# 布隆过滤器实验报告

**Fan Xinyan**

## 一、实验要求

根据一个包含 10,000 个 1 到 1,000,000 的随机数的数据集，在用户给定误判率 p（false positive rate）的前提下，设计一个布隆过滤器，使得用户在给定一个新的数据是能以 1-p 的概率正确判断这个数是否在原数据集合中。

## 二、实验设计

### （一）Bloom Filter 类

包含参数：位数组及其大小，误判率，哈希函数个数，数据集大小；
包含函数：主要函数为哈希函数、构造函数、析构函数、插入到布隆过滤器和查询函数等

```
class BloomFilter{
private:
    double pError;
    int numHash;
    int numBit;
    int numNum;
    bool *bitMap;
    vector<int> randNum1;
    vector<int> randNum2;
    int prime;
    void initRandNum();
    long long int IntHashFunction(long long int key);
public:
    BloomFilter(double pError);
    ~BloomFilter();
    void initBloomFilter();
    void addToBloomFilter(int num);
    bool queryInBloomFilter(int num);
    void printBitMap();
};
```

### （二）函数设计

#### 1、初始化布隆过滤器

通过公式可以得知，当给定误判率时，位数组大小和哈希函数个数可由下列公式得出：

$$numBit = -\frac{numNum * \ln p}{(\ln 2)^2}$$

$$numHash = 0.7 * \frac{numBit}{numNum}$$

其中，numBit 表示位数组的大小，numHash 表示哈希函数的个数，numNum 表示数据集的大小（=10000）

Bloom Filter 构造函数内初始化参数：

```
numNum = 10000;
double m_n;
m_n = -1 * (log(pError)) / (log(2)*log(2));
numBit = ceil(m_n * numNum);
numHash = ceil(0.7 * m_n);
bitMap = new bool[numBit]{false};
```

## 2、将数据集中的数字插入布隆过滤器中

构造 numHash 个哈希函数，将数字哈希到不同的 index 中，将每个 bitMap[index]置为 true.

```
void BloomFilter::addToBloomFilter(int num){
    for(int i = 0; i < numHash; i++){                    //compute indexes
        long long int index;
        long long int key = ((randNum1[i] * num) + randNum2[i]);//linear function:num->key
        index = IntHashFunction(key);
        index %= numBit;
        bitMap[index] = true;
    }
}
```

哈希函数的设计：

### （1）如何形成多个不同的哈希函数

由于是有用户给定误判率后才能得到哈希函数的个数，最初的想法是能随机生成 numHash 个数，构造出 numHash 个映射，将原始数据映射到不同的位置上。最简单的映射是线性函数 ax+b，因此考虑了生成 numHash 个随机数对（a,b）分别存在向量 randNum1,randNum2 中。注意随机数的选取不能过大，避免造成相乘后数据太大而溢出。

```
void BloomFilter::initRandNum(){
    for(int i = 0; i < numHash; i++){
        int a = random(67);
        randNum1.push_back(a);
        for(int j = 1; j < 10; j++){
            a = random(67);
        }
        a = random(23);
        randNum2.push_back(a);
    }
}
```

（2）优化映射

虽然形成了不同种的 index，但是线性映射往往效果很差，冲突率很高，因此需要优化。

第一种方式是取模运算，选取一个合适的素数取模后作为 index 值，但是实验中尝试了很多素数，计算出来的实际误判率都要高出理想误判率一些，因此放弃。

第二种方式：考虑线性映射后的数字是 32-bits 的整数，因此在网上参考了 Redis 所采用的一种针对 32-bits 整数的哈希函数，效果很好，实际误判率几乎等于理想误判率。函数具体如下：

```
long long int BloomFilter::IntHashFunction(long long int key){
	key += ~(key << 15);
	key ^= (key >> 10);
	key += (key << 3);
	key ^= (key >> 6);
	key += ~(key << 11);
	key ^= (key >> 16);
	return key;
}
```

## 3、查询一个数是否在数据集中

对于一个给定的数，对其进行 numHash 次哈希，判定对应的 bitMap[index]的值是否为 true.若所有 index 上的值都为 true，则证明这个数在原数据集中；反之，则不在。

```
bool BloomFilter::queryInBloomFilter(int num){
	bool flag = false;
	for(int i = 0; i < numHash; i++){
		long long int index;
		long long int key = ((randNum1[i] * num) + randNum2[i]);
		index = IntHashFunction(key);
		index %= numBit;
		if(bitMap[index] == false){
			flag = true;
			break;
		}
	}
	if(flag == false){
		return true;
	}else{
		return false;
	}
}
```

# 三、测试用例及运行结果

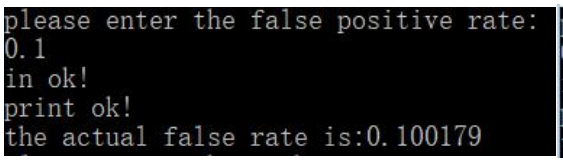
测试部分分为两个部分。

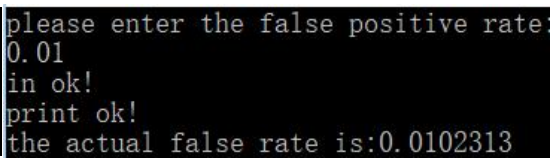
## （一）计算误判率

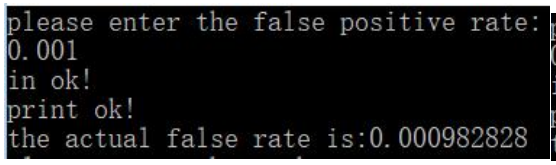
通过“numNotInSet.cpp”程序计算出所有不在给定数据集却是（1，1000000）中的数，存到"numNotIn.txt"中。

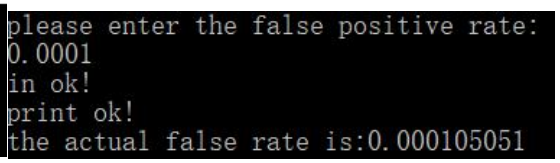
在将"numNotIn.txt"中的数据（共 numOfNotIn 个）放入“bloom_filter.cpp”中进行判断，统计判定错（即判断认为在原数据集中）的个数 conError，计算出实际的误判率 falseRate = conError / numOfNotIn，与理想误判率 p 进行对比。结果如下：

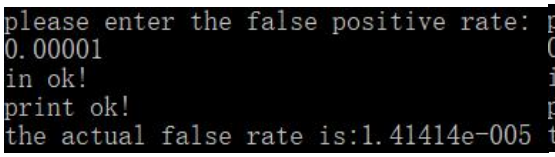
| 理想误判率 p | 实际误判率 falseRate |
|---|---|
| 0.1 | 0.100179 |
| 0.01 | 0.0102313 |
| 0.001 | 0.000982828 |
| 0.0001 | 0.000105051 |
| 0.00001 | 0.000014141 |
| 0.000001 | 0 |

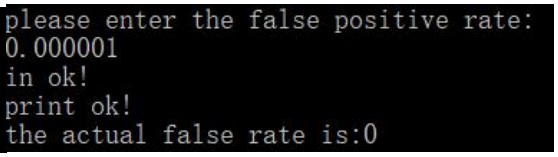


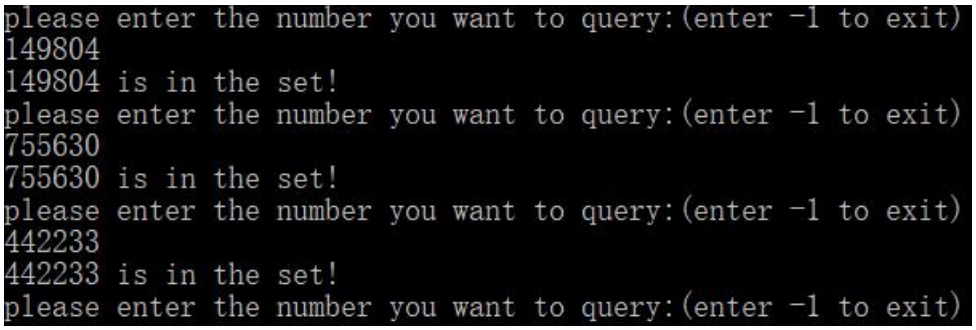




## （二）给定一个数进行查询

测试用例：误判率 p=0.00001

### 1、全在数据集中的测试用例

```
149804
755630
442233
```




## 2、不在数据集中的测试用例

```
858988
15
224656
165
72068
3827
```