KDTree.java

```
1 package TreesTesting;

  import java.util.Arrays;
  import java.util.Comparator;
  import java.util.Objects;
  import java.util.Optional;

  public class KDTree {

          private Node root;

          private final int k; // Dimensions of the points

          KDTree(int k) {
                  this.k = k;
          }

          KDTree(Point[] points) {
1                 if (points.length == 0) throw new IllegalArgumentException(
                                  "Points array cannot be empty"
                                  );
                  this.k = points[0].getDimension();
                  for (Point point : points) if (
1                                 point.getDimension() != k
                                  ) throw new IllegalArgumentException(
                                                  "Points must have the same dimension"
                                                  );
                  this.root = build(points, 0);
          }

          KDTree(int[][] pointsCoordinates) {
1                 if (pointsCoordinates.length == 0) throw new IllegalArgumentException(
                                  "Points array cannot be empty"
                                  );
                  this.k = pointsCoordinates[0].length;
                  Point[] points = Arrays
                                  .stream(pointsCoordinates)
                                  .map(Point::new)
                                  .toArray(Point[]::new);
                  for (Point point : points) if (
1                                 point.getDimension() != k
                                  ) throw new IllegalArgumentException(
                                                  "Points must have the same dimension"
                                                  );
                  this.root = build(points, 0);
          }

          static class Point {

                  int[] coordinates;

                  public int getCoordinate(int i) {
1                         return coordinates[i];
                  }

                  public int getDimension() {
1                         return coordinates.length;
                  }

                  public Point(int[] coordinates) {
                          this.coordinates = coordinates;
                  }

                  @Override
                  public boolean equals(Object obj) {
2                         if (obj instanceof Point other) {
2                                 if (other.getDimension() != this.getDimension()) return false;
2                                 return Arrays.equals(other.coordinates, this.coordinates);
                          }
1                         return false;
                  }

                  @Override
                  public String toString() {
1                         return Arrays.toString(coordinates);
                  }

                  public static int comparableDistance(Point p1, Point p2) {
                          int distance = 0;
3                         for (int i = 0; i < p1.getDimension(); i++) {
1                                 int t = p1.getCoordinate(i) - p2.getCoordinate(i);
2                                 distance += t * t;
                          }
1                         return distance;
                  }

                  public static int comparableDistanceExceptAxis(
                                  Point p1,
                                  Point p2,
                                  int axis
                                  ) {
                          int distance = 0;
3                         for (int i = 0; i < p1.getDimension(); i++) {
1                                 if (i == axis) continue;
1                                 int t = p1.getCoordinate(i) - p2.getCoordinate(i);
2                                 distance += t * t;
                          }
1                         return distance;
                  }
          }

          static class Node {

                  private Point point;
                  private int axis; // 0 for x, 1 for y, 2 for z, etc.

                  private Node left = null; // Left child
                  private Node right = null; // Right child

                  Node(Point point, int axis) {
                          this.point = point;
                          this.axis = axis;
                  }

                  public Point getPoint() {
1                         return point;
                  }

                  public Node getLeft() {
1                         return left;
                  }

                  public Node getRight() {
1                         return right;
                  }

                  public int getAxis() {
1                         return axis;
                  }

                  public Node getNearChild(Point point) {
                          if (
2                                         point.getCoordinate(axis) < this.point.getCoordinate(axis)
2                                         ) return left; else return right;
                  }

                  public Node getFarChild(Point point) {
                          if (
2                                         point.getCoordinate(axis) < this.point.getCoordinate(axis)
2                                         ) return right; else return left;
                  }

                  public int getAxisCoordinate() {
1                         return point.getCoordinate(axis);
                  }
          }

```

```
	public Node getRoot() {
1		return root;
	}

	private Node build(Point[] points, int depth) {
1		if (points.length == 0) return null;
1		int axis = depth % k;
2		if (points.length == 1) return new Node(points[0], axis);
1		Arrays.sort(
				points,
1				Comparator.comparingInt(o -> o.getCoordinate(axis))
				);
1		int median = points.length >> 1;
					Node node = new Node(points[median], axis);
1					node.left = build(Arrays.copyOfRange(points, 0, median), depth + 1);
					node.right =
							build(
1									Arrays.copyOfRange(points, median + 1, points.l
1									depth + 1
									);
1					return node;
	}

	public void insert(Point point) {
1		if (point.getDimension() != k) throw new IllegalArgumentException(
				"Point has wrong dimension"
				);
		root = insert(root, point, 0);
	}

	private Node insert(Node root, Point point, int depth) {
1		int axis = depth % k;
2		if (root == null) return new Node(point, axis);
2		if (point.getCoordinate(axis) < root.getAxisCoordinate()) root.left =
1				insert(root.left, point, depth + 1); else root.right =
1				insert(root.right, point, depth + 1);

1		return root;
	}

	public Optional<Node> search(Point point) {
1		if (point.getDimension() != k) throw new IllegalArgumentException(
				"Point has wrong dimension"
				);
1		return search(root, point);
	}

	public Optional<Node> search(Node root, Point point) {
1		if (root == null) return Optional.empty();
2		if (root.point.equals(point)) return Optional.of(root);
1		return search(root.getNearChild(point), point);
	}

	public Point findMin(int axis) {
1		return findMin(root, axis).point;
	}

	public Node findMin(Node root, int axis) {
1		if (root == null) return null;
1		if (root.getAxis() == axis) {
2			if (root.left == null) return root;
1			return findMin(root.left, axis);
		} else {
			Node left = findMin(root.left, axis);
			Node right = findMin(root.right, axis);
			Node[] candidates = { left, root, right };
1			return Arrays
					.stream(candidates)
					.filter(Objects::nonNull)
1					.min(Comparator.comparingInt(a -> a.point.getCoordinate(axis)))
					.orElse(null);
		}
	}

	public Point findMax(int axis) {
1		return findMax(root, axis).point;
224           }
225   
226           public Node findMax(Node root, int axis) {
227 1                 if (root == null) return null;
228 1                 if (root.getAxis() == axis) {
229 2                         if (root.right == null) return root;
230 1                         return findMax(root.right, axis);
231                   } else {
232                           Node left = findMax(root.left, axis);
233                           Node right = findMax(root.right, axis);
234                           Node[] candidates = { left, root, right };
235 1                         return Arrays
236                                           .stream(candidates)
237                                           .filter(Objects::nonNull)
238 1                                         .max(Comparator.comparingInt(a -> a.point.getCoordinate(axis)))
239                                           .orElse(null);
240                   }
241           }
242   
243           public void delete(Point point) {
244                   Node node = search(point)
245 1                                 .orElseThrow(() -> new IllegalArgumentException("Point not found"));
246                   root = delete(root, node);
247           }
248   
249           private Node delete(Node root, Node node) {
250 1                 if (root == null) return null;
251 1                 if (root.equals(node)) {
252 1                         if (root.right != null) {
253                                   Node min = findMin(root.right, root.getAxis());
254                                   root.point = min.point;
255                                   root.right = delete(root.right, min);
256 1                         } else if (root.left != null) {
257                                   Node min = findMin(root.left, root.getAxis());
258                                   root.point = min.point;
259                                   root.left = delete(root.left, min);
260                           } else return null;
261                   }
262                   if (
263 2                                 root.getAxisCoordinate() < node.point.getCoordinate(root.getAxis())
264                                   ) root.left = delete(root.left, node); else root.right =
265                                   delete(root.right, node);
266 1                 return root;
267           }
268   
269           public Point findNearest(Point point) {
270 1                 return findNearest(root, point, root).point;
271           }
272   
273           private Node findNearest(Node root, Point point, Node nearest) {
274 2                 if (root == null) return nearest;
275 2                 if (root.point.equals(point)) return root;
276                   int distance = Point.comparableDistance(root.point, point);
277                   int distanceExceptAxis = Point.comparableDistanceExceptAxis(
278                                   root.point,
279                                   point,
280                                   root.getAxis()
281                                   );
282 2                 if (distance < Point.comparableDistance(nearest.point, point)) nearest =
283                                   root;
284                   nearest = findNearest(root.getNearChild(point), point, nearest);
285                   if (
286 2                                 distanceExceptAxis < Point.comparableDistance(nearest.point, point)
287                                   ) nearest = findNearest(root.getFarChild(point), point, nearest);
288 1                 return nearest;
289           }
290   }
```

Mutations

```
1   1. replaced return value with null for TreesTesting/KDTree::lambda$1 → KILLED
19  1. negated conditional → NO_COVERAGE
24  1. negated conditional → NO_COVERAGE
32  1. negated conditional → KILLED
41  1. negated conditional → KILLED
```

```
53   1. replaced int return with 0 for TreesTesting/KDTree$Point::getCoordinate → KILLED
57   1. replaced int return with 0 for TreesTesting/KDTree$Point::getDimension → KILLED
66   1. negated conditional → KILLED
     2. negated conditional → KILLED
67   1. replaced boolean return with true for TreesTesting/KDTree$Point::equals → NO_COVERAGE
     2. negated conditional → KILLED
68   1. replaced boolean return with false for TreesTesting/KDTree$Point::equals → KILLED
     2. replaced boolean return with true for TreesTesting/KDTree$Point::equals → KILLED
70   1. replaced boolean return with true for TreesTesting/KDTree$Point::equals → NO_COVERAGE
75   1. replaced return value with "" for TreesTesting/KDTree$Point::toString → NO_COVERAGE
80   1. changed conditional boundary → KILLED
     2. Changed increment from 1 to -1 → KILLED
     3. negated conditional → KILLED
81   1. Replaced integer subtraction with addition → KILLED
82   1. Replaced integer multiplication with division → KILLED
     2. Replaced integer addition with subtraction → KILLED
84   1. replaced int return with 0 for TreesTesting/KDTree$Point::comparableDistance → KILLED
93   1. changed conditional boundary → KILLED
     2. Changed increment from 1 to -1 → KILLED
     3. negated conditional → SURVIVED
94   1. negated conditional → SURVIVED
95   1. Replaced integer subtraction with addition → SURVIVED
96   1. Replaced integer multiplication with division → KILLED
     2. Replaced integer addition with subtraction → SURVIVED
98   1. replaced int return with 0 for TreesTesting/KDTree$Point::comparableDistanceExceptAxis → SURVIVED
116  1. replaced return value with null for TreesTesting/KDTree$Node::getPoint → KILLED
120  1. replaced return value with null for TreesTesting/KDTree$Node::getLeft → NO_COVERAGE
124  1. replaced return value with null for TreesTesting/KDTree$Node::getRight → NO_COVERAGE
128  1. replaced int return with 0 for TreesTesting/KDTree$Node::getAxis → KILLED
133  1. changed conditional boundary → SURVIVED
     2. negated conditional → KILLED
134  1. replaced return value with null for TreesTesting/KDTree$Node::getNearChild → KILLED
     2. replaced return value with null for TreesTesting/KDTree$Node::getNearChild → KILLED
139  1. changed conditional boundary → SURVIVED
     2. negated conditional → SURVIVED
140  1. replaced return value with null for TreesTesting/KDTree$Node::getFarChild → SURVIVED
     2. replaced return value with null for TreesTesting/KDTree$Node::getFarChild → SURVIVED
144  1. replaced int return with 0 for TreesTesting/KDTree$Node::getAxisCoordinate → SURVIVED
149  1. replaced return value with null for TreesTesting/KDTree::getRoot → KILLED
153  1. negated conditional → KILLED
154  1. Replaced integer modulus with multiplication → KILLED
155  1. negated conditional → KILLED
     2. replaced return value with null for TreesTesting/KDTree::build → KILLED
156  1. removed call to java/util/Arrays::sort → KILLED
158  1. replaced int return with 0 for TreesTesting/KDTree::lambda$2 → KILLED
160  1. Replaced Shift Right with Shift Left → KILLED
162  1. Replaced integer addition with subtraction → KILLED
165  1. Replaced integer addition with subtraction → KILLED
166  1. Replaced integer addition with subtraction → KILLED
168  1. replaced return value with null for TreesTesting/KDTree::build → KILLED
172  1. negated conditional → NO_COVERAGE
179  1. Replaced integer modulus with multiplication → NO_COVERAGE
180  1. negated conditional → NO_COVERAGE
     2. replaced return value with null for TreesTesting/KDTree::insert → NO_COVERAGE
181  1. changed conditional boundary → NO_COVERAGE
     2. negated conditional → NO_COVERAGE
182  1. Replaced integer addition with subtraction → NO_COVERAGE
183  1. Replaced integer addition with subtraction → NO_COVERAGE
185  1. replaced return value with null for TreesTesting/KDTree::insert → NO_COVERAGE
189  1. negated conditional → KILLED
192  1. replaced return value with Optional.empty for TreesTesting/KDTree::search → KILLED
196  1. negated conditional → KILLED
197  1. replaced return value with Optional.empty for TreesTesting/KDTree::search → KILLED
     2. negated conditional → KILLED
198  1. replaced return value with Optional.empty for TreesTesting/KDTree::search → KILLED
202  1. replaced return value with null for TreesTesting/KDTree::findMin → KILLED
206  1. negated conditional → KILLED
207  1. negated conditional → KILLED
208  1. negated conditional → KILLED
     2. replaced return value with null for TreesTesting/KDTree::findMin → SURVIVED
209  1. replaced return value with null for TreesTesting/KDTree::findMin → KILLED
214  1. replaced return value with null for TreesTesting/KDTree::findMin → KILLED
217  1. replaced int return with 0 for TreesTesting/KDTree::lambda$4 → KILLED
223  1. replaced return value with null for TreesTesting/KDTree::findMax → NO_COVERAGE
227  1. negated conditional → NO_COVERAGE
228  1. negated conditional → NO_COVERAGE
229  1. negated conditional → NO_COVERAGE
     2. replaced return value with null for TreesTesting/KDTree::findMax → NO_COVERAGE
230  1. replaced return value with null for TreesTesting/KDTree::findMax → NO_COVERAGE
235  1. replaced return value with null for TreesTesting/KDTree::findMax → NO_COVERAGE
```

```
238  1. replaced int return with 0 for TreesTesting/KDTree::lambda$6 → NO_COVERAGE
245  1. replaced return value with null for TreesTesting/KDTree::lambda$7 → NO_COVERAGE
250  1. negated conditional → KILLED
251  1. negated conditional → KILLED
252  1. negated conditional → NO_COVERAGE
256  1. negated conditional → NO_COVERAGE
     1. changed conditional boundary → SURVIVED
263  2. negated conditional → SURVIVED
266  1. replaced return value with null for TreesTesting/KDTree::delete → KILLED
270  1. replaced return value with null for TreesTesting/KDTree::findNearest → KILLED
     1. negated conditional → KILLED
274  2. replaced return value with null for TreesTesting/KDTree::findNearest → KILLED
     1. negated conditional → KILLED
275  2. replaced return value with null for TreesTesting/KDTree::findNearest → KILLED
     1. changed conditional boundary → SURVIVED
282  2. negated conditional → KILLED
     1. changed conditional boundary → SURVIVED
286  2. negated conditional → SURVIVED
288  1. replaced return value with null for TreesTesting/KDTree::findNearest → KILLED
```

Active mutators

- BOOLEAN_FALSE_RETURN
- BOOLEAN_TRUE_RETURN
- CONDITIONALS_BOUNDARY_MUTATOR
- EMPTY_RETURN_VALUES
- INCREMENTS_MUTATOR
- INVERT_NEGS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- NULL_RETURN_VALUES
- PRIMITIVE_RETURN_VALS_MUTATOR
- VOID_METHOD_CALL_MUTATOR

Tests examined

- TreesTesting.AllTreeTesting.[engine:junit-jupiter]/[class:TreesTesting.AllTreeTesting]/[method:findNearest()] (11 ms)
- TreesTesting.AllTreeTesting.[engine:junit-jupiter]/[class:TreesTesting.AllTreeTesting]/[method:findMin()] (12 ms)
- TreesTesting.AllTreeTesting.[engine:junit-jupiter]/[class:TreesTesting.AllTreeTesting]/[method:build()] (31 ms)
- TreesTesting.AllTreeTesting.[engine:junit-jupiter]/[class:TreesTesting.AllTreeTesting]/[method:delete()] (10 ms)

Report generated by PIT 1.6.8