哈希表

郑悟强 PB22051082

2023.12.5

1 问题描述

1. 输入关键字序列；
2. 用除留余数法构建哈希函数，用线性探测法 (线性探测再散列) 解决冲突，构建哈希表 HT1；
3. 用除留余数法构建哈希函数，用拉链法 (链地址法) 解决冲突，构建哈希表 HT2；
4. 分别对 HT1 和 HT2 计算在等概率情况下查找成功和查找失败的 ASL；
5. 分别在 HT1 和 HT2 中查找给定的关键字，给出比较次数

2 算法的描述

2.1 数据结构的描述

HT1（采用线性探测法解决冲突）：

```
class HashTable{
private:
    int *elem;              //哈希表的内容
    int count;              //当前数据元素个数
    int size;               //哈希表的长度
    int *success;           //成功查找次数
    int *unsuccess;         //失败查找次数
```

HT2（采用链地址法解决冲突），每一个具体哈希表地址采用一个链表：

```
//链表定义
typedef struct LinkNode{
    int val;
    LinkNode *next;
}LinkNode, *LinkList;

class HashTable{
private:
    LinkList *elem;         //哈希表的内容
    int* depth;             //每个链表的深度
    int count;              //当前数据元素个数
    int size;               //哈希表的长度
    int *success;           //成功查找次数
    int *unsuccess;         //失败查找次数
    int maxdepth;           //最大的深度
```

2.2 程序结构的描述

1. 哈希函数的实现（采用除留余数法）：

```cpp
int HashTable::Hashing(int key){
    //通过除留余数法得到的哈希函数
    return key%size;
}
```

2.HT1 中哈希表的构造及成功查找 ASL 和失败查找 ASL 的求取：

基本原理：如果一个关键字的哈希函数得到的地址已经有元素，那么就将该地址 + d，d 从 1 开始不断增大，直到一个位置为空，就插入这个位置。

```cpp
void HashTable::initHash(){
    cout<<"请输入关键字个数：";
    cin>>count;
    cout<<"请输入所有关键字：";
    //使用nums数组临时存储所有的关键字
    int nums[count];
    for(int i=0;i<count;i++){
        cin>>nums[i];
    }
    cout<<"请输入哈希表的长度：";
    cin>>size;

    //构建哈希表
    //初始化内容数组和成功/失败查找次数数组
    elem = (int *)malloc(size*sizeof(int));
    success = (int *)malloc(size*sizeof(int));
    unsuccess = (int *)malloc(size*sizeof(int));
    for(int t=0;t<size;t++){
        //初始化哈希表内容，-1代表为空
        elem[t] = -1;
        success[t] = 0;
        unsuccess[t] = 1;
    }
    for(int j=0;j<count;j++){
        int key = Hashing(nums[j]);
        int d = 0;
        //线性探测再散列法处理冲突
        while(elem[(key+d)%size] != -1){
            d++;
        }
        elem[(key+d)%size] = nums[j];
        success[(key+d)%size] = d+1;    //成功查找次数为d+1
    }
    //再来生成失败查找次数
    //失败查找时，会先求哈希函数，会发现对应位置不匹配，
    //再反复线性探测，直到发现对应地址为空，说明查找失败，
    //或者另一种可能就是找一圈到原位都不匹配
    for(int i=0;i<size;i++){
        int un = 0;
        while(elem[(un+i)%size]!=-1 && un<=size){
            un++;
        }
        unsuccess[i] = un+1;
    }
}
```

3.HT2 中哈希表的构造及成功查找 ASL 和失败查找 ASL 的求取：

基本原理：如果一个关键字的哈希函数得到的地址已有元素，就根据链表的构造，在这个链表的末端插入新节点。

```
void HashTable::initHash(){
    cout<<"请输入关键字个数：";
    cin>>count;
    cout<<"请输入所有关键字：";
    //用nums数组临时存储所有内容
    int nums[count];
    for(int i=0;i<count;i++){
        cin>>nums[i];
    }
    cout<<"请输入哈希表长度：";
    cin>>size;

    //构造哈希表
    //初始化所有的哈希表结点和成功/失败查找次数
    maxdepth = 0;
    elem = (LinkList *)malloc(size*sizeof(LinkList));
    depth = (int*)malloc(sizeof(int)*size);
    success = (int*)malloc(size*sizeof(int));
    unsuccess = (int*)malloc(size*sizeof(int));
    for(int i=0;i<size;i++){
        elem[i] = nullptr;
        depth[i] = 0;
        success[i] = 0;
        unsuccess[i] = 1;
    }
    //通过链地址法将每个关键字存入哈希表中
    for(int i=0;i<count;i++){
        LinkNode *p = (LinkNode *)malloc(sizeof(LinkNode));
        p->val = nums[i];
        p->next = nullptr;
        if(elem[Hashing(nums[i])] == nullptr){
            elem[Hashing(nums[i])] = p;
        }
        else{
            LinkNode *q = elem[Hashing(nums[i])];
            //找到对应位置的末节点
            while(q->next!=nullptr){
                q = q->next;
            }
            q->next = p;
        }
        depth[Hashing(nums[i])]++;
        if(depth[Hashing(nums[i])]>maxdepth){
            maxdepth = depth[Hashing(nums[i])];
        }
        success[Hashing(nums[i])]++;
        unsuccess[Hashing(nums[i])]++;
    }
}
```

3 调试分析

HT1 的结果：

```
请输入关键字个数：8
请输入所有关键字：23 35 12 56 123 39 342 90
请输入哈希表的长度：11
哈希表的地址：  0     1     2     3     4     5     6     7     8     9     10
表中的关键字：  -     23    35    12    56    123   39    342   90    -     -
成功查找次数：  0     1     1     3     4     4     1     7     7     0     0
失败查找次数：  1     9     8     7     6     5     4     3     2     1     1
查找成功的平均查找长度：      3.5
查找失败的平均查找长度：      4.27273
请输入关键字个数：8
请输入所有关键字：23 35 12 56 123 39 342 90
请输入哈希表长度：11
哈希表的地址：  0     1     2     3     4     5     6     7     8     9     10
表中的关键字：  -     23    35    -     -     -     39    -     -     -     -
                      12    123
                      56    90
                      342
成功查找次数：  0     1     1     0     0     0     1     0     0     0     0
                      2     2
                      3     3
                      4
失败查找次数：  1     5     4     1     1     1     2     1     1     1     1
查找成功的平均查找长度：      2.125
查找失败的平均查找长度：      1.72727
```

4 算法时空分析

4.1 时间复杂度

HT1: 构造过程中，极端情况为所有元素的哈希函数得到的值相同，那么就需要 1+2+3+...+n 次循环查找，那么时间复杂度就是 O(n^2)，不过一般情况下，一个比较合理的哈希函数，构造过程时间复杂度不会很高，一般为 O(n) 左右。

HT2: 构造过程中，极端情况也为所有元素的哈希函数得到的值相同，那么时间复杂度也为 O(n^2)。

查找过程同上，极端情况复杂度为 O(n)。不过一般在哈希函数比较合理地情况下，时间复杂度为 O(1)。

4.2 空间复杂度

构造过程空间复杂度为 O(n)。

5 实验体会与分析

通过本次实验，我明白了哈希表的基本构造方法，解决冲突的方法和基本原理，也对查找的基本原理，ASL 的计算方法有了更详细的认识。