中国科学技术大学

University of Science and Technology of China

数据结构实验报告 5

哈希表

姓名：高茂航

学号：PB22061161

日期：2023 年 12 月 10 日

实验报告 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失败时的平均查找次数；

5. 分别在 HT1 和 HT2 中查找给定的关键字，给出查找次数。

2 算法描述

2.1 数据结构

2.1.1 线性探测再散列法

```
typedef struct{
  int Address[MAXNUM];//哈希表地址
  int Keyword[MAXNUM];//哈希表关键字
  int keynum;//关键字个数
  int SuccessFind[MAXNUM]={0};//成功查找次数
  int FailFind[MAXNUM]={0};//失败查找次数
  float AveSuccess=0;//查找成功的平均查找次数
  float AveFail=0;//查找失败时在各个位置上的平均查找次数
}HashTable;

```

Listing 1: 线性探测再散列法

2.1.2 链地址法

```
typedef struct Node{
  int data;
  Node* next=NULL;
}Node;
typedef struct SuccessNode{
  int SuccessNum;
  SuccessNode* next=NULL;
}SuccessNode;
typedef struct{
  int Address[MAXNUM];//哈希表地址
  int keynum=0;//关键字个数
  Node* Link[MAXNUM];//哈希表链地址
  SuccessNode* SuccessLink[MAXNUM];//成功查找次数
  int FailFind[MAXNUM]={0};//失败查找次数
  float AveSuccess=0;//查找成功的平均查找次数
  float AveFail=0;//查找失败时在各个位置上的平均查找次数
}HashTable;

```

Listing 2: 链地址法

2.2 程序结构

2.2.1 线性探测再散列法

```
1     int Hash(int);//哈希函数(除留余数法)
2     void InsertHash1(HashTable*,int);//线性探测再散列把关键字插入哈希表
3     void SearchHash1(HashTable*,int);//在哈希表中查找关键字
4     void JudgeFail1(HashTable*);//求在每个地址查找失败时的最小散列次数
5     int main(){
6       int i,key[MAXNUM];
7       HashTable *hashtable = new HashTable;
8       cin>>hashtable->keynum;
9       for(i=0;i<hashtable->keynum;i++)
10        cin>>key[i];
11      cin>>p;
12      for(i=0;i<p;i++){
13        hashtable->Address[i]=i;
14        hashtable->Keyword[i]=-1;//初始化哈希表中没有关键字的位置为-1
15      }
16      cout<<"哈希表地址: ";
17      for(i=0;i<p;i++)
18        cout<<hashtable->Address[i]<<"  ";
19      for(i=0;i<hashtable->keynum;i++)
20        InsertHash1(hashtable,key[i]);
21      cout<<"表中关键字: ";
22      for(i=0;i<p;i++)
23        if(hashtable->Keyword[i]!=-1)
24          cout<<i<<":"<<hashtable->Keyword[i]<<" ";
25      for(i=0;i<hashtable->keynum;i++)
26        SearchHash1(hashtable,key[i]);
27      cout<<"成功查找次数: ";
28      for(i=0;i<p;i++){
29        cout<<i<<":"<<hashtable->SuccessFind[i]<<" ";
30        hashtable->AveSuccess+=hashtable->SuccessFind[i];
31      }
32      JudgeFail1(hashtable);//求在每个地址查找失败时的最小散列次数
33      cout<<"失败查找次数: ";
34      for(i=0;i<p;i++){
35        cout<<i<<":"<<hashtable->FailFind[i]<<" ";
36        hashtable->AveFail+=hashtable->FailFind[i];
37      }
38      cout<<"查找成功的平均查找次数: "<<hashtable->AveSuccess/hashtable->keynum<<endl<<"查找
        失败的平均查找次数: "<<hashtable->AveFail/p<<endl;
39      delete hashtable;
40    }
41
```

Listing 3: 线性探测再散列法

```
1 InsertHash1(HashTable* H,int key){//用线性探测再散列把关键字插入哈希表
2     int address=Hash(key);
3     while (H->Keyword[address]!=-1)
4         address=(address+1)%p;//线性探测再散列
5     H->Keyword[address]=key;
6 }
7
```

Listing 4: 把关键字插入哈希表

```
void SearchHash1(HashTable* H,int key){//在哈希表中查找关键字
  int address=Hash(key),num=0;
  while (H->Keyword[address]!=key){
    num++;
    address=(address+1)%p;
    if(H->Keyword[address]==-1||address==Hash(key)){//假如查找到一个没有关键字的位置或者查找了一圈回到原来的位置，说明哈希表中没有该关键字
      cout<<"哈希表中没有该关键字"<<endl;
      break;
    }
  }
  H->SuccessFind[address]=++num;
}

```

Listing 5: 在哈希表中查找关键字

```
void JudgeFail1(HashTable* H){//求在每个地址查找失败时的最小散列次数
    int i,temp;
    for(i=0;i<p;i++){
        H->FailFind[i]++;
        temp=i;
        while (H->Keyword[temp]!=-1){
            temp=(temp+1)%p;
            if(H->Keyword[temp]!=H->Keyword[i])
                H->FailFind[i]++;
            else{
                H->FailFind[i]++;
                break;
            }
        }
    }
}

```

Listing 6: 求在每个地址查找失败时的最小散列次数

2.2.2 链地址法

```
    int main(){
      //前面与上述代码相同
      for(i=0;i<p;i++){
        hashtable->Address[i]=i;
        hashtable->Link[i]=NULL;
        hashtable->SuccessLink[i]=NULL;
      }
      for(i=0;i<hashtable->keynum;i++)
        InsertHash2(hashtable,key[i]);
      Node* pt[p];
      for(i=0;i<p;i++)
        pt[i]=hashtable->Link[i];
      cout<<"表中关键字: ";
      for(j=0;j<p;j++){
        if(pt[j]){
          cout<<j<<"(";
          while(pt[j]){
            if(pt[j]->next){
              cout<<pt[j]->data<<" ";
```

```
            pt[j]=pt[j]->next;
          }
          else{
            cout<<pt[j]->data;
            break;
          }
        }
        cout<<") ";
      }
    }
    SuccessNode* pt2[p];
    for(i=0;i<hashtable->keynum;i++)
      SearchHash2(hashtable,key[i]);
    for(i=0;i<p;i++)
      pt2[i]=hashtable->SuccessLink[i];
    cout<<"成功查找次数: ";
    for(j=0;j<p;j++){
      if(pt2[j]){
        cout<<j<<"(";
        while(pt2[j]){
          if(pt2[j]->next){
            cout<<pt2[j]->SuccessNum<<" ";
            pt2[j]=pt2[j]->next;
          }
          else{
            cout<<pt2[j]->SuccessNum;
            break;
          }
        }
        cout<<") ";
      }
      else
        cout<<j<<"(0) ";
    }
    //后面与上述代码相同
  }

```

Listing 7: 链地址法

```
void InsertHash2(HashTable* H,int key){//链地址法把关键字插入哈希表
  int address=Hash(key);
  Node* newNode = new Node;
  newNode->data = key;
  newNode->next = NULL;
  if(!H->Link[address])
    H->Link[address] = newNode;
  else {
    Node* pt = H->Link[address];
    while (pt->next)
      pt = pt->next;
    pt->next = newNode;
  }
}

```

Listing 8: 把关键字插入哈希表

```
void SearchHash2(HashTable* H,int key){//在哈希表中查找关键字
    int address=Hash(key),num=0;
```

```
3     Node* pt=H->Link[address];
4     if(!pt){
5         cout<<"哈希表中没有该关键字"<<endl;
6         return;
7     }
8     while(pt){
9         num++;
10        if(pt->data == key){
11            if(!H->SuccessLink[address]){
12                SuccessNode* newNode = new SuccessNode;
13                newNode->SuccessNum=num;
14                newNode->next = NULL;
15                H->AveSuccess+=num;
16                H->SuccessLink[address]=newNode;
17            }
18            else {
19                SuccessNode* pt2 = H->SuccessLink[address];
20                SuccessNode* newNode = new SuccessNode;
21                newNode->SuccessNum=num;
22                newNode->next = NULL;
23                H->AveSuccess+=num;
24                while (pt2->next)
25                    pt2 = pt2->next;
26                pt2->next = newNode;
27            }
28            return;
29        }
30        pt = pt->next;
31    }
32    cout<<"哈希表中没有该关键字"<<endl;
33 }
34
```

Listing 9: 在哈希表中查找关键字

```
1 void JudgeFail2(HashTable* H){//求在每个地址查找失败时的最小散列次数
2     int i,temp;
3     for(i=0;i<p;i++){
4         H->FailFind[i]++;
5         if(H->Link[i]){
6             Node* pt=H->Link[i];
7             while(pt){
8                 H->FailFind[i]++;
9                 pt=pt->next;
10            }
11        }
12    }
13 }
14
```

Listing 10: 求在每个地址查找失败时的最小散列次数

2.2.3 哈希函数

```
1     int p=1;//除留余数法中的除数
2     int Hash(int m){//哈希函数(除留余数法)
3       return m%p;
4     }
```

5

Listing 11: 哈希函数 (除留余数法)

3 调试分析

本程序遇到的主要问题是：

1. 在线性探测再散列查找失败时判断哈希表中是否存在该关键字的方法；

2. 在链地址法中，处理指针时出现 Segmentation fault 错误。

但在插入关键字时，由于已经默认了插入的关键字互不相同，没有判断哈希表中是否已经存在该关键字。

4 算法时空分析

哈希表的空间复杂度是 $O(n)$, 插入查找的时间复杂度是 $O(n)$, 线性探测再散列的平均查找成功和失败次数一般大于链地址法。

5 测试结果分析

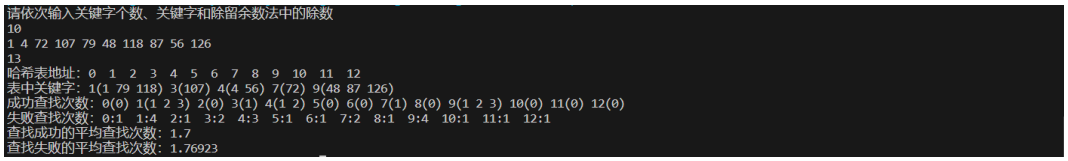
5.1 线性探测探测再散列法

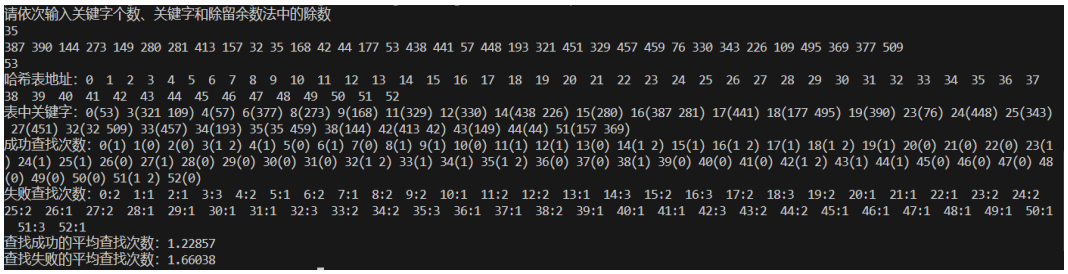
```
请依次输入关键字个数、关键字和除留余数法中的除数
10
1 4 72 107 79 48 118 87 56 126
13
哈希表地址: 0  1  2  3  4  5  6  7  8  9  10  11  12
表中关键字: 1:1 2:79 3:107 4:4 5:118 6:56 7:72 9:48 10:87 11:126
成功查找次数: 0:0 1:1 2:2 3:1 4:1 5:5 6:3 7:1 8:0 9:1 10:2 11:3 12:0
失败查找次数: 0:1 1:8 2:7 3:6 4:5 5:4 6:3 7:2 8:1 9:4 10:3 11:2 12:1
查找成功的平均查找次数: 2
查找失败的平均查找次数: 3.61538
```

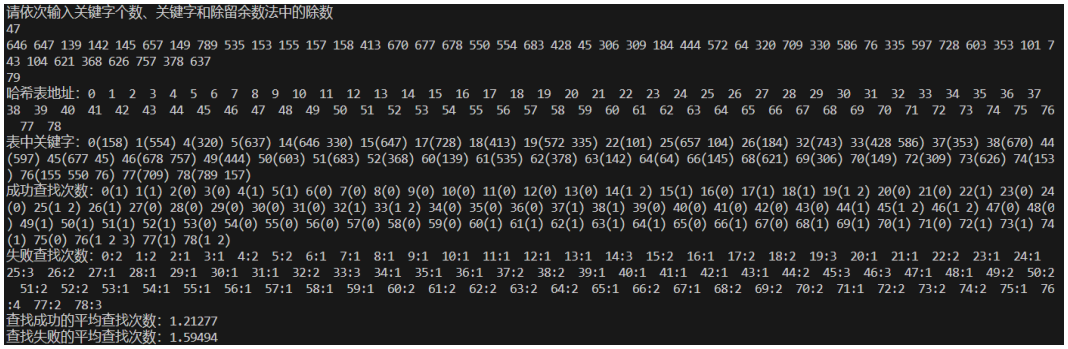
```
请依次输入关键字个数、关键字和除留余数法中的除数
35
387 390 144 273 149 280 281 413 157 32 35 168 42 44 177 53 438 441 57 448 193 321 451 329 457 459 76 330 343 226 109 495 369 377 509
53
哈希表地址: 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37
38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
表中关键字: 0:53 3:321 4:57 5:109 6:377 8:273 9:168 11:329 12:330 14:438 15:280 16:387 17:281 18:177 19:390 20:441 21:226 22:495 23:76 24:448 25:343 27:45
1 32:32 33:457 34:193 35:35 36:459 37:509 38:144 42:413 43:149 44:42 45:44 51:157 52:369
成功查找次数: 0:1 1:0 2:0 3:1 4:1 5:3 6:1 7:0 8:1 9:1 10:0 11:1 12:1 13:0 14:1 15:1 16:1 17:2 18:1 19:1 20:4 21:8 22:5 23:1 24:1 25:1 26:0 27:1 28:0 29:0
30:0 31:0 32:1 33:1 34:1 35:1 36:2 37:6 38:1 39:0 40:0 41:0 42:1 43:1 44:3 45:2 46:0 47:0 48:0 49:0 50:0 51:1 52:2
失败查找次数: 0:2 1:1 2:1 3:5 4:4 5:3 6:2 7:1 8:3 9:2 10:1 11:3 12:2 13:1 14:13 15:12 16:11 17:10 18:9 19:8 20:7 21:6 22:5 23:4 24:3 25:2 26:1 27:2 28:1 2
9:1 30:1 31:1 32:8 33:7 34:6 35:5 36:4 37:3 38:2 39:1 40:1 41:1 42:5 43:4 44:3 45:2 46:1 47:1 48:1 49:1 50:1 51:4 52:3
查找成功的平均查找次数: 1.77143
查找失败的平均查找次数: 3.62264
```

```
请依次输入关键字个数、关键字和除留余数法中的除数
47
646 647 139 142 145 657 149 789 535 153 155 157 158 413 670 677 678 550 554 683 428 45 306 309 184 444 572 64 320 709 330 586 76 335 597 728 603 353 101 7
43 104 621 368 626 757 378 637
79
哈希表地址: 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37
38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76
  77  78
表中关键字: 0:157 1:158 2:554 3:709 4:320 5:76 6:637 14:646 15:647 16:330 17:728 18:413 19:572 20:335 22:101 25:657 26:184 27:104 32:743 33:428 34:586 37:
353 38:670 44:597 45:677 46:678 47:45 48:757 49:444 50:603 51:683 52:368 60:139 61:535 62:378 63:142 64:64 66:145 68:621 69:306 70:149 72:309 73:626 74:15
3 76:155 77:550 78:789
成功查找次数: 0:2 1:2 2:2 3:6 4:1 5:9 6:2 7:0 8:0 9:0 10:0 11:0 12:0 13:0 14:1 15:1 16:3 17:1 18:1 19:1 20:2 21:0 22:1 23:0 24:0 25:1 26:1 27:3 28:0 29:0
30:0 31:0 32:1 33:1 34:2 35:0 36:0 37:1 38:1 39:0 40:0 41:0 42:0 43:0 44:1 45:1 46:1 47:3 48:3 49:1 50:1 51:1 52:1 53:0 54:0 55:0 56:0 57:0 58:0 59:0 60:1
 61:1 62:1 63:1 64:1 65:0 66:1 67:0 68:1 69:1 70:1 71:0 72:1 73:1 74:1 75:0 76:1 77:2 78:1
失败查找次数: 0:8 1:7 2:6 3:5 4:4 5:3 6:2 7:1 8:1 9:1 10:1 11:1 12:1 13:1 14:8 15:7 16:6 17:5 18:4 19:3 20:2 21:1 22:2 23:1 24:1 25:4 26:3 27:2 28:1 29:1
30:1 31:1 32:4 33:3 34:2 35:1 36:1 37:3 38:2 39:1 40:1 41:1 42:1 43:1 44:10 45:9 46:8 47:7 48:6 49:5 50:4 51:3 52:2 53:1 54:1 55:1 56:1 57:1 58:1 59:1 60:
6 61:5 62:4 63:3 64:2 65:1 66:2 67:1 68:4 69:3 70:2 71:1 72:4 73:3 74:2 75:1 76:11 77:10 78:9
查找成功的平均查找次数: 1.59574
查找失败的平均查找次数: 3.17722
```

5.2 链地址法







6 实验体会收获

通过本次实验掌握了用除留余数法构建哈希表、用线性探测再散列法和链地址法处理冲突的方法，加深了对哈希表和查找相关知识的认识。