一. BBF算法实现与对比

1.1 、问题背景

Best Bin First (BBF) 是一种基于k-d树改进的近似最近邻搜索算法，通过优先队列和搜索次数限制优化查询效率。本作业要求实现BBF算法并与暴力搜索、标准k-d树搜索进行对比分析。

1.2 、题目要求

(a) 实现k-d树构建算法，要求支持欧氏距离计算。
(b) 实现BBF搜索算法，要求：

- 使用最大堆优先队列管理搜索路径
- 动态维护候选最近邻距离
- 支持设置最大搜索叶子节点数参数t

(c) 编写测试代码，在不同维度的数据集上验证算法正确性。

1.3. 对比实验

在相同数据集上对比以下算法：

- 暴力搜索（Brute-force）
- 标准k-d树搜索
- BBF搜索（$t = 200$）
  记录以下指标：
  (a) 单次查询平均时间（运行100次取均值）
  (b) 准确率：返回结果与真实最近邻的欧氏距离比值（≤1.05视为

成功）

(c) 内存占用分析

绘制结果对比表格，并分析BBF的时空效率权衡。

1.4. 理论分析

(a) 解释当维度d增加时，BBF的准确率为何会下降?

(b) 推导BBF的渐进时间复杂度，并与k-d树的$O(\sqrt{n} + k \log n)$复杂度对比。

1.5、实验指导

1. **数据集生成**：

   附件是数据集生成的代码和已经生成的数据集。

2. BBF核心伪代码**：

```
PriorityQueue ← k-d tree root with priority=0
best_dist = ∞
while not queue.empty() and searched_nodes < t:
    node = queue.pop_max_priority()
    if node is leaf:
        searched_nodes += 1
        update best_dist with points in node
    else:
        split_dim = node.split_dimension
        query_val = query[split_dim]
        nearer_subtree, farther_subtree = decide_subtrees(node, query_val)
        queue.insert(farther_subtree, priority=1/(distance_to_split))
```

```
            queue.insert(nearer_subtree,
    priority=1/(distance_to_split))
return best_dist
```

1.6、提交要求

- 完整可运行代码（Python/C++/Java）
- Markdown报告含：算法描述、实现细节、实验结果图表、理论分析

2. 基于层次聚类树的图像检索

实现层次聚类树，对于300张以上的图像数据集，利用opencv的SIFT或者ORB特征提取方法对图像提取特征点，用层次聚类树方法构建词典，之后基于词向量进行图像最近邻查询。参考的算法伪代码如下：

算法: 层次聚类视觉词典图像检索系统

输入:
  图像数据集路径 dataset_path,
  查询图像路径 query_path,
  词典大小 K,
  返回结果数 top_k

输出: 前top_k相似图像路径及相似度

Begin:

1. 特征提取:
   for 每张图像 img in dataset_path:
   使用SIFT提取特征描述符 desc ← extract(img)
   保存 desc 到 all_descriptors 列表
2. 构建视觉词典:
   合并所有描述符 all_features ← concatenate(all_descriptors)
   层次聚类 cluster_labels ← AgglomerativeClustering(K).fit(all_features)
   for 每个簇 i in 0..K-1:
   visual_word[i] ← mean(all_features where cluster_labels == i)
   构建视觉词典 vocab ← [visual_word[0], ..., visual_word[K-1]]
3. 生成数据库词袋:
   构建KD树 vocab_tree ← KDTree(vocab)
   for 每张图像 img in dataset_path:
   desc ← 其对应特征描述符
   查找最近视觉单词 indices ← vocab_tree.query(desc, k=1)
   生成词频直方图 hist ← histogram(indices, bins=K)
   归一化 hist ← hist / $||hist||_2$
   保存 hist 到 database_bows

4. 查询处理:
   query_desc ← extract(query_path)　　# 提取查询特征
   query_hist ← 按步骤3生成查询词袋向量
   similarities ← cosine_similarity(query_hist, database_bows)
   top_indices ← argsort(similarities)[-top_k:]
   return 对应图像路径及相似度分数

End

提交时，请写一个Markdown实验报告，报告中请提交算法详细代码，设计思路，查询图像和查询结果图像，查询的分数。可以多做几组查询，调整算法中的不同参数，对算法的运行时间，效果等进行评测分析。