实验二 哈希表的设计

哈希表简介：

哈希表

一个哈希表（hash table）是包含如下模块的用于解决字典问题的数据结构：

- 一个有 n 个元素的表T，表的元素用下标$N = 0,1,\ldots,n-1$来访问
- 哈希函数h（hash function），它将 M 中的元素映射到 N 中，M 中包含m个键值

我们假设$n < m$。T中每个元素都是一个内存字，可以存下 M 中任何一个元素的编码。理想情况下，哈希函数将 $S \subset M$ 中的不同键值映射到表中不同的位置

实验内容：

> 1.编写属于2-通用簇的某个哈希函数代码（30 分）
>
> •输入：键值
>
> •输出：哈希值
>
> 2.给定一个较大的m值，例如10^9，和一个较小的n值，例如1000，通过实验观察多次插入操作后链表的平均长度（键值随机采样），并与理论结果进行对比分析（40 分）
>
> 3.对于（2）的结果，使用 2-通用哈希函数簇中不同的哈希函数，观察并分析结果的差异（30 分）

第一问：

一下是我编写的两个哈希函数：

```
long long get_hash1(long long x){
    assert(mod>0);
    return ((((long long)x<<1)%mod*(long long)x%mod)+x)%mod;
}
```

```
long long get_hash2(long long x){
    assert(mod>0);
    return x%mod;
}
```

第二问：

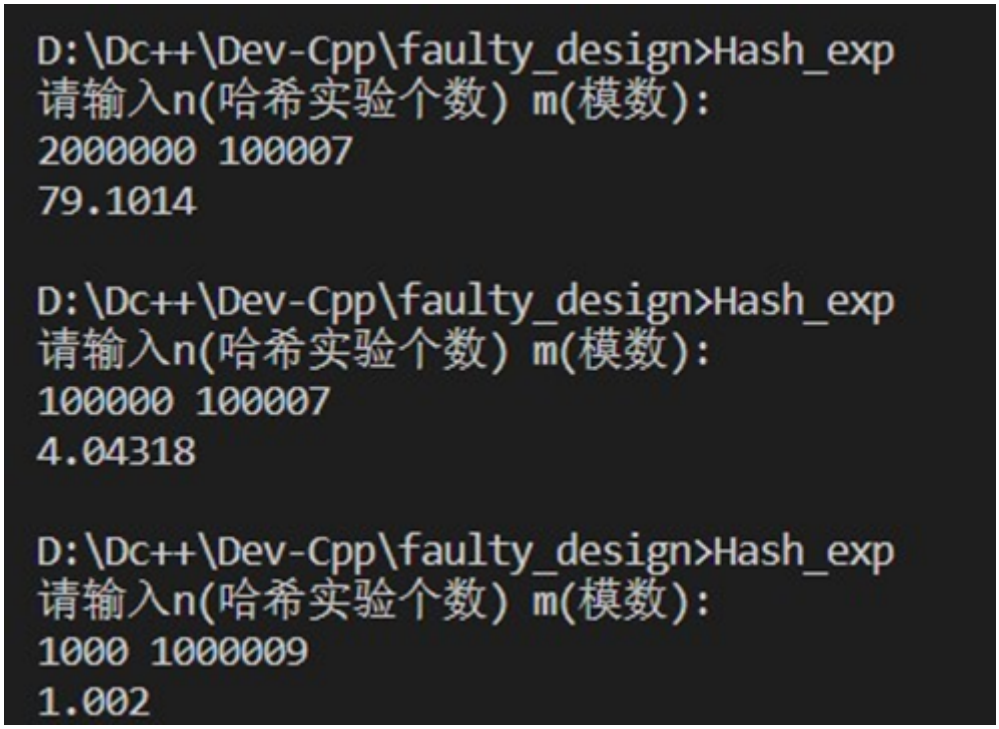
```
D:\Dc++\Dev-Cpp\faulty_design>Hash_exp
请输入n(哈希实验个数) m(模数):
100 10007
1.0101

D:\Dc++\Dev-Cpp\faulty_design>Hash_exp
请输入n(哈希实验个数) m(模数):
100000 100007
4.04008

D:\Dc++\Dev-Cpp\faulty_design>Hash_exp
请输入n(哈希实验个数) m(模数):
100  100009
1.0101

D:\Dc++\Dev-Cpp\faulty_design>Hash_exp
请输入n(哈希实验个数) m(模数):
1000 1000009
1.001
```

经过实验发现在m较大，n较小的情况下，平均长度趋于1.0，与理论结果相近

在m，n较为接近的情况下，平均长度因哈希函数而异。

第三问：

如图是哈希函数1的实验结果：



如图是哈希函数2的实验结果：

经过对比可以发现不论是大模数小数据抑或是大模数大数据哈希函数2的表现要优于哈希函数1.

Code

```
#include<unordered_set>
#include<unordered_map>
#include<functional>
#include<algorithm>
#include<string.h>
#include<iostream>
#include<iterator>
#include<cstring>
#include<numeric>
#include<assert.h>
#include<random>
#include<cstdio>
#include<vector>
#include<bitset>
#include<queue>
#include<stack>
#include<cmath>
#include<ctime>
#include<set>
#include<map>
// using namespace std;
// using long long = long long;
const long long N = 200010;
std::mt19937 rnd(time(0));
namespace MyHash{
    long long mod,idx,hash_number;
    struct Node{
        Node(long long _key,long long _val,long long _next):key(_key),val(_val),next(_next){};
        long long key,val,next;
    };
    std::vector<Node> node;
    std::vector<long long> head;
    void init(long long _mod){
        assert(_mod!=0);
        hash_number=idx=0,mod=_mod;
        node.clear();
        head.resize(mod, -1);
    }
    long long get_hash1(long long x){
        assert(mod>0);
        return ((((long long)x<<1)%mod*(long long)x%mod)+x)%mod;
    }
    long long get_hash2(long long x){
        assert(mod>0);
        return x%mod;
    }
    long long get_node(long long key,long long val){
        Node now=Node(key,val,-1);
        node.push_back(now);
        return idx++;
    }
    void insert(long long key, long long val){
        long long _hash = get_hash1(key);
        // long long _hash = get_hash2(key);
        long long now = get_node(key, val);
```

```
        if(head[_hash] == -1) hash_number ++;
        node[now].next = head[_hash];
        head[_hash]=now;
    }
    double average_len(){
        if(!hash_number) return 0;
        assert(hash_number);
        return (double)idx/hash_number;
    }
}


void solve(){
    long long n,m;
    std::cout << "请输入n(哈希实验个数) m(模数):" << std::endl;;
    std::cin >> n >> m;
    MyHash::init(m);
    for(long long i=1;i<=n;++i){
        auto key = rnd(),val = rnd();
        MyHash::insert(key, val);
    }
    auto ans = MyHash::average_len();
    if(ans<1e-9) std :: cout << "哈希表为空" << std :: endl;
    else std::cout << ans << std::endl;
}

signed main(){
    solve();
    return 0;
}
```