Lab 3 - Hash Table and Topological Sort

Group 23

Jan. 1^{st}, 2019

Contents

Chapter 1

Introduction

1.1 Description

Given a hash table of size N, we can define a hash function $H_i = A_i \ mod \ N$. 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.

However, 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.

Example see chapter 2-1

1.2 Purpose

1. Master how to insert and find elements from a hash table.
2. Learn how to build a graph module while solving a problem.
3. Use adjacency list to save a graph.
4. Solve problem with Topological sort.
5. Design algorithms which has different complexity of space and time.
6. Learn to analyze the complexity of space and time.
7. Test code, find bugs, and debug.

Chapter 2

Algorithm Specification and Complexity Analysis

2.1 Item analysis and Example

First of all, the purpose of this algorithm is, given a hash table and try to recover a possible insert order which satisfies that each time inserting an element, the one should be the minimum.

First, let's consider when is an element is available to insert.

Let $H_i = A_i \ mod \ N$. We can easily find that if an element in the hash table can be inserted, then all the elements from A_{H_i} to A_{i-1} should be already inserted before A_i, and there mustn't be a space(-1) in this range.

For example

Listing 2.1 Input

```
11
33 1 13 12 34 38 27 22 32 -1 21
```

In this input, we can first calculate array H.

```
i   0  1  2  3  4  5  6  7  8  9  10
A  33  1 13 12 34 38 27 22 32  -1 21
H   0  1  2  1  1  5  5  0  10 -1 10
```

In this case, some of these elements whose $H_i = i$ can be inserted directly, they are: $A[0] = 33, A[1] = 1, A[2] = 13, A[5] = 38, A[10] = 21$.

In order to show this method more clearly, suppose we have a priority queue (written by heap) to save those elements who can be printed directly. We have $Q = \{1, 13, 21, 33, 38\}$. Now we can output $1, 13$, let $O = \{1, 13\}$, $Q = \{21, 33, 38\}$.

Now, things become somehow different.

```
i   0  1  2  3  4  5  6  7  8  9  10
```

```
2: A  33  X  X 12 34 38 27 22 32  -1 21
3: H   0  X  X  1  1  5  5  0  10 -1 10
```

We can see $A[3] = 12$, satisfies that elements from $A[H[i]] = A[1]$ to $A[i-1] = A[2]$ are both printed, this means $A[3] = 12$ can be printed now. Let $Q = \{12, 21, 33, 38\}$. Then print 12, we have $Q = \{21, 33, 38\}$, $O = \{1, 13, 12\}$.

```
1: i   0  1  2  3  4  5  6  7  8  9  10
2: A  33  X  X  X 34 38 27 22 32  -1 21
3: H   0  X  X  X  1  5  5  0  10 -1 10
```

Then after printing $A[3]$, $A[4]$ can be printed as well. then insert $A[4] = 34$ into Q. Get $Q = \{21, 33, 34, 38\}$, print minimum element, $Q = \{33, 34, 38\}$, $O = \{1, 13, 12, 21\}$.

In this way, each time delete an element, detect if there is a possible element can be printed, and add these elements into Q, and then output the minimum (first) element in Q. Do this again and again until Q is empty.

2.2 Algorithm Description

2.2.1 Algorithm 1

Algorithm Description

The most basic algorithm is, make a heap Q, and each time an element is output, detect the whole table to find those elements are available to output.

Complexity Analysis

Time Complexity Altogether output $O(N)$ numbers, each time need $O(N^2)$ time to check the whole array. Each element will be inserted into the heap once, cost $O(NlogN)$ time. Therefore, the total complexity is $O(N * N^2 + NlogN) = O(N^3)$.

Space Complexity Obviously, space complexity is $O(N)$.

2.2.2 Algorithm 2

Algorithm Description

In algorithm 1, checking the whole array cost too much time. This time we consider in another way. Consider to solve this problem with graph data structure.

First, if an element $A[i]$ (called a vertex, or a node) is available to print, then it depends on whether the $O(N)$ vertices from $A[H[i]]$ to $A[i-1]$ are printed. Then, it's easy to add an edge to $A[i]$ from the vertices which $A[i]$ depends on. In this way, we use $O(N^2)$ edges and $O(N)$ vertices. Then run topological sort. Each time printing a node from the graph, delete it at the same time. See figure 2.2.2.1

Algorithm 1

```
1: function CheckWholeArray(A,Q)
2:     for i ← 0 to N − 1 do
3:         for j ← A[i] mod N to i − 1 do
4:             if A[i] is not output or A[i] is empty then
5:                 break
6:             end if
7:         end for
8:         if not break then Q.push(A[i])
9:         end if
10:    end for
11: end function
12:
13: function Algorithm1(A,Q)
14:    Check(A,Q)
15:    while ( doQ is not empty)
16:        Print Q.top()
17:        Q.pop()
18:        CheckWholeArray(A,Q)
19:    end while
20: end function
```

Meanwhile, keep pre-printed nodes with a heap(STL priority_queue) to ensure the minimum order.

Algorithm 2

```
1: function BuildGraph
2:     for i ← 0 to N − 1 do
3:         for j ← A[i] mod N to i − 1 do
4:             if A[j] >= 0 then
5:                 AddEdge(j,i)
6:             end if
7:         end for
8:     end for
9: end function
```

Complexity Analysis

Time Complexity Each node needs at worst $O(N)$ time to insert edges. So the time cost of building graph is $O(N^2)$. Then, we need $O(NlogN)$ time to keep a heap. So, the total complexity is $O(NlogN + N^2) = O(N^2)$.

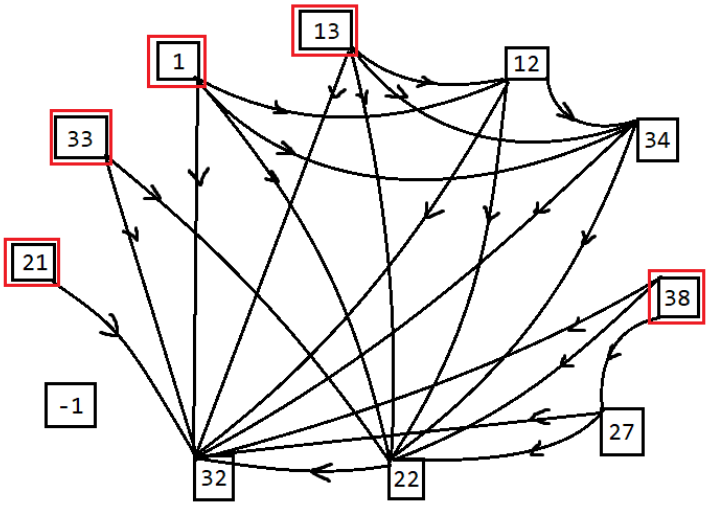

Figure 2.2.2.1: example in chapter 1, red vertices are those whose in-degree is 0

Space Complexity There are $O(N^2)$ edges and $O(N)$ vertices, this cost $O(N^2)$ space(no matter which method is used to store the graph). Heap need $O(N)$ space. Total space complexity is $O(N^2)$.

2.2.3 Algorithm 3

Algorithm Description

In algorithm 2, we find that building the graph has cost too much time and space. When N become large, $O(N^2)$ time and space complexity is not acceptable. This algorithm uses a data structure called ***Segment Tree***.

Segment Tree is a tree-shaped data structure, and it is a complete-binary-tree, which means the height of segment tree is at most $O(logN)$. Each node of segment tree represents a range of elements. Figure 2.2.3.1 is an example. As we see, each node has two children, each child represents the left or right half range of father-node. And it's easy to find out that, each node has an index(orange colored). Assume a node whose index is x, then its left child is $x * 2 = x << 1$, and right child is $x * 2 + 1 = x << 1|1$. Finding out this, suggests we can build the tree with $O(N)$ linear array, rather annoying pointers.

After building this tree, we can excitingly find that we can build the graph described in algorithm 2, in less time. Because using nodes in the tree can reduce the time spent on connecting $O(N)$ nodes to which is depends on them. This time is reduce to $O(logN)$.

For example, if 11th node depends on node 4th-10th, we don't need to connect the 7 nodes to it, we use 3 nodes in the tree whose indicis are 9,5 and 12, each represents 4-4,5-8 and 9-10. See figure 2.2.3.2.

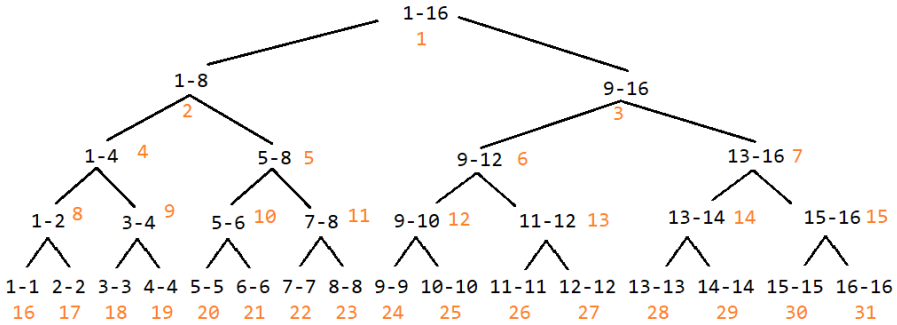

Figure 2.2.3.1: Segment Tree

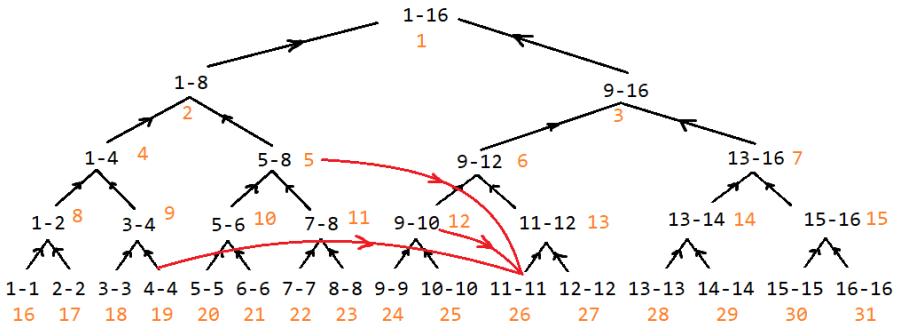

Figure 2.2.3.2: Suppose 11th node dependents on 4th-10th nodes

Build the graph as figure 2.2.3.2 shows, then run topological sort like what algorithm 2 does, would work.

Complexity Analysis

Time Complexity Though we built a large tree, the number of vertices is still $O(2^{\lceil log_2(N) \rceil}) = O(N)$. And the height of tree is $O(logN)$, therefore operations on the tree are $O(logN)$. And the number of edges is $O(NlogN)$, for original segment tree has $O(N)$ edges, and each node adds $O(logN)$ edges, totally is $O(N+N*logN) = O(NlogN)$. So, building the graph can be done in $O(NlogN)$ time.

Total time complexity is (time of build tree)+(time of connecting nodes)+(time of topological sort and keeping a heap)=$O(N + NlogN + (NlogN + NlogN)) = O(NlogN)$

Space Complexity As said above, there are $O(NlogN)$ edges, and $O(N)$ vertices. The total space complexity is (edges)+(vertices)+(heap)=$O(NlogN + N + N) = O(NlogN)$

Chapter 3

Testing Method and Result

Test can be divided into two parts: 1.hand-made special cases; 2.random data

3.1 Special cases test

Sample Sample input from pta.

```
Input:
11
33 1 13 12 34 38 27 22 32 -1 21
Output:
1 13 12 21 33 34 38 27 22 32
Status: Accepted
```

All empty input Input with an empty table

```
Input:
5
-1 -1 -1 -1 -1
Output(None):
Status: Accepted
```

All crashed All elements are crashed.

```
Input:
5
6 1 11 16 21
Output:
1 11 16 21 6
Status: Accepted
```

Non-crash All elements are at H[i], none crash happens.

```
Input:
5
10 1 22 18 9
Output:
1 9 10 18 22
Status: Accepted
```

One number Only one number is input.

```
Input:
1
233
Output:
233
Status: Accepted
```

Empty is not -1 In this problem, negative represents empty block, this case is to test whether -2,-3 etc. works.

```
Input:
5
10 -1 -2 -3 -4
Output:
10
Status: Accepted
```

Zero case For non-negative is available, 0 is a special case.

```
Input:
5
0 1 2 3 4
Output:
0 1 2 3 3
Status: Accepted
```

Back around Crashed elements fill up the table and go back to A[0].

```
Input:
10
29 28 39 38 49 48 59 58 18 19
Output:
18 19 29 28 39 38 49 48 59 58
Status: Accepted
```

3.2 Illegal Data

Case 1 Some data missing in answer.

```
Input:
5
6 1 2 3 -1
Output:
1 2 3
Status: Accepted
```

Case 2 Not unique.

```
Input:
5
1 1 2 2 3
Output:
1 2 2 3 1
Status: Accepted
```

3.3 Random Generator 1 (gen1.exe)

Usage:
`gen1 [N] [random seed]`
Input:
None
Output:
1.Out put a set of random input for this problem to stdout.
2.Out put the answer to ans.txt.
Generating method:
Generate 4/5N-N random integers, sort them, and input into a hash table.
Note:
Use this method to generate data, the answer is always in increasing order.

3.4 Random Generator 2 (gen2.exe)

Usage:
`gen2 [N] [random seed]`
Input:
None
Output:
Out put a set of random input for this problem to stdout.
Generating method:
Generate 0-N random integers, and input into a hash table.

Note:

1. None answer is generated, we use this gerator to compare whether two algorithms output the same way.
2. Empty block is always -1.
3. 0 is possible.

3.5 Random Generator 3 (gen3.exe)

Usage:

`gen3 [N] [random seed]`

Input:

None

Output:

Out put a set of random input for this problem to stdout.

Generating method:

Generate 0-N random integers, and input into a hash table.

Note:

1. None answer is generated, we use this generator to compare whether two algorithms output the same way.
2. Empty block is NOT always -1.
3. 0 is possible.

3.6 Tester 1 (test1.exe)

Usage:

Click to run.

Input:

Input file name of exe, for example "algorithm1". Then input N.

Output:

Tester will check your output and answer automatically, if a mistake is found, program will pause, then *in.txt* is the input, *out.txt* is your output, and *ans.txt* is answer.

Generator: gen1

3.7 Tester 2 (test2.exe)

Usage:

Click to run.

Input:

Input two file name of exe, for example "algorithm1" and "algorithm2". Then input N.

Output:

Tester will check two outputs automatically, if difference is found, program will pause, then *in.txt* is the input, *out1.txt* is your first output, and *out2.txt* is your second output.
Generator: gen2

3.8 Tester 3 (test3.exe)

The same as test2, use gen3.

3.9 Testing result

Hand-made data all pass.

Each algorithm were check by all 3 testors at $N = 3, 5, 10, 50, 100, 1000$. Tester run 10min for each case, all pass.

Algorithm 1 and 2 can run at most $N = 1000$ in 1 second, with answer correctly output. Algorithm 3 can run at most $N = 100000$ in 1 second, with answer correctly output.

In a word, all three algorithms are correct and stable enough to solve this problem.

Chapter 4

Appendix: Cpp Code

4.1 algorithm1.cpp

```
#include <bits/stdc++.h>

using namespace std;

typedef pair<int,int> PII;
priority_queue<PII,vector<PII>, greater<PII> > Q;
vector<int> vec;

int n,a[1100],cnt;
bool taken[1100];

void check()
{
  //check the whole array
  for(int i=0;i<n;++i)
  {
    //bad point continue
    if(a[i]<0 || taken[i])continue;
    int j; for(j=a[i]%n;j!=i;j=(j+1)%n)
      // must be taken away not empty at the beginning
      if(a[j]!=-2)break;
    // good point, ready to print
    // make a pair recording number and position
    if(j==i) Q.push(make_pair(a[i],i)),taken[i]=true;
  } return ;
}

int main()
{
  scanf("%d",&n);
  for(int i=0;i<n;++i)
```

```
32:   {
33:     scanf("%d",&a[i]);
34:     // -1 represents empty
35:     if(a[i]>=0)cnt++;
36:     else a[i]=-1;
37:   }
38:   while(cnt--)
39:   {
40:     check();
41:     if(Q.empty()) break;
42:     PII temp=Q.top(); Q.pop();
43:     // save answer
44:     if(temp.first>=0)vec.push_back(temp.first);
45:     a[temp.second]=-2;
46:   }
47:
48:   //output
49:   if(vec.size())printf("%d",vec[0]);
50:   for(int i=1;i<(int)vec.size();++i)
51:     printf(" %d",vec[i]);
52:   printf("\n");
53:   return 0;
54: }
```

4.2 algorithm2.cpp

```
1: #include <bits/stdc++.h>
2:
3: using namespace std;
4:
5: //template of Adjacency list
6: template<const int _n,const int _m>
7: // number of nodes and edges
8: struct Edge
9: {
10:   struct Edge_base { int to,next; }e[_m];
11:   int cnt,p[_n],in[_n];
12:   Edge() { clear(); }
13:   void clear()
14:   {
15:     cnt=0,memset(p,0,sizeof(p));
16:     memset(in,0,sizeof(in));
17:   }
18:   void insert(const int x,const int y)
19:   { e[++cnt].to=y; e[cnt].next=p[x]; p[x]=cnt; in[y]++; }
20:   int start(const int x) { return p[x]; }
21:   Edge_base& operator[](const int x) { return e[x]; }
22: };
```

```
23:
24: int n,a[1100],id[1100];
25: vector<int> vec;
26: Edge<1100,1100000> e;
27:
28: void Topo_sort()
29: {
30:   typedef pair<int,int> PII;
31:   priority_queue<PII,vector<PII>,greater<PII> > Q; // heap
32:   //init, find 0-in-degree-nodes
33:   for(int i=1;i<=n;++i)
34:     if(a[i]>=0 && e.in[i]==0)
35:       Q.push(make_pair(a[i],i));
36:   // start
37:   while(!Q.empty())
38:   {
39:     // each time output a node
40:     PII temp=Q.top(); Q.pop();
41:     for(int i=e.start(temp.second);i;i=e[i].next)
42:       if(--e.in[e[i].to]==0)
43:         Q.push(make_pair(a[e[i].to],e[i].to));
44:     if(temp.first>=0) vec.push_back(temp.first);
45:   } return ;
46: }
47:
48: int main()
49: {
50:   scanf("%d",&n);
51:   for(int i=1;i<=n;++i)
52:     scanf("%d",&a[i]);
53:
54:   for(int i=1;i<=n;++i)
55:   {
56:     if(a[i]<0) continue;
57:     // index+1 for index '0' is hard to deal with
58:     // link [h,i-1] to i
59:     if(i>a[i]%n+1)
60:       for(int j=a[i]%n+1;j<i;++j) e.insert(j,i);
61:     if(i<a[i]%n+1)// go back round
62:     {
63:       //link [h,n-1] to i
64:       for(int j=a[i]%n+1;j<=n;++j) e.insert(j,i);
65:       //link [0,i-1] to i
66:       for(int j=1;j<i;++j) e.insert(j,i);
67:     }
68:   }
69:
70:   Topo_sort();
71:
72:   // output answer
```

```
73:   if(vec.size())printf("%d",vec[0]);
74:   for(int i=1;i<(int)vec.size();++i)
75:     printf(" %d",vec[i]);
76:
77:   printf("\n");
78:
79:   return 0;
80: }
```

4.3 algorithm3.cpp

```
1: #include <bits/stdc++.h>
2:
3: using namespace std;
4:
5: //template of Adjacency list
6: template<const int _n,const int _m>i
7: // number of nodes and edges
8: struct Edge
9: {
10:   struct Edge_base { int to,next,w; }e[_m];
11:   int cnt,p[_n],in[_n];
12:   Edge() { clear(); }
13:   void clear()
14:   {
15:     cnt=0,memset(p,0,sizeof(p));
16:     memset(in,0,sizeof(in));
17:   }
18:   void insert(const int x,const int y)
19:   { e[++cnt].to=y; e[cnt].next=p[x];
20:     p[x]=cnt; e[cnt].w=-1; in[y]++; }
21:   int start(const int x) { return p[x]; }
22:   Edge_base& operator[](const int x) { return e[x]; }
23: };
24:
25: int n,a[110000],id[110000];
26: vector<int> vec;
27: Edge<270000,1700000> e;
28:
29: // build a segment tree
30: void Build(const int l,const int r,const int num)
31:   //range [l,r], index num
32: {
33:   if(l==r)
34:   {
35:     e.insert(num,num>>1);
36:     id[l]=num;
37:     return ;
```

```
38:   }
39:   int mid=(l+r)>>1;
40:   Build(l,mid,num<<1);//build left sub-tree
41:   Build(mid+1,r,num<<1|1);// build right sub-tree
42:   if(e.in[num]) e.insert(num,num>>1);
43:   //connect num to its father
44: }
45:
46: void Link(const int l,const int r,const int num,
47:     // current range [l,r]  index num
48:     const int s,const int t,const int d,const int w)
49:     // required range [s,t]  connect to vertex 'd'
50:     // w is weight of edge,
51:     // is used to mark if an edge is pointing to a leaf node
52: {
53:   if(s<=l && r<=t)
54:   {
55:     e.insert(num,d);
56:     e[e.start(num)].w=w;
57:     return ;
58:   }
59:   int mid=(l+r)>>1;
60:   if(s<=mid) Link(l,mid,num<<1,s,t,d,w);
61:   //if left half contains part of required range
62:   if(t>mid)  Link(mid+1,r,num<<1|1,s,t,d,w);
63:   //if right half contains part of required range
64: }
65:
66: void Topo_sort()
67: {
68:   typedef pair<int,int> PII;
69:   priority_queue<PII,vector<PII>,greater<PII> > Q;// heap
70:   queue<int> QQ;
71:   //init, find 0-in-degree-nodes
72:   for(int i=1;i<=n;++i)
73:     if(a[i]>=0 && e.in[id[i]]==0)
74:       Q.push(make_pair(a[i],id[i]));
75:   // start
76:   while(!Q.empty())
77:   {
78:     // each time output a node
79:     PII temp=Q.top(); Q.pop();
80:     for(int i=e.start(temp.second);i;i=e[i].next)
81:     {
82:       int tto=e[i].to;
83:       if(--e.in[tto]==0)
84:       {
85:         if(e[i].w>=0)Q.push(make_pair(e[i].w,tto));
86:         else QQ.push(tto);
87:       }
```

```
88:     }
89:     // clear non-leaf nodes
90:     while(!QQ.empty())
91:     {
92:       int tt=QQ.front(); QQ.pop();
93:       // for each node tt is connected to
94:       for(int i=e.start(tt);i;i=e[i].next)
95:       {
96:         int tto=e[i].to;
97:         if(--e.in[tto]==0)
98:         {
99:           if(e[i].w>=0)Q.push(make_pair(e[i].w,tto));
100:          else QQ.push(tto);
101:        }
102:      }
103:    }
104:    vec.push_back(temp.first);
105:  } return ;
106: }
107:
108: int main()
109: {
110:   scanf("%d",&n);
111:   for(int i=1;i<=n;++i)
112:     scanf("%d",&a[i]);
113:   Build(1,n,1);
114:   for(int i=1;i<=n;++i)
115:   {
116:     int h=a[i]%n,idd=id[i];
117:     if(a[i]<0) continue;
118:     // index+1 for index '0' is hard to deal with
119:     // link [h,i-1] to i
120:     if(i>h+1)Link(1,n,1,h+1,i-1,idd,a[i]);
121:     else if(i<h+1)// go back round
122:     {
123:       //link [h,n-1] to i
124:       Link(1,n,1,h+1,n,idd,a[i]);
125:       //link [0,i-1] to i
126:       if(i!=1)Link(1,n,1,1,i-1,idd,a[i]);
127:     }
128:   }
129:
130:   Topo_sort();
131:
132:   // output answer
133:   if(vec.size())printf("%d",vec[0]);
134:   for(int i=1;i<(int)vec.size();++i)
135:     printf(" %d",vec[i]);
136:
137:   printf("\n");
```

```
138:
139:   return 0;
140: }
```

4.4 gen1.cpp

```
1: #include <bits/stdc++.h>
2:
3: using namespace std;
4:
5: int h[110000],a[110000];
6:
7:
8: int main(int argv,char ** argc)
9: {
10:   FILE* out=fopen("ans.txt","w");
11:   map<int,int> Map;
12:   assert(argv>=2);
13:   int sd=(argv>=3?atoi(argc[2]):time(0));
14:   mt19937 RND(sd);
15:   int n=atoi(argc[1]);
16:   int m=RND()%(n/5)+(n*4/5);
17:   for(int i=1;i<=m;++i)
18:   {
19:     do{ a[i]=RND()%(n*10)+1; }while(Map[a[i]]);
20:     Map[a[i]]=1;
21:   }
22:   sort(a+1,a+m+1);
23:   for(int i=1;i<=m;++i)
24:   {
25:     int k;
26:     for(k=0;h[(a[i]+k)%n];++k);
27:     h[(a[i]+k)%n]=a[i];
28:   }
29:   printf("%d\n",n);
30:   for(int i=0;i<n;++i)
31:     printf("%d ",h[i]?h[i]:-1);
32:   printf("\n");
33:   for(int i=1;i<=m;++i)
34:     fprintf(out,"%d ",a[i]);
35:   printf("\n");
36:   return 0;
37: }
```

4.5 gen2.cpp

```cpp
#include <bits/stdc++.h>

using namespace std;

int h[110000],a[110000];


int main(int argv,char ** argc)
{
  map<int,int> Map;
  assert(argv>=2);
  int sd=(argv>=3?atoi(argc[2]):time(0));
  mt19937 RND(sd);
  int n=atoi(argc[1]);
  memset(h,0xff,sizeof(h));
  int m=RND()%n+1;
  for(int i=1;i<=m;++i)
  {
    do{ a[i]=RND()%(n*10); }while(Map[a[i]]);
    Map[a[i]]=1;
  }
  for(int i=1;i<=m;++i)
  {
    int k;
    for(k=0;~h[(a[i]+k)%n];++k);
    h[(a[i]+k)%n]=a[i];
  }
  printf("%d\n",n);
  for(int i=0;i<n;++i)
    printf("%d ",~h[i]?h[i]:-1);
  printf("\n");
  return 0;
}
```

4.6 gen3.cpp

```cpp
#include <bits/stdc++.h>

using namespace std;

int h[110000],a[110000];

int main(int argv,char ** argc)
{
  map<int,int> Map;
  assert(argv>=2);
  int n=atoi(argc[1]);
```

```
12:   int sd=(argv>=3?atoi(argc[2]):time(0));
13:   mt19937 RND(sd);
14:   memset(h,0xff,sizeof(h));
15:   int m=RND()%n+1;
16:   for(int i=1;i<=m;++i)
17:   {
18:     do{ a[i]=RND()%(n*10); }while(Map[a[i]]);
19:     Map[a[i]]=1;
20:   }
21:   for(int i=1;i<=m;++i)
22:   {
23:     int k;
24:     for(k=0;~h[(a[i]+k)%n];++k);
25:     h[(a[i]+k)%n]=a[i];
26:   }
27:   printf("%d\n",n);
28:   for(int i=0;i<n;++i)
29:     printf("%d ",~h[i]?h[i]:-(RND()%(n*10)));
30:   printf("\n");
31:   return 0;
32: }
```

4.7 test1.cpp

```
1: #include <bits/stdc++.h>
2:
3: using namespace std;
4:
5: int main()
6: {
7:   char S[100],cmd[100];
8:   int n;
9:   printf("Program:\t");
10:   scanf("%s",S);
11:   printf("N:\t");
12:   scanf("%d",&n);
13:   srand(time(0));
14:
15:   do
16:   {
17:     sprintf(cmd,"gen1 %d %d >in.txt",n,rand());
18:     system(cmd);
19:     sprintf(cmd,"%s <in.txt >out.txt",S);
20:     system(cmd);
21:     sprintf(cmd,"fc /w ans.txt out.txt");
22:   }while(!system(cmd));
23:
24:   system("pause");
```

```
25:
26:   return 0;
27: }
```

4.8 test2.cpp

```
1: #include <bits/stdc++.h>
2:
3: using namespace std;
4:
5: int main()
6: {
7:   char S[100],cmd[100],T[100];
8:   int n;
9:   printf("Program1:\t");
10:  scanf("%s",S);
11:  printf("Program2:\t");
12:  scanf("%s",T);
13:  printf("N:\t");
14:  scanf("%d",&n);
15:  srand(time(0));
16:
17:  do
18:  {
19:    sprintf(cmd,"gen2 %d %d >in.txt",n,rand());
20:    system(cmd);
21:    sprintf(cmd,"%s <in.txt >out.txt",S);
22:    system(cmd);
23:    sprintf(cmd,"%s <in.txt >out1.txt",T);
24:    system(cmd);
25:    sprintf(cmd,"fc /w out.txt out1.txt");
26:  }while(!system(cmd));
27:
28:  system("pause");
29:
30:  return 0;
31: }
```

4.9 test3.cpp

```
1: #include <bits/stdc++.h>
2:
3: using namespace std;
4:
5: int main()
```

```
{
  char S[100],cmd[100],T[100];
  int n;
  printf("Program1:\t");
  scanf("%s",S);
  printf("Program2:\t");
  scanf("%s",T);
  printf("N:\t");
  scanf("%d",&n);
  srand(time(0));

  do
  {
    sprintf(cmd,"gen3 %d %d >in.txt",n,rand());
    system(cmd);
    sprintf(cmd,"%s <in.txt >out.txt",S);
    system(cmd);
    sprintf(cmd,"%s <in.txt >out1.txt",T);
    system(cmd);
    sprintf(cmd,"fc /w out.txt out1.txt");
  }while(!system(cmd));

  system("pause");

  return 0;
}
```

4.10 Statement

We hereby declare that all the work done in this project is of our independent effort as a group.

4.11 Duty Assignments

Programmer: Jianglai Dai
Tester: Jiawei Peng
Report Writer: Shuting Guo