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　　任课老师：张子臻

年级:	2024级	专业(方向):	计算机科学与技术(人工智能与大数据)
学号:	24325155	姓名:	梁桂铭
电话:	15817681625	Email:	lianggm8@mail2.sysu.edu.cn
开始日期:	2025年10月31日	完成日期:	2025年10月31日

第一题

一、实验题目

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.

二、实验目的

1. 掌握使用哈希集合进行高效字符串查找的方法
2. 理解时间复杂度优化在大量数据处理中的重要性
3. 熟悉C++标准库中unordered_set的使用

三、算法设计

3.1 核心思路

本算法采用**空间换时间**的策略，通过构建哈希集合来实现$O(1)$时间复杂度的字符串查找。具体流程如下：

1. **预处理阶段**：将集合A中的所有字符串存入unordered_set中
2. **查询阶段**：遍历集合B中的每个字符串，检查其是否存在于哈希集合中
3. **输出阶段**：按集合B的原始顺序输出所有存在于集合A中的字符串

3.2 时间复杂度分析

- 构建哈希集合：$O(n)$，其中n为集合A的大小
- 查询处理：$O(m)$，其中m为集合B的大小
- 总体时间复杂度：$O(n+m)$

3.3 空间复杂度分析

- 哈希集合存储：$O(n)$
- 总体空间复杂度：$O(n)$

3.4 算法优势

- 利用哈希表的平均$O(1)$查找复杂度，适合大规模数据处理
- 保持查询结果的原始顺序，满足题目要求

四、程序运行与测试

4.1 核心代码实现

```
#include <iostream>
#include "query.h"
#include <unordered_set>
using namespace std;
void query(string A[], int n, string B[], int m) {
    unordered_set<string> setA;
    for (int i = 0; i < n; i++) {
        setA.insert(A[i]);
    }
    for (int i = 0; i < m; i++) {
        if (setA.find(B[i]) != setA.end()) {
            cout << B[i] << endl;
        }
    }
}
```

4.2 代码说明

1. **数据结构选择**：使用 `unordered_set<string>` 作为哈希集合，提供高效的插入和查找操作
2. **集合构建**：通过循环将集合A的所有元素插入哈希表
3. **查询处理**：遍历集合B，使用 `find()` 方法检查元素是否存在
4. **顺序保持**：按集合B的遍历顺序输出结果，自然保持原始顺序

五、实验总结

通过本次实验，我深入理解了哈希表在字符串查询中的应用价值。unordered_set提供了平均$O(1)$时间复杂度的查找操作，相比线性搜索的$O(n)$时间复杂度，在处理大规模数据时优势明显。同时，算法保持了查询结果的顺序一致性，体现了在实际应用中对数据完整性的重视。

第二题

一、实验题目

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)。

二、实验目的

1. 掌握哈希表的基本原理和线性探测法的实现
2. 学习如何处理哈希冲突和计算平均查找长度
3. 理解字符串数值化处理的方法
4. 熟悉多测试用例的输入输出处理

三、算法设计

3.1 核心算法原理

本系统采用**哈希函数+线性探测法**解决分房问题，主要包含以下步骤：

3.1.1 哈希函数设计

对于每个小朋友的英文名，计算其数值和：

$$numName = \sum_{i=1}^{k}(c_i -' a' + 1)$$

其中k为名字长度，c_i为第i个字符

初始房号计算公式：

$$hashValue = numName \mod m$$

3.1.2 冲突解决策略

采用线性探测法处理哈希冲突：

- 如果目标房号已被占用，依次检查后续房号
- 到达末尾后回到第0号房继续查找
- 直到找到空房或遍历所有房号

3.1.3 平均查找长度计算

$$ASL = \frac{\sum_{i=1}^{n} 查找次数}{n}$$

其中查找次数包括成功找到空房的比较次数

3.2 算法流程

1. **初始化阶段**：创建大小为m的字符串数组作为哈希表
2. **处理每个名字**：
    - 计算名字的数值和
    - 计算初始哈希值
    - 使用线性探测法寻找空房
    - 记录查找次数
3. **输出结果**：按房号顺序输出入住情况
4. **计算ASL**：统计总查找次数并计算平均值

3.3 时间复杂度分析

- 最佳情况：$O(n)$（无冲突）

- 最坏情况：$O(n \times m)$（严重冲突）
- 平均情况：$O(n)$

3.4 空间复杂度分析

- 哈希表存储：$O(m)$
- 总体空间复杂度：$O(m)$

四、程序运行与测试

4.1 核心代码实现

```
for (int i = 0; i < n; i++) {
    int sum = 0;
    cin >> name;
    for (char c : name) {
        sum += c - 'a' + 1;
    }

    if (h[sum % m].empty()) {
        h[sum % m] = name;
    } else {
        int flag = 0;
        l = sum % m + 1;
        while (l < m) {
            cnt++;
            if (h[l].empty()) {
                h[l] = name;
                flag = 1;
                break;
            }
            l++;
        }
        if (flag == 0) {
            l = 0;
            while (l < m) {
                cnt++;
                if (h[l].empty()) {
                    h[l] = name;
                    flag = 1;
                    break;
```

```
                }
                l++;
            }
        }
    }
}

for (int i = 0; i < m; i++) {
    if (h[i].empty()) {
        cout << i << ":NULL" << endl;
        continue;
    }
    cout << i << ":" << h[i] << endl;
}
printf("%.3f\n", (double)(cnt + n) / n);
}
```

4.2 代码功能说明

1. **多测试用例处理**：使用 `while(cin >> n)` 循环处理多个测试用例
2. **名字数值计算**：遍历名字字符，累加字母对应的数值
3. **哈希表操作**：使用字符串数组实现哈希表，空字符串表示空房
4. **线性探测实现**：分两段查找（从哈希位置到末尾，再从开头到哈希位置前）
5. **统计计数**：变量 `cnt` 记录冲突时的额外查找次数
6. **结果输出**：格式化输出房号状态和平均查找长度

五、实验总结

通过本次实验，我深入理解了哈希表的实际应用和冲突解决机制。线性探测法作为一种开放定址法，虽然实现简单但在高负载情况下性能会下降。实验中还学习了如何正确计算平均查找长度，这是评估哈希表性能的重要指标。

第三题

一、实验题目

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)。

二、实验目的

1. 掌握平方探测法解决哈希冲突的原理和实现
2. 学习不同冲突解决策略对哈希表性能的影响
3. 理解平均查找长度的统计和计算方法
4. 比较线性探测与平方探测的性能差异

三、算法设计

3.1 核心算法原理

本系统采用**哈希函数+平方探测法**解决分房问题，主要包含以下步骤：

3.1.1 哈希函数设计

对于每个小朋友的英文名，计算其数值和：

$$numName = \sum_{i=1}^{k}(c_i -' a' + 1)$$

其中k为名字长度，c_i为第i个字符

初始房号计算公式：

$$hashValue = numName \mod m$$

3.1.2 冲突解决策略

采用平方探测法处理哈希冲突，探测序列为：

$$h_i(key) = (h(key) + i^2) \mod m \quad (i = 1, 2, 3, \ldots)$$

3.1.3 平均查找长度计算

$$ASL = \frac{\sum_{i=1}^{n} \text{查找次数}}{n}$$

其中查找次数包括成功找到空房的比较次数

3.2 算法流程

1. **初始化阶段**：创建大小为m的字符串数组作为哈希表
2. **处理每个名字**：
    - 计算名字的数值和
    - 计算初始哈希值
    - 使用平方探测法寻找空房
    - 记录查找次数
3. **输出结果**：按房号顺序输出入住情况
4. **计算ASL**：统计总查找次数并计算平均值

3.3 时间复杂度分析

- 最佳情况：$O(n)$（无冲突）
- 平均情况：$O(n)$
- 平方探测法相比线性探测能减少聚集现象

3.4 空间复杂度分析

- 哈希表存储：$O(m)$
- 总体空间复杂度：$O(m)$

四、程序运行与测试

4.1 核心代码实现

```
for (int i = 0; i < n; i++) {
    sum = 0;
    times = 1;
    cin >> name;
    for (char c : name) {
        sum += c - 'a' + 1;
    }

    if (h[sum % m].empty()) {
        h[sum % m] = name;
    } else {
        l = (sum % m + times * times) % m;
        while (l < m) {
            cnt++;
            if (h[l].empty()) {
                h[l] = name;
                break;
            }
            times++;
            l = (sum % m + times * times) % m;
        }
    }
}

for (int i = 0; i < m; i++) {
    if (h[i].empty()) {
        cout << i << ":NULL" << endl;
        continue;
    }
    cout << i << ":" << h[i] << endl;
}
printf("%.3f\n", (double)(cnt + n) / n);
}
```

4.2 代码功能说明

1. **多测试用例处理**：使用 `while(cin >> n)` 循环处理多个测试用例
2. **名字数值计算**：遍历名字字符，累加字母对应的数值（a=1, b=2, ..., z=26）
3. **平方探测实现**：
   - 初始探测位置： `sum % m`
   - 冲突时探测位置： `(sum % m + i²) % m` ，其中i=1,2,3,...
   - 变量 `times` 记录探测次数
4. **统计计数**：变量 `cnt` 记录冲突时的额外查找次数
5. **结果输出**：格式化输出房号状态和平均查找长度

4.3 平方探测特点

- **减少聚集**：相比线性探测，平方探测能有效减少聚集现象
- **探测序列**：$h+1^2, h+2^2, h+3^2$, ... 模m
- **覆盖性**：平方探测可能无法探测所有位置，但在实际应用中效果良好

五、实验总结

通过本次实验，我深入理解了平方探测法在哈希冲突解决中的应用。相比线性探测法，平方探测法通过二次增量减少了"一次聚集"现象，提高了哈希表的性能。实验中还学习了如何正确实现探测序列和计算平均查找长度。

5.2 算法对比分析

平方探测 vs 线性探测：

- **线性探测**：简单易实现，但容易产生聚集
- **平方探测**：减少聚集，但可能无法探测所有位置
- **性能比较**：在负载因子较高时，平方探测通常有更好的性能

附录，提交文件清单

第一题

```
#include  <iostream>
#include  "query.h"
#include  <unordered_set>
using  namespace  std;
void  query(string  A[], int  n, string  B[], int  m) {
    unordered_set<string> setA;
    for (int i = 0; i < n; i++) {
        setA.insert(A[i]);
    }
    for (int i = 0; i < m; i++) {
        if (setA.find(B[i]) != setA.end()) {
            cout << B[i] << endl;
        }
    }
}
```

第二题

```
for (int i = 0; i < n; i++) {
    int sum = 0;
    cin >> name;
    for (char c : name) {
        sum += c - 'a' + 1;
    }

    if (h[sum % m].empty()) {
        h[sum % m] = name;
    } else {
        int flag = 0;
        l = sum % m + 1;
        while (l < m) {
            cnt++;
            if (h[l].empty()) {
                h[l] = name;
                flag = 1;
                break;
            }
            l++;
        }
```

```
            if (flag == 0) {
                l = 0;
                while (l < m) {
                    cnt++;
                    if (h[l].empty()) {
                        h[l] = name;
                        flag = 1;
                        break;
                    }
                    l++;
                }
            }
        }
    }

    for (int i = 0; i < m; i++) {
        if (h[i].empty()) {
            cout << i << ":NULL" << endl;
            continue;
        }
        cout << i << ":" << h[i] << endl;
    }
    printf("%.3f\n", (double)(cnt + n) / n);
    }
```

第三题

```
for (int i = 0; i < n; i++) {
    sum = 0;
    times = 1;
    cin >> name;
    for (char c : name) {
        sum += c - 'a' + 1;
    }

    if (h[sum % m].empty()) {
        h[sum % m] = name;
    } else {
        l = (sum % m + times * times) % m;
        while (l < m) {
            cnt++;
            if (h[l].empty()) {
                h[l] = name;
```

```
                    break;
                }
                times++;
                l = (sum % m + times * times) % m;
            }
        }
    }

    for (int i = 0; i < m; i++) {
        if (h[i].empty()) {
            cout << i << ":NULL" << endl;
            continue;
        }
        cout << i << ":" << h[i] << endl;
    }
    printf("%.3f\n", (double)(cnt + n) / n);
    }
```