Bài tập về Nhà K-d tree

Bài 1:Build K-d Tree from scratch

Bài 2: Find out balancing, removing elements operators (optional)

Bài làm

Bài 1:

Build K-d Tree

```
def build_kdtree(points, depth=0):
    n = len(points)

    if n <= 0:
        return None

    axis = depth % k

    sorted_points = sorted(points, key=lambda point: point[axis])

    return {
        'point': sorted_points[n // 2],
        'left': build_kdtree(sorted_points[:n // 2], depth + 1),
        'right': build_kdtree(sorted_points[n // 2 + 1:], depth + 1)
    }
```

Nearest Neighbor Search

```
def kdtree_native_closest_point(root, point, depth=0, best=None):
    if root is None:
        return best

    axis = depth % k

    next_best = None
    next_branch = None

    if best is None or distance_squared(point, best) > distance_squared(
point, root['point']):
        next_best = root['point']
    else:
        next_best = best

    if point[axis] < root['point'][axis]:
        next_branch = root['left']
    else:
        next_branch = root['right']

    return kdtree_naive_closest_point(next_branch, point, depth + 1, nex
t_best)



def closer_distance(pivot, p1, p2):
    if p1 is None:
        return p2

    if p2 is None:
        return p1

    d1 = distance_squared(pivot, p1)
    d2 = distance_squared(pivot, p2)

    if d1 < d2:
        return p1
    else:
        return p2
```

```
def kdtree_closest_point(root, point, depth=0):
    if root is None:
        return None

    axis = depth % k

    next_branch = None
    opposite_branch = None

    if point[axis] < root['point'][axis]:
        next_branch = root['left']
        opposite_branch = root['right']
    else:
        next_branch = root['right']
        opposite_branch = root['left']

    best = closer_distance(point,
                           kdtree_closest_point(next_branch,
                                                point,
                                                depth + 1),
                           root['point'])

    if distance_squared(point, best) > (point[axis] - root['point'][axis
]) ** 2:
        best = closer_distance(point,
                               kdtree_closest_point(opposite_branch,
                                                    point,
                                                    depth + 1),
                               best)

    return best
```

Range search

```
def search(self, point):
    if self.accept is None:
      point = np.asarray(point)
    if self.k != utils.check_dimensionality(point, accept=self.accept):
      raise ValueError("Point must be same dimensionality as the KDTree"
)
    elif np.all(self.value == point):
```

```
      return self
    elif point[self.axis] >= self.value[self.axis]:
      if self.right is None:
        return None
      else:
        return self.right.search(point)
    else:
      if self.left is None:
        return None
      else:
        return self.left.search(point)
```

Bài 2:

```
def balance(self):
    if not self.invariant():
      values = self.collect()
      return KDTree.initialize(values, k=self.k, init_axis=self.axis, ac
cept=self.accept)
    return self
```