目录

[01-复杂度1 最大子列和问题（20 分） 6](#_Toc498615752)

[输入格式: 6](#_Toc498615753)

[输出格式: 6](#_Toc498615754)

[输入样例: 6](#_Toc498615755)

[输出样例: 6](#_Toc498615756)

[01-复杂度2 Maximum Subsequence Sum（25 分） 6](#_Toc498615757)

[Input Specification: 7](#_Toc498615758)

[Output Specification: 7](#_Toc498615759)

[Sample Input: 7](#_Toc498615760)

[Sample Output: 7](#_Toc498615761)

[02-线性结构1 两个有序链表序列的合并（15 分） 7](#_Toc498615762)

[函数接口定义： 7](#_Toc498615763)

[裁判测试程序样例： 8](#_Toc498615764)

[输入样例： 8](#_Toc498615765)

[输出样例： 9](#_Toc498615766)

[02-线性结构2 一元多项式的乘法与加法运算（20 分） 9](#_Toc498615767)

[输入格式: 9](#_Toc498615768)

[输出格式: 9](#_Toc498615769)

[输入样例: 9](#_Toc498615770)

[输出样例: 9](#_Toc498615771)

[02-线性结构3 Reversing Linked List（25 分） 10](#_Toc498615772)

[Input Specification: 10](#_Toc498615773)

[Output Specification: 10](#_Toc498615774)

[Sample Input: 10](#_Toc498615775)

[Sample Output: 10](#_Toc498615776)

[02-线性结构4 Pop Sequence（25 分） 11](#_Toc498615777)

[Input Specification: 11](#_Toc498615778)

[Output Specification: 11](#_Toc498615779)

[Sample Input: 11](#_Toc498615780)

[Sample Output: 11](#_Toc498615781)

[03-树1 树的同构（25 分） 12](#_Toc498615782)

[输入格式: 12](#_Toc498615783)

[输出格式: 12](#_Toc498615784)

[输入样例1（对应图1）： 12](#_Toc498615785)

[输出样例1: 13](#_Toc498615786)

[输入样例2（对应图2）： 13](#_Toc498615787)

[输出样例2: 14](#_Toc498615788)

[03-树2 List Leaves（25 分） 14](#_Toc498615789)

[Input Specification: 14](#_Toc498615790)

[Output Specification: 14](#_Toc498615791)

[Sample Input: 14](#_Toc498615792)

[Sample Output: 14](#_Toc498615793)

[03-树3 Tree Traversals Again（25 分） 15](#_Toc498615794)

[Input Specification: 15](#_Toc498615795)

[Output Specification: 15](#_Toc498615796)

[Sample Input: 15](#_Toc498615797)

[Sample Output: 16](#_Toc498615798)

[04-树4 是否同一棵二叉搜索树（25 分） 16](#_Toc498615799)

[输入格式: 16](#_Toc498615800)

[输出格式: 16](#_Toc498615801)

[输入样例: 16](#_Toc498615802)

[输出样例: 16](#_Toc498615803)

[04-树5 Root of AVL Tree（25 分） 17](#_Toc498615804)

[Input Specification: 18](#_Toc498615805)

[Output Specification: 18](#_Toc498615806)

[Sample Input 1: 18](#_Toc498615807)

[Sample Output 1: 18](#_Toc498615808)

[Sample Input 2: 18](#_Toc498615809)

[Sample Output 2: 18](#_Toc498615810)

[04-树6 Complete Binary Search Tree（30 分） 19](#_Toc498615811)

[Input Specification: 19](#_Toc498615812)

[Output Specification: 19](#_Toc498615813)

[Sample Input: 19](#_Toc498615814)

[Sample Output: 19](#_Toc498615815)

[04-树7 二叉搜索树的操作集（30 分） 19](#_Toc498615816)

[函数接口定义： 19](#_Toc498615817)

[裁判测试程序样例： 20](#_Toc498615818)

[输入样例： 21](#_Toc498615819)

[输出样例： 22](#_Toc498615820)

[05-树7 堆中的路径（25 分） 22](#_Toc498615821)

[输入格式: 22](#_Toc498615822)

[输出格式: 22](#_Toc498615823)

[输入样例: 22](#_Toc498615824)

[输出样例: 23](#_Toc498615825)

[05-树8 File Transfer（25 分） 23](#_Toc498615826)

[Input Specification: 23](#_Toc498615827)

[Output Specification: 23](#_Toc498615828)

[Sample Input 1: 23](#_Toc498615829)

[Sample Output 1: 24](#_Toc498615830)

[Sample Input 2: 24](#_Toc498615831)

[Sample Output 2: 24](#_Toc498615832)

[05-树9 Huffman Codes（30 分） 24](#_Toc498615833)

[Input Specification: 25](#_Toc498615834)

[Output Specification: 25](#_Toc498615835)

[Sample Input: 25](#_Toc498615836)

[Sample Output: 26](#_Toc498615837)

[06-图1 列出连通集（25 分） 26](#_Toc498615838)

[输入格式: 26](#_Toc498615839)

[输出格式: 26](#_Toc498615840)

[输入样例: 26](#_Toc498615841)

[输出样例: 27](#_Toc498615842)

[06-图2 Saving James Bond - Easy Version（25 分） 27](#_Toc498615843)

[Input Specification: 27](#_Toc498615844)

[Output Specification: 28](#_Toc498615845)

[Sample Input 1: 28](#_Toc498615846)

[Sample Output 1: 28](#_Toc498615847)

[Sample Input 2: 28](#_Toc498615848)

[Sample Output 2: 28](#_Toc498615849)

[06-图3 六度空间（30 分） 28](#_Toc498615850)

[输入格式: 29](#_Toc498615851)

[输出格式: 29](#_Toc498615852)

[输入样例: 29](#_Toc498615853)

[输出样例: 30](#_Toc498615854)

[07-图4 哈利·波特的考试（25 分） 30](#_Toc498615855)

[输入格式: 30](#_Toc498615856)

[输出格式: 30](#_Toc498615857)

[输入样例: 31](#_Toc498615858)

[输出样例: 31](#_Toc498615859)

[07-图5 Saving James Bond - Hard Version（30 分） 31](#_Toc498615860)

[Input Specification: 32](#_Toc498615861)

[Output Specification: 32](#_Toc498615862)

[Sample Input 1: 32](#_Toc498615863)

[Sample Output 1: 32](#_Toc498615864)

[Sample Input 2: 33](#_Toc498615865)

[Sample Output 2: 33](#_Toc498615866)

[07-图6 旅游规划（25 分） 33](#_Toc498615867)

[输入格式: 33](#_Toc498615868)

[输出格式: 33](#_Toc498615869)

[输入样例: 33](#_Toc498615870)

[输出样例: 34](#_Toc498615871)

[08-图7 公路村村通（30 分） 34](#_Toc498615872)

[输入格式: 34](#_Toc498615873)

[输出格式: 34](#_Toc498615874)

[输入样例: 34](#_Toc498615875)

[输出样例: 35](#_Toc498615876)

[08-图8 How Long Does It Take（25 分） 35](#_Toc498615877)

[Input Specification: 35](#_Toc498615878)

[Output Specification: 35](#_Toc498615879)

[Sample Input 1: 35](#_Toc498615880)

[Sample Output 1: 36](#_Toc498615881)

[Sample Input 2: 36](#_Toc498615882)

[Sample Output 2: 36](#_Toc498615883)

[08-图9 关键活动（30 分） 36](#_Toc498615884)

[输入格式: 37](#_Toc498615885)

[输出格式: 37](#_Toc498615886)

[输入样例: 37](#_Toc498615887)

[输出样例: 37](#_Toc498615888)

[09-排序1 排序（25 分） 37](#_Toc498615889)

[输入格式: 38](#_Toc498615890)

[输出格式: 38](#_Toc498615891)

[输入样例: 38](#_Toc498615892)

[输出样例: 38](#_Toc498615893)

[09-排序2 Insert or Merge（25 分） 38](#_Toc498615894)

[Input Specification: 39](#_Toc498615895)

[Output Specification: 39](#_Toc498615896)

[Sample Input 1: 39](#_Toc498615897)

[Sample Output 1: 39](#_Toc498615898)

[Sample Input 2: 39](#_Toc498615899)

[Sample Output 2: 39](#_Toc498615900)

[09-排序3 Insertion or Heap Sort（25 分） 39](#_Toc498615901)

[Input Specification: 40](#_Toc498615902)

[Output Specification: 40](#_Toc498615903)

[Sample Input 1: 40](#_Toc498615904)

[Sample Output 1: 40](#_Toc498615905)

[Sample Input 2: 40](#_Toc498615906)

[Sample Output 2: 40](#_Toc498615907)

[10-排序4 统计工龄（20 分） 41](#_Toc498615908)

[输入格式: 41](#_Toc498615909)

[输出格式: 41](#_Toc498615910)

[输入样例: 41](#_Toc498615911)

[输出样例: 41](#_Toc498615912)

[10-排序5 PAT Judge（25 分） 41](#_Toc498615913)

[Input Specification: 41](#_Toc498615914)

[Output Specification: 42](#_Toc498615915)

[Sample Input: 42](#_Toc498615916)

[Sample Output: 43](#_Toc498615917)

[10-排序6 Sort with Swap(0, i)（25 分） 43](#_Toc498615918)

[Input Specification: 43](#_Toc498615919)

[Output Specification: 43](#_Toc498615920)

[Sample Input: 43](#_Toc498615921)

[Sample Output: 44](#_Toc498615922)

[11-散列1 电话聊天狂人（25 分） 44](#_Toc498615923)

[输入格式: 44](#_Toc498615924)

[输出格式: 44](#_Toc498615925)

[输入样例: 44](#_Toc498615926)

[输出样例: 44](#_Toc498615927)

[11-散列2 Hashing（25 分） 44](#_Toc498615928)

[Input Specification: 45](#_Toc498615929)

[Output Specification: 45](#_Toc498615930)

[Sample Input: 45](#_Toc498615931)

[Sample Output: 45](#_Toc498615932)

# 01-复杂度1 最大子列和问题（20 分）

给定*K*个整数组成的序列{ *N*​1​​, *N*​2​​, ..., *N*​*K*​​ }，“连续子列”被定义为{ *N*​*i*​​, *N*​*i*+1​​, ..., *N*​*j*​​ }，其中 1≤*i*≤*j*≤*K*。“最大子列和”则被定义为所有连续子列元素的和中最大者。例如给定序列{ -2, 11, -4, 13, -5, -2 }，其连续子列{ 11, -4, 13 }有最大的和20。现要求你编写程序，计算给定整数序列的最大子列和。

本题旨在测试各种不同的算法在各种数据情况下的表现。各组测试数据特点如下：

* 数据1：与样例等价，测试基本正确性；
* 数据2：102个随机整数；
* 数据3：103个随机整数；
* 数据4：104个随机整数；
* 数据5：105个随机整数；

输入格式:

输入第1行给出正整数*K* (≤100000)；第2行给出*K*个整数，其间以空格分隔。

输出格式:

在一行中输出最大子列和。如果序列中所有整数皆为负数，则输出0。

输入样例:

6

-2 11 -4 13 -5 -2

输出样例:

20

# 01-复杂度2 Maximum Subsequence Sum（25 分）

Given a sequence of *K* integers { *N*​1​​, *N*​2​​, ..., *N*​*K*​​ }. A continuous subsequence is defined to be { *N*​*i*​​, *N*​*i*+1​​, ..., *N*​*j*​​ } where 1≤*i*≤*j*≤*K*. The Maximum Subsequence is the continuous subsequence which has the largest sum of its elements. For example, given sequence { -2, 11, -4, 13, -5, -2 }, its maximum subsequence is { 11, -4, 13 } with the largest sum being 20.

Now you are supposed to find the largest sum, together with the first and the last numbers of the maximum subsequence.

Input Specification:

Each input file contains one test case. Each case occupies two lines. The first line contains a positive integer *K*(≤10000). The second line contains *K* numbers, separated by a space.

Output Specification:

For each test case, output in one line the largest sum, together with the first and the last numbers of the maximum subsequence. The numbers must be separated by one space, but there must be no extra space at the end of a line. In case that the maximum subsequence is not unique, output the one with the smallest indices *i* and *j* (as shown by the sample case). If all the *K* numbers are negative, then its maximum sum is defined to be 0, and you are supposed to output the first and the last numbers of the whole sequence.

Sample Input:

10

-10 1 2 3 4 -5 -23 3 7 -21

Sample Output:

10 1 4

# 02-线性结构1 两个有序链表序列的合并（15 分）

本题要求实现一个函数，将两个链表表示的递增整数序列合并为一个非递减的整数序列。

函数接口定义：

List Merge( List L1, List L2 );

其中List结构定义如下：

typedef struct Node \*PtrToNode;

struct Node {

ElementType Data; /\* 存储结点数据 \*/

PtrToNode Next; /\* 指向下一个结点的指针 \*/

};

typedef PtrToNode List; /\* 定义单链表类型 \*/

L1和L2是给定的带头结点的单链表，其结点存储的数据是递增有序的；函数Merge要将L1和L2合并为一个非递减的整数序列。应直接使用原序列中的结点，返回归并后的链表头指针。

裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

typedef int ElementType;

typedef struct Node \*PtrToNode;

struct Node {

ElementType Data;

PtrToNode Next;

};

typedef PtrToNode List;

List Read(); /\* 细节在此不表 \*/

void Print( List L ); /\* 细节在此不表；空链表将输出NULL \*/

List Merge( List L1, List L2 );

int main()

{

List L1, L2, L;

L1 = Read();

L2 = Read();

L = Merge(L1, L2);

Print(L);

Print(L1);

Print(L2);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

输入样例：

3

1 3 5

5

2 4 6 8 10

输出样例：

1 2 3 4 5 6 8 10

NULL

NULL

# 02-线性结构2 一元多项式的乘法与加法运算（20 分）

设计函数分别求两个一元多项式的乘积与和。

### 输入格式:

输入分2行，每行分别先给出多项式非零项的个数，再以指数递降方式输入一个多项式非零项系数和指数（绝对值均为不超过1000的整数）。数字间以空格分隔。

### 输出格式:

输出分2行，分别以指数递降方式输出乘积多项式以及和多项式非零项的系数和指数。数字间以空格分隔，但结尾不能有多余空格。零多项式应输出0 0。

### 输入样例:

4 3 4 -5 2 6 1 -2 0

3 5 20 -7 4 3 1

### 输出样例:

15 24 -25 22 30 21 -10 20 -21 8 35 6 -33 5 14 4 -15 3 18 2 -6 1

5 20 -4 4 -5 2 9 1 -2 0

# 02-线性结构3 Reversing Linked List（25 分）

Given a constant *K* and a singly linked list *L*, you are supposed to reverse the links of every *K* elements on *L*. For example, given *L* being 1→2→3→4→5→6, if *K*=3, then you must output 3→2→1→6→5→4; if *K*=4, you must output 4→3→2→1→5→6.

Input Specification:

Each input file contains one test case. For each case, the first line contains the address of the first node, a positive *N* (≤10​5​​) which is the total number of nodes, and a positive *K* (≤*N*) which is the length of the sublist to be reversed. The address of a node is a 5-digit nonnegative integer, and NULL is represented by -1.

Then *N* lines follow, each describes a node in the format:

Address Data Next

where Address is the position of the node, Data is an integer, and Next is the position of the next node.

Output Specification:

For each case, output the resulting ordered linked list. Each node occupies a line, and is printed in the same format as in the input.

Sample Input:

00100 6 4

00000 4 99999

00100 1 12309

68237 6 -1

33218 3 00000

99999 5 68237

12309 2 33218

Sample Output:

00000 4 33218

33218 3 12309

12309 2 00100

00100 1 99999

99999 5 68237

68237 6 -1

# 02-线性结构4 Pop Sequence（25 分）

Given a stack which can keep *M* numbers at most. Push *N* numbers in the order of 1, 2, 3, ..., *N* and pop randomly. You are supposed to tell if a given sequence of numbers is a possible pop sequence of the stack. For example, if *M* is 5 and *N* is 7, we can obtain 1, 2, 3, 4, 5, 6, 7 from the stack, but not 3, 2, 1, 7, 5, 6, 4.

### Input Specification:

Each input file contains one test case. For each case, the first line contains 3 numbers (all no more than 1000): *M* (the maximum capacity of the stack), *N* (the length of push sequence), and *K* (the number of pop sequences to be checked). Then *K* lines follow, each contains a pop sequence of *N* numbers. All the numbers in a line are separated by a space.

### Output Specification:

For each pop sequence, print in one line "YES" if it is indeed a possible pop sequence of the stack, or "NO" if not.

### Sample Input:

5 7 5

1 2 3 4 5 6 7

3 2 1 7 5 6 4

7 6 5 4 3 2 1

5 6 4 3 7 2 1

1 7 6 5 4 3 2

### Sample Output:

YES

NO

NO

YES

NO

# 03-树1 树的同构（25 分）

给定两棵树T1和T2。如果T1可以通过若干次左右孩子互换就变成T2，则我们称两棵树是“同构”的。例如图1给出的两棵树就是同构的，因为我们把其中一棵树的结点A、B、G的左右孩子互换后，就得到另外一棵树。而图2就不是同构的。

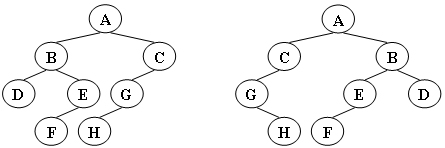


图1

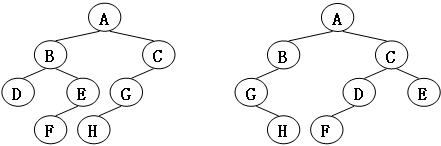


图2

现给定两棵树，请你判断它们是否是同构的。

输入格式:

输入给出2棵二叉树树的信息。对于每棵树，首先在一行中给出一个非负整数*N* (≤10)，即该树的结点数（此时假设结点从0到*N*−1编号）；随后*N*行，第*i*行对应编号第*i*个结点，给出该结点中存储的1个英文大写字母、其左孩子结点的编号、右孩子结点的编号。如果孩子结点为空，则在相应位置上给出“-”。给出的数据间用一个空格分隔。注意：题目保证每个结点中存储的字母是不同的。

输出格式:

如果两棵树是同构的，输出“Yes”，否则输出“No”。

输入样例1（对应图1）：

8

A 1 2

B 3 4

C 5 -

D - -

E 6 -

G 7 -

F - -

H - -

8

G - 4

B 7 6

F - -

A 5 1

H - -

C 0 -

D - -

E 2 -

输出样例1:

Yes

输入样例2（对应图2）：

8

B 5 7

F - -

A 0 3

C 6 -

H - -

D - -

G 4 -

E 1 -

8

D 6 -

B 5 -

E - -

H - -

C 0 2

G - 3

F - -

A 1 4

输出样例2:

No

# 03-树2 List Leaves（25 分）

Given a tree, you are supposed to list all the leaves in the order of top down, and left to right.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer *N* (≤10) which is the total number of nodes in the tree -- and hence the nodes are numbered from 0 to *N*−1. Then *N* lines follow, each corresponds to a node, and gives the indices of the left and right children of the node. If the child does not exist, a "-" will be put at the position. Any pair of children are separated by a space.

Output Specification:

For each test case, print in one line all the leaves' indices in the order of top down, and left to right. There must be exactly one space between any adjacent numbers, and no extra space at the end of the line.

Sample Input:

8

1 -

- -

0 -

2 7

- -

- -

5 -

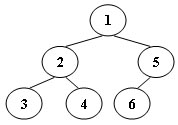
4 6

Sample Output:

4 1 5

# 03-树3 Tree Traversals Again（25 分）

An inorder binary tree traversal can be implemented in a non-recursive way with a stack. For example, suppose that when a 6-node binary tree (with the keys numbered from 1 to 6) is traversed, the stack operations are: push(1); push(2); push(3); pop(); pop(); push(4); pop(); pop(); push(5); push(6); pop(); pop(). Then a unique binary tree (shown in Figure 1) can be generated from this sequence of operations. Your task is to give the postorder traversal sequence of this tree.

  
Figure 1

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer *N* (≤30) which is the total number of nodes in a tree (and hence the nodes are numbered from 1 to *N*). Then 2*N* lines follow, each describes a stack operation in the format: "Push X" where X is the index of the node being pushed onto the stack; or "Pop" meaning to pop one node from the stack.

Output Specification:

For each test case, print the postorder traversal sequence of the corresponding tree in one line. A solution is guaranteed to exist. All the numbers must be separated by exactly one space, and there must be no extra space at the end of the line.

Sample Input:

6

Push 1

Push 2

Push 3

Pop

Pop

Push 4

Pop

Pop

Push 5

Push 6

Pop

Pop

Sample Output:

3 4 2 6 5 1

# 04-树4 是否同一棵二叉搜索树（25 分）

给定一个插入序列就可以唯一确定一棵二叉搜索树。然而，一棵给定的二叉搜索树却可以由多种不同的插入序列得到。例如分别按照序列{2, 1, 3}和{2, 3, 1}插入初始为空的二叉搜索树，都得到一样的结果。于是对于输入的各种插入序列，你需要判断它们是否能生成一样的二叉搜索树。

输入格式:

输入包含若干组测试数据。每组数据的第1行给出两个正整数*N* (≤10)和*L*，分别是每个序列插入元素的个数和需要检查的序列个数。第2行给出*N*个以空格分隔的正整数，作为初始插入序列。最后*L*行，每行给出*N*个插入的元素，属于*L*个需要检查的序列。

简单起见，我们保证每个插入序列都是1到*N*的一个排列。当读到*N*为0时，标志输入结束，这组数据不要处理。

输出格式:

对每一组需要检查的序列，如果其生成的二叉搜索树跟对应的初始序列生成的一样，输出“Yes”，否则输出“No”。

输入样例:

4 2

3 1 4 2

3 4 1 2

3 2 4 1

2 1

2 1

1 2

0

输出样例:

Yes

No

No

**鸣谢青岛大学周强老师补充测试数据！**

# 04-树5 Root of AVL Tree（25 分）

An AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Figures 1-4 illustrate the rotation rules.



Now given a sequence of insertions, you are supposed to tell the root of the resulting AVL tree.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer *N* (≤20) which is the total number of keys to be inserted. Then *N* distinct integer keys are given in the next line. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the root of the resulting AVL tree in one line.

Sample Input 1:

5

88 70 61 96 120

Sample Output 1:

70

Sample Input 2:

7

88 70 61 96 120 90 65

Sample Output 2:

88

# 04-树6 Complete Binary Search Tree（30 分）

A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

 The left subtree of a node contains only nodes with keys less than the node's key.

 The right subtree of a node contains only nodes with keys greater than or equal to the node's key.

 Both the left and right subtrees must also be binary search trees.

A Complete Binary Tree (CBT) is a tree that is completely filled, with the possible exception of the bottom level, which is filled from left to right.

Now given a sequence of distinct non-negative integer keys, a unique BST can be constructed if it is required that the tree must also be a CBT. You are supposed to output the level order traversal sequence of this BST.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer *N* (≤1000). Then *N* distinct non-negative integer keys are given in the next line. All the numbers in a line are separated by a space and are no greater than 2000.

Output Specification:

For each test case, print in one line the level order traversal sequence of the corresponding complete binary search tree. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

Sample Input:

10

1 2 3 4 5 6 7 8 9 0

Sample Output:

6 3 8 1 5 7 9 0 2 4

# 04-树7 二叉搜索树的操作集（30 分）

本题要求实现给定二叉搜索树的5种常用操作。

函数接口定义：

BinTree Insert( BinTree BST, ElementType X );

BinTree Delete( BinTree BST, ElementType X );

Position Find( BinTree BST, ElementType X );

Position FindMin( BinTree BST );

Position FindMax( BinTree BST );

其中BinTree结构定义如下：

typedef struct TNode \*Position;

typedef Position BinTree;

struct TNode{

ElementType Data;

BinTree Left;

BinTree Right;

};

* 函数Insert将X插入二叉搜索树BST并返回结果树的根结点指针；
* 函数Delete将X从二叉搜索树BST中删除，并返回结果树的根结点指针；如果X不在树中，则打印一行Not Found并返回原树的根结点指针；
* 函数Find在二叉搜索树BST中找到X，返回该结点的指针；如果找不到则返回空指针；
* 函数FindMin返回二叉搜索树BST中最小元结点的指针；
* 函数FindMax返回二叉搜索树BST中最大元结点的指针。

裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

typedef int ElementType;

typedef struct TNode \*Position;

typedef Position BinTree;

struct TNode{

ElementType Data;

BinTree Left;

BinTree Right;

};

void PreorderTraversal( BinTree BT ); /\* 先序遍历，由裁判实现，细节不表 \*/

void InorderTraversal( BinTree BT ); /\* 中序遍历，由裁判实现，细节不表 \*/

BinTree Insert( BinTree BST, ElementType X );

BinTree Delete( BinTree BST, ElementType X );

Position Find( BinTree BST, ElementType X );

Position FindMin( BinTree BST );

Position FindMax( BinTree BST );

int main()

{

BinTree BST, MinP, MaxP, Tmp;

ElementType X;

int N, i;

BST = NULL;

scanf("%d", &N);

for ( i=0; i<N; i++ ) {

scanf("%d", &X);

BST = Insert(BST, X);

}

printf("Preorder:"); PreorderTraversal(BST); printf("\n");

MinP = FindMin(BST);

MaxP = FindMax(BST);

scanf("%d", &N);

for( i=0; i<N; i++ ) {

scanf("%d", &X);

Tmp = Find(BST, X);

if (Tmp == NULL) printf("%d is not found\n", X);

else {

printf("%d is found\n", Tmp->Data);

if (Tmp==MinP) printf("%d is the smallest key\n", Tmp->Data);

if (Tmp==MaxP) printf("%d is the largest key\n", Tmp->Data);

}

}

scanf("%d", &N);

for( i=0; i<N; i++ ) {

scanf("%d", &X);

BST = Delete(BST, X);

}

printf("Inorder:"); InorderTraversal(BST); printf("\n");

return 0;

}

/\* 你的代码将被嵌在这里 \*/

输入样例：

10

5 8 6 2 4 1 0 10 9 7

5

6 3 10 0 5

5

5 7 0 10 3

输出样例：

Preorder: 5 2 1 0 4 8 6 7 10 9

6 is found

3 is not found

10 is found

10 is the largest key

0 is found

0 is the smallest key

5 is found

Not Found

Inorder: 1 2 4 6 8 9

# 05-树7 堆中的路径（25 分）

将一系列给定数字插入一个初始为空的小顶堆H[]。随后对任意给定的下标i，打印从H[i]到根结点的路径。

输入格式:

每组测试第1行包含2个正整数*N*和*M*(≤1000)，分别是插入元素的个数、以及需要打印的路径条数。下一行给出区间[-10000, 10000]内的*N*个要被插入一个初始为空的小顶堆的整数。最后一行给出*M*个下标。

输出格式:

对输入中给出的每个下标i，在一行中输出从H[i]到根结点的路径上的数据。数字间以1个空格分隔，行末不得有多余空格。

输入样例:

5 3

46 23 26 24 10

5 4 3

输出样例:

24 23 10

46 23 10

26 10

# 05-树8 File Transfer（25 分）

We have a network of computers and a list of bi-directional connections. Each of these connections allows a file transfer from one computer to another. Is it possible to send a file from any computer on the network to any other?

Input Specification:

Each input file contains one test case. For each test case, the first line contains *N* (2≤*N*≤10​4​​), the total number of computers in a network. Each computer in the network is then represented by a positive integer between 1 and *N*. Then in the following lines, the input is given in the format:

I c1 c2

where I stands for inputting a connection between c1 and c2; or

C c1 c2

where C stands for checking if it is possible to transfer files between c1 and c2; or

S

where S stands for stopping this case.

Output Specification:

For each C case, print in one line the word "yes" or "no" if it is possible or impossible to transfer files between c1 and c2, respectively. At the end of each case, print in one line "The network is connected." if there is a path between any pair of computers; or "There are k components." where k is the number of connected components in this network.

Sample Input 1:

5

C 3 2

I 3 2

C 1 5

I 4 5

I 2 4

C 3 5

S

Sample Output 1:

no

no

yes

There are 2 components.

Sample Input 2:

5

C 3 2

I 3 2

C 1 5

I 4 5

I 2 4

C 3 5

I 1 3

C 1 5

S

Sample Output 2:

no

no

yes

yes

The network is connected.

# 05-树9 Huffman Codes（30 分）

In 1953, David A. Huffman published his paper "A Method for the Construction of Minimum-Redundancy Codes", and hence printed his name in the history of computer science. As a professor who gives the final exam problem on Huffman codes, I am encountering a big problem: the Huffman codes are NOT unique. For example, given a string "aaaxuaxz", we can observe that the frequencies of the characters 'a', 'x', 'u' and 'z' are 4, 2, 1 and 1, respectively. We may either encode the symbols as {'a'=0, 'x'=10, 'u'=110, 'z'=111}, or in another way as {'a'=1, 'x'=01, 'u'=001, 'z'=000}, both compress the string into 14 bits. Another set of code can be given as {'a'=0, 'x'=11, 'u'=100, 'z'=101}, but {'a'=0, 'x'=01, 'u'=011, 'z'=001} is NOT correct since "aaaxuaxz" and "aazuaxax" can both be decoded from the code 00001011001001. The students are submitting all kinds of codes, and I need a computer program to help me determine which ones are correct and which ones are not.

Input Specification:

Each input file contains one test case. For each case, the first line gives an integer *N* (2≤*N*≤63), then followed by a line that contains all the *N* distinct characters and their frequencies in the following format:

c[1] f[1] c[2] f[2] ... c[N] f[N]

where c[i] is a character chosen from {'0' - '9', 'a' - 'z', 'A' - 'Z', '\_'}, and f[i] is the frequency of c[i]and is an integer no more than 1000. The next line gives a positive integer *M* (≤1000), then followed by *M*student submissions. Each student submission consists of *N* lines, each in the format:

c[i] code[i]

where c[i] is the i-th character and code[i] is an non-empty string of no more than 63 '0's and '1's.

Output Specification:

For each test case, print in each line either "Yes" if the student's submission is correct, or "No" if not.

Note: The optimal solution is not necessarily generated by Huffman algorithm. Any prefix code with code length being optimal is considered correct.

Sample Input:

7

A 1 B 1 C 1 D 3 E 3 F 6 G 6

4

A 00000

B 00001

C 0001

D 001

E 01

F 10

G 11

A 01010

B 01011

C 0100

D 011

E 10

F 11

G 00

A 000

B 001

C 010

D 011

E 100

F 101

G 110

A 00000

B 00001

C 0001

D 001

E 00

F 10

G 11

Sample Output:

Yes

Yes

No

No

# 06-图1 列出连通集（25 分）

给定一个有*N*个顶点和*E*条边的无向图，请用DFS和BFS分别列出其所有的连通集。假设顶点从0到*N*−1编号。进行搜索时，假设我们总是从编号最小的顶点出发，按编号递增的顺序访问邻接点。

### 输入格式:

输入第1行给出2个整数*N*(0<*N*≤10)和*E*，分别是图的顶点数和边数。随后*E*行，每行给出一条边的两个端点。每行中的数字之间用1空格分隔。

### 输出格式:

按照"{ *v*​1​​ *v*​2​​ ... *v*​*k*​​ }"的格式，每行输出一个连通集。先输出DFS的结果，再输出BFS的结果。

### 输入样例:

8 6

0 7

0 1

2 0

4 1

2 4

3 5

### 输出样例:

{ 0 1 4 2 7 }

{ 3 5 }

{ 6 }

{ 0 1 2 7 4 }

{ 3 5 }

{ 6 }

# 06-图2 Saving James Bond - Easy Version（25 分）

This time let us consider the situation in the movie "Live and Let Die" in which James Bond, the world's most famous spy, was captured by a group of drug dealers. He was sent to a small piece of land at the center of a lake filled with crocodiles. There he performed the most daring action to escape -- he jumped onto the head of the nearest crocodile! Before the animal realized what was happening, James jumped again onto the next big head... Finally he reached the bank before the last crocodile could bite him (actually the stunt man was caught by the big mouth and barely escaped with his extra thick boot).

Assume that the lake is a 100 by 100 square one. Assume that the center of the lake is at (0,0) and the northeast corner at (50,50). The central island is a disk centered at (0,0) with the diameter of 15. A number of crocodiles are in the lake at various positions. Given the coordinates of each crocodile and the distance that James could jump, you must tell him whether or not he can escape.

Input Specification:

Each input file contains one test case. Each case starts with a line containing two positive integers *N* (≤100), the number of crocodiles, and *D*, the maximum distance that James could jump. Then *N* lines follow, each containing the (*x*,*y*) location of a crocodile. Note that no two crocodiles are staying at the same position.

Output Specification:

For each test case, print in a line "Yes" if James can escape, or "No" if not.

Sample Input 1:

14 20

25 -15

-25 28

8 49

29 15

-35 -2

5 28

27 -29

-8 -28

-20 -35

-25 -20

-13 29

-30 15

-35 40

12 12

Sample Output 1:

Yes

Sample Input 2:

4 13

-12 12

12 12

-12 -12

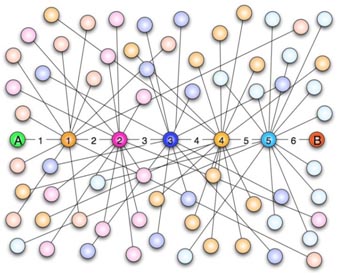
12 -12

Sample Output 2:

No

# 06-图3 六度空间（30 分）

“六度空间”理论又称作“六度分隔（Six Degrees of Separation）”理论。这个理论可以通俗地阐述为：“你和任何一个陌生人之间所间隔的人不会超过六个，也就是说，最多通过五个人你就能够认识任何一个陌生人。”如图1所示。

  
图1 六度空间示意图

“六度空间”理论虽然得到广泛的认同，并且正在得到越来越多的应用。但是数十年来，试图验证这个理论始终是许多社会学家努力追求的目标。然而由于历史的原因，这样的研究具有太大的局限性和困难。随着当代人的联络主要依赖于电话、短信、微信以及因特网上即时通信等工具，能够体现社交网络关系的一手数据已经逐渐使得“六度空间”理论的验证成为可能。

假如给你一个社交网络图，请你对每个节点计算符合“六度空间”理论的结点占结点总数的百分比。

输入格式:

输入第1行给出两个正整数，分别表示社交网络图的结点数*N*（1<*N*≤10​4​​，表示人数）、边数*M*（≤33×*N*，表示社交关系数）。随后的*M*行对应*M*条边，每行给出一对正整数，分别是该条边直接连通的两个结点的编号（节点从1到*N*编号）。

输出格式:

对每个结点输出与该结点距离不超过6的结点数占结点总数的百分比，精确到小数点后2位。每个结节点输出一行，格式为“结点编号:（空格）百分比%”。

输入样例:

10 9

1 2

2 3

3 4

4 5

5 6

6 7

7 8

8 9

9 10

输出样例:

1: 70.00%

2: 80.00%

3: 90.00%

4: 100.00%

5: 100.00%

6: 100.00%

7: 100.00%

8: 90.00%

9: 80.00%

10: 70.00%

# 07-图4 哈利·波特的考试（25 分）

哈利·波特要考试了，他需要你的帮助。这门课学的是用魔咒将一种动物变成另一种动物的本事。例如将猫变成老鼠的魔咒是haha，将老鼠变成鱼的魔咒是hehe等等。反方向变化的魔咒就是简单地将原来的魔咒倒过来念，例如ahah可以将老鼠变成猫。另外，如果想把猫变成鱼，可以通过念一个直接魔咒lalala，也可以将猫变老鼠、老鼠变鱼的魔咒连起来念：hahahehe。

现在哈利·波特的手里有一本教材，里面列出了所有的变形魔咒和能变的动物。老师允许他自己带一只动物去考场，要考察他把这只动物变成任意一只指定动物的本事。于是他来问你：带什么动物去可以让最难变的那种动物（即该动物变为哈利·波特自己带去的动物所需要的魔咒最长）需要的魔咒最短？例如：如果只有猫、鼠、鱼，则显然哈利·波特应该带鼠去，因为鼠变成另外两种动物都只需要念4个字符；而如果带猫去，则至少需要念6个字符才能把猫变成鱼；同理，带鱼去也不是最好的选择。

输入格式:

输入说明：输入第1行给出两个正整数*N* (≤100)和*M*，其中*N*是考试涉及的动物总数，*M*是用于直接变形的魔咒条数。为简单起见，我们将动物按1~*N*编号。随后*M*行，每行给出了3个正整数，分别是两种动物的编号、以及它们之间变形需要的魔咒的长度(≤100)，数字之间用空格分隔。

输出格式:

输出哈利·波特应该带去考场的动物的编号、以及最长的变形魔咒的长度，中间以空格分隔。如果只带1只动物是不可能完成所有变形要求的，则输出0。如果有若干只动物都可以备选，则输出编号最小的那只。

输入样例:

6 11

3 4 70

1 2 1

5 4 50

2 6 50

5 6 60

1 3 70

4 6 60

3 6 80

5 1 100

2 4 60

5 2 80

输出样例:

4 70

# 07-图5 Saving James Bond - Hard Version（30 分）

This time let us consider the situation in the movie "Live and Let Die" in which James Bond, the world's most famous spy, was captured by a group of drug dealers. He was sent to a small piece of land at the center of a lake filled with crocodiles. There he performed the most daring action to escape -- he jumped onto the head of the nearest crocodile! Before the animal realized what was happening, James jumped again onto the next big head... Finally he reached the bank before the last crocodile could bite him (actually the stunt man was caught by the big mouth and barely escaped with his extra thick boot).

Assume that the lake is a 100 by 100 square one. Assume that the center of the lake is at (0,0) and the northeast corner at (50,50). The central island is a disk centered at (0,0) with the diameter of 15. A number of crocodiles are in the lake at various positions. Given the coordinates of each crocodile and the distance that James could jump, you must tell him a shortest path to reach one of the banks. The length of a path is the number of jumps that James has to make.

### Input Specification:

Each input file contains one test case. Each case starts with a line containing two positive integers *N* (≤100), the number of crocodiles, and *D*, the maximum distance that James could jump. Then *N* lines follow, each containing the (*x*,*y*) location of a crocodile. Note that no two crocodiles are staying at the same position.

### Output Specification:

For each test case, if James can escape, output in one line the minimum number of jumps he must make. Then starting from the next line, output the position (*x*,*y*) of each crocodile on the path, each pair in one line, from the island to the bank. If it is impossible for James to escape that way, simply give him 0 as the number of jumps. If there are many shortest paths, just output the one with the minimum first jump, which is guaranteed to be unique.

### Sample Input 1:

17 15

10 -21

10 21

-40 10

30 -50

20 40

35 10

0 -10

-25 22

40 -40

-30 30

-10 22

0 11

25 21

25 10

10 10

10 35

-30 10

### Sample Output 1:

4

0 11

10 21

10 35

### Sample Input 2:

4 13

-12 12

12 12

-12 -12

12 -12

### Sample Output 2:

0

# 07-图6 旅游规划（25 分）

有了一张自驾旅游路线图，你会知道城市间的高速公路长度、以及该公路要收取的过路费。现在需要你写一个程序，帮助前来咨询的游客找一条出发地和目的地之间的最短路径。如果有若干条路径都是最短的，那么需要输出最便宜的一条路径。

输入格式:

输入说明：输入数据的第1行给出4个正整数*N*、*M*、*S*、*D*，其中*N*（2≤*N*≤500）是城市的个数，顺便假设城市的编号为0~(*N*−1)；*M*是高速公路的条数；*S*是出发地的城市编号；*D*是目的地的城市编号。随后的*M*行中，每行给出一条高速公路的信息，分别是：城市1、城市2、高速公路长度、收费额，中间用空格分开，数字均为整数且不超过500。输入保证解的存在。

输出格式:

在一行里输出路径的长度和收费总额，数字间以空格分隔，输出结尾不能有多余空格。

输入样例:

4 5 0 3

0 1 1 20

1 3 2 30

0 3 4 10

0 2 2 20

2 3 1 20

输出样例:

3 40

# 08-图7 公路村村通（30 分）

现有村落间道路的统计数据表中，列出了有可能建设成标准公路的若干条道路的成本，求使每个村落都有公路连通所需要的最低成本。

输入格式:

输入数据包括城镇数目正整数*N*（≤1000）和候选道路数目*M*（≤3*N*）；随后的*M*行对应*M*条道路，每行给出3个正整数，分别是该条道路直接连通的两个城镇的编号以及该道路改建的预算成本。为简单起见，城镇从1到*N*编号。

输出格式:

输出村村通需要的最低成本。如果输入数据不足以保证畅通，则输出−1，表示需要建设更多公路。

输入样例:

6 15

1 2 5

1 3 3

1 4 7

1 5 4

1 6 2

2 3 4

2 4 6

2 5 2

2 6 6

3 4 6

3 5 1

3 6 1

4 5 10

4 6 8

5 6 3

输出样例:

12

# 08-图8 How Long Does It Take（25 分）

Given the relations of all the activities of a project, you are supposed to find the earliest completion time of the project.

### Input Specification:

Each input file contains one test case. Each case starts with a line containing two positive integers *N* (≤100), the number of activity check points (hence it is assumed that the check points are numbered from 0 to *N*−1), and *M*, the number of activities. Then *M* lines follow, each gives the description of an activity. For the i-th activity, three non-negative numbers are given: S[i], E[i], and L[i], where S[i] is the index of the starting check point, E[i] of the ending check point, and L[i] the lasting time of the activity. The numbers in a line are separated by a space.

### Output Specification:

For each test case, if the scheduling is possible, print in a line its earliest completion time; or simply output "Impossible".

### Sample Input 1:

9 12

0 1 6

0 2 4

0 3 5

1 4 1

2 4 1

3 5 2

5 4 0

4 6 9

4 7 7

5 7 4

6 8 2

7 8 4

### Sample Output 1:

18

### Sample Input 2:

4 5

0 1 1

0 2 2

2 1 3

1 3 4

3 2 5

### Sample Output 2:

Impossible

# 08-图9 关键活动（30 分）

假定一个工程项目由一组子任务构成，子任务之间有的可以并行执行，有的必须在完成了其它一些子任务后才能执行。“任务调度”包括一组子任务、以及每个子任务可以执行所依赖的子任务集。

比如完成一个专业的所有课程学习和毕业设计可以看成一个本科生要完成的一项工程，各门课程可以看成是子任务。有些课程可以同时开设，比如英语和C程序设计，它们没有必须先修哪门的约束；有些课程则不可以同时开设，因为它们有先后的依赖关系，比如C程序设计和数据结构两门课，必须先学习前者。

但是需要注意的是，对一组子任务，并不是任意的任务调度都是一个可行的方案。比如方案中存在“子任务A依赖于子任务B，子任务B依赖于子任务C，子任务C又依赖于子任务A”，那么这三个任务哪个都不能先执行，这就是一个不可行的方案。

任务调度问题中，如果还给出了完成每个子任务需要的时间，则我们可以算出完成整个工程需要的最短时间。在这些子任务中，有些任务即使推迟几天完成，也不会影响全局的工期；但是有些任务必须准时完成，否则整个项目的工期就要因此延误，这种任务就叫“关键活动”。

请编写程序判定一个给定的工程项目的任务调度是否可行；如果该调度方案可行，则计算完成整个工程项目需要的最短时间，并输出所有的关键活动。

### 输入格式:

输入第1行给出两个正整数*N*(≤100)和*M*，其中*N*是任务交接点（即衔接相互依赖的两个子任务的节点，例如：若任务2要在任务1完成后才开始，则两任务之间必有一个交接点）的数量。交接点按1~*N*编号，*M*是子任务的数量，依次编号为1~*M*。随后*M*行，每行给出了3个正整数，分别是该任务开始和完成涉及的交接点编号以及该任务所需的时间，整数间用空格分隔。

### 输出格式:

如果任务调度不可行，则输出0；否则第1行输出完成整个工程项目需要的时间，第2行开始输出所有关键活动，每个关键活动占一行，按格式“V->W”输出，其中V和W为该任务开始和完成涉及的交接点编号。关键活动输出的顺序规则是：任务开始的交接点编号小者优先，起点编号相同时，与输入时任务的顺序相反。

### 输入样例:

7 8

1 2 4

1 3 3

2 4 5

3 4 3

4 5 1

4 6 6

5 7 5

6 7 2

### 输出样例:

17

1->2

2->4

4->6

6->7

# 09-排序1 排序（25 分）

给定*N*个（长整型范围内的）整数，要求输出从小到大排序后的结果。

本题旨在测试各种不同的排序算法在各种数据情况下的表现。各组测试数据特点如下：

 数据1：只有1个元素；

 数据2：11个不相同的整数，测试基本正确性；

 数据3：103个随机整数；

 数据4：104个随机整数；

 数据5：105个随机整数；

 数据6：105个顺序整数；

 数据7：105个逆序整数；

 数据8：105个基本有序的整数；

 数据9：105个随机正整数，每个数字不超过1000。

输入格式:

输入第一行给出正整数*N*（≤10​5​​），随后一行给出*N*个（长整型范围内的）整数，其间以空格分隔。

输出格式:

在一行中输出从小到大排序后的结果，数字间以1个空格分隔，行末不得有多余空格。

输入样例:

11

4 981 10 -17 0 -20 29 50 8 43 -5

输出样例:

-20 -17 -5 0 4 8 10 29 43 50 981

# 09-排序2 Insert or Merge（25 分）

According to Wikipedia:

**Insertion sort** iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

**Merge sort** works as follows: Divide the unsorted list into N sublists, each containing 1 element (a list of 1 element is considered sorted). Then repeatedly merge two adjacent sublists to produce new sorted sublists until there is only 1 sublist remaining.

Now given the initial sequence of integers, together with a sequence which is a result of several iterations of some sorting method, can you tell which sorting method we are using?

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer *N* (≤100). Then in the next line, *N* integers are given as the initial sequence. The last line contains the partially sorted sequence of the *N* numbers. It is assumed that the target sequence is always ascending. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in the first line either "Insertion Sort" or "Merge Sort" to indicate the method used to obtain the partial result. Then run this method for one more iteration and output in the second line the resuling sequence. It is guaranteed that the answer is unique for each test case. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

Sample Input 1:

10

3 1 2 8 7 5 9 4 6 0

1 2 3 7 8 5 9 4 6 0

Sample Output 1:

Insertion Sort

1 2 3 5 7 8 9 4 6 0

Sample Input 2:

10

3 1 2 8 7 5 9 4 0 6

1 3 2 8 5 7 4 9 0 6

Sample Output 2:

Merge Sort

1 2 3 8 4 5 7 9 0 6

# 09-排序3 Insertion or Heap Sort（25 分）

According to Wikipedia:

**Insertion sort** iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

**Heap sort** divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. it involves the use of a heap data structure rather than a linear-time search to find the maximum.

Now given the initial sequence of integers, together with a sequence which is a result of several iterations of some sorting method, can you tell which sorting method we are using?

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer *N* (≤100). Then in the next line, *N* integers are given as the initial sequence. The last line contains the partially sorted sequence of the *N* numbers. It is assumed that the target sequence is always ascending. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in the first line either "Insertion Sort" or "Heap Sort" to indicate the method used to obtain the partial result. Then run this method for one more iteration and output in the second line the resuling sequence. It is guaranteed that the answer is unique for each test case. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

Sample Input 1:

10

3 1 2 8 7 5 9 4 6 0

1 2 3 7 8 5 9 4 6 0

Sample Output 1:

Insertion Sort

1 2 3 5 7 8 9 4 6 0

Sample Input 2:

10

3 1 2 8 7 5 9 4 6 0

6 4 5 1 0 3 2 7 8 9

Sample Output 2:

Heap Sort

5 4 3 1 0 2 6 7 8 9

# 10-排序4 统计工龄（20 分）

给定公司*N*名员工的工龄，要求按工龄增序输出每个工龄段有多少员工。

输入格式:

输入首先给出正整数*N*（≤10​5​​），即员工总人数；随后给出*N*个整数，即每个员工的工龄，范围在[0, 50]。

输出格式:

按工龄的递增顺序输出每个工龄的员工个数，格式为：“工龄:人数”。每项占一行。如果人数为0则不输出该项。

输入样例:

8

10 2 0 5 7 2 5 2

输出样例:

0:1

2:3

5:2

7:1

10:1

# 10-排序5 PAT Judge（25 分）

The ranklist of PAT is generated from the status list, which shows the scores of the submissions. This time you are supposed to generate the ranklist for PAT.

Input Specification:

Each input file contains one test case. For each case, the first line contains 3 positive integers, *N* (≤10​4​​), the total number of users, *K* (≤5), the total number of problems, and *M* (≤10​5​​), the total number of submissions. It is then assumed that the user id's are 5-digit numbers from 00001 to *N*, and the problem id's are from 1 to *K*. The next line contains *K* positive integers p[i] (i=1, ..., *K*), where p[i] corresponds to the full mark of the i-th problem. Then *M* lines follow, each gives the information of a submission in the following format:

user\_id problem\_id partial\_score\_obtained

where partial\_score\_obtained is either −1 if the submission cannot even pass the compiler, or is an integer in the range [0, p[problem\_id]]. All the numbers in a line are separated by a space.

Output Specification:

For each test case, you are supposed to output the ranklist in the following format:

rank user\_id total\_score s[1] ... s[K]

where rank is calculated according to the total\_score, and all the users with the same total\_scoreobtain the same rank; and s[i] is the partial score obtained for the i-th problem. If a user has never submitted a solution for a problem, then "-" must be printed at the corresponding position. If a user has submitted several solutions to solve one problem, then the highest score will be counted.

The ranklist must be printed in non-decreasing order of the ranks. For those who have the same rank, users must be sorted in nonincreasing order according to the number of perfectly solved problems. And if there is still a tie, then they must be printed in increasing order of their id's. For those who has never submitted any solution that can pass the compiler, or has never submitted any solution, they must NOT be shown on the ranklist. It is guaranteed that at least one user can be shown on the ranklist.

Sample Input:

7 4 20

20 25 25 30

00002 2 12

00007 4 17

00005 1 19

00007 2 25

00005 1 20

00002 2 2

00005 1 15

00001 1 18

00004 3 25

00002 2 25

00005 3 22

00006 4 -1

00001 2 18

00002 1 20

00004 1 15

00002 4 18

00001 3 4

00001 4 2

00005 2 -1

00004 2 0

Sample Output:

1 00002 63 20 25 - 18

2 00005 42 20 0 22 -

2 00007 42 - 25 - 17

2 00001 42 18 18 4 2

5 00004 40 15 0 25 -

# 10-排序6 Sort with Swap(0, i)（25 分）

Given any permutation of the numbers {0, 1, 2,..., *N*−1}, it is easy to sort them in increasing order. But what if Swap(0, \*) is the ONLY operation that is allowed to use? For example, to sort {4, 0, 2, 1, 3} we may apply the swap operations in the following way:

Swap(0, 1) => {4, 1, 2, 0, 3}

Swap(0, 3) => {4, 1, 2, 3, 0}

Swap(0, 4) => {0, 1, 2, 3, 4}

Now you are asked to find the minimum number of swaps need to sort the given permutation of the first *N*nonnegative integers.

Input Specification:

Each input file contains one test case, which gives a positive *N* (≤10​5​​) followed by a permutation sequence of {0, 1, ..., *N*−1}. All the numbers in a line are separated by a space.

Output Specification:

For each case, simply print in a line the minimum number of swaps need to sort the given permutation.

Sample Input:

10

3 5 7 2 6 4 9 0 8 1

Sample Output:

9

# 11-散列1 电话聊天狂人（25 分）

给定大量手机用户通话记录，找出其中通话次数最多的聊天狂人。

输入格式:

输入首先给出正整数*N*（≤10​5​​），为通话记录条数。随后*N*行，每行给出一条通话记录。简单起见，这里只列出拨出方和接收方的11位数字构成的手机号码，其中以空格分隔。

输出格式:

在一行中给出聊天狂人的手机号码及其通话次数，其间以空格分隔。如果这样的人不唯一，则输出狂人中最小的号码及其通话次数，并且附加给出并列狂人的人数。

输入样例:

4

13005711862 13588625832

13505711862 13088625832

13588625832 18087925832

15005713862 13588625832

输出样例:

13588625832 3

# 11-散列2 Hashing（25 分）

The task of this problem is simple: insert a sequence of distinct positive integers into a hash table, and output the positions of the input numbers. The hash function is defined to be *H*(*key*)=*key*%*TSize* where *TSize*is the maximum size of the hash table. Quadratic probing (with positive increments only) is used to solve the collisions.

Note that the table size is better to be prime. If the maximum size given by the user is not prime, you must re-define the table size to be the smallest prime number which is larger than the size given by the user.

Input Specification:

Each input file contains one test case. For each case, the first line contains two positive numbers: *MSize* (≤10​4​​) and *N* (≤*MSize*) which are the user-defined table size and the number of input numbers, respectively. Then *N* distinct positive integers are given in the next line. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the corresponding positions (index starts from 0) of the input numbers in one line. All the numbers in a line are separated by a space, and there must be no extra space at the end of the line. In case it is impossible to insert the number, print "-" instead.

Sample Input:

4 4

10 6 4 15

Sample Output:

0 1 4 –

# 11-散列3 QQ帐户的申请与登陆（25 分）

实现QQ新帐户申请和老帐户登陆的简化版功能。最大挑战是：据说现在的QQ号码已经有10位数了。

输入格式:

输入首先给出一个正整数*N*（≤10​5​​），随后给出*N*行指令。每行指令的格式为：“命令符（空格）QQ号码（空格）密码”。其中命令符为“N”（代表New）时表示要新申请一个QQ号，后面是新帐户的号码和密码；命令符为“L”（代表Login）时表示是老帐户登陆，后面是登陆信息。QQ号码为一个不超过10位、但大于1000（据说QQ老总的号码是1001）的整数。密码为不小于6位、不超过16位、且不包含空格的字符串。

输出格式:

针对每条指令，给出相应的信息：

1）若新申请帐户成功，则输出“New: OK”；  
2）若新申请的号码已经存在，则输出“ERROR: Exist”；  
3）若老帐户登陆成功，则输出“Login: OK”；  
4）若老帐户QQ号码不存在，则输出“ERROR: Not Exist”；  
5）若老帐户密码错误，则输出“ERROR: Wrong PW”。

输入样例:

5

L 1234567890 myQQ@qq.com

N 1234567890 myQQ@qq.com

N 1234567890 myQQ@qq.com

L 1234567890 myQQ@qq

L 1234567890 myQQ@qq.com

输出样例:

ERROR: Not Exist

New: OK

ERROR: Exist

ERROR: Wrong PW

Login: OK

# 11-散列4 Hashing - Hard Version（30 分）

Given a hash table of size *N*, we can define 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 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.

Input Specification:

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 Specification:

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.

Sample Input:

11

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

Sample Output:

1 13 12 21 33 34 38 27 22 32

# 期末考试 还原二叉树（8 分）

给定一棵二叉树的先序遍历序列和中序遍历序列，要求计算该二叉树的高度。

### 输入格式:

输入首先给出正整数N（≤50），为树中结点总数。下面两行先后给出先序和中序遍历序列，均是长度为N的不包含重复英文字母（区别大小写）的字符串。

### 输出格式:

输出为一个整数，即该二叉树的高度。

### 输入样例:

9

ABDFGHIEC

FDHGIBEAC

### 输出样例:

5