**随机算法课程**

**实验报告**

**实验一：**用minHash实现集合的相似性连接

姓名：陈峰

学号：7203610323

班级：20WJ0313

评分表：（由老师填写）

|  |  |
| --- | --- |
| 最终得分： | |
| 对实验题目的理解是否透彻： | |
| 实验步骤是否完整、可信 ： | |
| 代码质量 ： | |
| 实验报告是否规范 ： | |
| 趣味性、难度加分 ： | |
| 特 色： | 1 |
| 2 |
| 3 |

**一、实验题目概述**

1. 使用传统的双重循环实现集合的相似性的计算（主要计算交集）、

2. 使用MiniHash随机算法计算集合相似性计算

3. 使用局部敏感哈希LSH优化MiniHash，从而实现大数据下的集合相似性计算

**二、对实验步骤的详细阐述**

1. 载入数据。由于使用的数据较大，使用c++传统的fout，fin输入太慢，需要使用更加高效的算法，这里我是用了c中fread直接读取缓冲区，从而加快了数据的读取。根据测试，数据比简单使用fin快5倍以上。这里使用的是c++STL中的vector和unordered\_set存储数据，曾尝试使用new开辟动态数组进行管理，但是并不相差太多，并且后面的进行一些操作更加复杂。

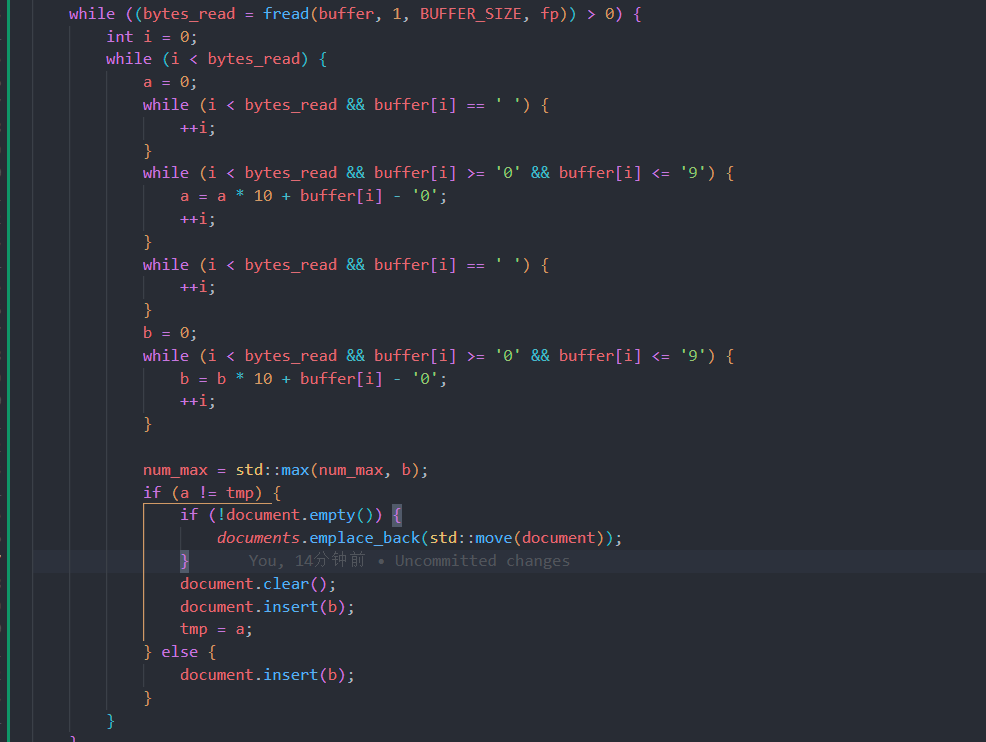
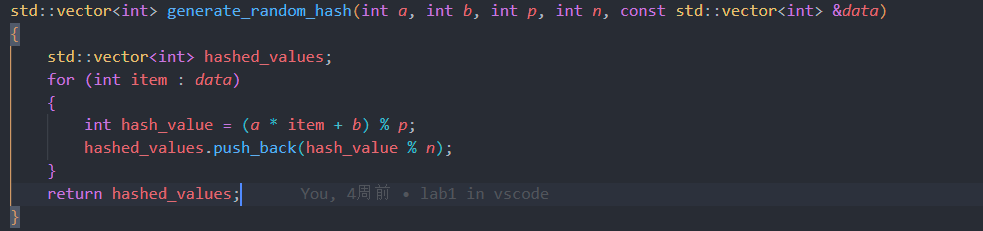


图 1读入优化

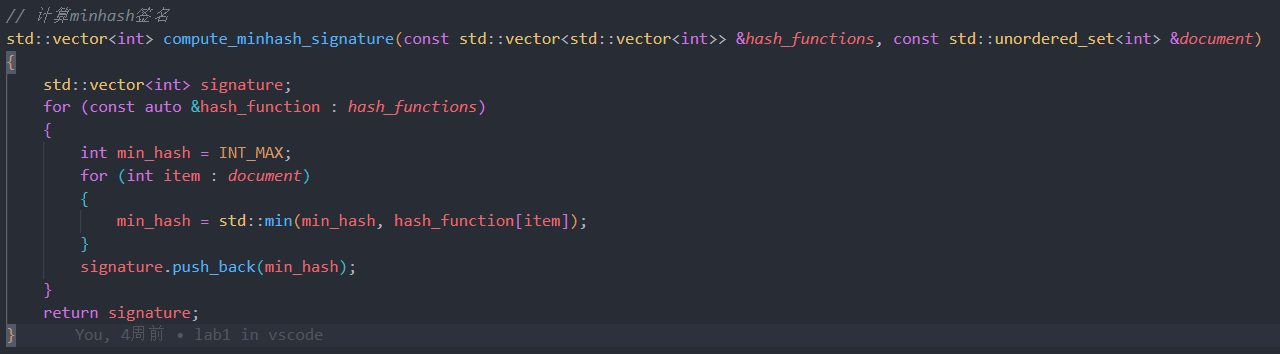
2. 实现传统的双重循环计算结合相似性。没什么好说的，一个优化就是数组排序之后然后计算交集，就可以把降低到.但是因为使用了c++STL模板函数set\_intersection也可以直接计算，它要求参与计算的集合满足一致的偏序。

3. 实现MiniHash算法。

（1）、创建哈希函数族；



（2）、对于给定的哈希函数族，产生哈希签名；



（3）、根据签名比较相似度。经过我们的实验，在测试机不超过10万组数据下，10个函数函数就可以达到90%以上的准确度，15个作用可以达到99%以上的准确度，更多具体信息参考实验结果中的分析。

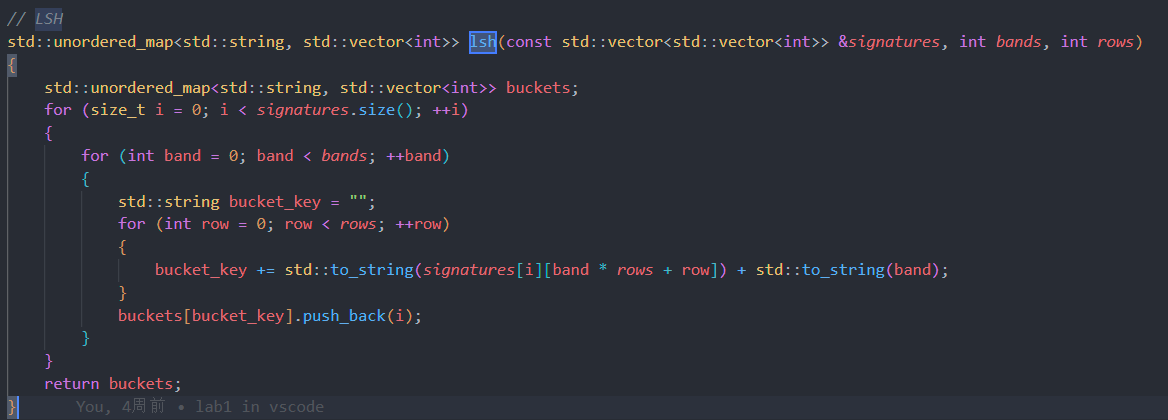
4. 实现局部敏感哈希LSH。使用MiniHash的签名，相当于我们只是将双重循环比较中的求解交集的复杂度由变为，但是我们依然需要使用

的循环进行两两比较，这在集合数量很大的时候时很难完成的，因此能否设计一个算法，将满足相似性条件的集合放在一起。基于这个思想提出了局部敏感哈希。局部敏感哈希算法：

（1）、选择更多的哈希函数，并将生成的哈希签名分成若干个部分（自己设置的超参数），我设置的是每个部分包含10个hash函数生成的签名，由此每个集合得到了若干个哈希值。

（2）、将每个集合映射到这些哈希值中。

（3）、对于每个哈希值，如果其中具有超过1个集合被映射到其中，那么对该哈希值中的集合之间进行两两比较，此时使用的是分割之前的hash值进行相似性的判断。



5. 验证MiniHash和局部敏感哈希算法的正确性。在大规模数据上进行实验之前，先要验证算法的正确性，我在100k数据规模中抽取了前10000组数据进行实验，实验结果验证了算法的正确性。

6. 对于给定的数据，进行实验。实验结果包含在experiment.txt文件中，实验的变量有：数据集、哈希函数个数和算法类别。由于在给定的大规模数据集上，完全无法对Naïve算法和MiniHash算法（不加LSH）运行，所以Naïve和MiniHash只在选取的少样本数据集上运行，MiniHash with LSH在完整的数据集上运行。

7. 总结实验结果，并使用python进行实验结果的可视化。对于c++输出的实验结果(即experiment.txt)使用了matplotlib库进行绘图。这里给出正确率和召回率的定义。

定义：在给定的集合相似度的阈值T之后，高于T的被认为是正样本，低于T的被认为是负样本。

计算正确率（Precision）：对于MiniHashWithLSH预测为正样本的，使用计算集合交集的算法进行验证，从而得到True Positive和False Positive的数量，从而计算正确率。

计算召回率：对于LSH哈希在同一个值下的集合，进行两两比较时被预测为负样本，我们也使用的算法计算相似度。从而得到True Negative 和False Negative。在这里定义和TN和FN其实和全局的TN和FN不同。因为在LSH中，如果两个集合没有被哈希在同一个值之中，就被认为是一个负样本，但是我们这里只统计了LSH被哈希在一个值中的集合。所以真正的召回率是要远高于我们计算的召回率。其实，我们通过实验结果可以看见，召回率通过都是很高的（接近于1），所以真是的召回率往往更加趋近于1.

**三、实验数据**

**1. 实验设置**

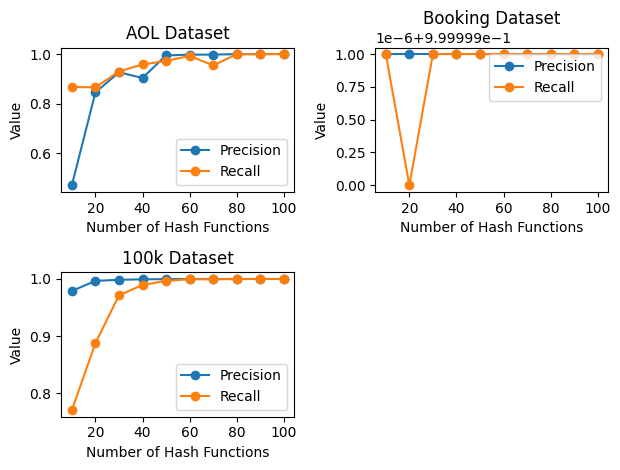
**实验环境**：

Window 10操作系统，Visual Studio2022

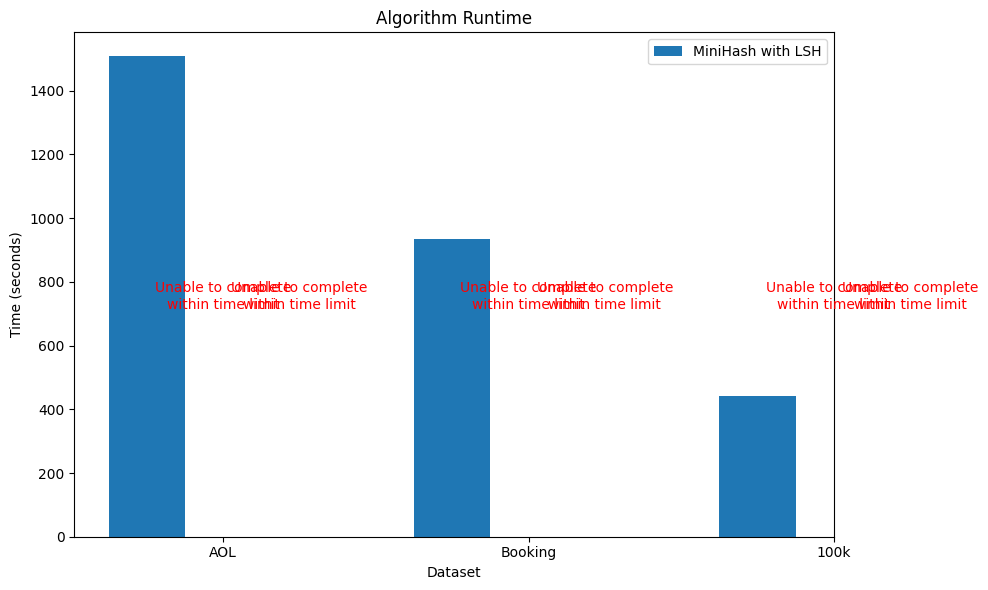
**数据**：

1. 从100k中抽取的10000组数据进行Naïve和MiniHash的实验  
 2. 在LSH上进行的E1\_AOL,E1\_Booking,E1\_Kosarak\_100k三个数据集

**2. 实验结果**

****

图表 1不同数据集的运行结果

****

图表 2不同算法的运行时间

**四、对实验结果的理解和分析**

在经过随机行打乱后，两个集合的最小哈希值相等的概率等于这两个集合的Jaccard相似度，证明如下：

现仅考虑集合S1和S2，那么这两列所在的行有下面3种类型：

1、S1和S2的值都为1，记为X

2、只有一个值为1，另一个值为0，记为Y

3、S1和S2的值都为0，记为Z

S1和S2交集的元素个数为x，并集的元素个数为x+y，所以sim(S1,S2) = Jaccard(S1,S2) = x/(x+y)。接下来计算h(S1)=h(S2)的概率，经过随机行打乱后，从上往下扫描，在碰到Y行之前碰到X行的概率为x/(x+y)，即h(S1)=h(S2)的概率为x/(x+y)。

对于使用Naïve算法和MiniHash Without LSh算法不能再一定时间内运行出结果，这个原因主要是因为集合的数量过多导致的结果，这两个算法都要对任意的两个集合进行比较。算法的时间复杂度为 。并且再LSH算法时，如果hash函数的数量过少，或者分块的超参数设置过大，也会导致hash冲突过多，多个集合被映射到同一个hash值，在进行二次比较的时候会严重影响算法的性能，甚至导致算法无法在有限时间运行。这里LSH的分块超参数设置为10，在哈希函数低于20的情况下，算法无法在20分钟内运行出结果。

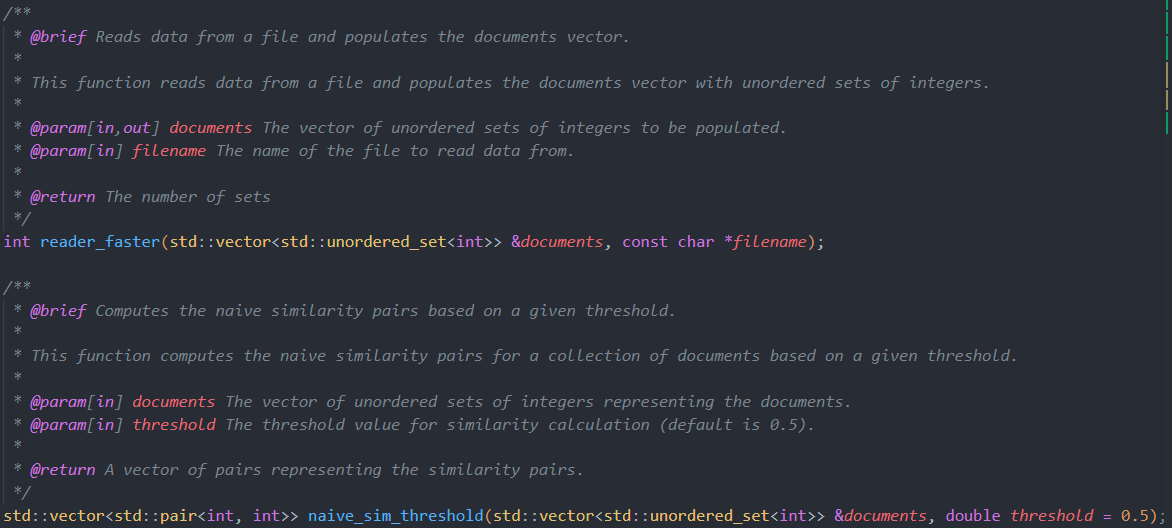
**五、实验过程中最值得说起的几个方面**

1. 使用c++编程。在实验课，老师建议使用c++编程，有利于了解算法的具体实现思路，虽然来说c++编程缺少很多实用的函数，代码量较大，但是确实可读性更强，更能理解算法思路。

2. 针对大数据的读入进行了优化。由于我们已经知道了输入的具体格式，可以直接从缓冲区中读入数据，而不用使用c++的流输入，这样可以加快数据的输入。

3. 使用了局部敏感哈希算法优化MiniHash。LSH本质上是一种居于距离的算法，可以将哈希签名的差别看作距离，只用一定哈希签名相同的点距离才被认为时邻居，然后在邻居之间进行比较。

4、对于代码，完全采用了doxygen风格的注释，增强了代码的可读性和复用性。



5、使用python对数据进行了可视化。

注：对每个大项，可自主根据需要添加小标题