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.

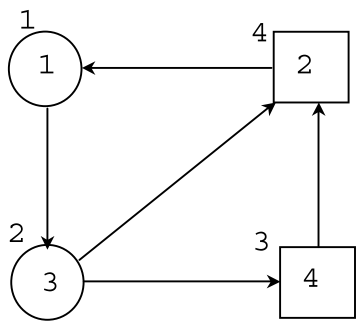

Figure 1.

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.