中山大学计算机学院本科生实验报告

（2025学年第1学期）

课程名称：数据结构与算法实验　　任课教师：张子臻

年级	2024级	专业（方向）	计算机科学与技术（人工智能）
学号	242325157	姓名	梁玮麟
电话	18620062182	Email	3371676041@qq.com
开始日期	2025.10.29	结束日期	2025.11.03

第一题

1、实验题目

z8-Query

题目描述

We have a set of strings (set A) and a set of queries (set B). Each query is to check whether a string exists in set A or not. If yes, output this string. Your task is to implement the following function:

```
void query(string A[], int n, string B[], int m);
```

Here, n is the number of strings in set A, m is the number of strings in set B. 1<=n,m<=500,000. Submit the function only.

输入描述

No input.

输出描述

Output the strings in set B which exist in set A. The output should follow the original order of the strings in set B.

提示

For example, A[0]="ABC", A[1]="CD", A[2]="D" B[0] = "A", B[1] ="CD", B[2]="BC", B[3]="ABC", then you should output: CD ABC

2、实验目的

- 掌握 **哈希表 (unordered_map / unordered_set)** 的查找与插入机制。
- 理解利用哈希实现 **集合成员查询（Set Membership Test）** 的方法。

- 学会分析哈希查找的平均与最坏复杂度。

3、算法设计

设计思路：

1. 将集合 `A` 中的所有字符串插入哈希表 `hash`；
2. 对集合 `B` 中的每一个字符串，判断是否存在于哈希表中；
3. 若存在，则按原顺序输出；
4. 哈希表的插入与查找平均时间复杂度均为 `O(1)`。

流程图：

```
// Pseudo flow of z8-Query
start
|
|--> create unordered_map<string,int> hash
|
|--> for each string in A:
|       if not exist in hash:
|           insert(A[i])
|
|--> for each string in B:
|       if hash.count(B[i]) != 0:
|           print(B[i])
|
---> end
```

复杂度分析：

- 时间复杂度：
  - 建表：O(n)
  - 查询：O(m)
  - 总体期望：**O(n + m)**
- 空间复杂度：O(n)

细节注意：

- **只提交函数**：题面要求仅提交 `query` 函数，不要带 `main()` 或额外 I/O 初始化。
- **容器选择**：查找仅需判存在，建议 `unordered_set<string>`；若需计数再用 `unordered_map<string,int>`。
- **重复元素**：`A` 中若有重复，哈希自然去重；`B` 中同一字符串出现多次，按 **B 的原顺序逐次输出。**

具体实现：

```
//z8-Query
#include<iostream>
#include"query.h"
```

```
#include<string>
#include<unordered_map>
using namespace std;
void query(string A[], int n, string B[], int m){
    unordered_map<string,int> hash;
    for(int i=0;i<n;i++){
        if(hash.count(A[i])==0) hash.emplace(A[i],1);
        else hash[A[i]]++;
    }
    for(int i=0;i<m;i++){
        if(hash.count(B[i])!=0){
            cout<<B[i]<<endl;
        }
    }
}
```

4、程序运行与测试

运行结果：

- 标准输入：

```

```

- 实际输出(数据太长，仅展示部分)：

```
DLXYBNNZNR
BOKEKGJFBR
DNMRQWQWNT
PHIADICRDJ
DCKVPBADXM
DLZFMFZNLA
BHOXOBOZLE
XTIPDSVATB
YGWRVTHSKL
JUFIZUTHEI
IFNKKQTTLT
BHOXOBOZLE
HYFKPUTWCW
XNMHFRHJDX
CQVPEGBXBL
HSYFNDFAKT
OWDHJBEANF
ROSRGXTEQK
IFNKKQTTLT
DNMRQWQWNT
ZJHFUKJXHH
MKVANPEBKC
```

```
CRVHBCFFIF
RJTOURMIBC
IAKYITTTQE
CGBAWRFEJT
KTWBDGRFSP
EPWMRLWEGS
PRLVSNLVQL
FLLLBQSSZR
UXVQCXCFEU
ESDOZHHLHJ
GMIWJKXFAA
CRVHBCFFIF
OWDHJBEANF
COQKUCVWZY
DCKVPBADXM
ZTUVAATLST
BIJQYHSPIY
UXVQCXCFEU
QJHYNCNVAP
ESDOZHHLHJ
IFNKKQTTLT
MHFENRCJNU
XTIPDSVATB
DMEDNAYRTN
AZNGATLTDX
......
POHMSFZMDDJCZBBNGUEIXWFTZKMOYGYOVBLKWBKXVMUBKHIPQOSSFLJTWRUFXKZGMHDYZDQOCAYJSKEPHR
HOOLVRQHMQOOGIHFBWSMTJODELDBEGLMZQOZELJEMRGFFIQJAAIKUHKWZRGSZEUAVVYPEZHPDGLQLGFLCH
VAIDUNWXSKPOEHAVLEYLINNLDJXNAQMLCYXLONCIIFSYXRTIFHIWIWRRTKYJZZSOJYBNMFPKFQFQTYHNFC
DPXRUXKRHISTENMMFSQZKNPSEJQSQSDGKQTNKDASJVLHTZXFROLGWGNPOBYDDEPXALCKSLOPRHTVYSCPGX
FCUFSUGEVGFDXUCVZJSJMGNAZDLEVAAUHYJLGSPNIOHBQAAWUADCCXAGDWCJURHZHFKWMNYLDKYHGJHEWM
RMCAHNEJMFBPEIEROQACVZYYBXIYRRIIDFCMPFGINWZQMKMVNMMOPGBGPBAGXUYSKOAKVRZEMOKPBRTKGE
DLWVSFFZLBVSWHZNCTIWVOMYKPTMDOVXVXVTFOISHOWBDMNLOFTARYORLUZXNEEZLFGRKWCJAOJVKZSYXF
NXNQNQLSZKNRSCLOIWEVYUGXEWDZFWFBDPOXOBWOOOAXCLINYOPLDZJLSPFKEYZXDTLJYQGDTLRTOOUYLN
FUOIIBHMYXKQQDIGOMZCASLTERMVZMCLAKGYRQUDOBVDEULAFSQAOUNTDIPPVILHMNGCZJPDHMQDZIBTAO
AAFXGQWDTEKUTBXTHUMVJZHNWIANRNXLOMTZPHHKRKZLJLREXMROPNCKFFEMZXUNLODHARZQZSEXAYIEFS
IMAZVOSCVAVFVYWMHSMNJLRGMFLNGXWRXGHCWGNGMXMVPQMXWEYCTUQFFXNWGZDNVBOSGFFFHHEWMPPYIA
LDUWJZECLLGQIICFWZCJIDCBOHIFGLGLKNAZOOVZUTSPHJTJVHQWVZBRJKLGVAUDQGTUJFQIXEUJKXLCQH
YIBVGPQBZYPBLBQQ
VCNFKGQCTLXTQJGHDRRMHRPKMGYXMUOQVLOFUFKVVLBGMXRGSFWSSYQMCHXYAGFQNTHMHBQFWLXSZBPXJL
KWMCHMEWPBGPYJLFRZHTKSUFLZSLICYXKKJQUBDXYZOCQXKZSRGMGSFSWLOOLJEPLWJOVBITZIJVULRSVS
CFBEHBRKVRAJBUKFOAZOZPSIDNUGPVCGHUVAIWGMIGQFJDAEXMLYKOBSDPDVPYCGAPVAAAREPZWTRPOZHY
FUECJIBJZDTGJCGOMZHJXXCKBPHEVEKIERTKRRNKJHUKUGHXFWEOETPCOCUFRDBKUMIHVNTQFMXOUQGAXB
BMTAMZNCDBNPEGMNBHVWAAWNOKANJWRQITRNERHLKISENUUUBXCHMVLXOVHSCUWURQZWHNENWVLKGLVZIZ
KKYRBIXMBJFLTPSUWJECRHYEDKLUYMAEEWHXXKRINUUNCRJMMBMSWSLEDAEQIEJXRFNMHJVIQRGBOPGIAO
BBQQITGTVNETMMCZCSAQBCRTFJAXYSKSVUENLEZBLZCDDTUVSHHAWSAASGHZEXCYQHEJJIXYOHZQJBCVJO
RNYLLAKJIWOIJSIUNVUDQHWYXHIYLNRPAJUASGDEAODFRNILBVGGTKNPYPZIZUQIWUGBRLNSMFJBWDCXDD
TOB
```

- 期望输出：

（如上，数据过长，不再重复展示）

5、实验总结与心得

- 哈希表可在 **O(1)** 平均复杂度内完成查找，适合集合判定类问题。
- 注意输出顺序需与集合 `B` 保持一致，而非哈希表插入顺序。
- 哈希函数冲突过多时性能下降，但在一般数据分布下表现优异。

第二题

1、实验题目

z8-爸爸去哪儿之一

题目描述

最近“爸爸去哪儿”节目很火，据说新一期节目分房的策略有所改变:共有m间房，序号从0到m-1，村长根据每个小朋友的英 文名来分配房子。

具体规则如下:

每个小朋友的英文名都能得到一个对应的数值:‘a’数值为1，‘b’数值为2，...，‘z’数值为26，小朋友的英文名的数值为各个字母的数值和，比如kimi的英文名的数值为11+9+13+9=42。注:规定输入的英文名均为小写字母

假设小朋友的英文名的数值为numName的话，那这个小朋友和他爸爸本期节目要住的房子就是(`numName mod m`)号房。 如果某小朋友的`numName mod m`得到的值和之前的小朋友的一样，则用哈希中的线性探测法:找下一号房直到找到一间还没有父 子入住的，若已经找到第m-1间还有人，则回到第0间找。

分配完房子之后，村长想知道这个分房策略的平均查找长度是多少，也就是说村长根据这个策略来查找每个人的房子时，平均需要查找多少房子(结果保留三位小数)。

比如有以下5个小朋友，并且有6间房:

name	nameNum	nameNum % 6	houseNum	findLength
kimi	42	0	0	1
tiantian	88	4	4	1
stone	73	1	1	1
angela	40	4	5	2
cindy	55	1	2	2

平均查找长度为:(1+1+1+2+2) / 5 = 1.400

输入描述

输入有多个测试用例，以EOF结束。

对于每个测试用例，输入分两部分:

第一部分是1行，有两个整数n和m(1<= n, m<=10,000)，中间用空格隔开，n代表有多少对明星父子参加节目，m代表一共 有多少间房

第二部分是n行，每行都是一个小朋友的英文名，小朋友的英文名彼此都不同，且中间没有空格。每个小朋友的英文名不超过 10个字母，且均为小写字母

输出描述

对于每个测试用例，输出m+1行 前m行，每行为房子序号 + ":" + 要入住的小朋友的英文名，没有人入住则输出房子序号 + ":NULL"

第m+1行输出一个数字，表示平均查找长度，保留三位小数

输入样例

```
5 6
kimi
tiantian
stone
angela
cindy
```

输出样例

```
0:kimi
1:stone
2:cindy
3:NULL
4:tiantian
5:angela
1.400
```

2、实验目的

1. 掌握哈希函数以及哈希表的具体应用。
2. 掌握利用线性探查解决clustering。

3、算法设计

设计思路：

1. 每个名字求字母和得到哈希值；
2. 初始地址为 `numName % m`；
3. 若被占用则 `(addr + 1) % m` 线性探查并**环回**；
4. 插入时累计本次查找步数，结束后计算平均值并按格式输出。

流程图：

```
start
|
|--> input n, m
|
|--> init house[m] = "NULL"
|
|--> count = 0
|
|--> for each name:
|       num = sum(char - 'a' + 1)
|       addr = num % m
|       step = 1
|       while house[addr] != "NULL":
|           addr = (addr + 1) % m
|           step++
|       house[addr] = name
|       count += step
|
|--> print all house[i]
|
|--> ASL = count / n
|       print ASL (保留三位小数)
|
---> end
```

复杂度分析：

- 时间复杂度：**O(n) 平均，O(nm) 最坏**；
- 空间复杂度：O(m)。

细节注意：

- **字母转数**：`a->1,...,z->26`，确保输入为小写字母。
- **环回**：线性探查必须 `(addr+1)%m`，否则越界或漏检末尾空位。
- **平均查找长度定义**：每个插入的查找步数取平均。
- **输出格式**：严格打印格式，不多输出空格。

具体实现：

```cpp
//z8-爸爸去哪儿之一
#include<iostream>
#include<string>
#include<iomanip>
using namespace std;
int main(){
    int n,m;
    while(cin>>n>>m){
        double count=0;
        string house[m];
        for(int i=0;i<m;i++){
            house[i]="NULL";
        }
        for(int i=0;i<n;i++){
            string name;
            cin>>name;
            int numName=0;
            for(int j=0;j<name.length();j++){
                numName+=name[j]-'a'+1;
            }
            int idx=numName%m;
            while(true){
                count++;
                if(house[idx]=="NULL"){
                    house[idx]=name;
                    break;
                }else{
                    idx++;
                    if(idx==m) idx=0;
                }
            }
        }
        for(int i=0;i<m;i++){
            cout<<i<<':'<<house[i]<<endl;
        }
        cout<<fixed<<setprecision(3);
        cout<<count/n<<endl;
    }
}
```

4、程序运行与测试

运行结果：

- 标准输入：

```
5 11
ee
df
fd
```

```
gc
cg
```

- 实际输出：

```
0:df
1:fd
2:gc
3:cg
4:NULL
5:NULL
6:NULL
7:NULL
8:NULL
9:NULL
10:ee
3.000
```

- 期望输出：

```
0:df
1:fd
2:gc
3:cg
4:NULL
5:NULL
6:NULL
7:NULL
8:NULL
9:NULL
10:ee
3.000
```

5、实验总结与心得

- 线性探查**简单直观**，但容易形成“clustering”。
- 冲突多时效率急剧下降，应控制负载率 α < 0.75。
- 实验帮助理解哈希性能与表长的关系，掌握线性探查**具体实现**。

第三题

1、实验题目

z8-爸爸去哪儿之二

题目描述

最近“爸爸去哪儿”节目很火，据说新一期节目分房的策略有所改变:共有m间房，序号从0到m-1，村长根据每个小朋友的英 文名来分配房子。

具体规则如下:

每个小朋友的英文名都能得到一个对应的数值:‘a’数值为1，‘b’数值为2，...，‘z’数值为26，小朋友的英文名的数值为各个字母的数值和，比如kimi的英文名的数值为11+9+13+9=42。注:规定输入的英文名均为小写字母

假设小朋友的英文名的数值为numName的话，那这个小朋友和他爸爸本期节目要住的房子就是(numName mod m)号房。

如果某小朋友的numName mod m得到的值和之前的小朋友的一样，则用哈希中的平方探测法。 分配完房子之后，村长想知道这个分房策略的平均查找长度是多少，也就是说村长根据这个策略来查找每个人的房子时，平均需要查找多少房子(结果保留三位小数)。

比如有以下5个小朋友，并且有6间房:

name	nameNum	nameNum % 6	houseNum	findLength
kimi	42	0	0	1
tiantian	88	4	4	1
stone	73	1	1	1
angela	40	4	5	2
cindy	55	1	2	2

平均查找长度为:`(1+1+1+2+2) / 5 = 1.400`

输入描述

输入有多个测试用例，以EOF结束。

对于每个测试用例，输入分两部分:

第一部分是1行，有两个整数n和m`(1<= n, m<=10,000)`，中间用空格隔开，n代表有多少对明星父子参加节目，m代表一共 有多少间房

第二部分是n行，每行都是一个小朋友的英文名，小朋友的英文名彼此都不同，且中间没有空格。每个小朋友的英文名不超过 10个字母，且均为小写字母

输出描述

对于每个测试用例，输出m+1行 前m行，每行为房子序号+":"+要入住的小朋友的英文名，没有人入住则输出房子序号+":NULL"

第m+1行输出一个数字，表示平均查找长度，保留三位小数

输入样例

```
5 6
kimi
tiantian
stone
angela
cindy
```

输出样例

```
0:kimi
1:stone
2:cindy
3:NULL
4:tiantian
5:angela
1.400
```

2、实验目的

- 掌握**平方探查法**（Quadratic Probing）；
- 比较不同冲突解决策略下的查找效率；

3、算法设计

设计思路：

1. 将名字转换为数值并取模，得初始地址 `h`；
2. 冲突时依次尝试 `(h + 1^2) % m, (h + 2^2) % m, …`；
3. 记录每个成功插入的探查次数，计算平均查找长度。

流程图：

```
start
|
|--> input n, m
|
|--> init house[m] = "NULL"
|
|--> totalSearch = 0
|
```

```
|--> for each name:
|       num = sum(char - 'a' + 1)
|       h = num % m
|       i = 0
|       while house[(h + i*i) % m] != "NULL":
|           i++
|       addr = (h + i*i) % m
|       house[addr] = name
|       totalSearch += (i + 1)
|
|--> print all house[i]
|
|--> ASL = totalSearch / n
|       print ASL (保留三位小数)
|
---> end
```

复杂度分析：

- **平均时间复杂度：O(n)**
- 最坏时间复杂度：O(nm)
- 空间复杂度：O(m)

细节注意:

- **m 取质数**：一般来说，m最好取质数。但是题目中m是数组长度。
- **探查序列起点**：i 从 0 开始，步数为 i+1。
- **查找停止处理**：查到**NULL**时停止。
- **记录查找次数**：记录查找次数**times**，实现对应数的平方。

具体实现：

```cpp
//z8-爸爸去哪儿之二
#include<iostream>
#include<string>
#include<iomanip>
#include<cmath>
using namespace std;
int main(){
    int n,m;
    while(cin>>n>>m){
        double count=0;
        string house[m];
        for(int i=0;i<m;i++){
            house[i]="NULL";
        }
        for(int i=0;i<n;i++){
            string name;
            cin>>name;
            int numName=0;
```

```
            for(int j=0;j<name.length();j++){
                numName+=name[j]-'a'+1;
            }
            int addr=numName%m;
            int idx=addr;
            int times=0;
            while(true){
                times++;
                if(house[idx]=="NULL"){
                    house[idx]=name;
                    break;
                }else{
                    idx=addr+pow(times,2);
                    idx%=m;
                }
            }
            count+=times;
        }
        for(int i=0;i<m;i++){
            cout<<i<<':'<<house[i]<<endl;
        }
        cout<<fixed<<setprecision(3);
        cout<<count/n<<endl;
    }
}
```

4、程序运行与测试

运行结果：

- 标准输入：

```
5 11
ee
df
fd
gc
cg
```

- 实际输出：

```
0:df
1:NULL
2:NULL
3:fd
4:cg
5:NULL
6:NULL
```

```
7:NULL
8:gc
9:NULL
10:ee
3.000
```

- 期望输出：

```
0:df
1:NULL
2:NULL
3:fd
4:cg
5:NULL
6:NULL
7:NULL
8:gc
9:NULL
10:ee
3.000
```

- 仔细观察可以发现，平方探查法的结果与线性探查法的结果不一样，

5、实验总结与心得

- 平方探查有效缓解了线性探查种拥有**相同哈希值的元素**聚集在一起的问题；
- 若 **m 为质数**，探查序列可覆盖全表；
- 在负载率较高时仍可能出现性能退化；
- 实验展示了不同冲突策略在效率上的差异，进一步深化了对哈希表查找机制的理解。