Fundamentals of Data Structures

Laboratory Project 2：

Hashing—Hard Version

Authors:

张立涵

吴驰域

陈昱彤

Date：2015-12-31

Chapter 1: Introduction

1.1 The background of the problem

Given a hash table of size *N* , we can define a hash function *H(x)* = *x%N* . Suppose that the linear probing is used to solve collisions, we can easily obtain the status of the hash table with a given sequence of input numbers.

1.2 The description of the problem

Now you are asked to solve the reversed problem: reconstruct the input sequence from the given status of the hash table. Whenever there are multiple choices, the smallest number is always taken.

1.3 The description of the input and the output

Input: Each input file contains one test case. For each test case, the first line contains a positive integer *N* (1000), which is the size of the hash table. The next line contains *N* integers, separated by a space. A negative integer represents an empty cell in the hash table. It is guaranteed that all the non-negative integers are distinct in the table.

Output: For each test case, print a line that contains the input sequence, with the numbers separated by a space. Notice that there must be no extra space at the end of each line.

Chapter 2: Algorithm Specification

2.1 Specification of the algorithm

To solve this problem, we use 3 functions which are main function, InsertionSort function and FindPos function. For each sequence of input numbers, we select the valid numbers (not -1) and sort them in an increasing order. Then we use hash function and solve the collisions by linear probing to find the current positions of these sorted numbers. For each number, if the position we find is the same as the position of the original sequence, we print this number and drop it from the sorted numbers sequence, then we break, restart from the beginning of the sorted numbers sequence; if not, we test the current position of the next number. Repeat these steps above until all of the numbers from the sorted sequence have been printed.

2.2 Specifications of main data structures

1) For the InsertionSort function, we initialize a sorted sequence with 0 element, then each time we select an element from the unsorted sequence and insert it into the sorted sequence. When all the elements from the unsorted sequence have been inserted, we get a sorted sequence including all of the elements. Here is the key code:

```
for(j=1;j<N;j++){
        tmp=A[j];
        for(i=j;i>0&&A[i-1]>tmp;i--)
            A[i]=A[i-1];
        A[i]=tmp;
    }
```

2) For the hash function, we use p (p = x mod n) to find the position of number n. If there is a collision, we should add 1 to p until the collision disappears. If p meets the end of the array, we should let p be equal to 0. Here is the key code:

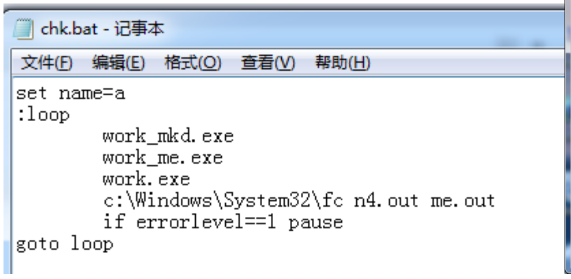
```
pos=X%N;
while(c[pos]>=0)
    if(++pos==N)   pos=0;
```

Chapter 3: Testing Result

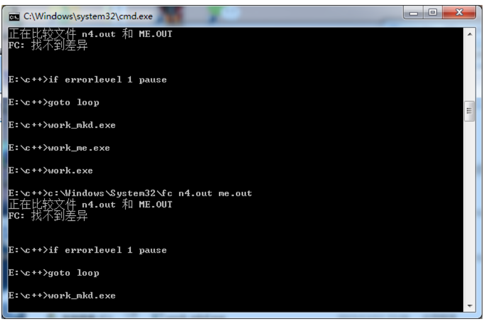
Input		Expected output	Actual output	Status
N	sequence			
11	33 1 13 12 34 38 27 22 32 -1 21	1 13 12 21 33 34 38 27 22 32	1 13 12 21 33 34 38 27 22 32	pass
	110 100 90 80 70 60 50 40 30 20 10	10 20 30 40 50 60 70 80 90 100 110	10 20 30 40 50 60 70 80 90 100 110	pass
	-1 12 23 34 15 26 -	7 12 15 19 18 23	7 12 15 19 18 23	pass

	1 7 19 18 -1	26 34	26 34	
5	-1 -1 -1 -1 -1	请按任意键继续. . .	请按任意键继续. . .	pass
0	/	请按任意键继续. . .	请按任意键继续. . .	pass

The first test case is the same as the sample. Since the program sorts the sequence from a smaller one to a larger one first, so the second test case is a queue of numbers which is sorted from a bigger one to a smaller one. Then two groups of border test cases which the input sequence is all -1 and the other is input 0 as N. Next, I design a random case which has many collisions. After testing special cases, I write two programs, one is to enumerate all possible sequences according to the input and check if it is the right as well as the best answer. The time complexity of the program can be terrible but it’s easy to write and it’s more likely to be correct. And the other program is to prosecute different input which is also a hash program. I use a “chk.bat” , file input and output to check if the program written by programmer is right. The pictures below show how I test the program.



```
set name=a
:loop
	work_mkd.exe
	work_me.exe
	work.exe
	c:\Windows\System32\fc n4.out me.out
	if errorlevel==1 pause
goto loop
```



```
正在比较文件 n4.out 和 ME.OUT
FC: 找不到差异


E:\c++>if errorlevel 1 pause

E:\c++>goto loop

E:\c++>work_mkd.exe

E:\c++>work_me.exe

E:\c++>work.exe

E:\c++>c:\Windows\System32\fc n4.out me.out
正在比较文件 n4.out 和 ME.OUT
FC: 找不到差异


E:\c++>if errorlevel 1 pause

E:\c++>goto loop

E:\c++>work_mkd.exe
```

After prosecuting new cases, running two programs which is written by me and the programmer and checking whether the answers are the same or not for an hour, we can draw a conclusion that the program is correct!

Chapter 4: Analysis and Comments

4.1 Analysis

The time complexity of this program is O(c*t*N) which t is the times of collisions and c is a large constant. However, the time complexity of the best situation is O(N) with a sorted and collisionless sequence which every figure is in the proper place. The time complexity of the worst situation can be O(n^2). Consequently, the time complexity is O(N^2).

The space complexity of this program is O(N) since the programmer doesn't use any recursion.

4.2 Comments

As for further possible improvements, since our programmer doesn't print the element i until all elements before it and element i undergo all collisions, I suggest we can do some preprocessing to calculate d[i] = a[i] % N and e[i] = abs(d[i] – its actual place), e[i] is the time of collisions which element i happens. After we sort the sequence, we can make use of e[i] to know how many collisions the element i will meet and we can calculate the actual place easily. We can handle each element by its size only once and the time complexity may become O(N). My idea is to use some math methods instead of simulating all collisions which may lead to waste too much unnecessary time. However, the feasibility of this new idea is under trying.

Appendix: Source Code (in C)

```c
#include <stdio.h>
#define MAX 1000

void InsertionSort(int A[],int N)     //use insertionsort algorithm to sort the numbers in
    the array
{
    int i,j;
    int tmp;
    for(j=1;j<N;j++){
        tmp=A[j];
        for(i=j;i>0&&A[i-1]>tmp;i--)    //find the correct position to store the number tmp
            A[i]=A[i-1];                //change the position
        A[i]=tmp;
    }
}

int FindPos(int X,int c[],int N)      //calculate hash value when insert the number to the
    c array
{
    int pos;
    if(X==-1) return -1;              //when the input number is equal to -1, then we
        return -1 as flag
    pos=X%N;
    while(c[pos]>=0)                 //use linear probing to solve the collision
        if(++pos==N)                 //when pos meets the end,then back to 0
            pos=0;
    return pos;
}


int main()
{
    int a[MAX];                      //this array is used to store the input numbers
    int b[MAX];                      //this array is used to store the valid numbers which is
        sorted
    int c[MAX];                      //this array is used as an assumed hash table
    int number;
    int N;
    int i;
    int pos;
    int flag=0;                      //this flag is used at output to control the output of
        blank
    scanf("%d",&N);
    for(i=0;i<N;i++){                //receive the data
        scanf("%d",&a[i]);
        b[i]=c[i]=-1;                //initialize b and c array by -1
    }
    for(i=0,number=0;i<N;i++)        //select the valid numbers in a array and store in b
        array
        if(a[i]>=0)
            b[number++]=a[i];        //variable number is used to store the number of data
    InsertionSort(b,number);
    while(1){
        for(i=0;i<number;i++){
            pos=FindPos(b[i],c,N);
            if(pos==-1) continue;
            if(a[pos]==b[i]){                  //if b[i] is in the correct position,
                if(flag==0){                   //then output the b[i]
                    printf("%d",b[i]);
                    flag=1;
                }
                else
                    printf(" %d",b[i]);
                c[pos]=b[i];                   //then store the b[i] in the correct
                    position in c array
                b[i]=-1;                       //then remove the b[i] by clearing the b[i]
                    to -1
                break;
            }
        }               //when a correct data is outputed, then continually calcualte the
            whole b array to find the next correct data
        if(i==number) break;           //if the whole data is outputed, then break
    }
    return 0;
}
```

Declaration

We hereby declare that all the work done in this project titled

“Fundamentals of Data Structures Laboratory Project 2: Hashing—Hard

Version” is of our independent effort as a group.

Duty Assignments:

Programmer: 张立涵

Tester: 吴驰域

Writer: 陈昱彤