National Olympiad in Informatics

NOI 2002 Problem Set

Here go the instructions.

Good Luck !

Task 1: ID

The **N**ational **I**dentity **C**ard number of the city state of **E**ropagnis, NICE, consists of seven digits and a letter appended behind. This letter is calculated from the digits using the Modulo Eleven method.

The steps involved in the computation are as follows:

1. Multiply each digit in the NICE number by its weight. The weight of the first digit is 2, the weight of the second is 7, the third is 6, the fourth is 5, the fifth is 4, the sixth is 3, and the seventh is 2.
2. Add all resulting products.
3. Find the remainder of dividing the sum by 11.
4. Map the remainder to a capital letter as follows: 0=J, 1=A, 2=B, 3=C, 4=D, 5=E, 6=F, 7=G, 8=H, 9=I, 10=Z

Example: Given the digits "6830907", the sum is $2 \times 6 + 7 \times 8 + 6 \times 3 + 5 \times 0 + 4 \times 9 + 3 \times 0 + 2 \times 7 = 12 + 56 + 18 + 0 + 36 + 0 + 14 = 136$. The number 136 has a remainder of 4, when divided by 11, and 4 is mapped to the letter "D". So, the actual NICE should be "6830907D".

Write a program to read the seven digits of the NICE, and output the letter that needs to be appended.

Input Format

The input file `ID.IN` consists of the seven digits of the NICE. In our example, the input file would contain

```
6830907
```

Output Format

The output file `ID.OUT` contains a single capital character, which is the letter to be appended to form the correct NICE. The output file for our example will look as follows.

```
D
```

Task 2: CARPET

Given a rectangular configuration representing a floor plan, where '#' indicates a pillar and '.' a clear space:

```
...#...#..
..##.#....
......#.#.
.#........
.........#
.....#.#..
```

What is the area (number of dots) of the largest rectangular carpet whose edges are parallel to the boundary of the floor that can be laid on this piece of land? For the example above, the largest carpet is represented with '*' symbols:

```
...#...#..
..##.#....
......#.#.
.#*******.
..*******#
.....#.#..
```

The area of this carpet is 14.

The configuration may be given as a list of positions of pillars (in no particular order), for example:

```
6 10   ← rows and columns of the floor
11     ← number of pillars
1 4    ← position of a pillar
1 8    ← position of another pillar
⋮
6 8    ← position of last pillar
```

The rows are numbered from top to bottom starting from row 1, and the columns are numbered from left to right starting from column 1.

You can assume that there are at most 20 rows and at most 40 columns.

Input Format

The input file `CARPET.IN` consists of the following lines:

1. The first line contains 2 integers indicating the number of rows and number of columns of the floor plan, respectively.
2. The second line consists of a positive integer n indicating the number of pillars.
3. Each of the subsequent n lines consists of the position of a pillar, represented as 2 integers separated by a space. The first integer represents the row number and the second integer represents the column number.

Thus the input file for the above example will look as follows.

```
6 10
11
1 4
1 8
```

```
2 6
5 10
4 2
2 4
2 3
6 6
3 7
3 9
6 8
```

Output Format

The output file `CARPET.OUT` contains a single integer value, which is the area of the largest carpet that can be laid on the floor.

The output file for our example will look as follows.

```
14
```

Task 3: CELLULAR

A square field is divided up into $n \times n$ cells. Each cell can only take one of two states: 0 or 1. At regular intervals, called generations, all cells update their state simultaneously, depending on the state that they and their neighbors had in the previous generation.

An interior cell has four neighbors, namely the cells above it and below it, and the cells to its right and to its left. Corner cells have only two neighbors. Other cells at the edge of the field have three neighbors.

One possible rule to update a cell is to look at the sum of the states that it and its neighbors had in the previous generation. This sum has a value in the range 0 through 5. In this case, the update rule for the cells can be encoded in 6 bits. For example, an update rule 0 0 1 0 0 1 means that the new state of each cell is

- 0 if the sum was 5,
- 0 if the sum was 4,
- 1 if the sum was 3,
- 0 if the sum was 2,
- 0 if the sum was 1, and
- 1 if the sum was 0,

where "the sum" means the sum of the old states of the cell and of all its neighbors. We can use binary code for uniquely identifying the rule. A 6-bit binary number $a\ b\ c\ d\ e\ f$ corresponds to the decimal value $(a \times 32) + (b \times 16) + (c \times 8) + (d \times 4) + (e \times 2) + f$. The above rule would have the binary number 001001, which corresponds to the decimal value 9.

For example, if n is 4 we could have the starting state

```
1111
1111
1111
1111
```

With rule 9, we get after 1 generation the states

```
1001
0000
0000
1001
```

and after 2 generations

```
0000
0110
0110
0000
```

Given a size n, a number of generations g, a starting state s and an ending state e, the goal is to find the rule with smallest decimal value that generates the ending state e from a starting state s after exactly g generations. The output required is the decimal value of the rule. If there is no such rule, the output should be -1.

You can assume that n is not bigger than 30 and g is not bigger than 50.

Input Format

The input file `CELLULAR.IN` consists of the following lines:

1. The first line contains two positive integers indicating the size n and number of generations g.
2. Each of the next n lines contains n digits which are 0 or 1. These n lines indicate the starting state s.
3. The next line is empty.
4. Each of the following n lines contains n digits which are 0 or 1. These n lines indicate the ending state e.

Thus the input file for the above example will look as follows.

```
4 2
1111
1111
1111
1111

0000
0110
0110
0000
```

Output Format

The output file `CELLULAR.OUT` contains a single integer value, which is the smallest decimal value of a rule that generates e from s after exactly g generations. The output file for the example above will contain:

```
9
```

Task 4: PROGRESS

An arithmetic progression is an ascending sequence a of n numbers $a_1 < a_2 < \ldots < a_n$ such that the difference of two consecutive elements is always the same. Example: The sequence $11 < 21 < 31 < 41 < 51$ is an arithmetic progression. A subsequence of an ascending sequence a of n numbers is a sequence b of m numbers, where $m \leq n$, such that every element of b occurs in a. Example: The sequences $21 < 41 < 51$, $11 < 41$ and $11 < 21 < 31 < 41 < 51$ are three subsequences of the sequence $11 < 21 < 31 < 41 < 51$.

Given is an ascending sequence c of k numbers $c_1 < c_2 < \ldots < c_k$, the task is to find the length of a longest arithmetic progression that is a subsequence of c. Note that there may be more than one such longest arithmetic progression, but the length is unique.

Example: Let c be the sequence $1 < 2 < 4 < 5 < 7 < 8 < 9 < 11 < 13 < 14 < 15 < 16$. There are many arithmetic progressions that are subsequences of c, such as $2 < 4$, $2 < 8 < 14$, and $13 < 14 < 15 < 16$. The longest arithmetic progression that is a subsequence of c is $5 < 7 < 9 < 11 < 13 < 15$, and therefore the answer is 6.

You can assume that the length of the sequence, k, is not smaller than 10 and not bigger than 500, and that the elements of the sequence are positive number smaller than 100000.

Input Format

The input file `PROGRESS.IN` consists of the following lines:

1. The first line contains an integer indicating the number k of elements of c.
2. The second line consists of the ascending sequence c, where the elements are separated by spaces.

In our example, the input file consists of:

```
12
1 2 4 5 7 8 9 11 13 14 15 16
```

Output Format

The output file `PROGRESS.OUT` contains a single positive integer that indicates the length of the longest arithmetic progression that is a subsequence of c. In our example, the output file contains

```
6
```

Task 5: ERP

The city of Eropagnis has just introduced a new electronic road pricing system. Every car has a device that can detect when you are turning left, turning right, moving straight ahead or making a u-turn and will charge you accordingly. The road transport authority has decided to charge \$1 for a left turn and \$5 for a right turn. Moving straight ahead is free while u-turns are forbidden except at ends of roads when it is no longer possible to move forward, turn left or turn right. Making such u-turns cost \$10 each time.

You decided to design and market a road guidance system that will give the cheapest route between any two points in the city. Luckily for you, all roads in Eropagnis go in either the North, South, East or West direction.

Example 1: In the following figure, a '`#`' symbol indicates a road segment while a '`.`' symbol indicates a non-road segment. Road segments can be traversed in both directions. The symbol '`E`' indicates the starting point with the car facing East while the symbol '`F`' indicates the finish point.

```
..........
....#####..
....#...#..
....#...#..
.#E######..
...#......
.##F#.....
..........
```

The cheapest route costing \$8 is to move forward, make 3 left turns, followed by a right turn to `F`. It is also possible to move forward and then make two right turns to reach `F` but this route costs \$10.

Example 2: In the following figure, the symbol '`S`' indicates the starting point with the car facing South. The cheapest route is to immediately turn left, make the first left turn, followed by the first right turn, costing \$7.

```
.....................
.#######.............
.#.....#.....#.......
.###...#.....#.....#.
...#...#.....#.....#.
.###...#.....#.....#.
.#.....#.....#.....#.
.############F#####..
.......#.........#...
.......#.........#...
...#...#...#####..#..
...#...#...#.#.#..#..
..#S########.#.#..#..
...#.....#.###..#....
...#.....#.....#.....
...........########..
.....................
```

The height of the map is at least 4 and at most 30. The width of the map is at least 4 and at most 30. There is exactly one starting point and one finish point. There is a route from the starting point to the finish point. There is always a frame of '`.`' surrounding the map so it is not possible to go outside the map boundary that is given.

Input Format

The input file `ERP.IN` consists of the following lines:

1. The first line contains two positive integers indicating the height h and the width w of the map.
2. Each of the following h lines contains w characters. The characters consist of either
   - '`.`' for non-road portions of the map, or
   - '`#`' for road portions of the map, or
   - '`E`' for the starting point with the car facing East, or
   - '`W`' for the starting point with the car facing West, or
   - '`N`' for the starting point with the car facing North, or
   - '`S`' for the starting point with the car facing South, or
   - '`F`' for the finish point.

Exactly one of the characters in the map is 'E', 'W', 'N' or 'S'.

For example, the input file for example 1 will look like:

```
8 11
...........
....#####..
....#...#..
....#...#..
.#E######..
....#......
.##F#......
...........
```

Output Format

The output file ERP.OUT will contain a single number which is the cost of the cheapest route from the starting point to the finish point. For example the output file for Example 1 will contain

```
8
```

Task 6: LEXICAL

A permutation of a set is a sequence in which every element of the set occurs exactly once. For example, the sequence 3201 is a permutation of the set $\{0, 1, 2, 3\}$, where the number 3 appears first, the number 2 appears second, the number 0 appears third and the number 1 appears last in the sequence. We can order the permutations of a set in a "lexicon" by looking at the first position where the permutations are different. For example, the permutation 3201 appears before 3210 in the lexicon, because at the first position where the permutations are different, the first permutation has a 0 whereas the second permutation has a bigger number, namely 1.

Given an integer n $(1 < n \leq 13)$ and a permutation of the set $\{0, 1, 2, ..., n-1\}$, determine the position of the permutation in the lexicon.

Example: For $n = 4$, the lexicon has 24 entries and looks like this:

0123, 0132, 0213, 0231, 0312, 0321, 1023, 1032, 1203, ..., 3201, 3210.

The permutation 3 2 0 1 appears at position 23.

The task is to determine at which position a given permutation appears in the lexicon.

Hint: If n is close to 13, it will be too slow to construct the entire lexicon, because the size of the lexicon is $1 \times 2 \times 3 \times \cdots \times n$.

Input Format

The input file LEXICAL.IN consists of the following two lines:

1. The first line contains an integer indicating the number n.
2. The second line consists of a permutation,
   namely a sequence of numbers from $\{0, 1, 2, ..., n-1\}$ separated by a space.

Thus the input file for the above example will look as follows:

```
4
3 2 0 1
```

Output Format

The output file LEXICAL.OUT contains a single integer value, which is the position of the permutation in the lexicon. The output file for our example consists of

```
23
```

More Examples

Example 1:

Input:

```
5
0 1 2 4 3
```

Output:

```
2
```

Example 2:

Input:

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

Output:

```
362141
```

Example 3:

Input:

```
13
4 0 6 7 2 12 11 8 10 3 1 9 5
```

Output:

```
1932053504
```