Kd trees

František Dráček
dracek1@uniba.sk

22. októbra 2025

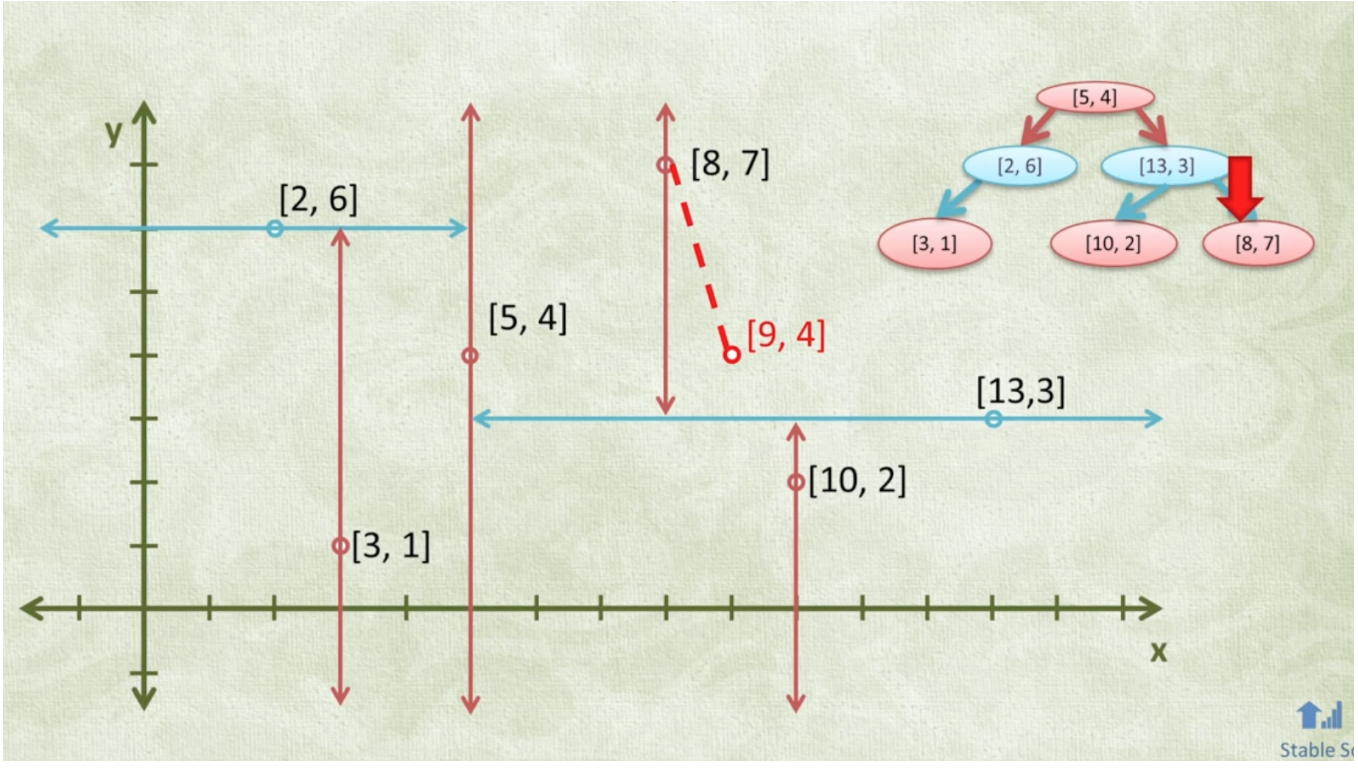
Kd-tree Construction and Nearest Neighbor Search

1. Given a set of n points.
2. Points are k-**dimensional** (i.e., in $\mathbb{R}^k$).
3. Construction starts at the **root node** with **depth** $d = 0$.
4. At each node/depth, we split the points into two regions using an **axis-aligned hyperplane** (a splitting **axis**).
5. The splitting axis i is selected by **cycling through the dimensions**: $i = d \pmod{k}$.
6. Sort the points along the current axis i and select the **median** point as the splitter.
7. The median point becomes the **current node**; points with a **smaller** i-th coordinate form the **left subtree**, and points with a **larger** coordinate form the **right subtree**.

Nearest Neighbor Search (NNS) Overview

8. **Search Traversal**: Recursively traverse the tree down to the leaf node that would contain the query point, tracking the **current best distance** and **closest point found**.
9. **Backtracking & Pruning**: Recursively go back up the tree. At each node, check the distance from the query point to the **splitting hyperplane**.
10. If this **hyperplane distance** is **smaller** than the **current best distance**, then the sphere defined by the current best distance **intersects** the hyperplane. Therefore, the **other branch** must be recursively searched.

Visualizing a 2D Kd-tree



Programming Assignment

- Implement the *k*-**d tree** data structure for a given set of *k*-dimensional points.
- Write a routine that, for a given **query point**, **locates the corresponding leaf node** in the tree (i.e., point **insertion** or **search**).
- Implement the **Nearest Neighbor Search** algorithm for an arbitrary query point, using **Euclidean distance** as the metric.
- **Constraint**: Do not use any existing, **off-the-shelf libraries** or packages that implement *k*-d trees.