TASK OVERVIEW SHEET / DAY-2

TASK	Score	Double	Depot
Task material directory/Linux	~/score	~/double	~/depot
Task material directory/Win98	C:\IOI\score	C:\IOI\double	C:\IOI\depot
Time limit per test	1 second	-	0.3 seconds
Memory limit	32MB	-	32MB
Maximum compilation time	*30 seconds*	-	*30 seconds*
Maximum source code size	*1MB*	-	*1MB*
Compiler options/C and C++	-O2 –static	-	-O2 –static
Compiler options/Pascal	-So -O2 –XS	-	-So -O2 -XS
Number of Tests	20	10	25
Maximum points per test	5	1 for case 1; 11 for cases 2..10.	4
Maximum total points	100	100	100

Program header comment when using Pascal	{
	PROG: score
	LANG: PASCAL
	}
		-	{
			PROG: depot
			LANG: PASCAL
			}

Program header comment when using C	/* PROG: score LANG: C */	-	/* PROG: depot LANG: C */
Program header comment when using C++	/* PROG: score LANG: C++ */	-	/* PROG: depot LANG: C++ */
Submission is accepted, if:	Game is played according to the rules - win or lose.	The file format is correct.	Score points from test case in Example 2 of task description.

Extra Information for Competition Day 2

General Information for Web Facility

- **You must not Submit or Test program source files exceeding 1 MB.**
- **Program compile time for grading and testing is restricted to 30 seconds.**

Task DOUBLE

Instructions for running `aestoolp.pas`

FreePascal command line

First compile:

```
fpc aestoolp.pas
```

Next execute `aestoolp`

FreePascal IDE (Linux: `xfp`; Windows: `fp`):

```
File-> Open->  File to open: aestoolp.pas  [OK]
Run-> Run->
```

Instructions for running `aestoolc.c`

GNU C command line

First compile:

```
gcc -o aestoolc aestoolc.c aeslibc.o
```

Next execute `aestoolc`

RHIDE (for GNU C, Linux and Windows: `rhide`)

```
File-> Open->  Name: aestoolc.c  [Open]
Options-> Linker Options->  Parameter: aeslibc.o  [OK]
Run-> Run->
```

Score

PROBLEM

Score is a board game for two players who move the same token from position to position on the board. The board has N positions, numbered 1 through N, and a set of arrows. Each arrow goes from one position to another. Each position is owned by one player or the other, whom we call the owner of that position. In addition, each position has a positive value. All values are different. Position 1 is the starting position. Initially, both players have a score 0.

The game is played as follows. We denote the current token position at the beginning of the move by C. At the beginning of the game, C is position 1. A move of the game consists of the following operations:

1. If the value of C is larger than the current score of the owner of C, then the value of C becomes the new score for the owner of C. Otherwise, the score of the owner of C remains the same. The score of the other player does not change in either case.
2. After this, the owner of C chooses one of the arrows out of the current token position and the destination of the arrow becomes the new current token position. Notice that a player may make several consecutive moves.

The game ends after the token is returned to the starting position. The winner is the player with the higher score when the game ends.

The arrows are always arranged so that the following conditions hold:

- It is always possible to choose an arrow out of the current token position.
- Each position P is reachable from the starting position, that is, there is a sequence of arrows from the starting position to P.
- The game is guaranteed to end after a finite number of moves.

Write a program, which plays this game and wins. All the games your program is made to play in evaluation are such that it is possible to win, whether or not you move first. The opponent in evaluation plays optimally, that is, once given a chance, it will win the game and your program will lose.

INPUT AND OUTPUT

Your program reads input from standard input and writes output to standard output. Your program is Player 1 and the opponent is Player 2. When your program is started, it should first read the following input from standard input.

The first line contains one integer: the number of positions N, $1 \le N \le 1000$. The following N lines each contain N integers with information about the arrows. If there is an arrow from position i to position j, then the jth number on the ith line of these N lines is 1, otherwise it is 0.

The next line contains N integers: the owners of the positions. If the position i is owned by Player 1 (you), then the ith integer is 1, otherwise the ith integer is 2.

The next line contains N integers, the values of the positions. If the ith integer is j, then the value of position i is j. For the values j of positions it holds that $1 \le j \le N$ and all values are different.

After this, the game starts with the current token position being 1. Your program should play as follows, and exit when the token returns to position 1:

- If it is your program's turn to move, then your program should write the number of the next position P, $1 \le P \le N$, to standard output
- If it is your program's opponent's turn to move, then your program should read the number of the next position P, $1 \le P \le N$, from standard input.

Consider the following example. The board is represented in Figure 1. The positions marked with a circle belong to Player 1 and the ones marked with a square belong to Player 2. Each position has its value drawn in the square or circle, and the positions number next to the square or circle. A game being played is represented below.

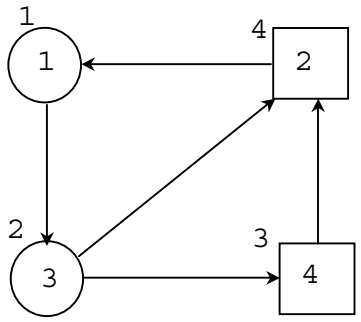

Figure 1.

stdin	stdout	explanation
`4`		*N*
`0 1 0 0`		Information on arrows from position 1
`0 0 1 1`		Information on arrows from position 2
`0 0 0 1`		Information on arrows from position 3
`1 0 0 0`		Information on arrows from position 4
`1 1 2 2`		Owners of positions
`1 3 4 2`		Values of positions
	`2`	Player 1 moves.
	`4`	Player 1 moves.
`1`		Player 2 moves to starting position – game ends.

After the game, Player 1 has score 3 and Player 2 has score 2. Player 1 wins.

PROGRAMMING INSTRUCTIONS

In the examples below, `target` is the integer variable for the position.

If you program in C++ and use iostreams, you should use the following implementation for reading standard input and writing to standard output:

```
cin>>target;
cout<<target<<endl<<flush;
```

If you program in C or C++ and use scanf and printf, you should use the following implementation for reading standard input and writing to standard output:

```
scanf ("%d", &target);
printf("%d\n",target); fflush (stdout);
```

If you program in Pascal, you should use the following implementation of reading standard input and writing to standard output:

```
Readln(target);
Writeln(target);
```

TOOLS

You are given a program (`score2` on Linux, `score2.exe` on Windows). The program reads the description of the game from file `score.in` in the format described on the previous page. The program will write this information to standard output in the same format. This output can be used as an input for your program for test purposes. After that, the program plays with a random strategy, reading your programs moves from standard input and writing its own moves to standard output.

SCORING AND EVALUATION

For a test case, if you win the game, you get full points, otherwise you get 0 points. In the evaluation, your program is first made to play against another program with the time limit 1 second higher than the task time limit. Your programs input and output are recorded. Then, your program is executed a second time with input directed from a file and the official evaluation execution time is recorded. Your program must produce the same output as in the first execution.

Double Crypt

PROBLEM

The Advanced Encryption Standard (AES) involves a new strong encryption algorithm. It works with three *blocks* of 128 bits. Given a message block p (plaintext) and a key block k, the AES encryption function E returns an encrypted block c (ciphertext):

$$c = E(p, k) .$$

The inverse of the AES encryption function E is the decryption function D such that

$$D (E(p, k), k) = p , \quad E (D(c, k), k) = c .$$

In *Double AES*, two independent key blocks k_1 and k_2 are used in succession, first k_1, then k_2:

$$c_2 = E (E(p, k_1), k_2).$$

In this task, an integer s is also given. Only the leftmost $4*s$ bits of all keys are relevant, while the other bits (the rightmost 128 minus $4*s$ bits) are all zero.

You are to recover the encryption key pairs for some messages encrypted by Double AES. You are given both the plaintext p and the corresponding double-encrypted ciphertext c_2, and the structure of the encryption keys as expressed by the integer s.

The AES encryption and decryption algorithms are available in a library.
You must submit the recovered keys, and not a recovery program.

INPUT

You are given ten problem instances in the text files named `double1.in` to `double10.in`. Each input file consists of three lines. The first line contains the integer s, the second line the plaintext block p, and the third line the ciphertext block c_2 obtained from p by Double AES encryption. Both blocks are written as strings of 32 hexadecimal digits ('0'..'9', 'A'..'F'). The library provides a routine to convert strings to blocks. All input files are solvable.

OUTPUT

You are to submit ten output files corresponding to the given input files. Each output file consists of three lines. The first line contains the text
`#FILE double I`
where `I` is the number of the respective input file. The second line contains the key block k_1, and the third line the key block k_2, such that

$$c_2 = E (E(p, k_1), k_2).$$

Both blocks must be written as strings of 32 hexadecimal digits ('0'..'9', 'A'..'F'). The library provides a routine to convert blocks to strings. If there are multiple solutions, you need submit only one of them.

EXAMPLE

As an example we use input file number 0 here.

`double0.in`

```
1
00112233445566778899AABBCCDDEEFF
6323B4A5BC16C479ED6D94F5B58FF0C2
```

A possible output file

```
#FILE double 0
A0000000000000000000000000000000
7000000000000000000000000000000
```

LIBRARY

FreePascal library (Linux: `aeslibp.p, aeslibp.ppu, aeslibp.o;` Windows: `aeslibp.p, aeslibp.ppw, aeslibp.ow`):

```
type
  HexStr = String [ 32 ]; { only '0'..'9', 'A'..'F' }
  Block  = array [ 0..15 ] of Byte; { 128 bits }

procedure HexStrToBlock ( const hs: HexStr; var b: Block );
procedure BlockToHexStr ( const b: Block; var hs: HexStr );
procedure Encrypt ( const p, k: Block; var c: Block );
  { c = E(p,k) }
procedure Decrypt ( const c, k: Block; var p: Block );
  { p = D(c,k) }
```

The program `aestoolp.pas` illustrates how to use the FreePascal library.

GNU C/C++ library (Linux and Windows: `aeslibc.h, aeslibc.o`):

```
typedef char HexStr[33]; /* '0'..'9', 'A'..'F', '\0'-terminated */
typedef unsigned char Block[16];  /* 128 bits */

void hexstr2block ( const HexStr hs, /* out-param */ Block b );
void block2hexstr ( const Block b, /* out-param */ HexStr hs );
void encrypt ( const Block p, const Block k, /* out-param */ Block c );
  /* c = E(p,k) */
void decrypt ( const Block c, const Block k, /* out-param */ Block p );
  /* p = D(c,k) */
```

The program `aestoolc.c` illustrates how to use the GNU C/C++ library.

CONSTRAINTS

For the number s of relevant hexadecimal digits in a key it holds that $1 \le s \le 5$.

HINT: A good program can recover keys in less than 10 seconds for any allowed input file.

Depot

PROBLEM

A Finnish high technology company has a big rectangular depot. The depot has a worker and a manager. The sides of the depot, in the order around it, are called left, top, right and bottom. The depot area is divided into equal-sized squares by dividing the area into rows and columns. The rows are numbered starting from the top with integers 1,2,... and the columns are numbered starting from the left with integers 1,2,...

The depot has containers, which are used to store invaluable technological devices. The containers have distinct identification numbers. Each container occupies one square. The depot is so big, that the number of containers ever to arrive is smaller than the number of rows and smaller than the number of columns. The containers are not removed from the depot, but sometimes a new container arrives. The entry to the depot is at the top left corner.

The worker has arranged the containers around the top left corner of the depot in such a way that he will be able to find them by their identification numbers. He uses the following method.

Suppose that the identification number of the next container to be inserted is k (container k, for short). The worker travels the first row starting from the left and looks for the first container with identification number larger than k. If no such container is found, then container k is placed immediately after the rightmost of the containers previously in the row. If such a container l is found, then container l is replaced by container k, and l is inserted to the following row using the same method. If the worker reaches a row having no containers, the container is placed in the leftmost square of that row.

Suppose that containers `3,4,9,2,5,1` have arrived to the depot in this order. Then the placement of the containers at the depot is as follows.

```
1 4 5
2 9
3
```

The manager comes to the worker and they have the following dialogue:
Manager: Did container 5 arrive before container 4?
Worker: No, that is impossible.
Manager: Oh, so you can tell the arrival order of the containers by their placement.
Worker: Generally not. For instance, the containers now in the depot could have arrived in the order `3,2,1,4,9,5` or in the order `3,2,1,9,4,5` or in one of 14 other orders.

As the manager does not want to show that the worker seems much smarter, he goes away. You are to help the manager and write a program which, given a container placement, computes all possible orders in which they might have arrived.

INPUT

The input file name is `depot.in`. The first line contains one integer R: the number of rows with containers in them. The following R lines contain information about rows 1,...,R starting from the top as follows. First on each of those lines is an integer M: the number of containers in that row. Following that, there are M integers on the line: the identification numbers of the containers in the row starting from the left. All container identification numbers I satisfy $1 \leq I \leq 50$. Let N be the number of containers in the depot, then $1 \leq N \leq 13$.

OUTPUT

The output file name is `depot.out`. The output file contains as many lines as there are possible arrival orders. Each of these lines contains N integers: the identification numbers of the containers in the potential arrival order described by that line. All lines describe an arrival order not described in any other line.

EXAMPLE INPUTS AND OUTPUTS

Example 1: depot.in

```
3
3 1 4 5
2 2 9
1 3
```

depot.out

```
3 2 1 4 9 5
3 2 1 9 4 5
3 4 2 1 9 5
3 2 4 1 9 5
3 2 9 1 4 5
3 9 2 1 4 5
3 4 2 9 1 5
3 4 9 2 1 5
3 2 4 9 1 5
3 2 9 4 1 5
3 9 2 4 1 5
3 4 2 9 5 1
3 4 9 2 5 1
3 2 4 9 5 1
3 2 9 4 5 1
3 9 2 4 5 1
```

Example 2: depot.in

```
2
2 1 2
1 3
```

depot.out

```
3 1 2
1 3 2
```

SCORING

If the output file contains impossible orders or no orders at all, your score is 0 for that test case. Otherwise the score for a test case is computed as follows. If the output file contains all possible orders exactly once, your score is 4. If the output file contains at least half of the possible orders and each of them exactly once, your score is 2. If the output file contains less than half of the possible orders or some of them appear more than once, your score is 1.