

Project 3:

Hashing – Hard

Version

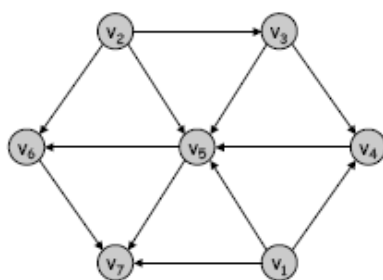
Date:2016-1-1

Chapter 1: Introduction

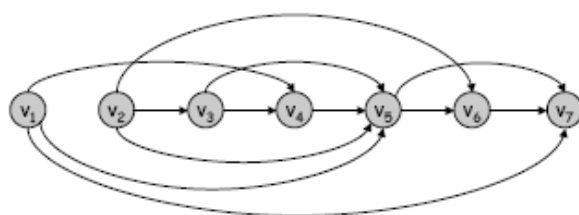
Generally speaking, we use a hash to solve collisions, obtaining the status of the hash table with a given sequence of input numbers. Now, we want to solve the problem: Given a hash table. Whenever there are multiple choices, the smallest number is always taken. About input, the first line contains a positive integer N (≤ 1000), which is the size of the hash table. The next line contains N integers, separated by a space. Then, print a line that contains the input sequence, with the numbers separated by a space.

Chapter 2: Algorithm Specification

Our program gets an output in the sequence by a Topological. First, define a degree of the input hash index, then create a map to sort the input node with no collisions. If the input node causes collisions, keep the degree and add the input node into the vector of previous nodes. Second, we are going to print the node in the hash table in the order. It always gets the smallest degree node which has been sorted by the key in the map, until there are no nodes with a 0 degree. When a node is borrowed out from the map, the degree of nodes in its next array will decrease one. Any node with degree 0 will then be stored in the map.



a DAG



a topological ordering

Chapter 3: Test case

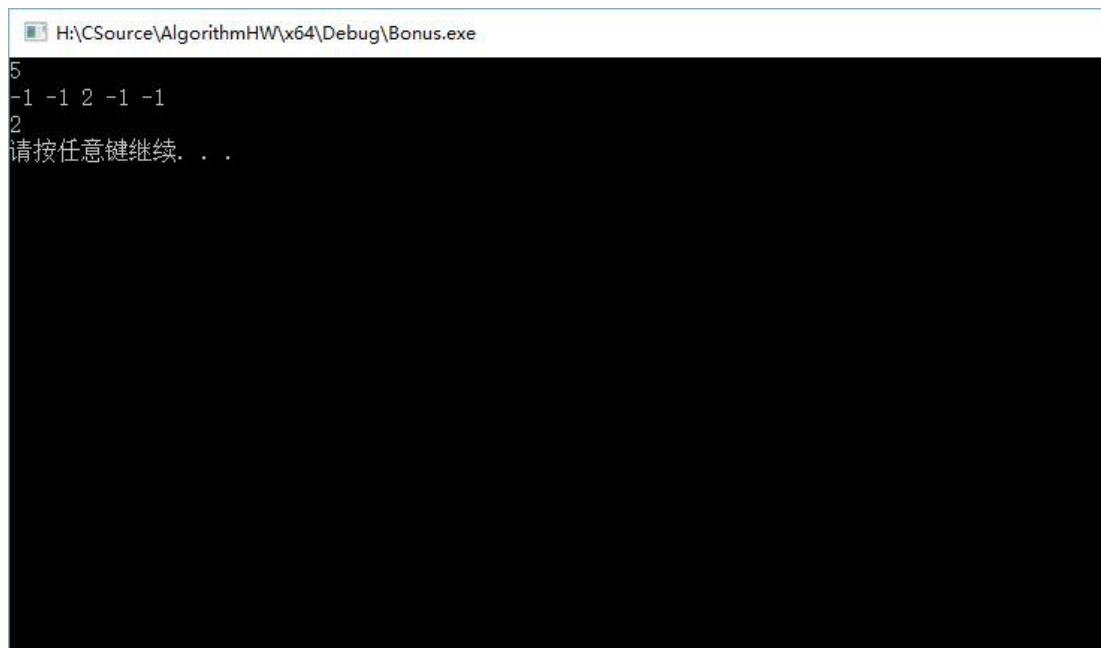
1)

```
H:\CSource\AlgorithmHW\x64\Debug\Bonus.exe
11
33 1 2 24 4 25 37 48 9 31 42
1 2 4 24 25 31 33 37 42 48 9请按任意键继续. . .
```

2)

```
H:\CSource\AlgorithmHW\x64\Debug\Bonus.exe
11
33 1 13 12 34 38 27 22 32 -1 21
1 13 12 21 33 34 38 27 22 32请按任意键继续. . .
```

3)




```
H:\CSource\AlgorithmHW\x64\Debug\Bonus.exe
5
-1 -1 2 -1 -1
2
请按任意键继续. . .
```

4)



```
H:\CSource\AlgorithmHW\x64\Debug\Bonus.exe
5
-1 -1 -1 -1 -1
请按任意键继续. . .
```

5)



```
选择H:\CSource\AlgorithmHW\x64\Debug\Bonus.exe
5
0 1 2 3 4
0 1 2 3 4
请按任意键继续. . .
```

6)



```
H:\CSource\AlgorithmHW\x64\Debug\Bonus.exe
1
0
0
请按任意键继续. . .
```

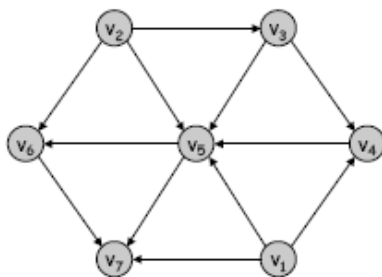
Obviously, all these test cases are right. The program has no bug.

Chapter4:Analysis and comments

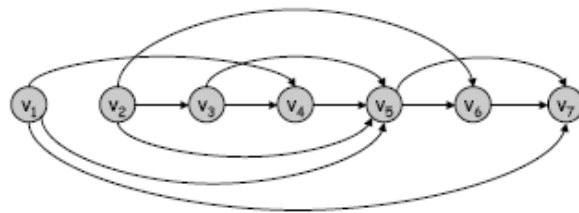
The program use a Topological sort to get a output in the sequence we want. It can be separated into two parts as fellows.

The first is to define the degree of the input hash index(collisions), then create a map to sort the input node with no collisions. If the input node cause collisions, keep the degree and add the input node into the vector of previous node;

The second is going to print the node in the hash table in the order that It was before being inserted. It always gets the smallest degree node which has been sort by the key in the map, until there is no nodes with a 0 degree. When a node is borrowed out from map, the degree of node in its next array will decrease one. Any node with degree 0, will then be store in the map.



a DAG



a topological ordering

It can be easily acknowledged that the time complexity of the program is $O(n + e)$; In this case, the n means the size of the hash table, while the e means the number of the collisions caused during insertion. The program has to scan the total of the table, so it take $O(n)$. The program needs to deal with the collisions, which takes $O(e)$;

According to the reason above, it quiet considerable and reasonable for us to assert that the complexity of the program is $O(n+e)$;

Appendix: Source Code

```
#include <iostream>

#include <vector>

#include <map>

using namespace std;

//拓扑排序

struct node{

    int num;//结点数值

    int degree;//结点的入度，表示当前位置与首次哈希位置的偏移值，即冲突次数

    vector<int> next;//后继结点

};

int main() {

    int N;

    cin >> N;

    map<int, int> map;

    int count = 0;

    vector<node> a(N);

    for (int i = 0; i<N; i++){//结点序号 0 到 N-1

        cin >> a[i].num;

        if (a[i].num >= 0){
```

```
count++; //有效结点的个数，用于输出控制
```

```
if (a[i].num%N == i){ //一次命中，入度为 0
```

```
    a[i].degree = 0;
```

```
    map[a[i].num] = i; //数值为 key，自动升序排列，便于
```

直接找到最小值,位置为 value

```
}
```

```
else{
```

```
    a[i].degree = ((i + N) - a[i].num%N) % N; //与首次
```

hash 的偏移值，即线性探测的次数

```
    int k = a[i].degree;
```

```
    while (k){
```

```
        a[(i - k + N) % N].next.push_back(i); //该结点
```

push 到所有前驱结点的 next 数组

```
        k--;
```

```
    }
```

```
}
```

```
}
```

```
}
```

```
while (!map.empty()){ //map 中存储所有入度为 0 的结点
```

```
    auto it = map.begin();
```

```
    cout << it->first << (--count ? " " : ""); //first 是结点数值，second 是
```

结点序号


```
        for (auto index = a[it->second].next.begin(); index != a[it-
>second].next.end(); index++){

            a[*index].degree--;//所有后继节点入度减 1

            if (a[*index].degree == 0)

                map[a[*index].num] = *index;

        }

        map.erase(it);

    }

    printf("\n");

    system("pause");

    return 0;

}
```