

**2021** 级

《大数据存储与管理》课程

**实 验 报 告**

**姓 名 黄宇**

**学 号 U202115436**

**班 号 CS2104班**

**日 期 2024.04.20**

[一、选题 3](#_Toc12000)

[二、实验结构 3](#_Toc6757)

[2.1 数据结构的设计 3](#_Toc5879)

[2.2 操作流程分析 4](#_Toc7507)

[2.3 理论分析 5](#_Toc10682)

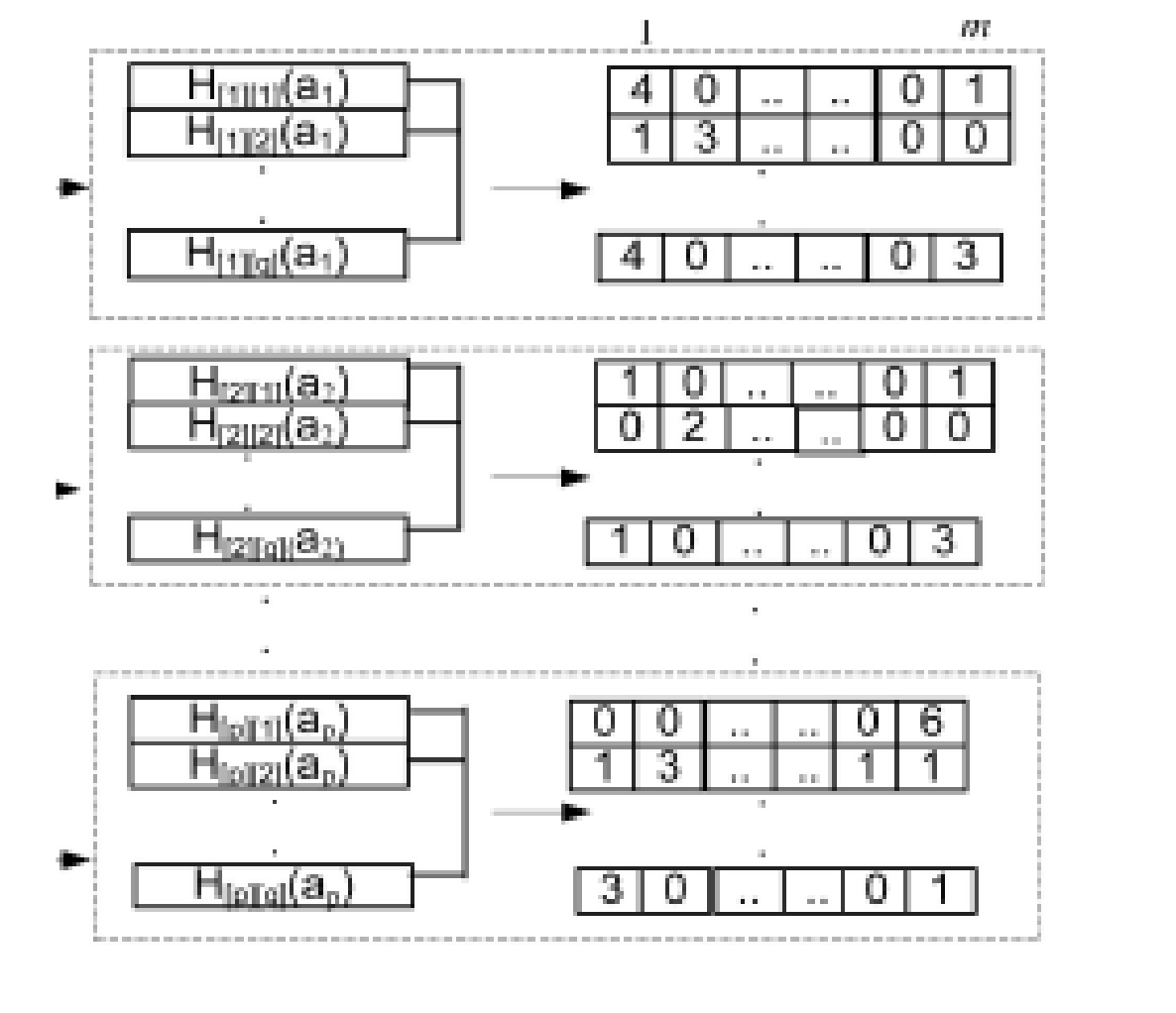
[2.4 实验测试性能 6](#_Toc11007)

[三、实验总结与心得 6](#_Toc25842)

# 一、选题

选题1： 基于Bloom Filter的设计

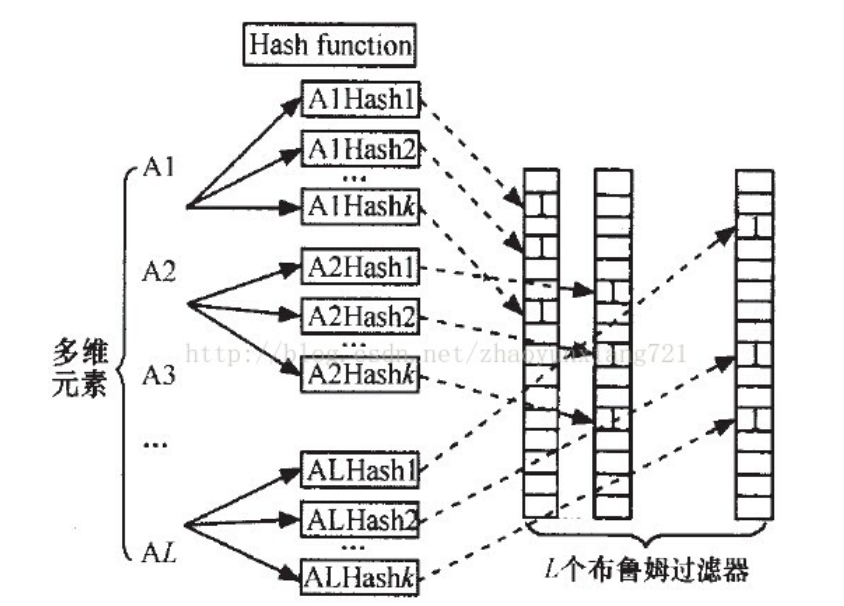
基于Bloom Filter的多维数据属性表示和索引



# 二、实验结构

## 2.1 数据结构的设计

在处理多维数据时，若采用基于Bloom Filter的数据结构，则会用到MDBF，也即多维布隆过滤器。在MDBF中采用多个标准的Bloom Filter组成，其个数等同于所需存储数据的维数。在元素查询过程中，通过多维元素各个属性值是否都存在相应过滤器中。其大体结构如下图所示。



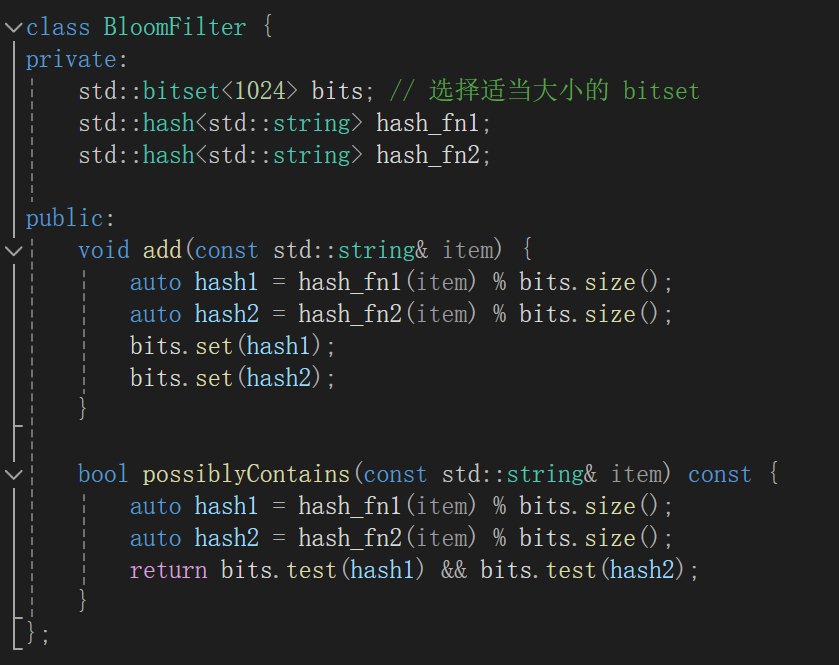
由于多维数据的整体性，通过上述MDBF得到的信息会由于单维属性的误判从而导致元素的整体误判，因此我们需要在上述MDBF的前提下，再使用一个Bloom Filter来存储所有属性联合哈希值。这样是我们能够更精确地判断多维数据地存在性。

举一例说明，但我们存在三维元素{1，2，3}，{4，5，6}时，若不使用额外Bloom Filter，则当我们检测{1，5，3}时会出现false positive，而改善后则不会出现该问题。

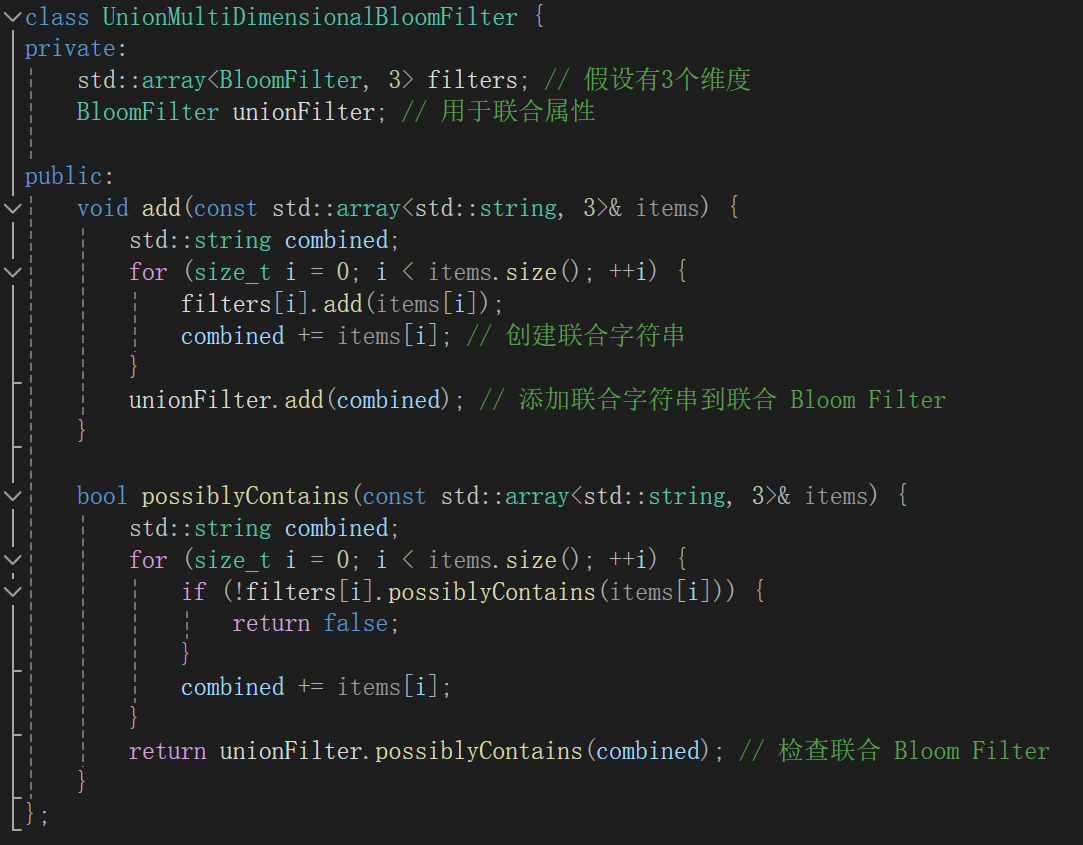
## 2.2 操作流程分析

在具体的代码设计中，我采用的是C++语言。

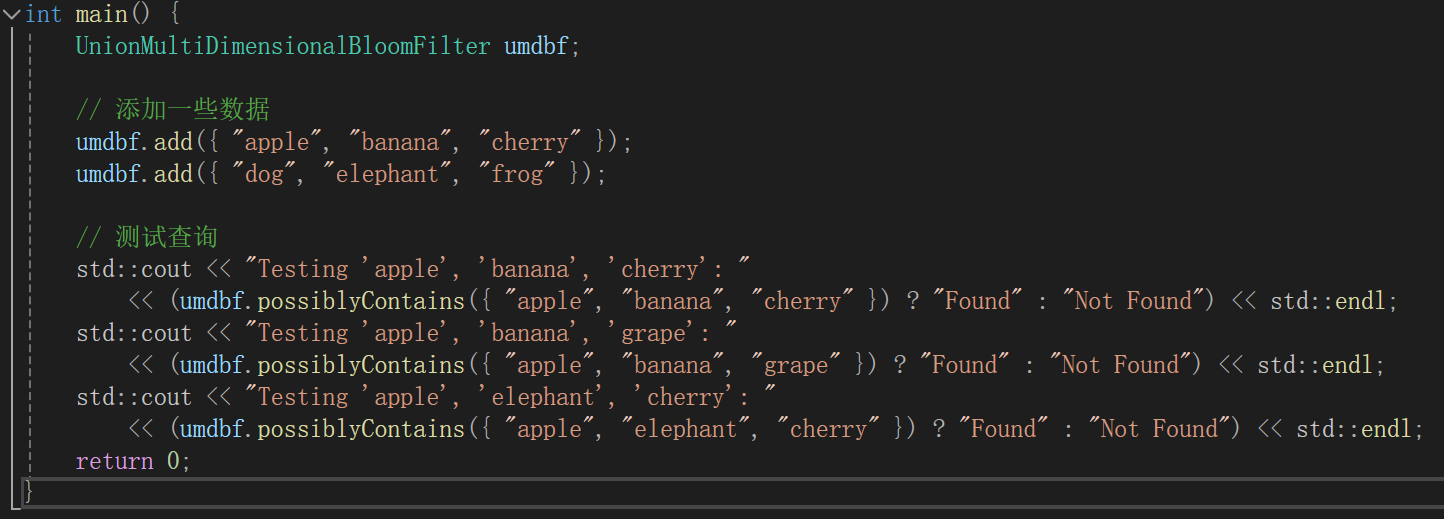
首先需要定义Bloom Filter的类。可以使用C++标准库中的bitset存储位数组，而哈希函数可以使用C++标准库中提供的std::hash计算哈希值（当然也可以使用其它哈希函数，本实验中采用库中函数）。得到如下代码：



之后为了处理多维数据，我们需要创建一个包含多个Bloom Filter的结构，其中每个维度都有一个对应的Bloom Filter。最后再额外使用一个Bloom Filter用于存储属性的联合值。其代码如下所示：



最后定义一个主函数来测试UMDBF以验证其功能，其代码如下图所示：



## 2.3 理论分析

由于该实验中采用的是标准Bloom Filter实现，因此不会发生False Negative。因为在Bloom Filter中一旦位置为1，就不会再进行更改，因此总能返回正确值。

对于False Positive，在我们的联合Bloom Filter中增加了额外的检查层次，理论上可以减少因低维错误肯定而认为元素存在的情况。但由于这个额外检查层自身也有可能出现错误肯定，尤其是多个不同属性组合的情况下。

对于其影响因素，大概有以下三点：

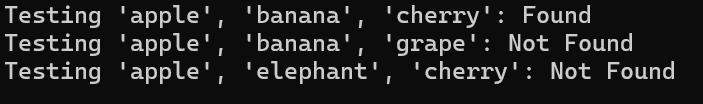
****哈希函数的选择****：选择好的哈希函数可以减少哈希冲突，从而降低错误肯定的概率。

****Bloom Filter 的大小****：增加 Bloom Filter 的大小可以降低错误肯定的概率，因为它提供了更多的位来存储信息，减少了不同元素哈希值的重叠概率。

****哈希函数的数量****：使用更多的哈希函数可以更均匀地分布哈希值，但也可能增加设置位的数量，导致更高的错误肯定概率。

## 2.4 实验测试性能

根据测试集可以得到结果如下图：



对于简单测试集，能得到相应结果。由于加入了额外的检查层，空间开销相较于前将会增大。

假设我们的Bloom Filter使用的是长度为m位的位数组。在原始的多维Bloom Filter中，如果有d个维度，每个维度使用一个独立的Bloom Filter，那么总的位数为d\*m。增加了一个维度后，总的位数变为（d+1）\*m。由此可知，当我们的数据位数越长，所需的空间开销则越多。

# 三、实验总结与心得

本次实验其数据结构主要完成了对多维联合Bloom Filter的代码实现，其具体实现原理并不复杂，本实验中所呈现的联合Bloom Filter较为简易，总体上能反映出算法的基本思想。