实验⼀：DHT开发

Kademlia DHT 是分布式哈希表的⼀种实现，它拥有⼀些很好的特性，如下: 节点 ID 与KEY是同样的值

域，都是使⽤ SHA-1 算法⽣成的 160 位摘要，这样⼤⼤简化了查询时的信息量，更便于查询。 可以使

⽤ XOR，计算任意两个节点的距离或节点和关键字的距离。 查找⼀条请求路径的时候，每个节点的信

息是完备的，只需要进⾏Log(n) 量级次跳转。 可根据查询速度和存储量的需求调整每个节点需要维护

的 DHT ⼤⼩。

⽽Kademlia中，最为关键的就是K\_Bucket算法。

以节点u为例，其路由表的⽣成过程如下:

1. 最初，u的路由表为⼀个单个的K桶，覆盖了整个160bit ID 空间

2. 当学习到新的节点信息后，则u会尝试把新节点的信息，根据其前缀值插⼊对应的K桶中。

a. 该桶没有满，则新节点直接插⼊这个K桶中;

b. 该K桶已经满了:

i. 如果该K桶覆盖范围包含了节点的ID，则把该K桶分裂为两个⼤⼩相同的新K桶，并对原桶内

的节点信息按照新的K桶前缀值进⾏重新分配;

ii. 如果该K桶覆盖范围没有包含节点的ID，则直接丢弃该新节点信息。

3. 上述过程不断重复，直到满⾜路由表的要求。达到距离近的节点的信息多、距离远的节点的信息少

的结果，这样就保证了路由查询过程能快速收敛。

实验⽬的

通过实现Kademlia DHT中的K\_Bucket算法，加深对分布式存储中的Kademlia DHT算法的理解，并提

升其编程能⼒和分布式系统设计能⼒。

实验前提：

1. 掌握了Kademlia DHT算法的基本原理和相关概念；

2. 学⽣应具备基本的编程知识和对Golang或TypeScript编程语⾔的了解；

3. 假设地址⻓度是20byte， 每个桶中节点的数量为3个。

实验内容

本次实验主要集中在Kademlia DHT中的K\_Bucket算法的实现，学⽣需要使⽤Golang或TypeScript完

成以下任务：

1. K\_Bucket算法实现：

a. 学⽣需要实现Kademlia DHT中的K\_Bucket数据结构，包括桶（Bucket）、节点（Node）等相

关数据结构 。

b. 学⽣应能够正确处理节点的插⼊、删除和更新等操作，根据节点ID将其分配到正确的桶中。

2. 接⼝实现：

需要为K\_Bucket结构提供两个接⼝：

◦ insertNode(nodeId string)：将给定的NodeId插⼊到正确的桶中。

◦ printBucketContents()：打印每个桶中存在的NodeID。

加分项：多节点模拟

1. 实现Peer结构，每个Peer都拥有上述实验中的内容；

2. 为之前的K\_Bucket结构增加⼀个接⼝：FindNode（nodeId string）bool。当别⼈调⽤这个接⼝

时，先执⾏⼀次insertNode操作，并查找⾃⼰桶中是否有这个节点：

a. 如果有这个节点，则返回true；

b. 如果没有这个节点，则从对应的桶中随机抽选2个节点，发送FindNode(nodeId)操作，并返回

false。

3. 当有节点加⼊时，通过其中⼀个peer对⾃⼰的节点信息进⾏⼴播；

4. 在主程序中初始化多个Peer（暂定5个），然后⽣成200个新的Peer，通过之前的5个初始化的节

点，加⼊这个⽹络；

5. 打印出这205个节点每个节点的桶的信息。

功能测试

1. 需要⾃⾏编写测试代码，对实现的K\_Bucket算法进⾏功能测试，验证插⼊和打印功能的正确性。

2. 可以设计额外的测试⽤例，评估算法在不同场景下的性能表现和正确性。

实验报告

1. 需要撰写实验报告，记录实现过程中的思考、遇到的问题和解决⽅案。

2. 报告中应包括实验结果的截图和分析，验证实现的正确性和性能表现。