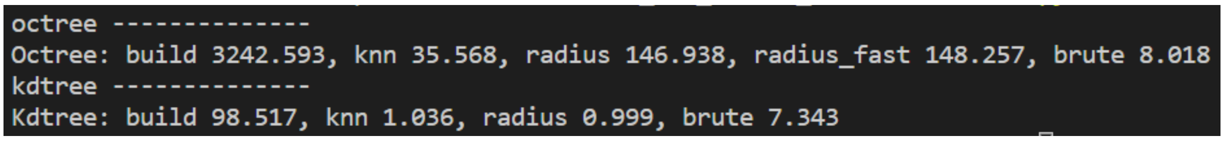
三维点云作业 第二章

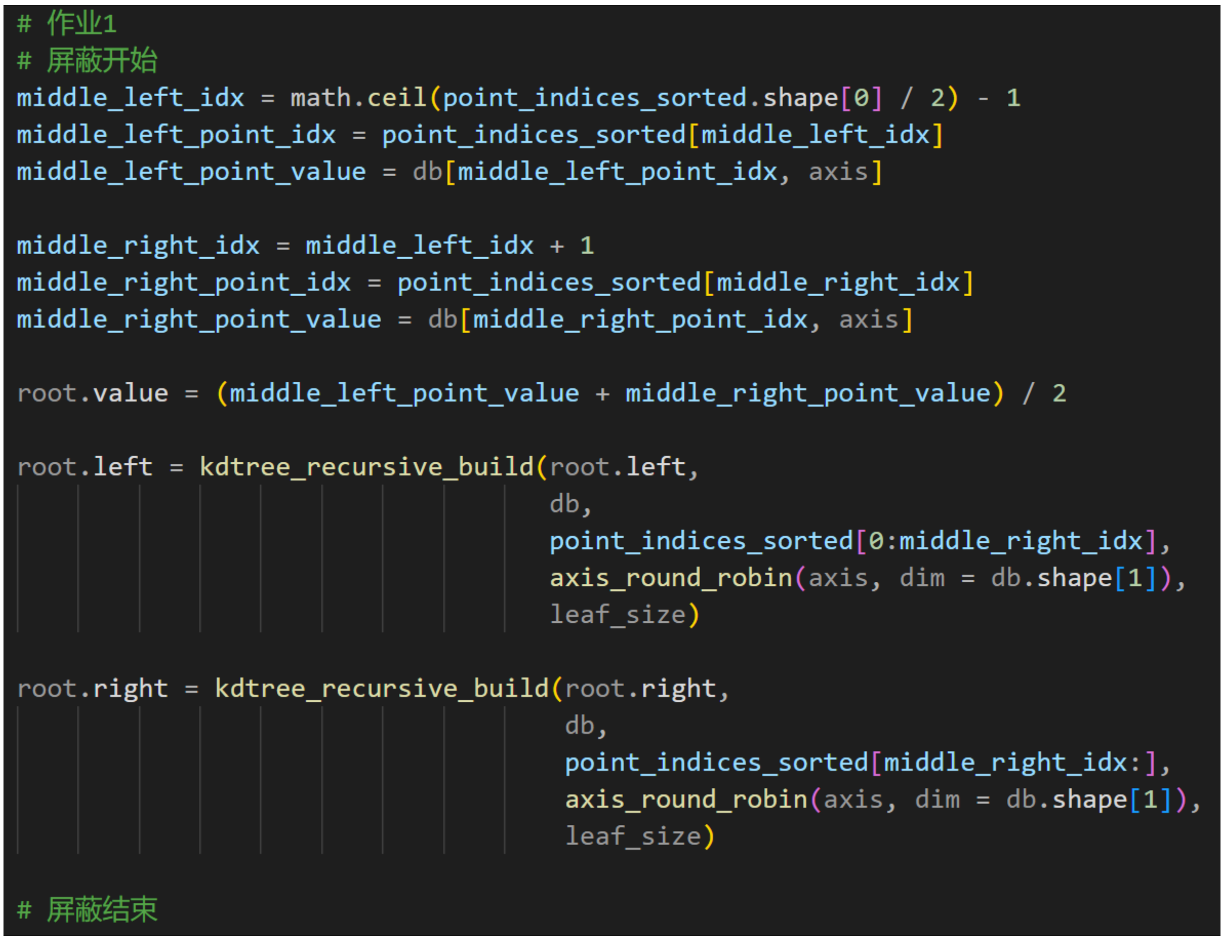
学习了KD-tree，Octree等数据结构及其思想，完成了课程作业。
通过使用课程提供的点云和代码，对两种算法的效率进行了比较，结果如下



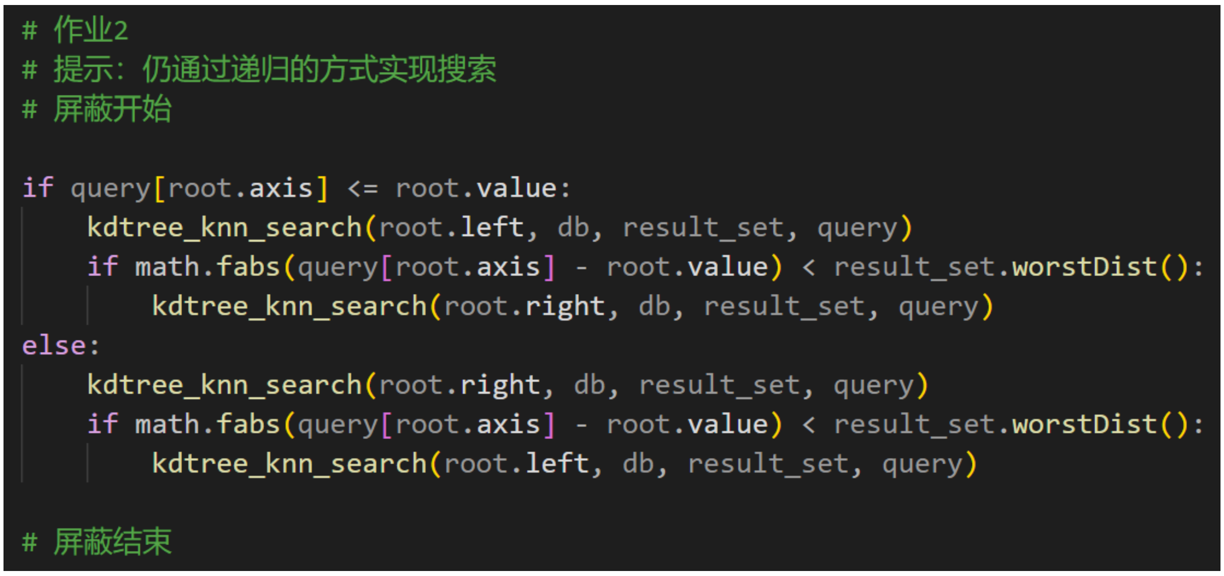
```
octree --------------
Octree: build 3242.593, knn 35.568, radius 146.938, radius_fast 148.257, brute 8.018
kdtree --------------
Kdtree: build 98.517, knn 1.036, radius 0.999, brute 7.343
```

发现Kdtree在效率上明显优于Octree
但是Octree中radius与radius_fast并无区别，不确定是数据特殊性还是写错了

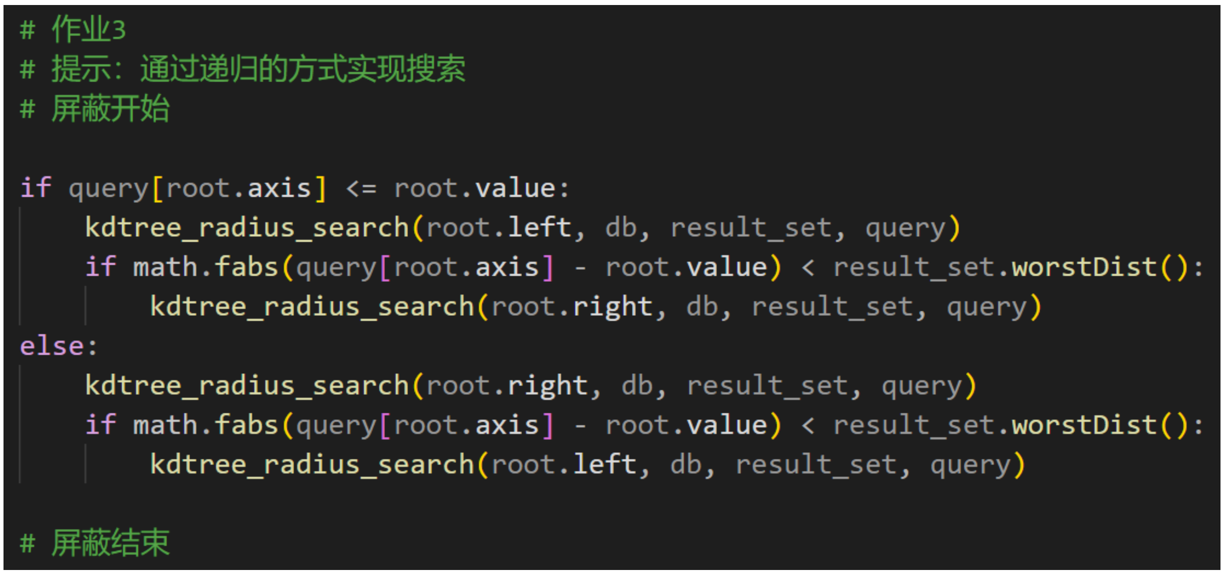
核心代码：



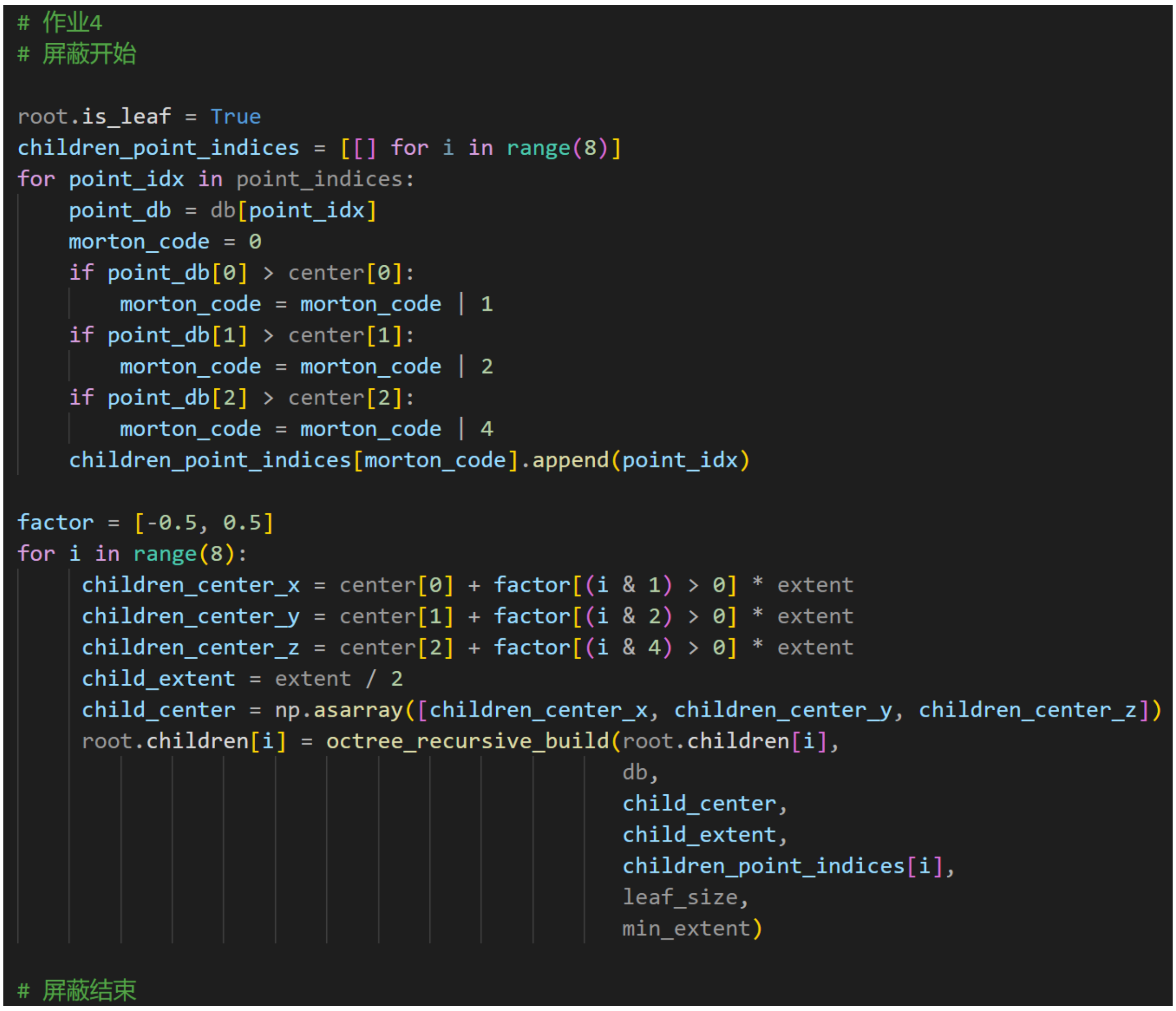
```python
# 作业1
# 屏蔽开始
middle_left_idx = math.ceil(point_indices_sorted.shape[0] / 2) - 1
middle_left_point_idx = point_indices_sorted[middle_left_idx]
middle_left_point_value = db[middle_left_point_idx, axis]

middle_right_idx = middle_left_idx + 1
middle_right_point_idx = point_indices_sorted[middle_right_idx]
middle_right_point_value = db[middle_right_point_idx, axis]

root.value = (middle_left_point_value + middle_right_point_value) / 2

root.left = kdtree_recursive_build(root.left,
                                   db,
                                   point_indices_sorted[0:middle_right_idx],
                                   axis_round_robin(axis, dim = db.shape[1]),
                                   leaf_size)

root.right = kdtree_recursive_build(root.right,
                                    db,
                                    point_indices_sorted[middle_right_idx:],
                                    axis_round_robin(axis, dim = db.shape[1]),
                                    leaf_size)

# 屏蔽结束
```



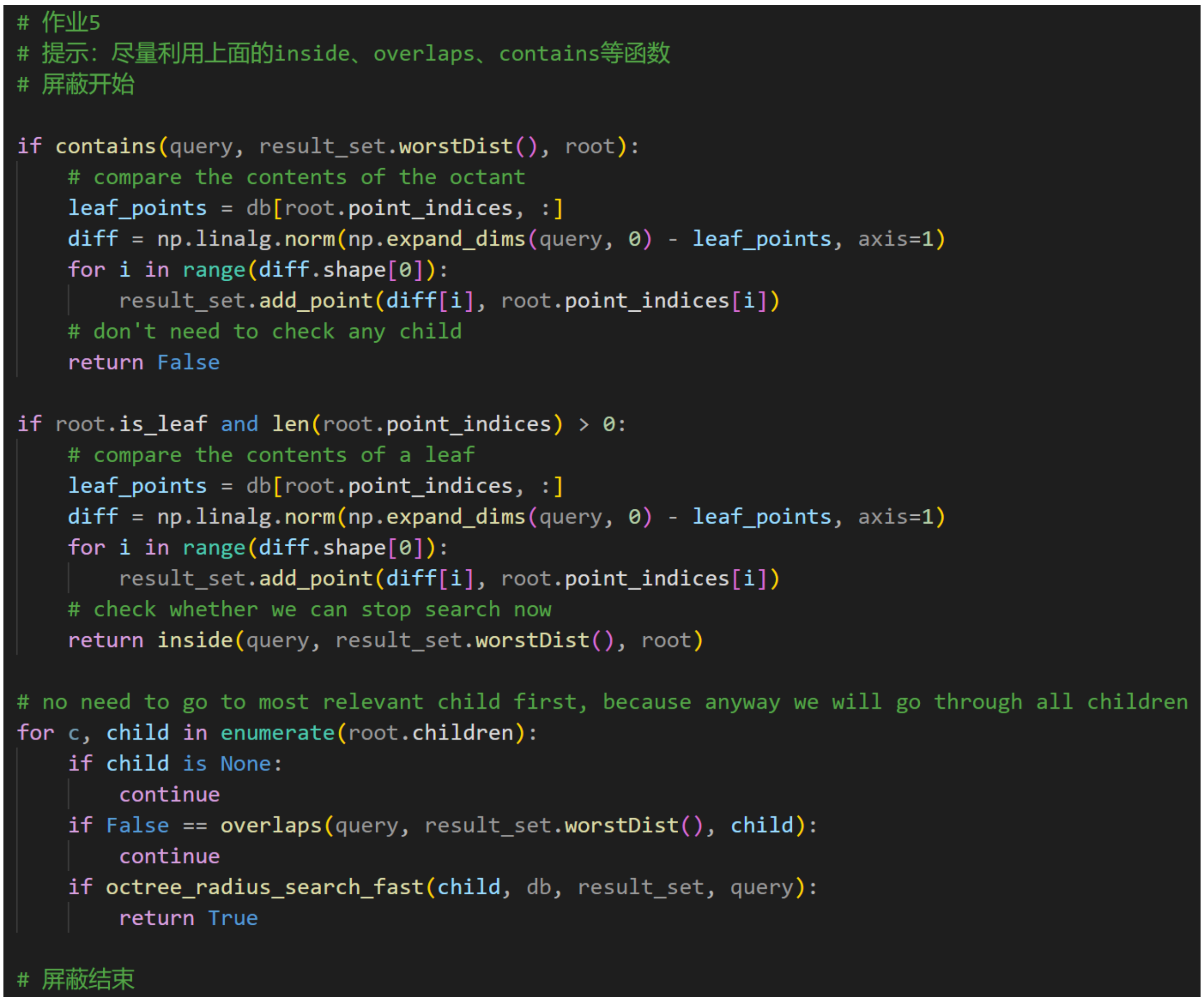
```python
# 作业2
# 提示：仍通过递归的方式实现搜索
# 屏蔽开始

if query[root.axis] <= root.value:
    kdtree_knn_search(root.left, db, result_set, query)
    if math.fabs(query[root.axis] - root.value) < result_set.worstDist():
        kdtree_knn_search(root.right, db, result_set, query)
else:
    kdtree_knn_search(root.right, db, result_set, query)
    if math.fabs(query[root.axis] - root.value) < result_set.worstDist():
        kdtree_knn_search(root.left, db, result_set, query)

# 屏蔽结束
```



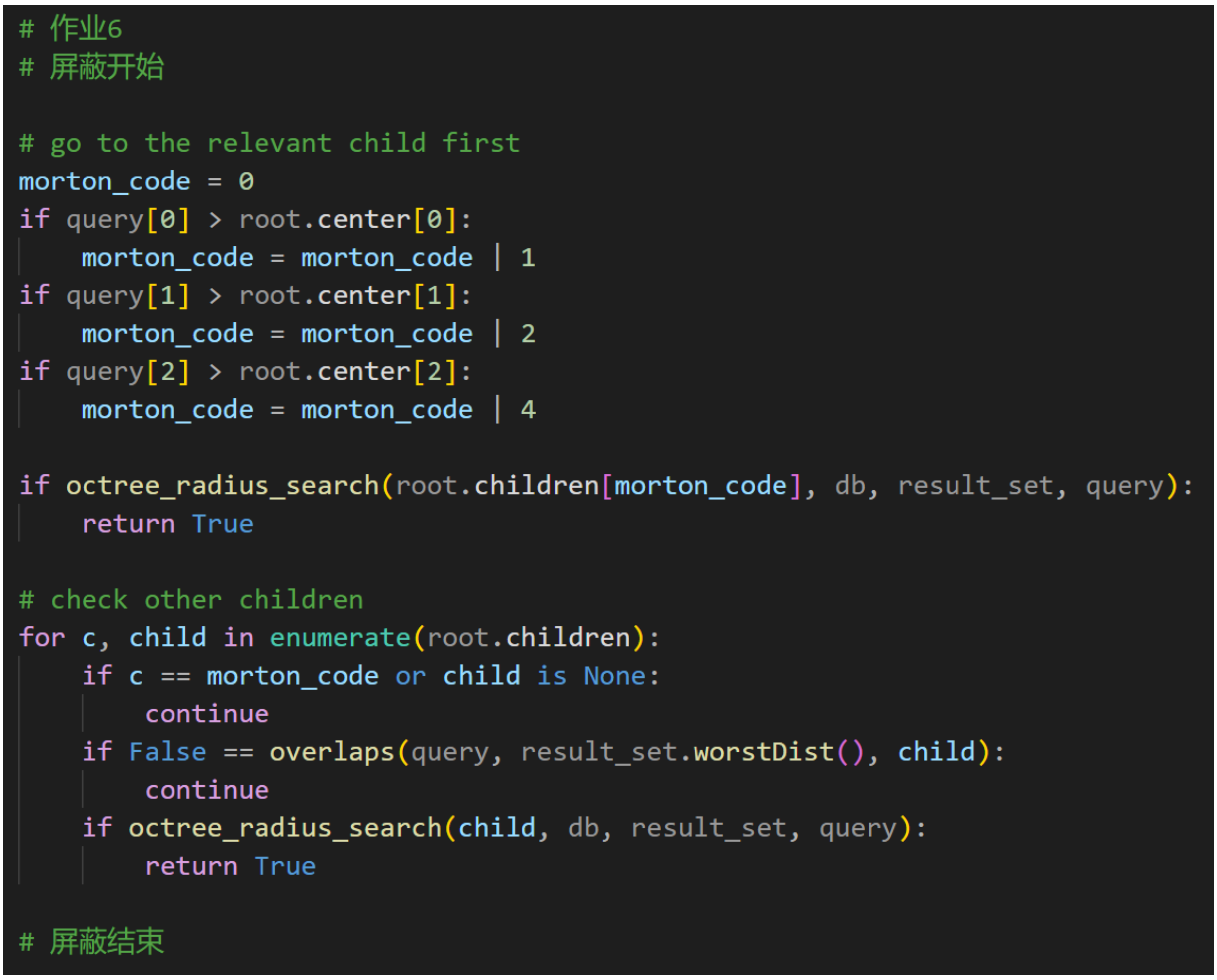
```python
# 作业3
# 提示：通过递归的方式实现搜索
# 屏蔽开始

if query[root.axis] <= root.value:
    kdtree_radius_search(root.left, db, result_set, query)
    if math.fabs(query[root.axis] - root.value) < result_set.worstDist():
        kdtree_radius_search(root.right, db, result_set, query)
else:
    kdtree_radius_search(root.right, db, result_set, query)
    if math.fabs(query[root.axis] - root.value) < result_set.worstDist():
        kdtree_radius_search(root.left, db, result_set, query)

# 屏蔽结束
```



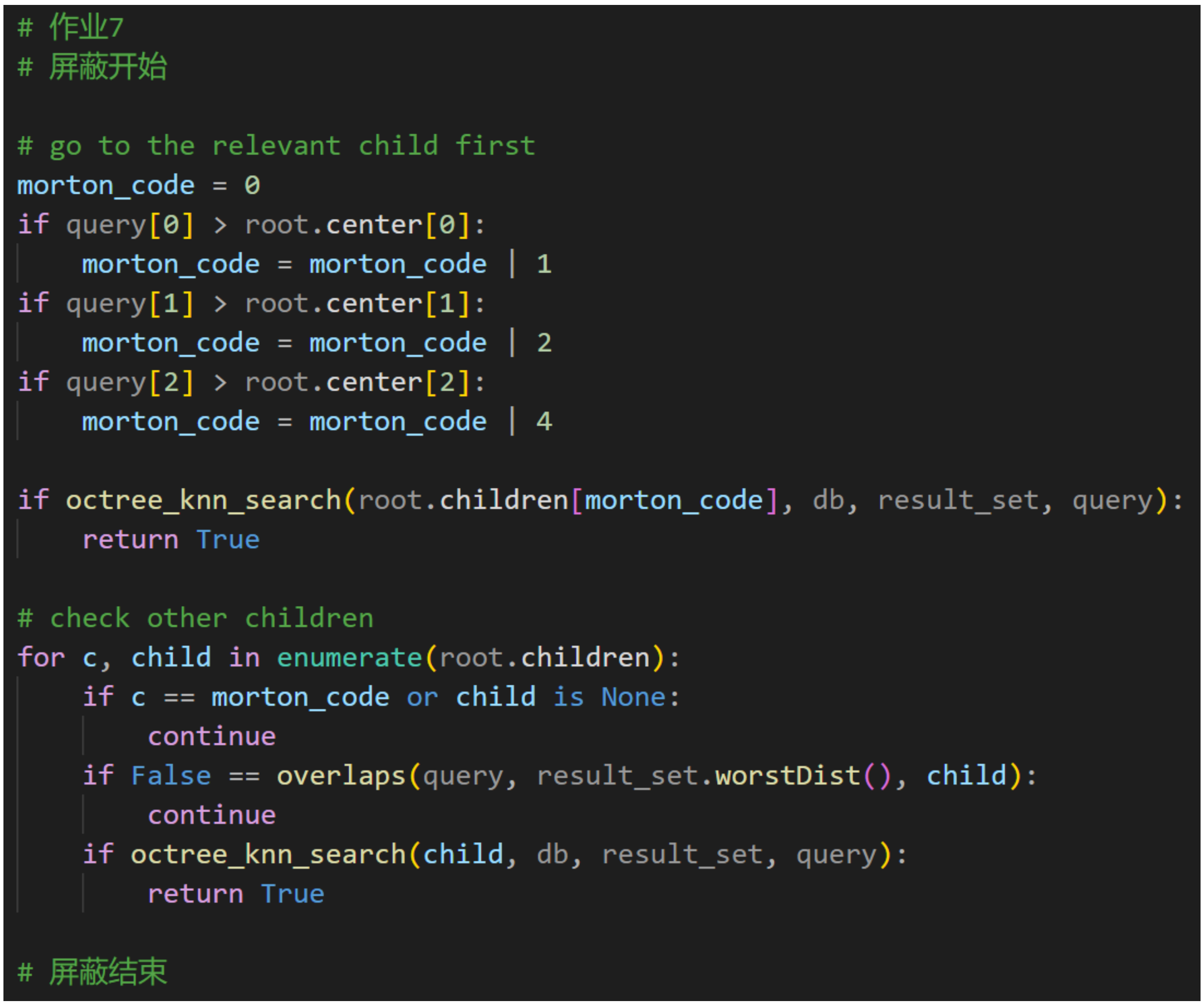
```python
# 作业4
# 屏蔽开始

root.is_leaf = True
children_point_indices = [[] for i in range(8)]
for point_idx in point_indices:
    point_db = db[point_idx]
    morton_code = 0
    if point_db[0] > center[0]:
        morton_code = morton_code | 1
    if point_db[1] > center[1]:
        morton_code = morton_code | 2
    if point_db[2] > center[2]:
        morton_code = morton_code | 4
    children_point_indices[morton_code].append(point_idx)

factor = [-0.5, 0.5]
for i in range(8):
    children_center_x = center[0] + factor[(i & 1) > 0] * extent
    children_center_y = center[1] + factor[(i & 2) > 0] * extent
    children_center_z = center[2] + factor[(i & 4) > 0] * extent
    child_extent = extent / 2
    child_center = np.asarray([children_center_x, children_center_y, children_center_z])
    root.children[i] = octree_recursive_build(root.children[i],
                                              db,
                                              child_center,
                                              child_extent,
                                              children_point_indices[i],
                                              leaf_size,
                                              min_extent)

# 屏蔽结束
```



```python
# 作业5
# 提示：尽量利用上面的inside、overlaps、contains等函数
# 屏蔽开始

if contains(query, result_set.worstDist(), root):
    # compare the contents of the octant
    leaf_points = db[root.point_indices, :]
    diff = np.linalg.norm(np.expand_dims(query, 0) - leaf_points, axis=1)
    for i in range(diff.shape[0]):
        result_set.add_point(diff[i], root.point_indices[i])
    # don't need to check any child
    return False

if root.is_leaf and len(root.point_indices) > 0:
    # compare the contents of a leaf
    leaf_points = db[root.point_indices, :]
    diff = np.linalg.norm(np.expand_dims(query, 0) - leaf_points, axis=1)
    for i in range(diff.shape[0]):
        result_set.add_point(diff[i], root.point_indices[i])
    # check whether we can stop search now
    return inside(query, result_set.worstDist(), root)

# no need to go to most relevant child first, because anyway we will go through all children
for c, child in enumerate(root.children):
    if child is None:
        continue
    if False == overlaps(query, result_set.worstDist(), child):
        continue
    if octree_radius_search_fast(child, db, result_set, query):
        return True

# 屏蔽结束
```



```python
# 作业6
# 屏蔽开始

# go to the relevant child first
morton_code = 0
if query[0] > root.center[0]:
    morton_code = morton_code | 1
if query[1] > root.center[1]:
    morton_code = morton_code | 2
if query[2] > root.center[2]:
    morton_code = morton_code | 4

if octree_radius_search(root.children[morton_code], db, result_set, query):
    return True

# check other children
for c, child in enumerate(root.children):
    if c == morton_code or child is None:
        continue
    if False == overlaps(query, result_set.worstDist(), child):
        continue
    if octree_radius_search(child, db, result_set, query):
        return True

# 屏蔽结束
```


```python
# 作业7
# 屏蔽开始

# go to the relevant child first
morton_code = 0
if query[0] > root.center[0]:
    morton_code = morton_code | 1
if query[1] > root.center[1]:
    morton_code = morton_code | 2
if query[2] > root.center[2]:
    morton_code = morton_code | 4

if octree_knn_search(root.children[morton_code], db, result_set, query):
    return True

# check other children
for c, child in enumerate(root.children):
    if c == morton_code or child is None:
        continue
    if False == overlaps(query, result_set.worstDist(), child):
        continue
    if octree_knn_search(child, db, result_set, query):
        return True

# 屏蔽结束
```