

ACM Regional Programming Contest 2000.

Problem #2. Convoy.

Program: PROGRAM2.EXE, PROGRAM2.CLASS

Source Program: PROGRAM2.CPP, PROGRAM2.C

PROGRAM2.JAVA, PROGRAM2.PAS

Input file: PROGRAM2.IN Output file: PROGRAM2.OUT

Problem Description

A convoy of vehicles has lined up on a single lane and one-way street in front of a single lane and one-way bridge over a river. Note that since the street is single lane, no vehicle can overtake any other. The bridge can sustain a given maximum load. To control the traffic on the bridge, operators are stationed on either end of the bridge. The convoy of vehicles is to be divided into groups, such that all the vehicles in any group can cross the bridge together. When a group reaches the other side, the operator on that side of the bridge uses a telephone to inform the operator on this side that the next group can start its journey over the bridge. The weight of each vehicle is known. The sum of the weights of the vehicles in any group cannot exceed the maximum load sustainable by the bridge. Associated with each vehicle is the maximum speed with which it can travel over the bridge. The time taken by a group of vehicles is calculated as the time taken by the slowest vehicle in the group to cross the bridge. The problem is to find the minimum amount of time in which the entire convoy can cross the bridge.

Input

The first line of input contains three positive integers (separated by blanks): the first one represents the maximum load that the bridge can sustain (in tonnes); the second one represents the length of the bridge (in kms); and the third one is the number of vehicles (n) in the convoy. Each of the next n lines of input contains a pair of positive integers, w s (separated by blanks), where w is the weight of the vehicle (in tonnes) and s is the maximum speed (in kmph) with which this vehicle can travel over the bridge. The weights and speeds of the vehicles are specified in the same order as the order in which the vehicles are queued up. You can assume that $n < 1000$.

Output

The output of the program should be a single real number specifying the minimum time in minutes in which the convoy can cross the bridge.

INPUT FILE

```
100 5 10
40 25
50 20
50 20
70 10
12 50
9 70
49 30
38 25
27 50
19 70
```

OUTPUT FILE

```
78.0
```

ACM Regional Programming Contest 2000.

Problem #3. Digital Lab

Program: PROGRAM3.EXE, PROGRAM3.CLASS

Source Program: PROGRAM3.CPP, PROGRAM3.C

PROGRAM3.JAVA, PROGRAM3.PAS

Input file: PROGRAM3.IN Output file: PROGRAM3.OUT

Problem Description

Assume that you work for the Digital Processing Lab. They ask you to write a program with an input binary matrix A, which contains the pattern to search on other binary matrix B. The input file include the size and elements for both A and B.

The recognition process consists in scanning row by row (horizontal scanning) the matrix B, when a pattern is located on B you must mark this pattern.

To mark a located pattern change 1 to 2 and 0 to * on B.

The output file of your program will be the matrix B with the located patterns marked.

Input

The first line of the input contains the size of A, next lines contains the matrix A row by row, next line contains the size of B and next lines contains the matrix B row by row.

Output

The output is the matrix B with the located patterns marked.

INPUT FILE

```
2 2
1 0
1 1
5 5
1 1 0 0 0
0 1 1 0 0
1 0 0 1 0
1 1 1 1 0
0 0 1 1 1
```

Note: The input file contains the size of the matrix A, the matrix A, the size of the matrix B and the matrix B.

OUTPUT FILE

```
1 2 * 0 0
0 2 2 0 0
2 * 0 1 0
2 2 1 2 *
0 0 1 2 2
```

INPUT FILE

```
1 1
```

1
55
11000
01100
10010
11110
00111

OUTPUT FILE

22000
02200
20020
22220
00222

INPUT FILE

11
0
55
11000
01100
10010
11110
00111

OUTPUT FILE

11***
*11**
1**1*
1111*
**111

INPUT FILE

26
100101
111010
55
11000
01100
10010
11110
00111

OUTPUT FILE

11000
01100
10010
11110
00111

ACM Regional Programming Contest 2000.

Problem #4. Generating permutations.

Program: PROGRAM4.EXE, PROGRAM4.CLASS

Source Program: PROGRAM4.CPP, PROGRAM4.C

PROGRAM4.JAVA, PROGRAM4.PAS

Input file: PROGRAM4.IN Output file: PROGRAM4.OUT

Problem Description

Consider a set $D = \{1, 2, 3, \dots, n\}$.

A **permutation** of the elements of D is an ordered array $\mathbf{a} = a_1 a_2 a_3 \dots a_n$ of all the distinct elements of D .

The number of permutations of D is $n!$.

Sample #1:

Let $D = \{1, 2, 3\}$, then #Permutations = $3! = 6$ and

Permutations = { 123, 132, 213, 231, 312, 321 }

Sample #2:

Let $D = \{1, 2, 3, 4\}$, then #Permutations = $4! = 24$ and

Permutations = { 1234, 1243, 1324, 1342, 1423, 1432, 2134, 2143,
2314, 2341, 2413, 2431, 3124, 3142, 3214, 3241,
3412, 3421, 4123, 4132, 4213, 4231, 4312, 4321 }

Lexicographical order

Let $\mathbf{a} = a_1 a_2 a_3 \dots a_n$ and $\mathbf{b} = b_1 b_2 b_3 \dots b_n$ permutations of D , then $\mathbf{a} < \mathbf{b}$ if only if

for some m with $1 \leq m < n$ and $a_1 = b_1$ $a_2 = b_2$ $a_3 = b_3$ $a_{m-1} = b_{m-1}$ and $a_m < b_m$

Check for the samples that the permutations are in lexicographical order !!!.

Generating Permutations

If the permutation \mathbf{b} is the lexicographical successor of the permutation \mathbf{a} , then we can get \mathbf{b} from \mathbf{a} using the next procedure:

1. Look for the greater m that $a_m < a_{m+1}$
2. Make $b_1 = a_1$, $b_2 = a_2, \dots, b_{m-1} = a_{m-1}$
3. Make b_m equal to the minor value of $a_{m+1}, a_{m+2}, \dots, a_n$, that is greater than a_m
4. Make $b_{m+1} < b_{m+2} < \dots < b_n$

Make a program that accept in an input file the number of elements n ($1 \leq n \leq 6$) of D and generate in an output file all the permutations of D lexicographically ordered.

Input

The input is the number of elements n ($1 \leq n \leq 6$) of the set D . There might be several runs in the input file. your program should stop when $n=0$.

Output

The output will be the name of the run, the value of n , follow by the list of all the permutations of D lexicographically ordered. There should be a blank line between runs.

INPUT FILE

1
2
4
0

OUTPUT FILE

Run 1 n=1

1

Run 2 n=2

12

21

Run 3 n=4

1 2 3 4

1 2 4 3

1 3 2 4

1 3 4 2

1 4 2 3

1 4 3 2

2 1 3 4

2 1 4 3

2 3 1 4

2 3 4 1

2 4 1 3

2 4 3 1

3 1 2 4

3 1 4 2

3 2 1 4

3 2 4 1

3 4 1 2

3 4 2 1

4 1 2 3

4 1 3 2

4 2 1 3

4 2 3 1

4 3 1 2

4 3 2 1

ACM Regional Programming Contest 2000.

Problem #5. Island's Vegetation.

Program: PROGRAM5.EXE, PROGRAM5.CLASS

Source Program: PROGRAM5.CPP, PROGRAM5.C

PROGRAM5.JAVA, PROGRAM5.PAS

Input file: PROGRAM5.IN Output file: PROGRAM5.OUT

Problem Description

An aerial photograph represents a square portion of an island's vegetation as a positive integer matrix. Each number in the matrix corresponds to a square subsection of the total area and contains relevant information about the plants that inhabit that particular place. For example, plant species that are unique to the island are represented with prime numbers, while species that can be found elsewhere are represented with non-prime numbers.

The Botanical Institute of the island wants to divide the area into regions that have unique vegetation, and those that do not. To accomplish this task, the Institute has requested you to analyze the supplied information and determine the number of different regions that exist and their sizes. Two subsections belong to the same region if they lie contiguously on the same row or column, and if they both are either prime or non-prime.

Input

The input to this problem is given out as sets of square areas. Each set contains the size of the area to analyze in a line by itself and is followed by the corresponding integer matrix, with each row occupying one line. A non-positive integer for the area size indicates that no more sets should be analyzed. The size of the matrix will not exceed 100 units per side and integers will not be larger than 100,000,000.

Output

For each set write the set number on the first line. On the next line output the number of regions of unique vegetation followed by the number of cells sorted in increasing order, that corresponds to each region. On the third line write the number of regions with non-unique vegetation. Output should be formatted as in the sample and an empty line must separate output from different sets.

INPUT FILE

```
3
2 4 9
17 6 37
29 8 11
4
2 3 12 15
5 7 21 33
4 6 11 17
8 9 13 29
-1
```

OUTPUT FILE

```
Area 1:
2 unique vegetation regions: 2 3
1 non-unique vegetation regions

Area 2:
2 unique vegetation regions: 4 4
2 non-unique vegetation regions
```

ACM Regional Programming Contest 2000.

Problem #6. Treasure Hunt.

Program: PROGRAM6.EXE, PROGRAM6.CLASS

Source Program: PROGRAM6.CPP, PROGRAM6.C

PROGRAM6.JAVA, PROGRAM6.PAS

Input file: PROGRAM6.IN Output file: PROGRAM6.OUT

Problem Description

In this problem you will be given a map of a rectangular maze with square blocks. From each block you can move in four directions (N, E, W, S) and you lose some energy for every walk from one block to an adjacent one. Some blocks of the maze are really blocked - that is you cannot move to those blocks. Some other blocks contain some treasures that you will have to collect. Each treasure has a particular pickup cost and carrying cost associated with it. The pickup cost is the energy required to pick up the treasure from the floor and the carrying cost is the energy required to carry the treasure from one block to an adjacent one.

Now given a starting and ending location in the maze you will have to plan a single walk from the starting location to collect and carry all the treasures to the ending location at the expense of minimum energy.

Input

The first line of the input contains two integers R and C (each ≤ 20) describing the dimensions of the maze. Then follows R lines of C characters each representing the map of the maze. Each character corresponds to a square block and represents its property ('.' : an empty block, '#' : a blocked block, '*' : a block containing a treasure, 'S' : the starting block, 'T' : the ending block).

The next line contains an integer representing the energy required in calories for a walk from a block to an adjacent one.

The next line contains pairs of integers (P_i, C_i) representing the pickup and carrying cost in calories for the treasures given in the map from top to bottom and for the same row from left to right. There will be at most 10 treasures in the maze.

The input may contain multiple test cases and ends with two zeros for R and C .

Output

For each test case first output the hunt number. In the next line print the minimum energy required for the hunt. The third line will contain the description of the hunt as a sequence of characters containing 'N', 'E', 'W', 'S' and 'P'. 'N', 'E', 'W' and 'S' represent a walk to the north, east, west and south respectively and 'P' means that the treasure is picked up from the current location. If the hunt is impossible just output the sentence 'The hunt is impossible.' in a line by itself. Each test case must be followed by a blank line.

INPUT FILE

5 8

#.....T

..#*..#.

..#####

...*...#

####S.#*

5

10 50 50 100 30 80

10 10

#.....*

..#*..#..

..#####..

.....#..

####S..##.

.*.#...#..

.....#..

.##.#...#

.*...#.#

...*...#.T

10

100 400 20 50 150 250 30 70 4 5

0 0

OUTPUT FILE

Hunt #1

The hunt is impossible.

Hunt #2

Minimum energy required = 17539 cal

NWWWNNEESPWWSSSEESSWSSSESPWWNPWNNENPESEEESEENENNNNNN

PSSSSWSSSE