Problem 1

348 - Optimal Array Multiplication Sequence

Time limit: 3.000 seconds

Given two arrays *A* and *B*, we can determine the array *C* = *A B* using the standard definition of matrix multiplication:

$$C_{i,j} = \sum_{k} A_{i,k} \times B_{k,j}$$

The number of columns in the *A* array must be the same as the number of rows in the *B* array. Notationally, let's say that *rows*(*A*) and*columns*(*A*) are the number of rows and columns, respectively, in the *A* array. The number of individual multiplications required to compute the entire *C* array (which will have the same number of rows as *A* and the same number of columns as *B*) is then *rows*(*A*) *columns*(*B*) *columns*(*A*). For example, if *A* is a 10×20 array, and *B* is a 20×15 array, it will take $10 \times 15 \times 20$, or 3000 multiplications to compute the *C* array.

To perform multiplication of more than two arrays we have a choice of how to proceed. For example, if *X*, *Y*, and *Z* are arrays, then to compute*X Y Z* we could either compute (*X Y*) *Z* or *X* (*Y Z*). Suppose *X* is a 5×10 array, *Y* is a 10×20 array, and *Z* is a 20×35 array. Let's look at the number of multiplications required to compute the product using the two different sequences:

(*X Y*) *Z*

- $5 \times 20 \times 10 = 1000$ multiplications to determine the product (*X Y*), a 5×20 array.
- Then $5 \times 35 \times 20 = 3500$ multiplications to determine the final result.
- Total multiplications: 4500.

X (*Y Z*)

- $10 \times 35 \times 20 = 7000$ multiplications to determine the product (*Y Z*), a 10×35 array.
- Then $5 \times 35 \times 10 = 1750$ multiplications to determine the final result.
- Total multiplications: 8750.

Clearly we'll be able to compute (*X Y*) *Z* using fewer individual multiplications.

Given the size of each array in a sequence of arrays to be multiplied, you are to determine an optimal computational sequence. Optimality, for this problem, is relative to the number of individual multiplications required.

Input

For each array in the multiple sequences of arrays to be multiplied you will be given only the dimensions of the array. Each sequence will consist of an integer *N* which indicates the number of arrays to be multiplied, and then *N* pairs of integers, each pair giving the number of rows and columns in an array; the order in which the dimensions are given is the same as the order in which the arrays are to be multiplied. A value of zero for *N* indicates the end of the input. *N* will be no larger than 10.

Output

Assume the arrays are named $A1, A2, \ldots AN$. Your output for each input case is to be a line containing a parenthesized expression clearly indicating the order in which the arrays are to be multiplied. Prefix the output for each case with the case number (they are sequentially numbered, starting with 1). Your output should strongly resemble that shown in the samples shown below. If, by chance, there are multiple correct sequences, any of these will be accepted as a valid answer.

Sample Input

```
3
1 5
5 20
20 1
3
5 10
10 20
20 35
6
30 35
35 15
15 5
5 10
10 20
20 25
0
```

Sample Output

```
Case 1: (A1 x (A2 x A3))
Case 2: ((A1 x A2) x A3)
Case 3: ((A1 x (A2 x A3)) x ((A4 x A5) x A6))
```

Code

```
//============================================================================
// Name        : OptimalArrayMultiplicationSequence.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
#include <sstream>
using namespace std;


string output(long *s, int i, int j, int n)
{
	if(i == j)
	{
		ostringstream ss;
		ss << "A" << i + 1;
		return ss.str();
	}
	else
	{
		int k = *(s + i * n + j);
		string left = "(" + output(s, i, k, n) + " x ";
		string right = output(s, k + 1, j, n) + ")";
		return left + right;
	}
}

int main()
{
	int caseNr = 0;

	while(true)
	{
		int n;
		// how many matrices should be multiplied
		cin >> n;
		if(n == 0) break;
		// read row and column dimensions in the following format:
		// A1row, A1col = A2row, ..., Anrow = An+1col, An+1row
		caseNr++;
		int p[n + 1];
		for(int i = 0; i < n; i++) cin >> p[i] >> p[i + 1];

		long m[n][n];			// number of multiplications required
		long s[n][n];			// where to split between matrices
		// reset all multiplication counters to 0
		for(int i = 0; i < n; i++) m[i][i] = 0;

		for(int len = 1; len < n; len++)	// len = length of chain - 1
		{					// e.g. A1 x A2 => len = 1
			// i counts from the left boundary to the splitting point
			for(int i = 0; i < n - len; i++)
			{
			// j counts from the splitting point to the right boundary
				int j = i + len;
```

```
                        // m[i][j] == -1 means there has been no result yet
                        m[i][j] = -1;
                        // the splitting point moves from i (left) to j (right)
                        for(int k = i; k < j; k++)
                        {
                        // calculate the number of required multiplications
                              int q = m[i][k] + m[k + 1][j] + p[i] * p[k + 1] *
                              p[j + 1];
                              if(m[i][j] < 0 || q < m[i][j])
                  // if there hasn't been a result yet or the result is lower
                              {
                        // keep the number of multiplications as new lowest
                                    m[i][j] = q;
                        // keep the splitting point that achieved this number
                                    s[i][j] = k;
                              }
                        }
                  }
            }

            cout << "Case " << caseNr << ": " << output(&s[0][0], 0, n - 1, n) <<
            endl;
      }
      return 0;
}
```

Problem 2
369 – Combinations

Time limit: 3.000 seconds

Computing the exact number of ways that *N* things can be taken *M* at a time can be a great challenge when *N* and/or *M* become very large. Challenges are the stuff of contests. Therefore, you are to make just such a computation given the following:

GIVEN:

$$5 \le N \le 100, \quad \text{and} \quad 5 \le M \le 100, \quad \text{and} \quad M \le N$$

Compute the **EXACT** value of:

$$C = \frac{N!}{(N-M)! \times M!}$$

You may assume that the final value of *C* will fit in a 32-bit Pascal LongInt or a C long.

For the record, the exact value of 100! is:

```
93,326,215,443,944,152,681,699,238,856,266,700,490,715,968,264,381,621,
468,592,963,895,217,599,993,229,915,608,941,463,976,156,518,286,253,
697,920,827,223,758,251,185,210,916,864,000,000,000,000,000,000,000,000
```

Input and Output

The input to this program will be one or more lines each containing zero or more leading spaces, a value for *N*, one or more spaces, and a value for *M*. The last line of the input file will contain a dummy *N*, *M* pair with both values equal to zero. Your program should terminate when this line is read.

The output from this program should be in the form:

N `things taken` *M* `at a time is` *C* `exactly.`

Sample Input

```
100  6
 20  5
 18  6
  0  0
```

Sample Output

100 things taken 6 at a time is 1192052400 exactly.
20 things taken 5 at a time is 15504 exactly.
18 things taken 6 at a time is 18564 exactly.

Code

```
//============================================================================
// Name        : Combinations.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 369 - Combinations
//============================================================================

#include <iostream>
using namespace std;

unsigned long triangle[101];

/**
 * find constructs a pascal's triangle by iteratively adding up neighboring
numbers within the triangle
 *     level 0:                          1
 *     level 1:                    1           1
 *     level 2:              1           2           1
 *     level 3:        1           3           3           1
 *     level 4:    1           4           6           4           1
 *                 depth 1     depth 2     depth 3     depth 2     depth 1
 */
unsigned long find(int n, int m)
{
	int depth, level;
	// fill the border of the triangle with ones (depth = 0)
	for(depth = 0; depth <= m ; depth++) triangle[depth] = 1;
	// the depth of the triangle is determined by the difference between n and m
	for(depth = 1; depth <= (n - m); depth++)
	{
	// the next level of the triangle is calculated by adding the value of...
	// ...current depth and previous level + previous depth and current level
		for(level = 1; level <= n; level++)
			triangle[level] = triangle[level] + triangle[level - 1];
	}
	return triangle[m];
}

int main()
{
	int m, n;
	while(true)
	{
		cin >> n >> m;
		if( m == 0 && n == 0 ) break;
		cout << n << " things taken " << m << " at a time is " << find(n, m)
		<< " exactly." << endl;
	}
	return 0;
}
```

Problem 3

412 - Pi

Time limit: 3.000 seconds

Professor Robert A. J. Matthews of the Applied Mathematics and Computer Science Department at the University of Aston in Birmingham, England has recently described howthe positions of stars across the night skymay be used to deduce a surprisingly accurate value of π . This result followed from the application of certain theorems in number theory.

Here, we don'thave the night sky, but can use the same theoretical basis to form an estimate for π :

Given any pair of whole numbers chosen from a large, random collection of numbers, the probability that the twonumbers have no common factor other than one (1) is

$$\frac{6}{\pi^2}$$

For example, using the *small* collection of numbers: 2, 3, 4, 5, 6; there are 10 pairs that can be formed: (2,3), (2,4), etc. Six of the 10 pairs: (2,3), (2,5), (3,4), (3,5), (4,5) and (5,6) have no common factor other than one. Using the ratio of the counts as the probability we have:

$$\frac{6}{\pi^2} \approx \frac{6}{10}$$

$$\pi \approx 3.162$$

In this problem, you'll receive a series of data sets. Each data set contains a set of pseudo-random positive integers. For each data set, find the portion of the pairs which may be formed that have nocommon factor other than one (1), and use the method illustrated above to obtain an estimate for π . Report this estimate for each data set.

Input

The input consists of a series of data sets.

The first line of each data set contains a positive integer value, *N*, greater than one (1) and less than 50.

There is one positive integer per line for the next *N* lines that constitute the set for which the pairs are to be examined. These integers are each greater than 0 and less than 32768.

Each integer of the input stream has its first digit as the first character on the input line.

The set size designator, *N*, will be zero to indicate the end of data.

Output

A line with a single real value is to be emitted for each input data set encountered. This value is the estimate for π for the data set. An output format like the sample below should be used. Answers must be rounded to six digits after the decimal point.

For some data sets, it may be impossible to estimate a value for π. This occurs when there are *no* pairs without common factors. In these cases, emit the single-line message:

```
No estimate for this data set.
```

exactly, starting with the first character, `"N"`, as the first character on the line.

Sample Input

```
5
2
3
4
5
6
2
13
39
0
```

Sample Output **(Your output for the float/real, using your chosen language, may be default-formatted differently).**

```
3.162278
No estimate for this data set.
```

Code

```
//============================================================================
// Name        : Pi.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 412 - Pi
//============================================================================

#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

// as per requirement the program should read between 1 and 49 numbers
int numbers[49];

/**
 * reads numbers into the numbers array and returns the amount of numbers read
this way
 */
int input()
{
	int i, n;
	cin >> n;
	if(n == 0) return 0;
	for(i = 0; i < n; i++) cin >> numbers[i];
	return n;
}

/**
 * calculates the greatest common divisor (GCD) which is synonymous to
 * the greates common factor (GFC) using the Euclidean algorithm
 */
int getGCD(int a, int b)
{
	int m;
	while(a != 0)
	{
		m = a;
		a = b % a;
		b = m;
	}
	return b;
}

int main()
{
	int n, i, j;
	int noCommonFactor;	// number of pairs with common factor
	int total;		// total number of pairs
	while(true)
	{
		n = input();	// read a set of numbers
		if(n == 0) break;	// program end condition
	// calculate the amount of pairs, that can be generated from the numbers
		else total = n * (n - 1) / 2;
	// counts up for every pair without a common factor greater than 1
```

```
            noCommonFactor = 0;
            for(i = 0; i < n; i++)    // generate pairs
            {
                  for( j = i + 1; j < n; j++)
                  {
            // if getGCD returns 1 both numbers have only 1 as common factor
                        if(getGCD(numbers[i], numbers[j]) == 1)
                        noCommonFactor++;
                  }
            }
            if(noCommonFactor == 0)
            cout << "No estimate for this data set." << endl;
            else cout << setprecision(6) << fixed << sqrt(6.0 * ((double)total /
            (double)noCommonFactor)) << endl;
      }
      return 0;
}
```

Problem 4

729 - The Hamming Distance Problem

Time limit: 3.000 seconds

The Hamming distance between two strings of bits (binary integers) is the number of corresponding bit positions that differ. This can be found by using XOR on corresponding bits or equivalently, by adding corresponding bits (base 2) without a carry. For example, in the two bit strings that follow:

```
     A       0 1 0 0 1 0 1 0 0 0
     B       1 1 0 1 0 1 0 1 0 0
A XOR B = 1 0 0 1 1 1 1 1 0 0
```

The Hamming distance (*H*) between these 10-bit strings is 6, the number of 1's in the XOR string.

Input

Input consists of several datasets. The first line of the input contains the number of datasets, and it's followed by a blank line. Each dataset contains *N*, the length of the bit strings and *H*, the Hamming distance, on the same line. There is a blank line between test cases.

Output

For each dataset print a list of all possible bit strings of length *N* that are Hamming distance *H* from the bit string containing all 0's (origin). That is, all bit strings of length *N* with exactly *H* `1`'s printed in ascending lexicographical order.

The number of such bit strings is equal to the combinatorial symbol *C*(*N*,*H*). This is the number of possible combinations of *N*-*H* zeros and *H*ones. It is equal to

$$\frac{N!}{(N-H)!H!}$$

This number can be very large. The program should work for $1 \leq H \leq N \leq 16$.

Print a blank line between datasets.

Sample Input

```
1
```

```
4 2
```

Sample Output

```
0011
0101
0110
1001
1010
1100
```

Code

```
//============================================================================
// Name        : TheHammingDistanceProblem.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 729 - The Hamming Distance Problem
//============================================================================

#include <iostream>
using namespace std;

int main()
{
	char num[17];		// buffer for the binary number string + null terminator
	int i, j, k;
	int n, h;			// number length and hamming distance
	int datasets;		// number of data sets to calculate
	bool newline = false;	// set to true when a newline needs to be printed
	cin >> datasets;	// read number od data sets
	// repeat loop until all data sets have been calculated
	for(i = 0; i < datasets; i++)
	{
		if(newline) cout << endl;
		else newline = true;

		cin >> n >> h;		// read length and hamming distance
		// insert null terminator at the end of the binary number string
		num[n] = 0;
		// fill the first n - h bits with zeros
		for(j = 0; j < (n - h); j++) num[j] = '0';
	// the other bits filled with ones to finish the starting binary number
		while(j < n) num[j++] = '1';

		while(true)
		{
			cout << num << endl;		// print the current binary number
			j = n - 1;	// j points to the last digit of the binary number
			while(num[j] == '0') j--; // find the first one from the end
			// k points in front of the first digit of the binary number
			k = -1;
			while(num[j] == '1')// repeat until j points to the first zero
			{
				j--;
				k++;
			}
			// if j gets smaller than 0 all permutations have been printed
			if(j < 0) break;
			num[j] = '1';		// otherwise change this zero to one
			// fill the n - k bits after the changed bit to zero
			for(j += 1; j < (n - k); j++) num[j] = '0';
			// the other bits are filled with ones
			while(j < n) num[j++] = '1';
		}
	}
	return 0;
}
```

Problem 5

750 - 8 Queens Chess Problem

Time limit: 3.000 seconds

In chess it is possible to place eight queens on the board so that no one queen can be taken by any other. Write a program that will determine all such possible arrangements for eight queens given the initial position of one of the queens.

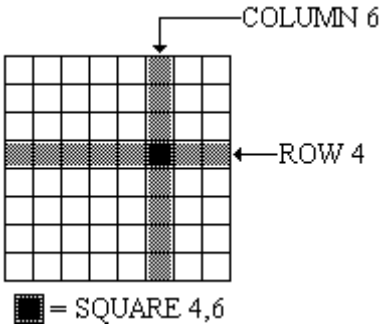
Do not attempt to write a program which evaluates every possible 8 configuration of 8 queens placed on the board. This would require 8^8evaluations and would bring the system to its knees. There will be a reasonable run time constraint placed on your program.

Input

The first line of the input contains the number of datasets, and it's followed by a blank line. Each dataset will be two numbers separated by a blank. The numbers represent the square on which one of the eight queens must be positioned. A valid square will be represented; it will not be necessary to validate the input.

To standardize our notation, assume that the upper left-most corner of the board is position (1,1). Rows run horizontally and the top row is row 1. Columns are vertical and column 1 is the left-most column. Any reference to a square is by row then column; thus square (4,6) means row 4, column 6.

Each dataset is separated by a blank line.



Output

Output for each dataset will consist of a one-line-per-solution representation.

Each solution will be sequentially numbered $1 \ldots N$. Each solution will consist of 8 numbers. Each of the 8 numbers will be the ROW coordinate for that solution. The column coordinate will be indicated by the order in which the 8 numbers are printed. That is, the first number represents the ROW in which the queen is

positioned in column 1; the second number represents the ROW in which the queen is positioned in column 2, and so on.

The sample input below produces 4 solutions. The full 8 × 8 representation of each solution is shown below.

DO NOT SUBMIT THE BOARD MATRICES AS PART OF YOUR SOLUTION!

```
   SOLUTION 1           SOLUTION 2           SOLUTION 3           SOLUTION 4

1 0 0 0 0 0 0 0      1 0 0 0 0 0 0 0 0      1 0 0 0 0 0 0 0      1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 1 0      0 0 0 0 0 1 0 0      0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0      0 0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 1      0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1      0 0 0 0 0 1 0 0 0      0 0 1 0 0 0 0 0      0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0 1      0 0 0 0 0 0 1 0      0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0      0 1 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0      0 0 0 0 1 0 0 0 0      0 1 0 0 0 0 0 0      0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0      0 0 1 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 1 0 0 0 0
```

Submit only the one-line, 8 digit representation of each solution as described earlier. Solution #1 below indicates that there is a queen at Row 1, Column 1; Row 5, Column 2; Row 8, Column 3; Row 6, Column 4; Row 3,Column 5; ... Row 4, Column 8.

Include the two lines of column headings as shown below in the sample output and print the solutions in lexicographical order.

Print a blank line between datasets.

Sample Input

```
1

1 1
```

Sample Output

```
SOLN       COLUMN
 #       1 2 3 4 5 6 7 8

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

Code

```
//============================================================================
// Name        : 8QueensChessProblem.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 750 - 8 Queens Chess Problem
//============================================================================

#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int queens[8];
int n, fcol, fline;
int solved;

/**
 * prints the solution in the required layout
 */
void output()
{
	cout << endl << setw(2) << ++solved << "     ";
	for(int i = 0; i < 8; i++) cout << setw(2) << queens[i] + 1;
}

/**
 * checks if a queen at the passed line and column would be attackable
 * given the current state of the queens array
 */
bool isAttackable(int line, int col)
{
	for(int i = 0; i < 8; i++)
	{
		// is another queen in the same column?
		if(queens[i] == col) return true;
		// is another queen diagonally above?
		if(line > i && ((line - i) == abs(queens[i] - col))) return true;
		// is another queen diagonally below?
		if(line < i && ((i - line) == abs(queens[i] - col))) return true;
	}
	return false; // if it's neither of the above queen is not attackable
}

/**
 * Backtracking algorithm to find all solutions
 */
void solve(int line)
{
	// don't check the line with the user defined queen
	if(line == fline) line++;
	for(int i = 0; i < 8; i++)
	{
		if(!isAttackable(line, i))
		{
			queens[line] = i;  // put a queen at current position
// if this has been the last line without queen one valid solution has been found!
```

```
                    if(line == 7 || (line == 6 && fline == 7)) output();
                    // otherwise try to find a solution with the given field
                    else solve(line + 1);
                    // if the program comes here it means
                    // all valid solutions for this setting have been found
                    queens[line] = 99;

             }
      }
}

int main()
{
      cin >> n;                        // get number of data sets to be solved
      for(int i = 0; i < n; i++)       // repeat for every data set
      {
             solved = 0;  // will be increased for every solution that is found
             // set all fields of queens to 99 (no queen set)
             for(int j = 0; j < 8; j++) queens[j] = 99;
             // get position of the queen in format line/column is read
             cin >> fline >> fcol;
             cout << "SOLN       COLUMN" << endl << " #      1 2 3 4 5 6 7 8" <<
             endl;
             fline--; fcol--;          // array indices start at 0
             queens[fline] = fcol;     // save position in queens array
             solve(0);                 // solve the problem starting with line 1
             cout << endl;
      }
      return 0;
}
```

Problem 6

989 - Su Doku

Time limit: 3.000 seconds

Problem

In many newspapers we may find some puzzles to solve, one of those is Su Doku. Given a grid *9×9* with some of entries filled, the objective is to fill in the grid so that every row, every column, and every *3×3* box contains the digits *1* through *9*.

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

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

source: http://www.sudoku.com

Input

Input contains several test cases separated by a blank line. Each of them contains an integer n such that $1 \leq n \leq 3$ and a grid $n^2 \times n^2$ with some of the entries filled with digits from *1* to n^2 (an entrie not filled will have *0*). In this case, the objective is to fill in the grid so that every row, every column, and every $n \times n$ box contains the digits *1* through n^2.

Output

A solution for the problem. If exists more than one, you should give the lower one assuming a lexicographic order. If there is no solution, you should print '`NO SOLUTION`'. For lexicographic comparison you should consider lines in first place. Print a blank line between test cases.

Sample input

```
3
0 6 0 1 0 4 0 5 0
0 0 8 3 0 5 6 0 0
```

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

Sample output

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

Code

```
//============================================================================
// Name        : SuDoku.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 989 - Su Doku
//============================================================================

#include <iostream>

using namespace std;

// used for the bitwise manipulations of sl, sc and sb
const int B[10] = {0, 1, 2, 4, 8, 16, 32, 64, 128, 256};
// b = number of blocks, n = dimension of the matrix
int n, b;
// sequence of the 81 fields of the su doko top to bottom, left to right
int in[81];
// v[a][b] = Su Doku matrix (a = line, b = column)
int v[9][9];
// sl[a] = set lines (a = line, first bit of
// int = 1 is set, second bit = 2 is set...)
int sl[9];
// sc[a] = set columns (a = column, first bit of
// int = 1 is set, second bit = 2 is set...)
int sc[9];
// sb[a][b] = set block (a = block line, b = block column,
// first bit of int = 1 is set, second bit = 2 is set...)
int sb[3][3];
// is set to true if a solution is found
bool solution;
// sets the given value at line l and column c if it doesn't break Su Doku rules
bool set(int l, int c, int value)
{
// if the given value is already set in the line, column or block return false
	if((sl[l] | sc[c] | sb[(int)(l / b)][(int)(c / b)]) & B[value]) return
false;
	else
	{
	// sets the corresponding bit in sl, sc and sb
	// e.g. the first bit is 2^0=1, the fifth bit which is 2^4=16 if value is 5
		sl[l] = sl[l] | B[value];
		sc[c] = sc[c] | B[value];
		sb[(int)(l / b)][(int)(c / b)] = sb[(int)(l / b)][(int)(c / b)] |
		B[value];
		// the value is set in the field of the su doku
		v[l][c] = value;
		// if the value has been set the function returns true
		return true;
	}
}

// unsets the given value at line l and column c
void unset(int l, int c, int value)
{
	// xor always results in 0 if both bits are set
	// which they have to be in order for unset to be called
	sl[l] = sl[l] ^ B[value];
```

```
	sc[c] = sc[c] ^ B[value];
	sb[(int)(l / b)][(int)(c / b)] = sb[(int)(l / b)][(int)(c / b)] ^ B[value];
	// value of the field in the su doku is set to 0 again
	v[l][c] = 0;
}

// puts out the whole matrix saved in v
void output()
{
	for(int l = 0; l < n; l++)
	{
		for(int c = 0; c < n; c++)
		{
			cout << v[l][c];
			if(c < (n - 1)) cout << " ";
			else cout << endl;

		}
	}
	cout << endl;
}

// transforms the sequential input in the in array to the matrix format of v and
sets the bits in sl, sc and sb
void transform()
{
	int k = 0;
	for(int l = 0; l < n; l++)
	{
		for(int c = 0; c < n; c++)
		{
			// l = line index, c = column index
			v[l][c] = in[k];
			// save non empty entries to corresponding
			if(in[k] > 0)
			{
			// set lines, columns and blocks
				sl[l] = sl[l] | B[in[k]];
				sc[c] = sc[c] | B[in[k]];
				sb[(int)(l / b)][(int)(c / b)] = sb[(int)(l /
				b)][(int)(c / b)] | B[in[k]];
			}
			k++;
		}
	}
}

// reads the su doko from stdin
bool input()
{
	// b = number of blocks in x and y direction
	cin >> b;
	// if 0 is entered the program ends.
	if(b == 0) return false;
	// A 9x9 Su Doku consists has 3x3 (n=3) rows and columns.
	n = b * b;
	// solution is set to true once the first valid solution has been found
	solution = false;

	for(int l = 0; l < n; l++)
	{
```

```
			for(int c = 0; c < n; c++)
			{
				cin >> in[l * n + c];
			}
		}

		for(int k = 0; k < n; k++)
		{
			sl[k] = 0;	// reset lines,
			sc[k] = 0;	// reset columns and
		}
		for(int k = 0; k < b; k++)
			for(int l = 0; l < b; l++) sb[k][l] = 0;	// reset blocks

		transform();		// set v, sl, sc and sb according to input
		return true;
}

// returns the solution if there is only one valid one for the field (e.g. all but
1 of the first n bits are set to 1)
int hasOneSolution(int x)
{
		char num = 0;
		char solution = 0;
		for(int i = 1; i <= n; i++)
		{
			// if the bit in question is not set in x, i is a valid solution
			if(!(x & B[i]))
			{	// increase the number of solutions by one
				num++;
				// remember the found solution
				solution = i;
			}
			// if num > 1, more than one solution
			// has been found and hasOneSolution returns 0
			if(num > 1) return 0;
		}
		// if exactly one solution has been found
		// hasOneSolution returns that solution
		if(solution == 1) return solution;
		// if no solutions have been found it returns 0
		else return 0;
}

// solves as much of the Su Doku as possible with fields where
// the solution is definite (e.g. fields that only have one solution)
void smartSolve()
{
		int l = 0;
		int c = 0;
		int k = 0;
		int s = 0;

		/* loop from top to bottom and left to right until all fields of
		 * the su doku have been checked once
		 * since the last field with exactly one solution had been found.
		 */
		while(k < (n*n))
		{
			if(v[l][c] == 0 && (s = hasOneSolution((sl[l] | sc[c] | sb[(int)(l /
b)][(int)(c / b)])))
```

```
		{
			// hasOneSolution has found definite solution
			// s which is set at position l and c
			set(l, c, s);
			// k is reset to 0, if no solutions are found
			// for the next n*n fields there are no more definite solutions
			k = 0;
		}
		c++;
		if(c == n)
		{
			c = 0;
			l++;
			if(l == n) l = 0;
		}
		// regardless of the result, each time the algorith
		// continues to the next field k is increased by one
		k++;
	}

}

// recursive backtracking function to find valid values for
// all zeros and put out the first solution to the whole Su Doku
void solve(int l, int c)
{
	do
	{
		c++;
		if(c == n)
		{
			c = 0;
			l++;
			if(l == n)
			{
				output();
				solution = true;
				return;
			}
		}
	}
	// repeat until either the next zero is found or
	// there are no zeros left (in which case a solution has been found)
	while(v[l][c] != 0);

	// if a zero has been found, try all values from 1 to n (dimension)
	for(int value = 1; value <= n; ++value)
	{
		// for each value check if it can be set and do so if true
		if(set(l, c, value))
		{
			// try to solve the rest of the Su Doku
			// (starting with the next column)
			solve(l, c);
			// if a solution has already been found
			// (after returning here), don't continue
			if(solution) return;
			// if no solution has been found, unset the
			// last set value and go on with the for loop
			unset(l, c, value);
		}
```

```
	}
}

int main()
{
	while(true)
	{
		// get input
		if(!input()) break;
		// solve all definite fields (those with only one solution)
		smartSolve();
		// solve the rest via backtracking algorithm
		solve(0, -1);
		if(!solution) cout << "NO SOLUTION" << endl;
		break;
	}

	return 0;
}
```

Problem 7

10066 - The Twin Towers

Time limit: 3.000 seconds

Once upon a time, in an ancient Empire, there were two towers of dissimilar shapes in two different cities. The towers were built by putting circular tiles one upon another. Each of the tiles was of the same height and had integral radius. It is no wonder that though the two towers were of dissimilar shape, they had many tiles in common.

However, more than thousand years after they were built, the Emperor ordered his architects to remove some of the tiles from the two towers so that they have exactly the same shape and size, and at the same time remain as high as possible. The order of the tiles in the new towers must remain the same as they were in the original towers. The Emperor thought that, in this way the two towers might be able to stand as the symbol of harmony and equality between the two cities. He decided to name them the *Twin Towers*.

Now, about two thousand years later, you are challenged with an even simpler problem: given the descriptions of two dissimilar towers you are asked only to find out the number of tiles in the highest twin towers that can be built from them.

Input

The input file consists of several data blocks. Each data block describes a pair of towers.

The first line of a data block contains two integers N1 and N2 (1 <= N1, N2 <= 100) indicating the number of tiles respectively in the two towers. The next line contains N1 positive integers giving the radii of the tiles (from top to bottom) in the first tower. Then follows another line containing N2 integers giving the radii of the tiles (from top to bottom) in the second tower.

The input file terminates with two zeros for N1 and N2.

Output

For each pair of towers in the input first output the twin tower number followed by the number of tiles (in one tower) in the highest possible twin towers that can be built from them. Print a blank line after the output of each data set.

Sample Input

```
7 6
20 15 10 15 25 20 15
15 25 10 20 15 20
8 9
10 20 20 10 20 10 20 10
20 10 20 10 10 20 10 10 20
```

```
0 0
```

Sample Output

```
Twin Towers #1
Number of Tiles : 4

Twin Towers #2
Number of Tiles : 6
```

Code

```
//============================================================================
// Name        : TheTwinTowers.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 10066 - The Twin Towers
//============================================================================


#include <iostream>
using namespace std;

int main()
{
	int towerNr = 0;
	int i, j;

	while(true)
	{
		int m, n;
		// Tower X has m pieces, Tower Y has n pieces
		cin >> m >> n;
		// if both m and n are 0, stop the program
		if(m == 0 && n == 0) break;
		// count the number of Twin Towers calculated
		towerNr++;

		int X[m];
		int Y[n];

		// Read the actual pieces for Tower X
		for(i = 0; i < m; i++) cin >> X[i];
		// Read the actual pieces for tower Y
		for(i = 0; i < n; i++) cin >> Y[i];

		// c[a][b] saves the longest distance
		int c[m + 1][n + 1];

		// reset the distances to 0
		for(i = 1; i <= m; i++) c[i][0] = 0;
		// for all relevant indexes
		for(j = 0; j <= n; j++) c[0][j] = 0;

		for (i = 1; i <= m; i++)
		{
			for (j = 1; j <= n; j++)
			{
				if (X[i - 1] == Y[j - 1]) c[i][j] = c[i - 1][j - 1] + 1;
				else if (c[i - 1][j] >= c[i][j - 1]) c[i][j] = c[i -
				1][j];
				else c[i][j] = c[i][j - 1];
			}
		}
		cout << "Twin Towers #" << towerNr << endl << "Number of Tiles : " <<
		c[m][n] << endl << endl;
	}
	return 0;
}
```

Problem 8

10929 - You can say 11

Time limit: 3.000 seconds

Introduction to the problem

Your job is, given a positive number N, determine if it is a multiple of eleven.

Description of the input

The input is a file such that each line contains a positive number. A line containing the number 0 is the end of the input. The given numbers can contain up to 1000 digits.

Description of the output

The output of the program shall indicate, for each input number, if it is a multiple of eleven or not.

Sample input:

112233
30800
2937
323455693
5038297
112234
0

Sample output

112233 is a multiple of 11.
30800 is a multiple of 11.
2937 is a multiple of 11.
323455693 is a multiple of 11.
5038297 is a multiple of 11.
112234 is not a multiple of 11.

Code

```
//============================================================================
// Name        : YouCanSay11.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 10929 - You can say 11
//============================================================================

#include <iostream>
#include <string>
using namespace std;

int main()
{
	int crosstotal, digit;
	string number;

	do
	{
		cin >> number;
		if(number == "0") break;
		crosstotal = 0;
		for(unsigned int i = 0; i < number.length(); i++)
		{
			digit = number[i] - 48;
			if((i % 2) == 0) crosstotal += digit;
			else crosstotal -= digit;
		}
		if((crosstotal % 11) == 0) cout << number << " is a multiple of 11."
		<< endl;
		else cout << number + " is not a multiple of 11." << endl;
	}
	while(true);

	return 0;
}
```

Problem 9

11172 - Relational Operator

Time limit: 3.000 seconds

Some operators checks about the relationship between two values and these operators are called relational operators. Given two numerical values your job is just to find out the relationship between them that is (i) First one is greater than the second (ii) First one is less than the second or (iii) First and second one is equal.

Input

First line of the input file is an integer t (t<15) which denotes how many sets of inputs are there. Each of the next t lines contain two integers a and b (|a|,|b|<1000000001).

Output

For each line of input produce one line of output. This line contains any one of the relational operators ">", "<" or "=", which indicates the relation that is appropriate for the given two numbers.

Sample Input	Output for Sample Input
3 10 20 20 10 10 10	< > =

Code

```
//============================================================================
// Name        : RelationalOperators.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 11172 - Relational Operator
//============================================================================

#include <iostream>
using namespace std;

int main()
{
	int i, t;
	cin >> t;
	int a[t], b[t];
	for(i = 0; i < t; i++) cin >> a[i] >> b[i];
	for(i = 0; i < t; i++)
	{
		if(a[i] < b[i]) cout << "<" << endl;
		else if(a[i] > b[i]) cout << ">" << endl;
		else cout << "=" << endl;
	}
	return 0;
}
```

Problem 10

12342 - Tax Calculator

Time limit: 3.000 seconds

People of a certain country have a little interest in paying tax. This is not the case that they do not love the country or they do not want to obey the law, but the problem is the complex calculations for payable tax. Most of the people do not understand the rule and some lawyers take high charges to help(?) them. So, most of the people just avoid paying tax. Realizing this, the Government decided to automate the taxpaying system and appointed you to program a tax calculator that takes a person's yearly income as input and calculate the payable tax for him. Here is the rule for calculate payable tax for an individual:

1. The first 180,000/- of income is tax free.
2. Next 300,000/- will have 10% tax.
3. Next 400,000/- will have 15% tax.
4. Next 300,000/- will have 20% tax.
5. The rest will have 25% tax.
6. The minimum payable tax will be 2,000/-. That is if anyone's tax is below 2,000/- (but of course greater than zero) he must pay 2,000/-
7. The payable tax must be an integer. If the calculated tax is a floating point then it must be replaced by the smallest integer greater than the payable tax.

Input

The first line of input will contain an integer **T** **($T \leq 5000$)** which denotes the number of test cases. Each of the following **T** lines contains an integer **k** **($1 \leq k \leq 10^9$)**.

Output

For each line of input output the case number and an integer which is the payable tax for the given income.

Sample Input	Output for Sample Input
3 180001 12345 615000	Case 1: 2000 Case 2: 0 Case 3: 50250

Illustration of 3rd sample:

Amount	Rate	Tax
180000	0%	0
300000	10%	30000
135000	15%	20250
Total		50250

Code

```
//============================================================================
// Name        : TaxCalculator.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 12342 - Tax Calculator
//============================================================================

#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;

int main()
{
	int max;
	long amount;
	double tax = 0;
	cout << setprecision(0);

	cin >> max;
	for(int i = 0; i < max; i++)
	{
		cin >> amount;

		// First 180k are free
		if(amount > 180000) amount -= 180000;
		else amount = 0;

		// Next 300k have 10% tax
		if(amount > 300000)
		{
			amount -= 300000;
			tax += 30000;
		}
		else if(amount > 0)
		{
			tax += amount * 0.1;
			amount = 0;
		}

		// Next 400k have 15% tax
		if(amount > 400000)
		{
			amount -= 400000;
			tax += 60000;
		}
		else if(amount > 0)
		{
			tax += amount * 0.15;
			amount = 0;
		}

		// Next 300k have 20% tax
		if(amount > 300000)
		{
			tax += 60000;
```

```
                amount -= 300000;
            }
            else if(amount > 0)
            {
                tax += amount * 0.2;
                amount = 0;
            }

            // Rest amount has 25% tax
            if(amount > 0)
            {
                tax += amount * 0.25;
                amount = 0;
            }

            // if any tax has been paid it has to be at least 2000
            if(tax > 0 && tax < 2000) tax = 2000;

            cout << "Case " << i + 1 << ": " << fixed << ceil(tax) << endl;
            tax = 0;
        }
        return 0;
}
```

Problem 11

12506 - Shortest Names

Time limit: 3.000 seconds

In a strange village, people have very long names. For example: aaaaa, bbb and abababab.

You see, it's very inconvenient to call a person, so people invented a good way: just call a prefix of the names. For example, if you want to call `aaaaa', you can call `aaa', because no other names start with `aaa'. However, you can't call `a', because two people's names start with `a'. The people in the village are smart enough to always call the shortest possible prefix. It is guaranteed that no name is a prefix of another name (as a result, no two names can be equal).

If someone in the village wants to call every person (including himself/herself) in the village exactly once, how many characters will he/she use?

Input

The first line contains T ($T \leq 10$), the number of test cases. Each test case begins with a line of one integer n ($1 \leq n \leq 1000$), the number of people in the village. Each of the following n lines contains a string consisting of lowercase letters, representing the name of a person. The sum of lengths of all the names in a test case does not exceed 1,000,000.

Output

For each test case, print the total number of characters needed.

Sample Input

```
1
3
aaaaa
bbb
abababab
```

Sample Output

```
5
```

Code

```
//============================================================================
// Name        : ShortestNames.cpp
// Author      : ae652720
// Version     :
// Copyright   : Your copyright notice
// Description : 12506 - Shortest Names
//============================================================================

#include <iostream>
#include <algorithm>
using namespace std;


// compares and returns the minimum required letters to distinguish both
int compare(string name1, string name2)
{
	// every name needs at least one letter
	int length = 1;
	// process the first string letter by letter
	for(unsigned int i = 0; i < name1.length(); i++)
	{
		// if letter is different in both names, the length has been found
		if(name1[i] != name2[i]) break;
		// else increase the length by one
		length++;
	}
	return length;
}


int main()
{
	int cases;
	int names;

	cin >> cases;

	// outer loop repeats once for every case
	for(int i = 0; i < cases; i++)
	{
		// read number of names
		cin >> names;
		string *name = new string[names];

		// read names
		for(int j = 0; j < names; j++) cin >> name[j];

		// sort the list of names alphabetically
		sort(name, name + names);
		int lastLength = 1;
		int nextLength = 1;
		int sum = 0;
		// inner loop compares each name with the following one
		for(int j = 0; j < names - 1; j++)
		{
			// compare two neighboring names and keep the
			// result in nextLength
```

```
                lastLength = nextLength;
                // the result of last iteration is kept in lastLength
                nextLength = compare(name[j], name[j + 1]);
                // the comparison that required most letters is the
                // one necessary to identify the name
                if(nextLength > lastLength) sum += nextLength;
                // this result is added to the sum
                else sum += lastLength;
            }
            // required length for the last name is always
            // the result of the last comparison
            sum += nextLength;

            cout << sum << endl;
            delete [] name;
        }

        return 0;
}
```