UW Madison's

2006 ACM-ICPC Team Test #1

October 22, 1:00-6:00pm

Overview:

This test consists of nine problems, which will be referred to by the following names (respective of order):

borg, brownie, chandelier, code, graduate, grass, lizards, reverse, rockers

As this is a team competition, you are allowed one computer for each team. Also, you may only use 25 pages of printed documentation in addition to the online STL and Java documentation.

Input/Output:

Your programs should take input from standard in (the keyboard) and output to standard out (the terminal). As is standard practice for the ICPC, you may assume that the input strictly follows the description in the problems. If a problem description does not indicate how the input will be terminated, you may assume it will be with the EOF character. It is your responsibility to ensure that your output **precisely** matches that described in the problems, or else risk your program being rejected with a "Presentation Error".

Problem Submission:

To submit a program, send e-mail to mwa+icpc@cs.wisc.edu and attach your source code. The subject line should contain only the problem name, **prob,** that you are submitting, and the attached source code should be named **prob**.{c|cpp|java}. For example, if you are using C++ and submitting the problem "borg", the file should be named "borg.cpp". You will receive the results of your submission as soon as possible via e-mail. You may submit from any e-mail account of a team member, but please try to be consistent and use only one throughout the contest.

Clarifications:

As in the ICPC, you may submit clarification requests as well. They should be sent to mwa+icpc@cs.wisc.edu, with a subject of "Clarification-**prob**", where **prob** is the name of the problem you wish to be clarified. Replace **prob** with "general" if there is an issue with the contest as a whole. Accepted clarification requests will be answered to all those taking the test via e-mail. Experience of previous years learns that most clarification requests are rejected, and receive a simple response such as "Read the problem description".

Printing:

You may print to the printer at any time during the test.

GOOD LUCK!

Killing Aliens in a Borg Maze

The Borg is an immensely powerful race of enhanced humanoids from the delta quadrant of the galaxy. The Borg collective is the term used to describe the group consciousness of the Borg civilization. Each Borg individual is linked to the collective by a sophisticated subspace network that insures each member is given constant supervision and guidance.

Your task is to help the Borg (yes, really) by developing a program which helps the Borg to estimate the minimal cost of scanning a maze for the assimilation of aliens hiding in the maze, by moving in north, west, east, and south steps. The tricky thing is that the beginning of the search is conducted by a large group of over **100** individuals. Whenever an alien is assimilated, or at the beginning of the search, the group may split in two or more groups (but their consciousness is still collective.). The cost of searching a maze is defined as the total distance covered by all the groups involved in the search together. That is, if the original group walks five steps, then splits into two groups each walking three steps, the total distance is **11=5+3+3**.

Input

On the first line of input there is one integer, **N <= 50**, giving the number of test cases in the input. Each test case starts with a line containing two integers **x, y** such that **1 <= x, *y* <= 50**. After this, **y** lines follow, each which **x** characters. For each character, a space ``  '' stands for an open space, a hash mark ``**#**'' stands for an obstructing wall, the capital letter ``**A**'' stand for an alien, and the capital letter ``**S**'' stands for the start of the search. The perimeter of the maze is always closed, i.e., there is no way to get out from the coordinate of the ``**S**''. At most **100** aliens are present in the maze, and everyone is reachable.

Output

For every test case, output one line containing the minimal cost of a successful search of the maze leaving no aliens alive.

Sample Input

```
2
6 5
#####
#A#A##
# # A#
#S  ##
#####
7 7
#####
#AAA###
#    A#
# S ###
#     #
#AAA###
#####
```

Sample Output

```
8
11
```

(Joint Effort Contest, Problem Source: Swedish National Programming Contest, arranged by department of Computer Science at Lund Institute of Technology.)

Brownie Points

Stan and Ollie play the game of Odd Brownie Points. Some brownie points are located in the plane, at integer coordinates. Stan plays first and places a vertical line in the plane. The line must go through a brownie point and may cross many (with the same x-coordinate). Then Ollie places a horizontal line that must cross a brownie point already crossed by the vertical line.

Those lines divide the plane into four quadrants. The quadrant containing points with arbitrarily large positive coordinates is the top-right quadrant.

The players score according to the number of brownie points in the quadrants. If a brownie point is crossed by a line, it doesn't count. Stan gets a point for each (uncrossed) brownie point in the top-right and bottom-left quadrants. Ollie gets a point for each (uncrossed) brownie point in the top-left and bottom-right quadrants.

Stan and Ollie each try to maximize his own score. When Stan plays, he considers the responses, and chooses a line which maximizes his smallest-possible score.

Input contains a number of test cases. The data of each test case appear on a sequence of input lines. The first line of each test case contains a positive odd integer $1 < n < 200000$ which is the number of brownie points. Each of the following n lines contains two integers, the horizontal (x) and vertical (y) coordinates of a brownie point. No two brownie points occupy the same place. The input ends with a line containing 0 (instead of the n of a test).

For each input test, print a line of output in the format shown below. The first number is the largest score which Stan can assure for himself. The remaining numbers are the possible (high) scores of Ollie, in increasing order.

Sample input

```
11
3 2
3 3
3 4
3 6
2 -2
1 -3
0 0
-3 -3
-3 -2
-3 -4
3 -7
0
```

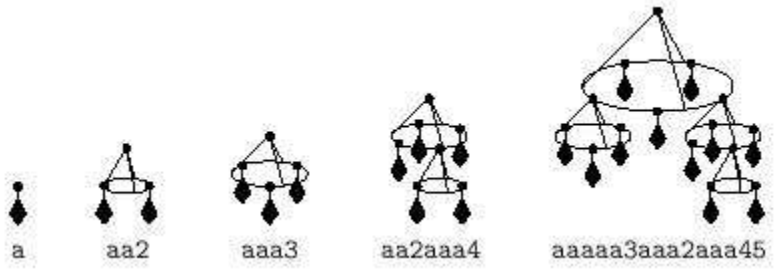
Output for sample input

```
Stan: 7; Ollie: 2 3;
```

P. Rudnicki

Chandelier

Lamps-O-Matic company assembles very large chandeliers. A chandelier consists of multiple levels. On the first level crystal pendants are attached to the rings. Assembled rings and new pendants are attached to the rings of the next level, and so on. At the end there is a single large ring -- the complete chandelier with multiple smaller rings and pendants hanging from it. A special-purpose robot assembles chandeliers. It has a supply of crystal pendants and empty rings, and a stack to store elements of a chandelier during assembly. Initially the stack is empty. Robot executes a list of commands to assemble a chandelier.



On command ``a'' robot takes a new crystal pendant and places it on the top of the stack. On command ``1'' to ``9'' robot takes the corresponding number of items from the top of the stack and consecutively attaches them to the new ring. The newly assembled ring is then placed on the top of the stack. At the end of the program there is a single item on the stack -- the complete chandelier. Unfortunately, for some programs it turns out that the stack during their execution needs to store too many items at some moments. Your task is to optimize the given program, so that the overall design of the respective chandelier remains the same, but the maximal number of items on the stack during the execution is minimal. A pendant or any complex multi-level assembled ring count as a single item of the stack. The design of a chandelier is considered to be the same if each ring contains the same items in the same order. Since rings are circular it does not matter what item is on the top of the stack when the robot receives a command to assemble a new ring, but the relative order of the items on the stack is important. For example, if the robot receives command ``4'' when items $\langle i_1, i_2, i_3, i_4 \rangle$ are on the top of the stack in this order (i_1 being the topmost), then the same ring is also assembled if these items are arranged on the stack in the following ways: $\langle i_2, i_3, i_4, i_1 \rangle$, or $\langle i_3, i_4, i_1, i_2 \rangle$, or $\langle i_4, i_1, i_2, i_3 \rangle$.

Input

Input file contains several test cases. Each of them consists of a single line with a valid program for the robot. The program consists of at most 10 000 characters.

Output

For each test case, print two output lines. On the first line write the minimal required stack capacity (number of items it can hold) to assemble the chandelier. On the second line write some program for the assembly robot that uses stack of this capacity and results in the same chandelier.

Sample Input

```
aaaaa3aaa2aaa45
```

Sample Output

```
6
aaa3aaa2aaa4aa5
```

Northeastern Europe & Russian Republic 2004-2005

Code Formatting

Programmers are known to wage religious wars when issues of proper code formatting are discussed. When new team of programmers starts working on a project, it often brings slightly different code formatting style and wants to reformat old source code according to their own style. Moreover, inexperienced programmers often neglect the importance of good and consistent code style, thus complicating the work of their teammates and themselves. This situation creates thriving market for code formatting tools.

You are taking part in a proof-of-concept project for a new code formatting tool code named Salvation. This is only a pilot project aimed not for a practical usefulness, but to demonstrate your ability to parse and format code of a high-level language. Your task is to write code formatter for a language called TRIVIAL (The Rival Implementation-Agnostic Language). This language has trivial lexical and grammatical structures. It does not have any keywords and control structures, because all constructs of the language are represented as function calls and closures.

The lexical structure consists of identifiers, opening and closing parenthesis and curly braces, commas, and semi-colons. Identifiers consist only of digits `0'-'9' and Latin letters `a'-`z', `A'-`Z'. Lexical terms may be separated by whitespaces, leading and trailing whitespaces in the file are also allowed. Whitespace may consist of spaces, tab characters (ASCII code 9), and line separators (a pair of ASCII 13, 10).

The structure of the valid trivial program is derived from the following productions:

- Program ::= Block
- Block ::= `{' Statements `}'
- Statements ::= Statement | Statement Statements
- Statement ::= Expression `;'
- Expression ::= *identifier* [`(' Arguments `)'] [Block]
- Arguments ::= Expression | Expression `,' Arguments

Properly formatted trivial program additionally conforms to the following rules:

- There are no empty lines.
- Tab characters are not used.
- The first character of the file is opening curly brace `{' (no preceding whitespaces), and the last character of the file is closing curly brace '}' (no trailing whitespaces).
- Each line is preceded by 4*N* space characters, where *N* is called *indentation level*.
- The first and the last lines of the program have zero indentation level.
- Lines that constitute block body and are enclosed in curly braces `{'... `}' have one more indentation level.
- No whitespace is allowed inside the line with the exception of the following two cases where a single space character is mandatory: before opening curly brace character `{' and after comma `,'.
- Lines (with the only exception of the last line) end with semicolon `;' or opening curly brace '{' characters. These characters cannot appear in the middle or at the beginning of any line (including the last one).
- Closing curly brace `}' characters appear only at the beginning of lines after indentation spaces.

See sample output section for an example of properly formatted trivial program.

Input

The input contains several valid trivial program, separated by a blank line. Size of the input file does not exceed 2000 bytes.

Output

Write to the output file properly formatted trivial code for the programs given in the input. Print a blank line between different programs.

Sample Input

```
{class(Point)
{
  member ( int ( x ) ) ; member ( int ( y ) ) ;
  member ( fun ( Length )
  {
    return ( sqrt ( sum ( sqr ( x ),sqr ( y ) ) ) );
  } ) ;
};
Main
{
  repeat
  {
    set ( n,input ( int ) ) ;
    for ( int ( i,0 ) , lt ( i,n ) , inc ( i ) )
    {
      print ( mult ( n,n ) ) ;
    };
  };
}; }
```

Sample Output

```
{
    class(Point) {
        member(int(x));
        member(int(y));
        member(fun(Length) {
            return(sqrt(sum(sqr(x), sqr(y))));
        });
    };
    Main {
        repeat {
            set(n, input(int));
            for(int(i, 0), lt(i, n), inc(i)) {
                print(mult(n, n));
            };
        };
    };
}
```

Northeastern 2003-2004

Time to Graduate

Consider the following example. A student is required to take 4 courses, mt42, cs123, cs456, and cs789. mt42 is only offered in the fall semester and has no prerequisites. Similarly, cs123 is only offered in the spring semester and has no prerequisites. cs456 is only offered in the spring semester and has both cs123 and mt42 as prerequisites. Finally, cs789 is offered in both fall and spring and has cs456 as its only prerequisite. The shortest time to graduate is 5 semesters, by taking mt42 in the fall, cs123 in the next spring, cs456 the following spring (since it is not offered in the fall) and finally cs789 the following fall.

For this problem, there are only two semesters, fall and spring. Always start counting semesters from the fall.

In addition to the fall/spring scheduling issues, there is one slight complication. In order to keep the dormitories full, each university limits the number of courses that can be taken in any semester. This limit appears as part of the input data. The third example below illustrates this issue.

Input

There are one to twenty-five data sets, followed by a final line containing only the integers `-1 -1'. A data set starts with a line containing two positive integers n , $1 \leq n \leq 12$, which is the number of courses in this data set and m , $2 \leq m \leq 6$, which is the maximum number of courses that can be taken in any single semester.

The next line contains the n course identifiers. Each is a 1-5 character string from the set {a-z, 0-9}. Following the course identifiers is the individual course information. This consists of n lines, one line for each course, containing the course identifier, semester offered (`F'=Fall, `S'=Spring, `B'=Both semesters), the number of prerequisite courses, p , $0 \leq p \leq 5$, and finally p prerequisite course identifiers. The first example data set below corresponds to the problem described above.

Output

The output contains one line for each data set, formatted as shown in the sample output.

Sample Input

```
4 6
cs123 mt42 cs456 cs789
mt42 F 0
cs123 S 0
cs456 S 2 cs123 mt42
cs789 B 1 cs456
3 6
math1 comp2 comp3
comp3 S 1 comp2
math1 S 0
comp2 F 1 math1
4 3
m10 m20 c33 c44
m10 B 0
m20 B 0
c33 B 0
c44 B 0
```

```
-1 -1
```

Sample Output

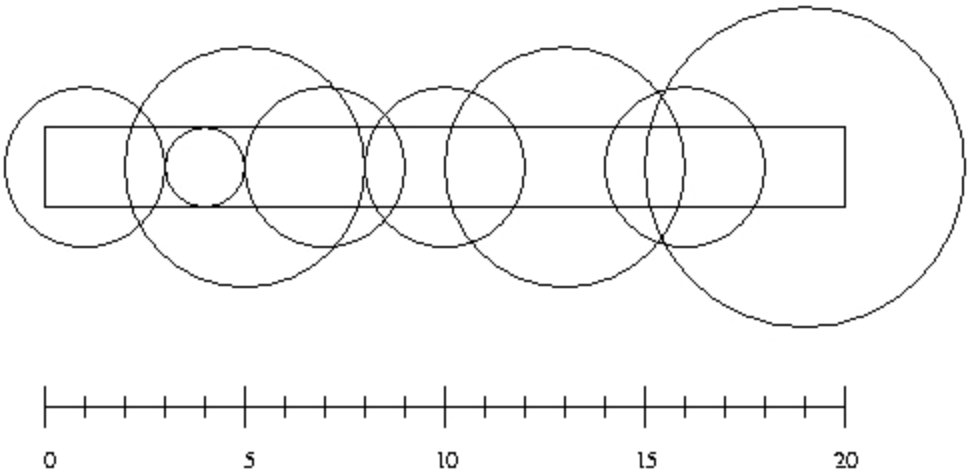
```
The minimum number of semesters required to graduate is 5.
The minimum number of semesters required to graduate is 4.
The minimum number of semesters required to graduate is 2.
```

Mid Central 2005-2006

Watering Grass

n sprinklers are installed in a horizontal strip of grass *l* meters long and *w* meters wide. Each sprinkler is installed at the horizontal center line of the strip. For each sprinkler we are given its position as the distance from the left end of the center line and its radius of operation.

What is the minimum number of sprinklers to turn on in order to water the entire strip of grass?



Input

Input consists of a number of cases. The first line for each case contains integer numbers *n*, *l* and *w* with *n* <= 10000. The next *n* lines contain two integers giving the position of a sprinkler and its radius of operation. (The picture above illustrates the first case from the sample input.)

Output

For each test case output the minimum number of sprinklers needed to water the entire strip of grass. If it is impossible to water the entire strip output -1.

Sample input

```
8 20 2
5 3
4 1
1 2
7 2
10 2
13 3
16 2
19 4
3 10 1
3 5
9 3
6 1
3 10 1
5 3
1 1
9 1
```

Sample Output

```
6
2
-1
```

(Regionals 2002 Warm-up Contest, Problem setter: Piotr Rudnicku)

Leapin' Lizards

Your platoon of wandering lizards has entered a strange room in the labyrinth you are exploring. As you are looking around for hidden treasures, one of the rookies steps on an innocent-looking stone and the room's floor suddenly disappears! Each lizard in your platoon is left standing on a fragile looking pillar, and a fire begins to rage below...

Leave no lizard behind! Get as many lizards as possible out of the room, and report the number of casualties.

The pillars in the room are aligned as a grid, with each pillar one unit away from the pillars to its east, west, north and south. Pillars at the edge of the grid are one unit away from the edge of the room (safety). Not all pillars necessarily have a lizard. A lizard is able to leap onto any unoccupied pillar that is within *d* units of his current one. A lizard standing on a pillar within leaping distance of the edge of the room may always leap to safety... but there's a catch: each pillar becomes weakened after each jump, and will soon collapse and no longer be usable by other lizards. Leaping onto a pillar does not cause it to weaken or collapse; only leaping off of it causes it to weaken and eventually collapse. Only one lizard may be on a pillar at any given time.

Input

The input file will begin with a line containing a single integer representing the number of test cases, which is at most 25. Each test case will begin with a line containing a single positive integer *n* representing the number of rows in the map, followed by a single non-negative integer *d* representing the maximum leaping distance for the lizards. Two maps will follow, each as a map of characters with one row per line. The first map will contain a digit (0-3) in each position representing the number of jumps the pillar in that position will sustain before collapsing (0 means there is no pillar there). The second map will follow, with an `L' for every position where a lizard is on the pillar and a `.' for every empty pillar. There will never be a lizard on a position where there is no pillar.

Each input map is guaranteed to be a rectangle of size $n \times m$, where $1 \leq n \leq 20$ and $1 \leq m \leq 20$. The leaping distance is always $1 \leq d \leq 3$.

Output

For each input case, print a single line containing the number of lizards that could not escape. The format should follows the samples provided below.

Note: Brute force methods examining every path will likely exceed the allotted time limit.

Sample Input

```
4
3 1
```

```
1111
1111
1111
LLLL
LLLL
LLLL
3 2
00000
01110
00000
.....
.LLL.
.....
3 1
00000
01110
00000
.....
.LLL.
.....
5 2
00000000
02000000
00321100
02000000
00000000
........
........
..LLLL..
........
........
```

Sample Output

```
Case #1: 2 lizards were left behind.
Case #2: no lizard was left behind.
Case #3: 3 lizards were left behind.
Case #4: 1 lizard was left behind.
```

Mid Central 2005-2006

Reverse and Add

The Problem

The "reverse and add" method is simple: choose a number, reverse its digits and add it to the original. If the sum is not a palindrome (which means, it is not the same number from left to right and right to left), repeat this procedure.

For example:
195 Initial number
591

786
687

1473
3741

5214
4125

9339 Resulting palindrome

In this particular case the palindrome 9339 appeared after the 4th addition. This method leads to palindromes in a few step for almost all of the integers. But there are interesting exceptions. 196 is the first number for which no palindrome has been found. It is not proven though, that there is no such a palindrome.

Task :
You must write a program that give the resulting palindrome and the number of iterations (additions) to compute the palindrome.

You might assume that all tests data on this problem:
- will have an answer ,
- will be computable with less than 1000 iterations (additions),
- will yield a palindrome that is not greater than 4,294,967,295.

The Input

The first line will have a number N with the number of test cases, the next N lines will have a number P to compute its palindrome.

The Output

For each of the N tests you will have to write a line with the following data : minimum number of iterations (additions) to get to the palindrome and the resulting palindrome itself separated by one space.

Sample Input

```
3
195
265
750
```

Sample Output

4 9339
5 45254
3 6666

Raucous Rockers

You just inherited the rights to *n* previously unreleased songs recorded by the popular group Raucous Rockers. You plan to release a set of *m* compact disks with a selection of these songs. Each disk can hold a maximum of *t* minutes of music, and a song can not overlap from one disk to another. Since you are a classical music fan and have no way to judge the artistic merits of these songs, you decide on the following criteria for making the selection:

1. The songs will be recorded on the set of disks in the order of the dates they were written - All songs on disk i must be written before those on disk i+1 for 1 <= i < m, and the songs must appear in order on each of these disks.
2. The total number of songs included will be maximized.

Input

The input consists of several datasets. The first line of the input indicates the number of datasets, then there is a blank line and the datasets separated by a blank line. Each dataset consists of a line containing the values of *1 <= n <= 1000*, *1 <= t <= 1,000,000,000* (yes, they like their power ballads) and *1 <= m <= 1,000* (integer numbers) followed by a line containing a list of the length of *n* songs, $t_1, t_2, ..., t_n$ ordered by the date they were written (Each t_i is between 1 and t minutes long, both inclusive, and $\sum_{i=1}^{n} t_i > m \times t$.)

Output

The output for each dataset consists of one integer indicating the number of songs that, following the above selection criteria will fit on *m* disks. Print a blank line between consecutive datasets.

Sample Input

```
2

10 5 3
3, 5, 1, 2, 3, 5, 4, 1, 1, 5

1 1 1
1
```

Sample Output

```
6

1
```