Fundamentals of Data Structures
Laboratory Project 3

Hashing – Hard Version

COMPLETE DATE: 2018/12/28

CONTENTS

Chapter 1

Introduction

1.1 Background

Over the past few weeks, we have learned a new kind of data structure, which is called Hashing. Hashing is the transformation of arbitrary length input (also known as pre-image) into fixed length output by hashing algorithm, which is the hash value. This transformation is a compression mapping, that is, the hash value space is usually much smaller than the input space, different inputs may be hashed into the same output, so it is impossible to determine the unique input value from the hash value. To sum up, it is a function that compresses messages of arbitrary length into message digests of a fixed length.

Until now we have a clear understanding of it, and have mastered some basic operations. However, this project is not about simple operation such as insertion when knowing the sequence. It is based on the consequence of insertion, and require us to get the insertion sequence before. This reverse approach has greatly tempered our thinking and helped us to grasp the knowledge of hashing more deeply.

Here is a brief introduction to this project: We are given a hash table of size N. After defining a hash function, 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. However, now we 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.

1.2 Algorithm requirements

In order to reconstruct the input sequence from the given status of the hash table, we have to divide this problem into several parts using different functions.

During the main function, the basic algorithm is to define some arrays to save data produced in the following process. While dealing with the data, the elements in these arrays are changing and adapting. Therefore, we can acquire a sequence from the array and output them to get the input sequence, which is required by the problem.

Moreover, we have created several functions. For example, we define a judgment function to determine whether the elements in the original sequence are inserted into the new hash at the same location each time. Also, in order to simplify the codes, we define a function that will be used to insert elements into the hash.

1.3 Achieve results

After the programmer finished the codes, we test the program and here is the achieving results. As for the 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.

Then we get the output by running it, which is greatly corresponding to the requirements, namely that printing a line that contains the input sequence, with the numbers separated by a space. Also, we have got full mark on PTA, which is shown below.

提交结果

提交时间	状态	分数	题目	编译器	耗时
2018/12/22 20:13:26	答案正确	20	P3	C (gcc)	31 ms

测试点	结果	耗时	内存
0	答案正确	4 ms	392KB
1	答案正确	5 ms	256KB
2	答案正确	12 ms	256KB
3	答案正确	4 ms	256KB
4	答案正确	31 ms	256KB

代码

```
#include<stdio.h>
int judge(int m,int *newcell,int *cell,int N)  //Define a judgment function
{
  int k=m;
  k%=N;
  while(newcell[k]>=0)  //When conflict occurs, move backwards.
  {
    k++;
    k%=N;
  }
```

Chapter 2

Algorithm Specification

To begin with, we define two functions in advance. One is *judge*(), which is a judgment function to determine whether the elements in the original sequence are inserted into the new hash at the same location each time, the other is *insert*(), which is a function that inserts elements into a hash.

Let's talk about the main function. First and foremost, we define several parameters, such as the size of the hash table, some cycles, counts and flags. Three arrays are defined to accept the sequence of elements, reinsert and output. Then we initialize all elements of the newly created hash array to all -1 and start accepting element sequences.

We use a loop to solve the whole problem. The key is to find the values that hold in the subsequent loop and select the smaller values from them. When finding the next element, we should make some changes to the new and former hashes. More specifically, the first step is to find the location of minimum in the former sequence. The second step is to change the element stored in the corresponding location of the former hash to a negative number. The third step is to insert this element into the new hash. Finally, we use a circulation to output results and the work is done completely.

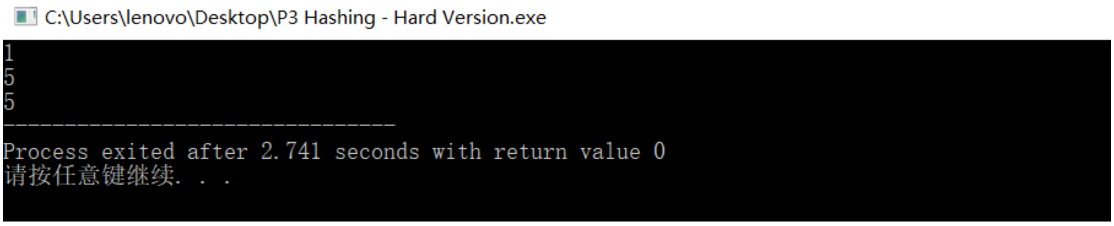
In summary, here is a brief specification. First and foremost, create a new empty hash, and then insert each element again. From the newly inserted elements, find the smallest elements whose insertion position is the same as the order given by the title. From this, we can know that the smallest element is the first element in the required order. After continuous circulation and repeated searching for the smallest element, we can find the Minimum insertion order, which is the requirement of this project.

Chapter 3

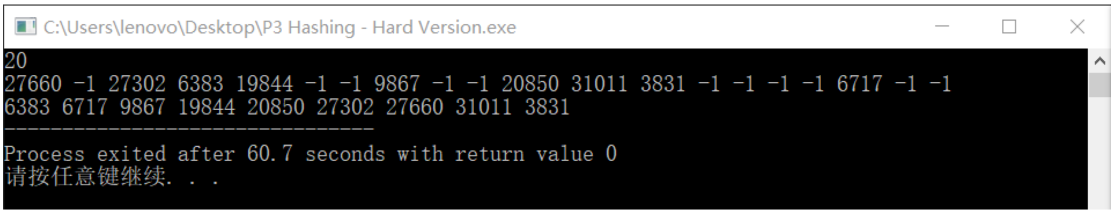
Testing Results

1. N = 10, no number

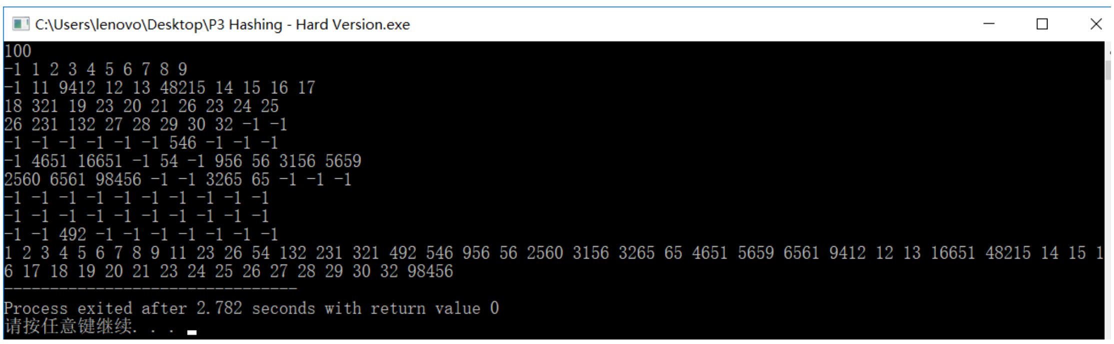
2. N = 1, random numbers



3. N = 20, random numbers



4. N = 100, random numbers



5. N = 1000, random numbers

C:\Users\lenovo\Desktop\P3 Hashing - Hard Version.exe

```
1000
2000 -1 -1 5003 21004 31004 -1 15007 18008 11009
28010 9011 -1 -1 -1 -1 -1 -1 -1 14019
28020 11021 5022 24022 10022 20025 28023 11024 1019 -1
-1 13031 4032 13032 -1 -1 17036 3036 20038 6039
-1 9041 42 4042 12044 12045 -1 -1 -1 -1
-1 -1 -1 12053 54 20054 20056 -1 -1 -1
-1 18061 13062 31061 -1 13065 -1 -1 -1 -1
-1 28071 -1 19073 -1 -1 5076 -1 -1 -1
-1 -1 -1 2083 -1 24085 -1 17087 18088 27089
-1 -1 -1 -1 3094 -1 -1 -1 5098 -1
4100 -1 31102 3103 -1 -1 16106 30107 31108 -1
5110 -1 -1 -1 -1 18115 31116 -1 -1 16119
21120 -1 -1 -1 124 -1 2126 -1 18128 -1
7130 -1 -1 -1 -1 -1 -1 -1 -1 -1
16140 141 15142 20143 143 4145 28146 30146 -1 -1
-1 1151 -1 23153 154 2155 26155 19157 27158 -1
20160 -1 9162 2162 -1 7165 -1 -1 -1 29169
19170 4170 2169 29171 11174 32171 -1 -1 8178 -1
24180 -1 14182 24183 -1 -1 31186 13187 19188 -1
17190 22191 30192 18191 6192 194 3196 23196 23197 17193
23200 25201 10196 16203 10203 -1 -1 -1 -1 -1
32210 -1 -1 -1 29214 -1 -1 -1 -1 -1
-1 -1 24222 20223 17223 6225 -1 -1 30228 -1
-1 -1 -1 -1 -1 -1 236 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
5250 -1 -1 -1 28254 17254 15256 -1 -1 -1
-1 -1 13262 -1 12264 19265 -1 -1 -1 -1
-1 6271 32271 24273 14271 -1 -1 -1 -1 -1
16280 -1 23282 8282 15282 16283 10286 28287 12288 289
```

C:\Users\lenovo\Desktop\P3 Hashing - Hard Version.exe

```
-1 -1 -1 27983 -1 -1 13986 23987 -1 -1
14990 -1 -1 -1 -1 -1 2996 25997 -1 31999
42 54 124 141 154 236 289 293 482 492 779 901 1151 1417 1544 1588 1764 1843 1870 1927 2000 2083 2126 2155 2307 2369 2511
 2635 2696 2996 3094 3103 3196 3298 3431 3435 3549 3558 3603 3626 3729 3738 3789 3903 4032 4042 4100 4145 4314 4415 4475
 4597 4640 4665 4679 4735 4803 4834 4887 4967 5003 5022 5076 5098 5110 5250 5437 5448 5536 5538 5700 5706 5787 5830 5845
 6039 6225 6271 6335 6360 6423 6484 6618 6619 6730 6869 6901 7130 7165 7377 7442 7449 7488 7519 7617 7628 7712 7883 7901
 6903 7959 8178 8361 8481 8493 8724 8910 9011 9041 9162 2162 9375 9504 9515 9742 9759 9790 9833 1833 9895 9906 9931 9962
 10286 10292 10384 10467 10556 10586 10713 10809 11009 11021 11174 11324 11338 11479 11512 11539 11702 11834 11841 11943
 8943 12044 12045 12053 12264 12288 12293 12317 12383 12456 12624 12860 12950 13031 13032 13062 13065 13187 13262 13291
13402 13459 13695 13932 13967 13978 13986 14019 14182 14310 14311 14344 14475 14605 14689 14772 14894 14946 14990 15007
15142 15256 15351 15458 15574 15575 15725 15891 16106 16119 16140 16203 10203 16280 16414 16513 9513 16520 16542 16688 1
6828 4828 16942 16945 17036 3036 17087 17190 17372 17411 17422 2422 16424 17438 17452 17506 17674 17774 17808 17862 1795
9 18008 18061 18088 18115 18128 18468 468 18589 18637 18652 18717 18757 18763 18788 18876 18897 18936 19073 19157 19170
4170 19188 19265 19559 19590 19630 19669 2669 19712 19719 19797 19816 19867 19896 19913 913 19955 19977 20025 20038 2005
4 20056 20143 143 20160 20223 17223 20329 20417 20451 20473 20581 20601 2601 20672 21004 21120 21426 21539 20538 18539 2
1549 16550 21660 21695 21719 21725 21727 21882 22191 22356 22387 22414 22467 22550 12551 22647 22649 22705 22726 22799 1
4799 20799 2801 22814 22889 22930 23153 23196 23197 23200 23282 8282 15282 16283 23623 23647 23656 1656 23806 23812 2384
5 23852 23987 24022 10022 24085 24180 24183 24222 24273 24356 24371 24373 24390 24394 24465 24485 22484 20486 24489 2459
7 24627 2626 24649 20650 24768 24947 20946 25201 25485 3488 25548 25668 4668 25722 25735 25825 25875 25997 26155 26293 2
6300 26303 26309 26363 26419 26478 26501 26577 9577 26778 26925 1925 26963 14963 27089 27158 27349 27351 24351 27433 274
47 27507 27530 12530 27594 22594 27596 27625 21625 2626 25628 5630 27645 27754 3754 27893 27939 27983 28010 28020 28023
11024 1019 28071 28146 28254 17254 28287 28298 28322 28434 28465 28477 28521 28618 28693 28704 28746 4746 29169 2169 291
71 29214 29315 4314 29335 29359 29511 29566 29578 29658 29659 22659 29870 26870 29973 30107 30146 30192 18191 6192 194 1
7193 10196 30228 30304 30334 30524 30528 30837 30834 30933 30975 31004 31061 31102 31108 31116 31186 31287 7286 31317 83
14 9315 20316 31323 10323 31330 31427 12424 25424 31557 31674 31929 31999 32171 32210 32271 14271 32392 7392 32440 26440
 32526 32592 32610 32663 32703 32758 23758 27757 25761 4758 22759
--------------------------------

Process exited after 2.213 seconds with return value 0
请按任意键继续. . .
```

6. N = 50, the numbers all have the SAME hash value

C:\Users\lenovo\Desktop\P3 Hashing - Hard Version.exe

```
50
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
--------------------------------

Process exited after 23.57 seconds with return value 0
请按任意键继续. . .
```

7. N =100, the numbers all have the SAME hash value

```
C:\Users\lenovo\Desktop\P3 Hashing - Hard Version.exe
100
-1 -1 -1 -1 -1 -1 -1 -1 -1 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9
--------------------------------
Process exited after 1.798 seconds with return value 0
请按任意键继续. . .
```

8. N =100, the numbers are some of the hash value

```
C:\Users\lenovo\Desktop\P3 Hashing - Hard Version.exe
100
-1 -1 2 2 3 5 5 6 6 7
3 11 12 12 4 15 9 6 18 19
19 18 22 23 23 25 26 23 28 28
30 22 27 7 12 19 10 23 18 20
26 25 24 22 3 4 4 15 22 2
24 29 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
2 2 3 5 5 6 6 7 3 11 12 12 4 15 9 6 18 19 19 18 22 23 23 25 26 23 28 28 30 22 27 7 12 19 10 23 18 20 26 25 24 22 3 4 4 1
5 22 2 24 29
--------------------------------
Process exited after 1.929 seconds with return value 0
请按任意键继续. . .
```

9. N =1000, the numbers are some of the hash value

```
C:\Users\lenovo\Desktop\P3 Hashing - Hard Version.exe
950 -1 -1 -1 -1 1955 -1 -1 -1 1959
2959 -1 962 2963 2963 -1 -1 1967 1967 -1
-1 -1 -1 2973 -1 975 -1 1977 1978 -1
-1 -1 -1 983 -1 -1 1986 2987 -1 -1
2990 -1 -1 -1 -1 -1 2996 1997 -1 1999
4 7 8 11 39 41 42 44 45 53 54 61 85 88 89 94 103 107 115 120 124 128 141 142 154 158 162 180 183 192 196 222 225 228 236
 256 264 271 273 288 289 293 293 298 304 317 334 335 349 351 351 351 360 371 373 375 383 390 394 423 426 431 433 435 447
 456 458 465 468 468 482 484 485 489 492 504 507 515 524 528 530 530 549 549 558 574 575 589 594 596 603 618 619 624 625
 626 627 637 645 652 660 717 725 725 727 729 730 738 742 754 754 757 759 763 768 779 788 789 790 833 837 860 869 876 882
 891 893 895 897 901 901 903 906 931 933 936 939 947 950 962 975 983 1004 1010 1020 1031 1032 1032 1042 1062 1061 1065 1
071 1073 1100 1102 1106 1108 1116 1119 1130 1140 1145 1146 146 1151 1157 1165 1170 1170 1186 1187 1188 1191 191 192 194
1201 1203 1203 1254 1262 1265 1280 1286 1287 1287 1291 1292 1298 1314 1317 1322 1323 1330 1356 356 1377 1384 1387 1402 1
414 1415 1414 1417 1427 1434 1442 1449 1459 1465 1467 1467 1475 1477 1484 1488 1513 513 1519 1520 1521 1542 1544 1548 15
50 1550 551 1556 1557 1559 1586 1588 1590 1594 1597 597 1617 1618 1628 1630 1640 1647 1649 649 1665 1668 1669 1679 1688
1693 1695 1704 1705 1712 1713 1712 1719 719 1722 1726 1735 1735 1746 1746 1764 1797 1799 1803 1809 1814 1816 1825 1828 1
828 1834 1833 1843 1867 1870 1875 1883 1887 1889 1896 1901 903 1913 913 1927 1929 1930 1932 1942 1945 1955 1959 1967 196
7 1977 1978 1986 1997 1999 2000 2003 2009 2019 2021 2022 22 1022 2025 1023 2024 1019 2036 36 2038 2054 2056 2076 2083 20
87 2098 2110 2126 2143 143 2153 2155 2155 2160 2162 2169 2169 2171 2174 2171 2178 2182 2190 2196 2197 2193 2200 1196 221
0 2214 2223 2223 2250 2254 2271 2271 2282 2282 282 1283 2293 1286 2300 2303 2307 2309 2310 2311 2315 1314 2314 315 2316
2324 1323 2329 2335 2338 2344 2359 2361 2363 2369 2372 2392 1392 2411 2417 2419 2422 2422 1424 424 1424 2437 2438 2440 2
440 2448 2451 2452 2473 2475 2478 2479 2481 2486 1485 488 2493 2501 2506 2511 2512 2511 2526 2536 2538 2539 539 2538 539
 2566 2577 577 2578 2581 2592 2601 2601 2605 2610 2623 2626 625 2626 1628 2630 2635 2647 2650 2656 1656 2658 2659 1659 2
663 2669 1668 2672 2674 1674 2689 2696 695 2700 2702 2703 2706 2724 2758 2758 757 1761 1758 1759 2772 2774 2778 2787 279
9 2799 2801 2806 2808 2812 2830 2834 834 2841 2845 2845 2852 2862 2870 2870 2894 2910 2925 1925 2943 2943 2946 2946 2959
 2963 2963 2973 2987 2990 2996
--------------------------------
Process exited after 4.176 seconds with return value 0
请按任意键继续. . .
```

Chapter 4

Analysis and Comments

4.1 Analysis

In this project, the code written by the programmer uses two functions, one main function.

The main ideas of the algorithm are as follows:

Create a new empty hash, and then insert each element again. From the newly inserted elements, find the smallest elements whose insertion position is the same as the order given by the title. From this, we can know that the smallest element is the first element in the required order. After continuous circulation and repeated searching for the smallest element, we can find the order of the title——Minimum insertion order.

We use judge function to re-insert the hash table given by the title, and find the smallest element in the rearranged hash list. Then we insert the smallest element into the new array through insert function, repeat and complete the whole program.

The first function, judge function, is used to determine the order given by the title. By judging the elements from cell [0] to cell [N], we can find the elements in the same order as the title, and then jump out of the function.

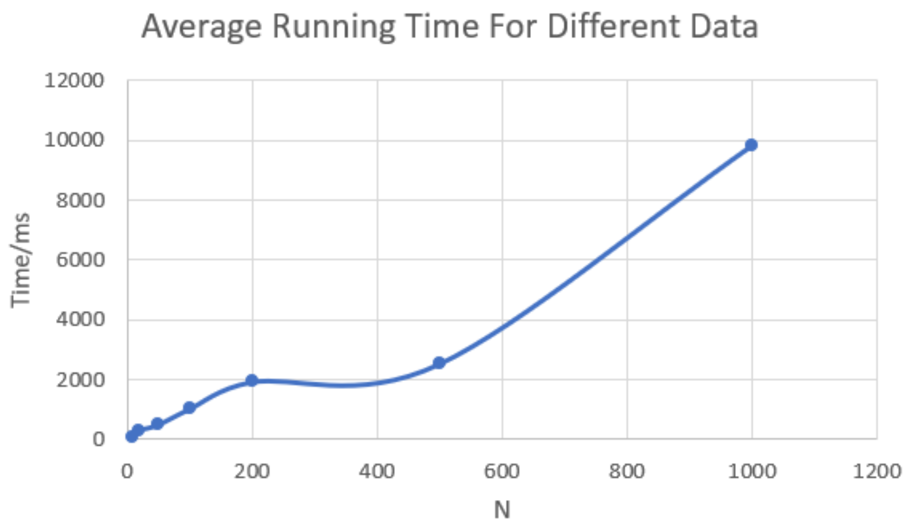
Then in the main function, find the smallest element of the element judged as 1 by the judge function, and insert it into the new array new cell through the insert function.

4.1.1 Analysis the time complexity of the algorithm

Time table (Average Time):

N	10	20	50	100	200	500	1000
Random number Time/ms	101	420	1030	1869	2398	2509	6256

The chart is as follows:



The chart above has successfully show the trend of average time complexity and worst case time complexity which will be discussed in the next. Considering that we use 3 superposition loops in a single run, the worst case time complexity is $O(N^3)$.

4.1.2 Analysis of the space complexity of the algorithm

Both time complexity and space complexity are necessary to consider. The former could influence the time that would be use to solve the problem. The latter has impact on storage space. The less the time complexity as well as the space complexity, it would be necessity to a better algorithm. As for the space complexity of our codes, on account that two structures are used to store the graph, the space complexity of the algorithm is $O(N)$, which is not so high.

4.2 Comments on further possible improvement

The whole program is very clear and organized, contains the idea of modular problem solving, is also conducive to other people's understanding, at the same time, the code has enough and detailed comments, reflects the programmer's thinking process and ideas, and makes it easier for readers to understand the whole program thoroughly.

The current array we used in this problem is cell[MAX_SIZE]. If we can define the array dynamically, some space may be saved when the size of problem is small. The read in process can be fasten using “putchar” instead of “scanf”. However, neither improvement is very significant as the size of N is rather small. And the whole algorithm can use less time complexity algorithm.

All in all, the programmer has done a good job.

Appendix

Source Code (in C)

```
#include<stdio.h>

int judge(int m, int *newcell, int *cell, int N)
//Define a judgment function to determine whether the elements in the original sequence are
inserted into the new hash at the same location each time.
{
	int k=m;
	k%=N;
	while(newcell[k]>=0)	//When conflict occurs, move backwards.
	{
		k++;
		k%=N;
	}
	if(cell[k]==m)		//If the insertion position in the new hash is the same as that in the
old hash, the condition can be satisfied
	{
		return 1;
	}
	else return 0;
}

void insert(int m, int *newcell, int N)	//Define a function that inserts elements into a hash
{
	int k=m;
	k%=N;
	while(newcell[k]>=0)	//When conflict occurs, move backwards.
	{
		k++;
		k%=N;
	}
	newcell[k]=m;		//After the corresponding position array subscript is found, the element
is assigned to the array unit.
}
```

```
int main()
{
    int N;          //define the size of the hash table
    scanf("%d",&N);
    int cell[1001];         //Define an array that accepts a sequence of elements
    int newcell[1001];          //Define a re-inserted hash
    int result[1001];           //Define the output result array
    int i,j,min,num=-1,sum=0,k,flag=0;      //Define some cycles, counts, and flags
    for(i=0;i<N;i++)
    newcell[i]=-1;          //Initialize all elements of the newly created hash array to -1
    for(i=0;i<N;i++)        //Start accepting element sequences
    {
        scanf("%d",&cell[i]);
        if(cell[i]>=0)
        sum++;              //Sum is used to record the number of elements in a table
    }
    for(i=0;i<sum;i++)          //Total cyclic sum times
    {
        for(j=0;j<N;j++)        //Start at the head of the cell every time.
        {
            if(cell[j]>=0)      //If the value of the element is less than 0 and is expressed
as null, the following loop is not necessary
            {
                if(flag==0&&judge(cell[j],newcell,cell,N))   //Find the first valid element value
                {
                    min=cell[j];
                    flag=1;
                }
                else if(flag==1&&judge(cell[j],newcell,cell,N))     //Find the values that hold
in the subsequent loop and select the smaller values from them
                {
                    if(min>cell[j])
                    min=cell[j];
                }
            }
        }
            if(flag!=0)             //When you find the next element, make some changes
to the new and old hashes
            {
                result[++num]=min;
                for(k=0;k<N;k++)  //Find the location of min in the old sequence
                {
                    if(cell[k]==min)
```

```
                    break;
                }
                cell[k]=-1;           //Change the element stored in the corresponding
location of the old hash to a negative number
                insert(min,newcell,N);    //Insert this element into the new hash
            }
            flag=0;   //Change the marker variable of whether to find the next element to 0
    }
    for(i=0;i<=num;i++)      //Circulation of output results
    {
        if(i==0)        //When the output is the first, there is no space
        printf("%d",result[i]);
        else
        printf(" %d",result[i]);
    }
}
```

Declaration

We hereby declare that all the work done in this project titled " Hashing – Hard Version " is of our independent effort as a group.

Duty Assignments:

Programmer: Mu Qingfeng
Tester: Guo Han
Report Writer: Wang Ruopeng