真の最近傍点
- もう少し効率のよい方法：深さ優先探索
  - 現在の領域から木を上に向かっていきながら分岐点をチェック
  - 分岐点の反対側の領域が、 $P$ から半径 $r$ 以内の領域と被っていれば、他方の分岐先も調べる
  - より近い点を見つけたら、 $P'$ と $r$ を更新
  - 根において、探索すべき方向がなくなったら終了

# $k$ -d木による最近傍点の探索： 枝刈りによって効率的に探索を行う



# 高次元の探索：

$k$ -d木は高次元で効率が悪いので次元削減が必要

---

- $k$ -d木が有効なのは数次元程度
  - データ数  $n \gg 2^k$  が望ましい
- 高次元の場合に、探索効率が悪くなる（多くの点を調べることになる）
- 次元削減によって次元を落としてから  $k$ -d木を適用する
  - ランダム射影
  - 主成分分析
  - ...