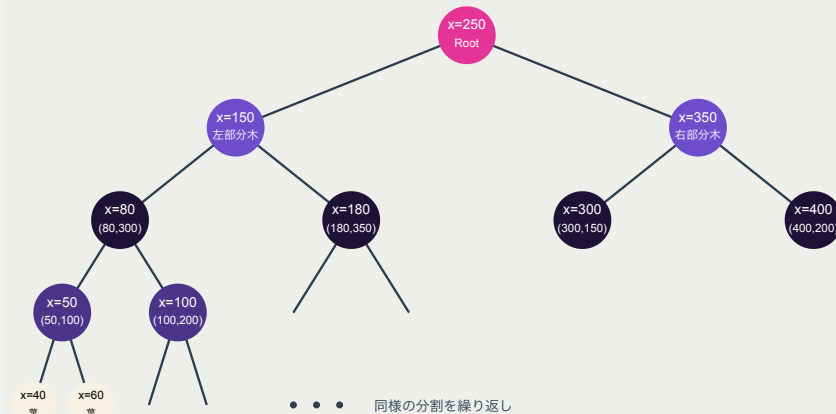
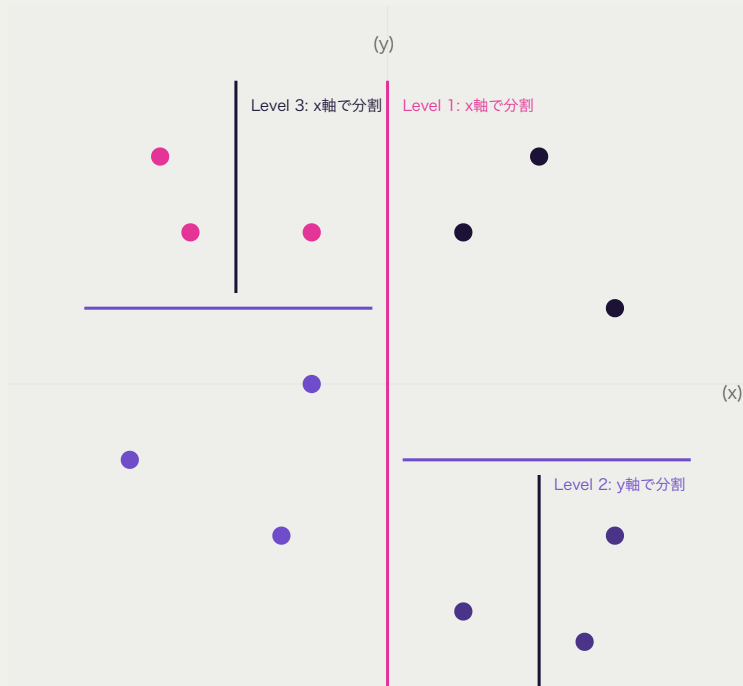


# 10万個の点から 一番近い点を見つける

～KD treeを例とした効率的なアルゴリズムの設計～





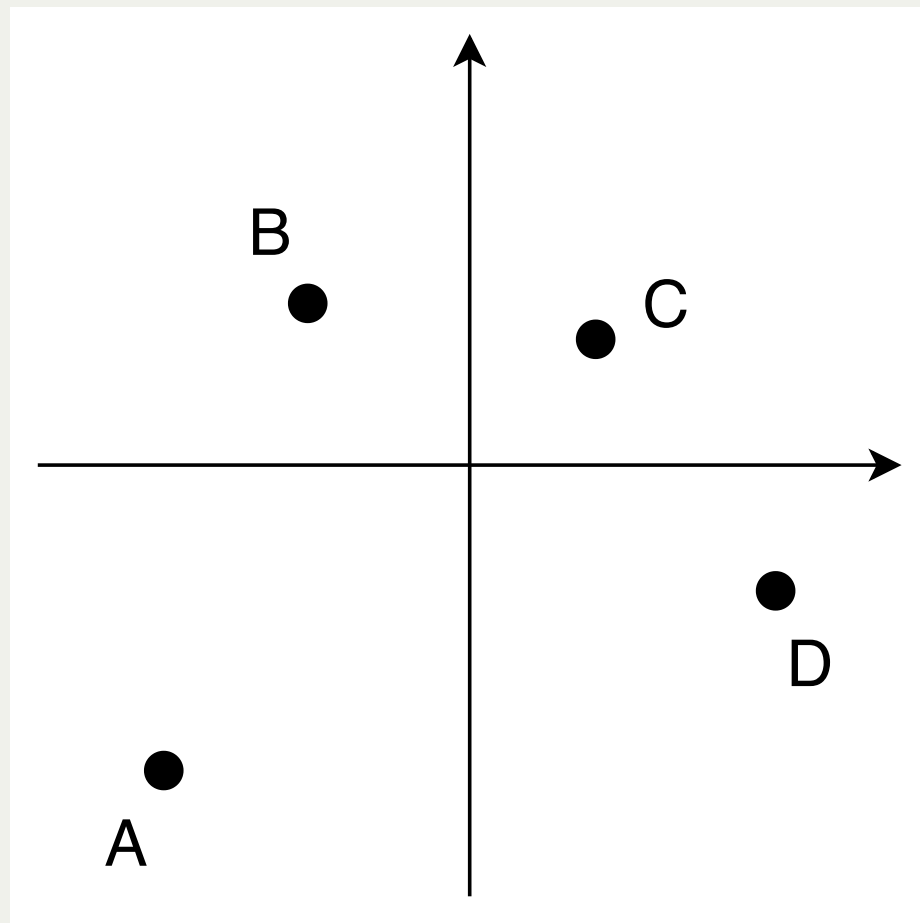
# 自己紹介😊

- さめ (meg-ssk)
- 🧑💻 フリーランスのソフトウェアエンジニア
- 得意分野:
  - 📷 コンピュータビジョン (画像認識/点群処理)
  - 🌐 空間情報処理 (GIS/リモートセンシング)
  - ☁️ クラウドインフラ設計/IaC (AWS)
- GitHub: s-sasaki-earthsea-wizard
- Speaker Deck: syotasasaki593876
- LinkedIn: syota-sasaki-878901320



# 問題提起: 一番近い点はどれ？

- 平面上に4つの点があります (A, B, C, D)
  - 点Aに一番近いのはどの点でしょう？
  - 点Bに一番近いのはどの点でしょう？
  - 点Cに一番近いのはどの点でしょう？
  - 点Dに一番近いのはどの点でしょう？
  - **すべての点の一番近い点を計算するには？**



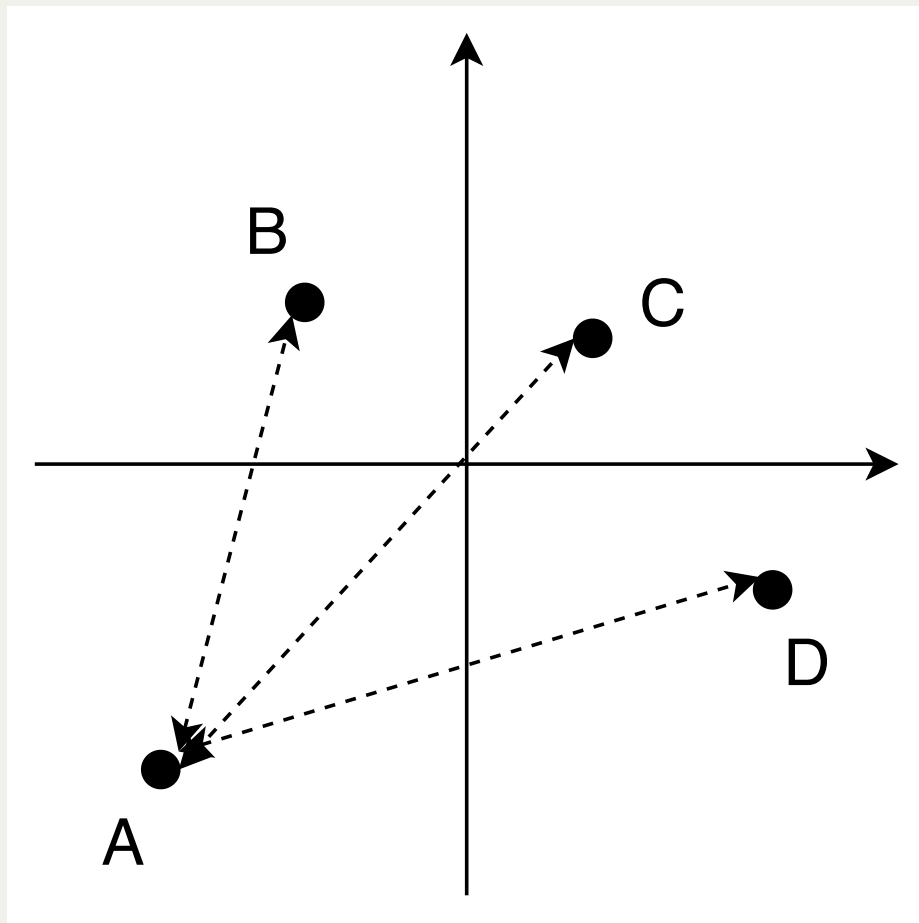
## シンプルな解き方

点Aに一番近い点を探す

- AB, AC, ADの長さ(ノルム)を計算する
- 言うなれば「定規で長さを測る」
- あとはこの中の最小値を選べばOK！

$$\min(||AB||, ||AC||, ||AD||)$$

合計3回の計算で解決！

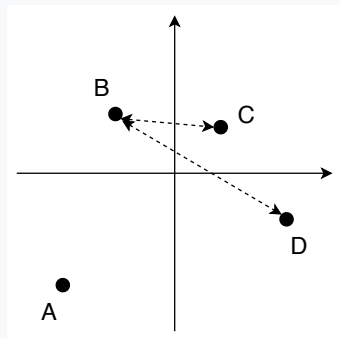




# 長さを測ることを繰り返す

点Bに一番近い点を探す

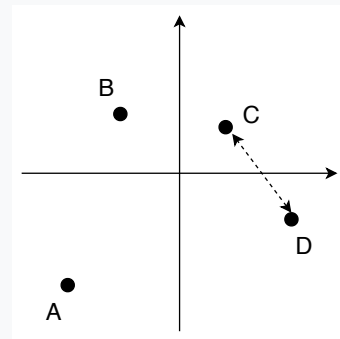
- BC, BDの長さを計算する
- BAの長さはすでに計算済み



合計2回の計算で解決！

点Cに一番近い点を探す

- CDの長さを計算する
- CA, CBの長さはすでに計算済み



合計1回の計算で解決！

- 3回 + 2回 + 1回 = **6回の計算で解決！**
- なんだ簡単じゃん！めでたしめでたし！ ...ではない！

# 💣 計算量の爆発

点の数が10万個になったら？

- 点Aに一番近い点を計算するためには99,999回の計算が必要
- 点Bに一番近い点を計算するためには99,998回の計算が必要
- (以下略...)
- 合計約50億回の計算が必要！



- 💣 計算量が爆発する！
- ❌💻 現実的な時間、計算リソースでは計算不可能！
- ❌🕒 リアルタイムでの計算は不可能！
- 🤔 もっと賢く計算できないかな？

!?! 突然ですがクイズです！

- 4本のワインがあります 🍷🍷🍷🍷
- その中に毒入りワインが1本あります 🍷💀
- 飲んでから1日後に毒の効果が現れます 🍷🤮
- **毒入りワインを見つけるためには何人の毒見係が必要？**

























- 👤👤👤👤 **4人以下の毒見係で毒入りワインを見つける方法があります！**
- 🙋 **有名なクイズなので、答えを知ってる人は手を挙げてください！**

 答え: 2人 

-  アリスと  ボブの2人が毒見係をします
- 以下の左の表のように2人がワインを飲みます

		
 1	○	○
 2	○	×
 3	×	○
 4	×	×

1日後...

-   かつ   →  1 が 
-   かつ   →  2 が 
-   かつ   →  3 が 
-   かつ   →  4 が 



ワインが8本に増えたら？ 🍷 x 8

3人の毒見係(👩🏻 👨🏻 👦🏻)で毒ワイン 🍷💀を特定できる

	👩🏻	👨🏻	👦🏻
🍷1	○	○	○
🍷2	○	○	×
🍷3	○	×	○
🍷4	○	×	×
🍷5	×	○	○
🍷6	×	○	×
🍷7	×	×	○
🍷8	×	×	×

• まったく同じ方法で:

- 16本のワイン 🍷x16
  - 4人の毒味係で毒ワイン 🍷💀を発見可能
- 32本のワイン 🍷x32
  - 5人の毒味係で毒ワイン 🍷💀を発見可能
- 🤔 10万本の 🍷があったら？

# $n$ 人の毒見係がいれば $2^n$ 本のワインを毒見できる

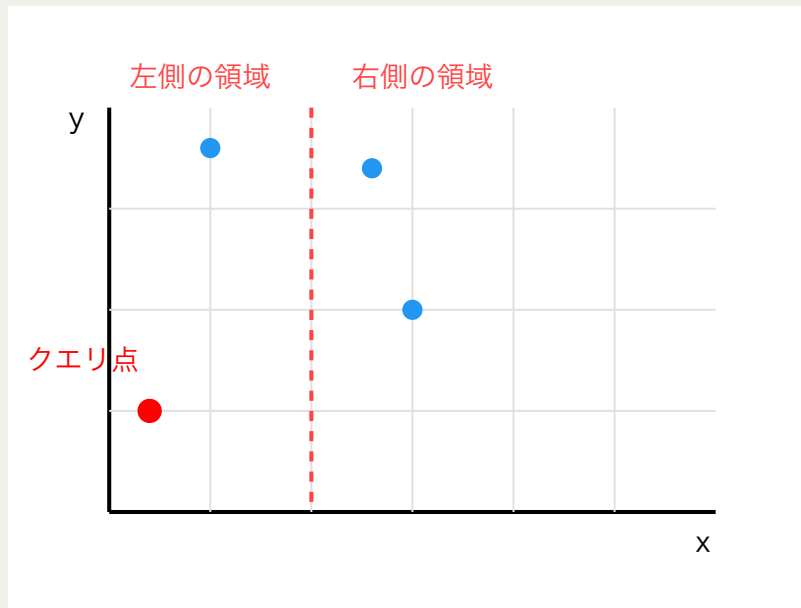
- 組み合わせの工夫で $n$ 人の毒見係がいれば $2^n$ 本のワインから1本の毒入りワインを発見できる
- 10万本のワインがあっても、17人の毒見係がいれば1本の毒入りワインを発見できる！

$$2^{17} = 131072 > 100000$$

- 少ない人数で多くのワインを毒見できる！
- 効率的な毒見係の配置が重要！
- 🤔これを応用して、10万個の点の中から一番近い点を探す方法はないかな？

# 効率化の鍵: KD Tree 🌳✂️🌿

- x軸に平行な線で空間を分割！
- 1回の分割でおおまかに候補を半分に絞り込める！



- 🤔 この分割を繰り返したらどうなる？
- 💡 候補がどんどん減って、より効率的な探索ができそう！

# 分割のチカラ

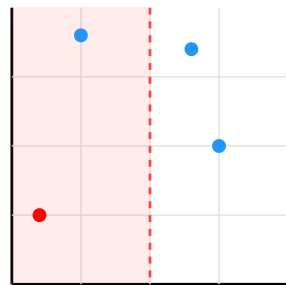
- 全探索:

- 4点から最近傍ペアを求める場合、すべての組み合わせ（6通り）の距離計算が必要

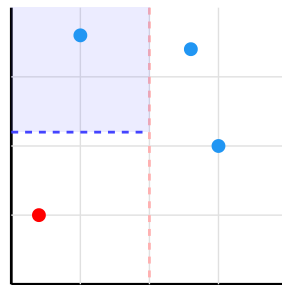
- KD-treeでの探索 (理想的なケース):

- 軸方向に沿った分割で探索候補を大幅に減らせる
- 全探索よりずっと少ない計算で最近接ペアを見つけられる

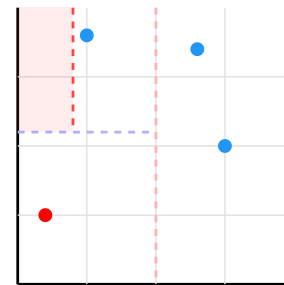
Step 1: X軸で分割し、左側の領域へ



Step 2: Y軸で分割し、上側へさらに絞る



Step 3: 再びX軸で分割し、A付近の領域へ特



- この図では「調べなくてもいい領域」がわかった!

# ー 引き算型 vs ÷ 割り算型

## • 引き算型(線形探索):

- 1回の計算で「候補を1つずつ」しか減らせない
- 10万個の候補点があれば10万回もチェックを繰り返す
- すべてのペアを計算:
  - $10万 \times 10万 / 2 =$  約50億回の計算😓

## • 割り算型(KD-Treeでの探索):

- 1回の分割で候補を約半分に減らせる
  - 10万個が1回で約5万個、2回で約2万5千個...
  - わずか17回で探索終了✅
- すべてのペアを計算:
  - $10万 + 10万/2 + 10万/4 + 10万/8 + \dots =$  約20万回の計算😊

- 候補が引き算で減る vs 候補が割り算で減る
  - 分割を繰り返すほど、探索範囲が爆発的に縮む (理想的には!)
- 割り算で減らしていく方が圧倒的に速い

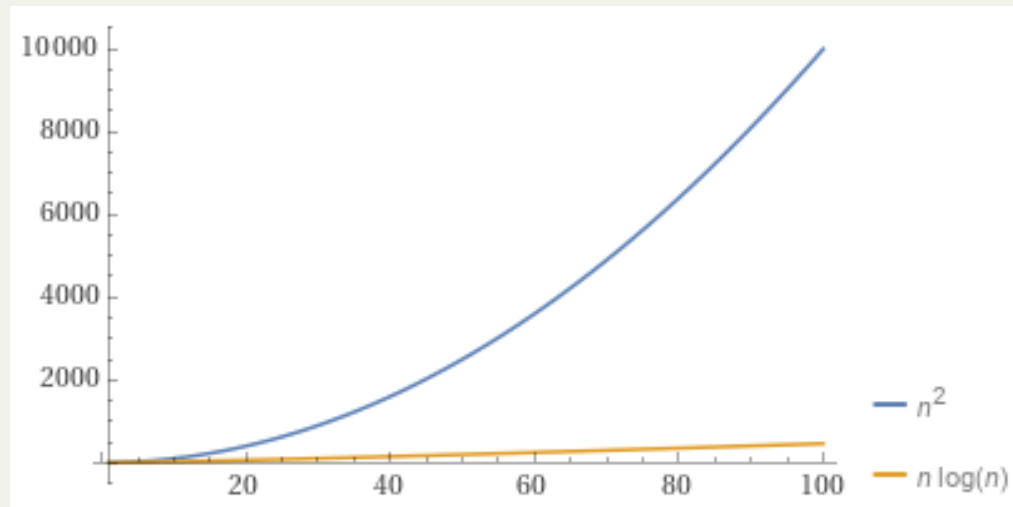
# 計算量と Big-O 記号

- 全探索:

- $n$ 個の点がある場合、すべてのペア( $n \times n$ )を調べるから  $O(n^2)$ 
  - ここで  $O(\dots)$  は  $n$  に対する計算量の増え方の目安 (Big-O 記号)

- KD-Tree(理想的な場合):

- 分割を重ねて候補を絞るから、平均的に  $O(n \log n)$ 
  - $n$  が大きくなっても、全探索よりずっと速くなる



## □ 平面から空間へ

- KD-Treeは、 $R^2$ (2次元)だけでなく、 $R^3$ (3次元)にも拡張可能
- 分割する平面を交互に変えながら、3次元空間を効率的に絞り込む
  - 3次元:
    - $y$ - $z$ 平面  $\rightarrow$   $z$ - $x$ 平面  $\rightarrow$   $x$ - $y$ 平面  $\rightarrow$   $y$ - $z$ 平面  $\rightarrow$  ... でサイクリックに平面で分割

- 3Dデータの解析などで威力を発揮！
- 筆者は普段のお仕事でよく使ってます！

## さらなる高次元へ

- KD-treeは $R^d$ ( $d$ 次元)にも拡張可能！
- 超平面 $H_0 \rightarrow H_1 \rightarrow \dots \rightarrow H_d \rightarrow H_0 \rightarrow \dots$  で分割

$$H_0 : (\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^d)$$

$$H_1 : (\mathbf{x}^0, \mathbf{x}^2, \dots, \mathbf{x}^d)$$

$\vdots$





$$H_d : (\mathbf{x}^0, \mathbf{x}^1, \dots, \mathbf{x}^{d-1})$$

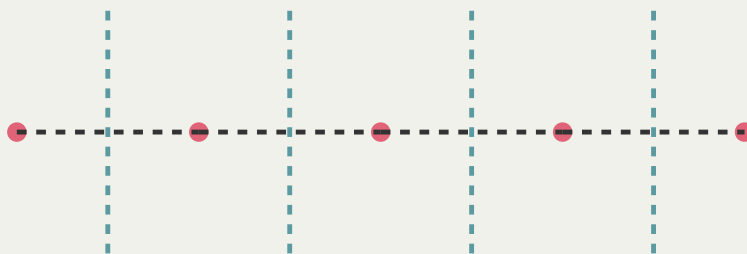
- 高次元空間の探索でこそ、KD-Treeの真価が発揮される！





# KD-treeが苦手な場面

- 一直線に並んだ点 - . - . - . - . -
  - 全探索と同じ  $O(n^2)$  の計算量に
  -  現実の3Dデータでは局所的に点が一直線に並ぶことは**十分あり得る**
    -  ビル、 道路、 堤防、などの人工構造物



- 横方向の分割に意味がない！縦方向の分割だけでは全探索と同じ！
- 対策: 軸を回転させる、ランダムな分割をする、など

# KD-treeの応用例

## 1. 点群データ解析

- 3DスキャンやLiDARデータの効率的な解析
- 点群の間引き、ノイズ除去

## 2. 特徴量ベクトルの類似度

- 趣味や好みの近いユーザーの探索
- 画像の類似度計算

## 3. Hausdorff距離の計算

- 機械学習モデルのトレーニングの損失関数
- 物体の形状比較

# アルゴリズムの現実的な使い方

- 🥱 新しいアルゴリズムを作るのは本当に大変
  - 正しさの証明
  - 性能の評価
  - アイディアの独自性
- そもそも新しいアルゴリズムを開発すれば自分の名前がつくレベルの難しさ！
  - エドガー・ダイクストラの"ダイクストラ法"
  - カジミェシュ・クラワトスキの"クラワトスキ定理"
  - ティム・ピーターズの"ティムソート"

- ✨ 既存のアルゴリズムの宝庫を活用しよう
  - 数学、物理、電気電子工学...様々な分野で培われた知恵
  - KD-treeも計算幾何学の分野から生まれた
    - 現在の応用例は開発者も想定してなかったはず！
  - 自分の課題に使えるアルゴリズムがあるはず！



# 現代のエンジニアの強み

- **充実したライブラリ**
  - scikit-learn: `sklearn.neighbors.KDTree`
  - SciPy: `scipy.sparse.csgraph.dijkstra`
  - OpenCV: `cv2.FlannBasedMatcher`
- **実装済みの高品質なコード**
  - テスト済み
  - 多くのユーザーによるフィードバック
  - ドキュメントが豊富

## 🌟 まとめ: 宝物は足元にある

- KD-treeを例に見てきたこと:
  - 単純な全探索 → **賢い分割**で効率化
  - 理想的な場合は**劇的な性能向上**
  - 最悪ケースを理解し、**対策をする**

- **先人の知恵を活用しよう！**
  - アルゴリズムの宝庫が既にそこにある
  - 異分野のアイディアを組み合わせる
  - 実装済みライブラリを有効活用
- 一緒に宝物を探しに行きましょう！ 🤝🚀🎉