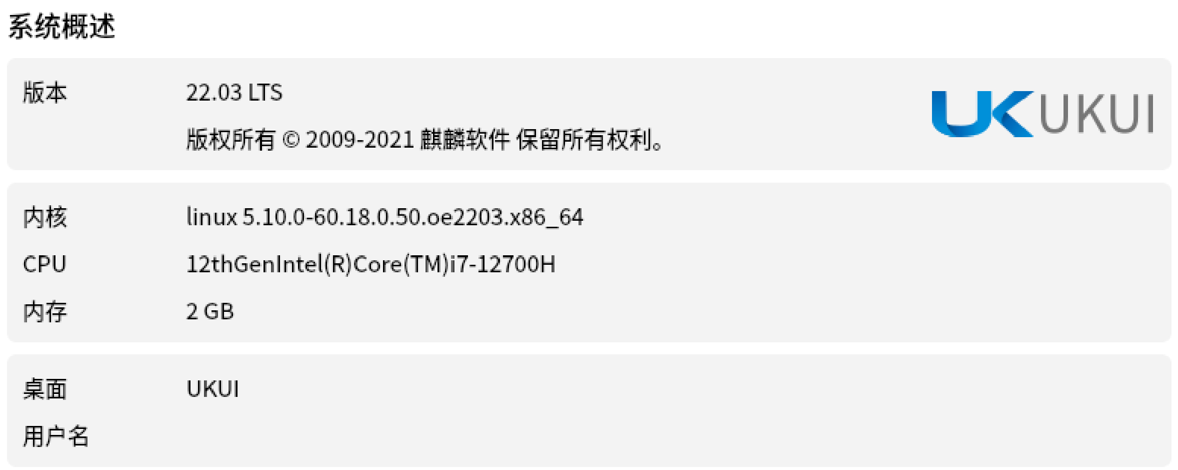
522031910747+李若彬+hw2

实验环境



系统概述

版本 22.03 LTS

版权所有 © 2009-2021 麒麟软件 保留所有权利。

内核 linux 5.10.0-60.18.0.50.oe2203.x86_64

CPU 12thGenIntel(R)Core(TM)i7-12700H

内存 2 GB

桌面 UKUI

用户名

实验设计

`Bloom Filter`的实现

- `BloomFilter`类的实现

```
class BloomFilter{
private:
    int m;//哈希数组的大小
    int k;//hash函数的个数
    uint64_t hash[2] = {0};
    uint64_t *array;//哈希数组
public:
    //构造函数
    BloomFilter(int set_m, int set_k){
        m = set_m;
        k = set_k;
        array = new uint64_t[set_m];
        for(int i = 0; i < m; i++){
            array[i] = 0;
        }
    }
    // 插入一个元素
    void insertNum(uint64_t num){
        for(int i = 0; i < k; i++){
            MurmurHash3_x64_128(&num, sizeof(num), i, hash);
            array[hash[1]%(m-1)] = 1;
        }
    }
    // 返回是否存在某个元素
    bool findNum(uint64_t num){
        for(int i = 0; i < k; i++){
            MurmurHash3_x64_128(&num, sizeof(num), i, hash);
            if(array[hash[1]%(m-1)] == 0){
                return false;
            }
        }
        return true;
    }
};
```

误报率测试函数

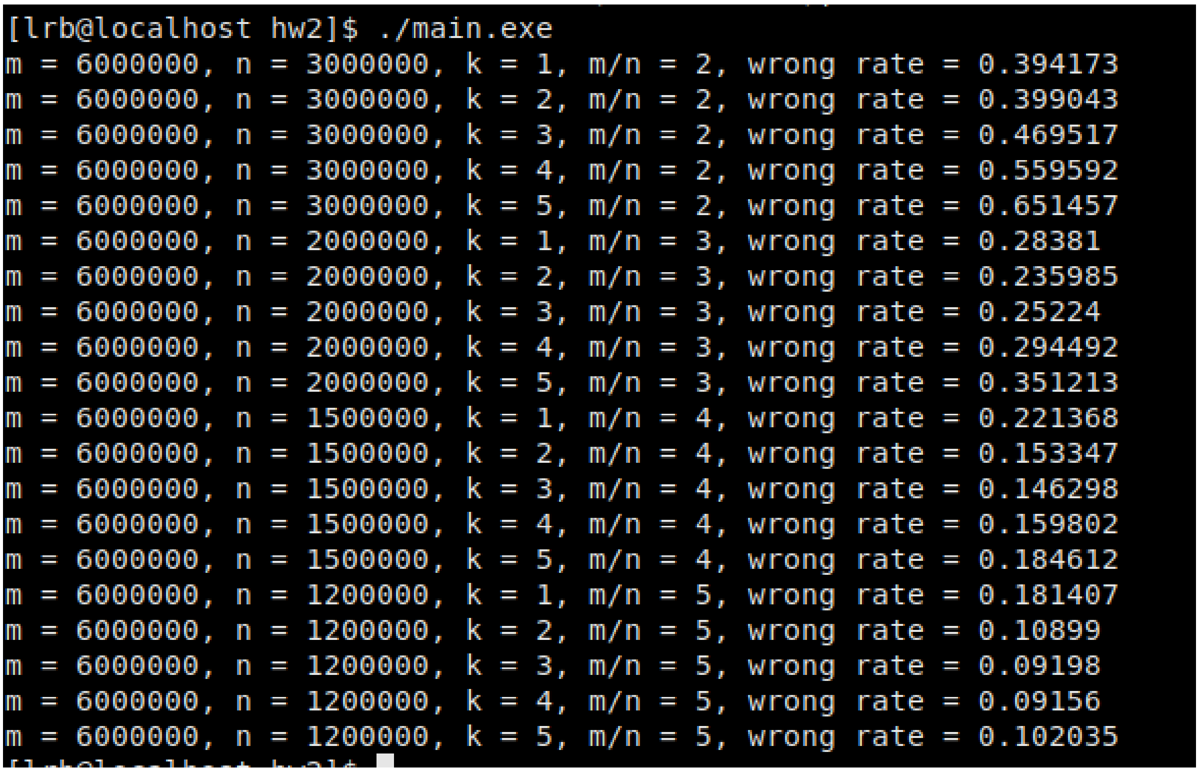
- 首先初始化哈希数组
- 创建一个`map`对象用于存储插入过程中已经出现过的元素，确保每次插入时以及后续查找时的元素都是不同的，插入`n`个元素，再创建一个`vector`对象用于存储查找的`t`个元素
- 创建一个`int`对象`count`，遍历用于存储查找元素的`vector`，如果其能被`bloom filter`查找到，则`count++`
- 误报率表示为$count \div t$

实验测试

```
int main() {
    srand(time(NULL));//初始化随机数
    int m = 6000000;
    for (int i = 2; i <=5; i++) {
        for (int j = 1; j <= 5; j++) {
            std::cout << "m = " << m << ", n = " << m / i << ", k = " << j << ", m/n = " << i << ", wrong rate = " << filterTest(m, m / i, j, m / 10) << std::endl;
        }
    }
    return 0;
}
```

- 初始化随机数，选取`m=6000000`(能被2、3、4、5、6整除)
- 使用2个嵌套循环，外层循环设定`m/n = 2~5`，内层循环相当于依次增加`k`值，达到增加哈希函数的个数的作用，每次循环插入`n`个元素，查找`m/10`个元素

实验结果



```
[lrb@localhost hw2]$ ./main.exe
m = 6000000, n = 3000000, k = 1, m/n = 2, wrong rate = 0.394173
m = 6000000, n = 3000000, k = 2, m/n = 2, wrong rate = 0.399043
m = 6000000, n = 3000000, k = 3, m/n = 2, wrong rate = 0.469517
m = 6000000, n = 3000000, k = 4, m/n = 2, wrong rate = 0.559592
m = 6000000, n = 3000000, k = 5, m/n = 2, wrong rate = 0.651457
m = 6000000, n = 2000000, k = 1, m/n = 3, wrong rate = 0.28381
m = 6000000, n = 2000000, k = 2, m/n = 3, wrong rate = 0.235985
m = 6000000, n = 2000000, k = 3, m/n = 3, wrong rate = 0.25224
m = 6000000, n = 2000000, k = 4, m/n = 3, wrong rate = 0.294492
m = 6000000, n = 2000000, k = 5, m/n = 3, wrong rate = 0.351213
m = 6000000, n = 1500000, k = 1, m/n = 4, wrong rate = 0.221368
m = 6000000, n = 1500000, k = 2, m/n = 4, wrong rate = 0.153347
m = 6000000, n = 1500000, k = 3, m/n = 4, wrong rate = 0.146298
m = 6000000, n = 1500000, k = 4, m/n = 4, wrong rate = 0.159802
m = 6000000, n = 1500000, k = 5, m/n = 4, wrong rate = 0.184612
m = 6000000, n = 1200000, k = 1, m/n = 5, wrong rate = 0.181407
m = 6000000, n = 1200000, k = 2, m/n = 5, wrong rate = 0.10899
m = 6000000, n = 1200000, k = 3, m/n = 5, wrong rate = 0.09198
m = 6000000, n = 1200000, k = 4, m/n = 5, wrong rate = 0.09156
m = 6000000, n = 1200000, k = 5, m/n = 5, wrong rate = 0.102035
[lrb@localhost hw2]$
```

m/n	k=1	k=2	k=3	k=4	k=5
2	0.394173	0.399043			
3	0.28381	0.235985	0.25224		
4	0.221368	0.153347	0.146298	0.159802	
5	0.181407	0.10899	0.09198	0.09156	0.102035

分析

- 实验结果显示当`k`值接近理论值`k`$=\ln 2 \cdot (\frac{m}{n})$ 时，`Bloom Filter`的误报率最低

  > $\frac{m}{n}=2 \ \rightarrow\ k=\ln 2 \cdot (\frac{m}{n})=1.386$
  >
  > $\frac{m}{n}=3 \ \rightarrow\ k=\ln 2 \cdot (\frac{m}{n})=2.079$
  >
  > $\frac{m}{n}=4 \ \rightarrow\ k=\ln 2 \cdot (\frac{m}{n})=2.773$
  >
  > $\frac{m}{n}=5 \ \rightarrow\ k=\ln 2 \cdot (\frac{m}{n})=3.466$

- 若`k`较小，则插入元素生成的哈希数较为单一，使得哈希数组中被标记为`1`的地方较为固定，更易发生哈希碰撞从而增加了误报率
- 若`k`较大，则插入元素生成的哈希数很庞杂，会使得哈希数组中几乎所有位置都被标记为`1`，从而引发哈希碰撞
- 当`k`的大小固定时，随`n`即插入元素的数量的增加，哈希碰撞的概率也相应增加从而产生误报