Se alguem tiver um link mais rapido, seria aconselhavel dar uma olhada em http://www.acm.org/contest/96/finals/problems.html, pois la tem algumas figuras. De qualquer jeito, os problemas abaixo contem a parte de texto da pagina acima.

Problem A

10-20-30

Input File: 10-20-30.in

A simple solitaire card game called 10-20-30 uses a standard deck of 52 playing cards in which suit is irrelevant. The value of a face card (king, queen, jack) is 10. The value of an ace is one. The value of each of the other cards is the face value of the card (2, 3, 4, etc.). Cards are dealt from the top of the deck. You begin by dealing out seven cards, left to right forming seven piles. After playing a card on the rightmost pile, the next pile upon which you play a card is the leftmost pile.

For each card placed on a pile, check that pile to see if one of the following three card combinations totals 10, 20, or 30.

1.the first two and last one,
2.the first one and the last two, or
3.the last three cards.

If so, pick up the three cards and place them on the bottom of the deck. For this problem, always check the pile in the order just described. Collect the cards in the order they appear on the pile and put them at the bottom of the deck. Picking up three cards may expose three more cards that can be picked up. If so, pick them up. Continue until no more sets of three can be picked up from the pile.

For example, suppose a pile contains 5 9 7 3 where the 5 is at the first card of the pile, and then a 6 is played. The first two cards plus the last card (5 + 9 + 6) sum to 20. The new contents of the pile after picking up those three cards becomes 7 3. Also, the bottommost card in the deck is now the 6, the card above it is the 9, and the one above the 9 is the 5.

If a queen were played instead of the six, 5 + 9 + 10 = 24, and 5 + 3 + 10 = 18, but 7 + 3 + 10 = 20, so the last three cards would be picked up, leaving the pile as 5 9.

If a pile contains only three cards when the three sum to 10, 20, or 30, then the pile "disappears" when the cards are picked up. That is, subsequent play skips over the position that the now-empty pile occupied. You win if all the piles disappear. You lose if you are unable to deal a card. It is also possible to have a draw if neither of the previous two conditions ever occurs.

Write a program that will play games of 10-20-30 given initial card decks as input.

Input

Each input set consists of a sequence of 52 integers separated by spaces and/or ends of line. The integers represent card values of the initial deck for that game. The first integer is the top card of the deck. Input is terminated by a single zero (0) following the last deck.

Output

For each input set, print whether the result of the game is a win, loss, or a draw, and print the number of times a card is dealt before the game results can be determined. (A draw occurs as soon as the state of the game is repeated.) Use the format shown in the "Output for the Sample Input" section.

Sample Input

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

Output for the Sample Input

```
Win : 66
Loss: 82
Draw: 73
```

Problem B

Calling Circles

Input file: circles.in

If you’ve seen television commercials for long-distance phone companies lately, you’ve noticed that many companies have been spending a lot of money trying to convince people that they provide the best service at the lowest cost. One company has "calling circles." You provide a list of people that you call most frequently. If you call someone in your calling circle (who is also a customer of the same company), you get bigger discounts than if you call outside your circle. Another company points out that you only get the big discounts for people in your calling circle, and if you change who you call most frequently, it’s up to you to add them to your calling circle.

LibertyBell Phone Co. is a new company that thinks they have the calling plan that can put other companies out of business. LibertyBell has calling circles, but they figure out your calling circle for you. This is how it works. LibertyBell keeps track of all phone calls. In addition to yourself, your calling circle consists of all people whom you call and who call you, either directly or indirectly. For example, if Ben calls Alexander, Alexander calls Dolly, and Dolly calls Ben, they are all within the same circle. If Dolly also calls Benedict and Benedict calls Dolly, then Benedict is in the same calling circle as Dolly, Ben, and Alexander. Finally, if Alexander calls Aaron but Aaron doesn’t call Alexander, Ben, Dolly, or Benedict, then Aaron is not in the circle.

You’ve been hired by LibertyBell to write the program to determine calling circl

Input

The input file will contain one or more data sets. Each data set begins with a line containing two integers, n and m. The first integer, n, represents the number of different people who are in the data set. The maximum value for n is 25. The remainder of the data set consists of m lines, each representing a phone call. Each call is represented by two names, separated by a single space. Names are first names only (unique within a data set), are case sensitive, and consist of only alphabetic characters; no name is longer than 25 letters. For example, if Ben called Dolly, it would be represented in the data file as

Ben Dolly

Input is terminated by values of zero (0) for n and m.

Output

For each input set, print a header line with the data set number, followed by a line for each calling circle in that data set. Each calling circle line contains the names of all the people in any order within the circle, separated by comma-space (a comma followed by a space). Output sets are separated by blank lines.

Sample Input

```
5 6
Ben Alexander
Alexander Dolly
Dolly Ben
Dolly Benedict
Benedict Dolly
Alexander Aaron
14 34
John Aaron
Aaron Benedict
Betsy John
Betsy Ringo
Ringo Dolly
Benedict Paul
John Betsy
John Aaron
Benedict George
Dolly Ringo
Paul Martha
George Ben
Alexander George
Betsy Ringo
Alexander Stephen
Martha Stephen
Benedict Alexander
Stephen Paul
Betsy Ringo
Quincy Martha
Ben Patrick
Betsy Ringo
Patrick Stephen
Paul Alexander
Patrick Ben
Stephen Quincy
Ringo Betsy
Betsy Benedict
Betsy Benedict
Betsy Benedict
Betsy Benedict
Betsy Benedict
Betsy Benedict
Quincy Martha
0 0
```

Output for the Sample Input

```
Calling circles for data set 1:
Ben, Alexander, Dolly, Benedict
Aaron

Calling circles for data set 2:
John, Betsy, Ringo, Dolly
Aaron
Benedict
Paul, George, Martha, Ben, Alexander, Stephen, Quincy, Patrick
```

Problem C

Cutting Corners

Input file: corner.in

Bicycle messengers who deliver documents and small items to businesses have long been part of the guerrilla transportation services in several major U.S. cities. The cyclists of Boston are a rare breed of riders. They are notorious for their speed, their disrespect for one-way streets and traffic signals, and their brazen disregard for cars, taxis, buses, and pedestrians.

Bicycle messenger services are very competitive. Billy’s Bicycle Messenger Service is no exception. To boost its competitive edge and to determine its actual expenses, BBMS is developing a new scheme for pricing deliveries that depends on the shortest route messengers can travel. You are to write a program to help BBMS determine the distances for these routes.

The following assumptions help simplify your task:

Messengers can ride their bicycles anywhere at ground level except inside buildings. Ground floors of irregularly shaped buildings are modeled by the union of the interiors of rectangles. By agreement any intersecting rectangles share interior space and are part of the same building. The defining rectangles for two separate buildings never touch, although they can be quite close. (Bicycle messengers- skinny to a fault-can travel between any two buildings. They can cut the sharpest corners and run their skinny tires right down the perimeters of the buildings.) The starting and stopping points are never inside buildings. There is always some route from the starting point to the stopping point.

Your program must be able to process several scenarios. Each scenario defines the buildings and the starting and stopping points for a delivery route. The picture below shows a bird’s-eye view of a typical scenario.

The input file represents several scenarios. Input for each scenario consists of lines as follows:

First line: n The number of rectangles describing the buildings in the scenario. 0 <= n <= 20 Second line: x1 y1 x2 y2 The x- and y-coordinates of the starting and stopping points of the route. Remaining n lines: x1 y1 x2 y2 x3 y3 The x- and y-coordinates of three vertices of a rectangle.

The x- and y-coordinates of all input data are real numbers between 0 and 1000 inclusive. Successive coordinates on a line are separated by one or more blanks. The integer -1 follows the data of the last scenario.

Output should number each scenario (Scenario #1, Scenario #2, etc.) and give the distance of the shortest route from starting to stopping point as illustrated in the Sample Output below. The distance should be written with two digits to the right of the decimal point. Output for successive scenarios should be separated by blank lines.

Sample Input

```
5
6.5   9     10  3
1     5     3   3     6  6
5.25  2     8   2     8  3.5
6     10    6   12    9  12
7     6     11  6     11 8
10    7     11  7     11 11
-1
```

Sample Output

```
Scenario #1
route distance: 7.28
```

Problem D

Bang the Drum Slowly

Input File: drum.in

Many years ago the "primary memory" of most computer systems was a magnetic drum. Read/write heads were placed so they could access data from the magnetic outer surface as the drum rotated along its horizontal axis. The following illustration gives the basic idea:

As the drum rotated, the data word under the read/write head(s) could be accessed. The drum continued to rotate after an instruction was fetched. After the execution of an instruction, the word ready to be accessed by the read/write head(s) was typically many words away. To minimize the delay that would occur if instructions to be executed sequentially were placed in consecutive words on the drum, designers of these machines frequently included the next instruction's drum address as a field in the instruction (that is, each instruction included an explicit "next instruction" address). Then "optimizing" assemblers could fill in the next instruction field with the address of the first available word ready to be read by the drum as soon as the current instruction was completed.

In this problem we want to determine the average execution times of simple programs without loops. We will consider only a single read/write head on a single track. Assume that the words on that track have sequential addresses numbered 1 through n. All instructions require the same length of time to execute, specifically the same time as it takes the drum to rotate past t words. t does not include the time to read the instruction from the drum, nor does it include the additional rotational delay that might be required if the next instruction isn't at the "optimum" address. However, these factors must be included in calculating the average execution time.

There are three types of instructions: terminal, conditional and unconditional. Terminal instructions don't have a "next instruction" address, since they terminate the execution of a program. Conditional instructions have two "next instruction" addresses, and unconditional instructions have only one "next instruction" address.

The execution time of a program run is the time taken from beginning to read the first instruction until the terminal instruction has executed. To calculate the average execution time of a program, every possible run time is weighted (multiplied) by the probability of the run. We assume equal probability of taking each path of a branch in a conditional instruction. The sum of all weighted run times is the average execution time of the program.

Assumptions

At the beginning of each test case the drum is positioned so that the instruction at location 1 is about to be read. Each program begins execution with the word in location 1. The time to read an instruction is one time unit. There will always be at least one terminal instruction, but there may be several.

Input

The input consists of a number of test cases. The input for each test case begins with a line containing integer values for n (1 < n < 50) and t (0 < t < n). This line is followed by a sequence of lines each of which contains integers representing an instruction address and zero, one, or two branch addresses. Specifically, for each instruction there is a location (between 1 and n), the number of "next instruction" addresses (0 for a terminal instruction, 1 for an unconditional instruction, and 2 for a conditional instruction), and that many branch addresses. The last instruction is followed by 0 on a line by itself. The input set is terminated by values of 0 for both n and t.

Output

For each test case, print the case number (they are numbered sequentially starting with 1) and the average execution time for the program. Execution times must be accurate to and displayed with four fractional digits.

Sample Input

```
10 5
1 0
0
10 5
1 1 6
6 0
0
10 5
1 1 7
7 0
0
10 5
1 2 7 8
7 0
8 0
0
10 6
8 0
7 1 3
3 0
1 2 7 8
0
0 0
```

Output for the Sample Input

```
Case 1. Execution time = 6.0000
Case 2. Execution time = 21.0000
Case 3. Execution time = 12.0000
Case 4. Execution time = 12.5000
Case 5. Execution time = 26.5000
```

Problem E

Pattern Matching Prelims

Input file: pattern.in

Some algorithms for optical character recognition compare a scanned image with templates of "perfect" characters. Part of the difficulty with such comparisons is deciding where to start the comparison. This is because the characters in the scanned image are subject to noise and distortion, resulting in changes in size, position, and orientation. A procedure that is sometimes used to deal with changes in position matches the "center of gravity" of the scanned character and the templates against which it is compared. In this problem you are to determine the "centers of gravity" of scanned images of characters.

For our purposes, a scanned image will be a rectangular array of real numbers, each of which represents the gray-scale value of a point in a scanned image. Each gray-scale value will be between 0 (representing a totally white region) and 1 (representing a totally black region). The array will have no more than 25 rows and 25 columns.

The center of gravity is a particular element of an array. Suppose a center of gravity is in the ith row and jth column. Then the difference between the sum of the elements of the two array portions above and below the ith row is minimal. Likewise, the difference of the sums of the elements in the two array portions to the left and to the right of the jth column is minimal.

Consider the array shown below, which might have resulted from scanning a lower case "o." The center of gravity for this array is uniquely in row 3, column 3. The difference of the sum of the elements in each array portion formed by ignoring the third row is 0.1 (the sums are 5.55 and 5.65). The difference of the sum of each array portion formed by ignoring the third column is 0.0 (the sums are both 5.60).

Input

The input will consist of a sequence of scanned character images. Input for each image will begin with two integers specifying the number of rows and columns in the scanned data. This will be immediately followed by the scanned gray-scale data given in row-major order. A pair of zeroes will follow the data for the last input image.

Output

For each input character image, display its number (they are sequentially numbered starting with 1) and the row and column corresponding to the center of gravity. If there is more than one center of gravity, the one with the largest row and column should be displayed. The sample that follows illustrates a reasonable output format.

Sample Input

```
5 5
0.1 0.2 0.1 0.2 0.1
0.1 0.2 0.3 0.1 0.1
0.2 0.3 0.1 0.1 0.3
0.4 0.1 0.1 0.1 0.2
0.2 0.2 0.3 0.3 0.1

5 10
0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3
0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.6

0 0
```

Output for the Sample Input

```
Case 1: center at (3, 3)
Case 2: center at (4, 6)
```

Problem F

Nondeterministic Trellis Automata

Input file: trellis.in

A nondeterministic trellis automaton (NTA) is a kind of parallel machine composed of identical finite-state processors arranged in an infinite triangular trellis. The top, or apex, of the triangle is a single processor. The next row has two processors and each successive row of an NTA has one more processor than the row above it. Each processor in an NTA is connected to two children in the row below it. Computation in an NTA occurs bottom up; the state of each processor in a row is based on the state of the processor's children and a transition table. The input to an NTA is the initial configuration of one row of processors. The input is specified by a string that gives the initial state of each processor in a row so that an n-character string specifies the initial configuration for a row of n processors. Computation proceeds up the NTA to the apex by nondeterministically calculating the state of each processor in a row based on the transition table and the state of the processor's children in the row below.

Some states are identified as accepting states. Some transitions are computed nondeterministically. An input is accepted if some computation puts the apex processor into an accepting state. An input is rejected if no computation puts the apex processor into an accepting state. For example, the table below shows transitions for a 3-state NTA. States are labeled by characters "a", "b", and "c"; the only accepting state is "c".

The diagram below shows two computations for the input "bba". The computation on the left rejects the input since the state of the apex is "a"; but the computation on the right accepts the input since the state of the apex is "c". Since some computation results in an accepting state for the apex, the input "bba" is accepted by the NTA. The input "bbb" would be rejected by this NTA since the only computation results in the state "a" for the apex.

Input

The states (and inputs) of an NTA are consecutive lowercase letters. Thus the states for a 5-state NTA are "a", "b", "c", "d", and "e". Accepting states are grouped at the end of the letters so that if a 5-state NTA has two accepting states, the accepting states are "d" and "e".

The input for your program is a sequence of NTA descriptions and initial configurations. An NTA description is given by the number of states n followed by the number of accepting states on one line separated by whitespace. The n x n transition table follows in row-major order; each transition string is given on a separate line. Each NTA description is followed by a sequence of initial configurations, one per line. A line of "#" terminates the sequence of initial configurations for that NTA. An NTA description in which the number of states is zero terminates the input for your program.

NTAs will have at most 15 states, initial configuration will be at most 15 characters.

Output

For each NTA description, print the number of the NTA (NTA 1, NTA 2, etc.). For each initial configuration of an NTA print the word "accept" or "reject" followed by a copy of the initial configuration.

Sample Input

```
3 1
a
a
c
ca
a
b
c
b
a
bba
aaaaa
abab
babbba
a
baaab
abbbaba
baba
bcbab
#
3 2
ab
a
c
a
ab
b
c
b
ab
abc
cbc
#
0 0
```

Output for the Sample Input

```
NTA 1
accept  bba
reject  abab
accept  babbba
reject  a
reject  aaaaa
accept  baaab
accept  abbbaba
accept  baba
reject  bcbab

NTA 2
reject  abc
accept  cbc
```

Problem G

Trucking

Input file: truck.in

Allied Container Movers (ACM) is a trucking company that provides overnight freight delivery. ACM has a distribution network with several intermediate container processing centers (ICPCs). At an ICPC, an incoming trailer is unloaded at a stripping door. Freight destined for that center is simply acknowledged as received. Onward shipments are distributed to relay doors based on their next destinations, where they are loaded onto waiting trailers.

Each ICPC has several stripping doors for unloading incoming trailers. When the number of trailers to be stripped exceeds the number of stripping doors, incoming trailers are queued until a door is available. A single trailer may have freight for several different ICPCs. Trailers with freight destined only for the local ICPC receive a lower priority for access to a stripping door than trailers with relay freight. In a similar fashion, trailers with relay freight having a closer final destination have lower priority than trailers with relay freight having a distant final destination. The time to unload a container and, if necessary, reload all its shipments onto one or more relay trailers is always 2 hours regardless of the size and number of shipments. A relay trailer is immediately routed to its next destination when it is full or when all shipments for the day expected for that destination have been loaded onto the trailer. Shipments are measured as a percent of a trailer volume and may be subdivided to the nearest percent in order to fill the trailer. There is no delay between a trailer departing and another trailer becoming available at the relay or stripping doors. There is never a shortage of trailers for onward distribution.

In order to help ACM assess the efficiency of their network, you must write a program to determine the average time a trailer waits for access to a stripping door and identify those shipments which will not arrive in their entirety at their intermediate or final destinations on time.

Input

The input describes a possibly disjoint subset of the network's ICPCs and traffic patterns that must be analyzed.

The first line of the input contains an integer n which specifies the number of ICPC descriptions to be processed, 1 <= n <= 100. This is followed by n descriptions, each describing one ICPC. Each description begins with a line containing three integers, c s d, where c is the center number, 0 <= c <= 99, s is the number of stripping doors at center c, 0 <= s <= 10, and d is the number of relay doors at center c, 0 <= d <= 10. There then follow d lines, one for each relay door. Each of these lines contains three integers, r v l, where r is the relay center's number, 0 <= r <= 99, v is the total volume of shipments to that center for the day expressed as a percentage of trailer volume, 0 <= v <= 900 and l is the latest acceptable time for shipments to arrive at center r, expressed as minutes since the start of the day, 0 <= l <= 1440. (v > 100 indicates that more than a single trailer must be used.)

The second part of the input describes some of the day's traffic. This part begins with one integer m on a line by itself indicating the number of trailer arrival records that follow, 1 <= m <= 100. Each record begins with a line containing three integers, a c s, where a is the trailer's arrival time expressed as minutes since the start of the day, 0 <= a <= 1440, at center number c, and s is the number of shipments in the trailer, 0 <= s <= 10. Then all s shipments are described by s lines of 5 integers, i o r v t, representing the shipment identification code i, 0 <= The second part of the input describes some of the day's traffic. This part begins with one integer m on a line by itself indicating the number of trailer arrival records that follow, 1 <= m <= 100. Each record begins with a line containing three integers, a c s, where a is the trailer's arrival time expressed as minutes since the start of the day, 0 <= a <= 1440, at center number c, and s is the number of shipments in the trailer, 0 <= s <= 10. Then all s shipments are described by s lines of 5 integers, i o r v t, representing the shipment identification code i, 0 <= i <= 99, the origin and next relay center numbers o and r respectively, the volume of the shipment v as a percentage of trailer volume and the time t taken to travel from center c to destination r measured in minutes. t is zero if c equals r. Arrival records are in order of ascending values of a. No two records have the same pair (a,c). All center numbers used as values for c and r will have an appropriate corresponding definition in the first part of the input, though the center numbers used for o need not.

Output

For each of the n ICPCs, your program must write out a line describing the average wait time for stripping doors in the appropriate one of these two forms:

The average wait for a stripping door at ICPC c is ###.# minutes.
There is no wait for a stripping door at ICPC c.

The average wait time is affected only by trailers which wait at least one minute for a stripping door.

Your program should then list all the shipments any part of which will not arrive at their intermediate or final destinations by any of the latest arrival times given along the route. This report should appear in columns headed as shown:

The late shipments are:
Id Origin Destination Volume

Sample Input:

```
2
0 1 1
8  40  600
8 3 4
6 115 1200
2  95 1260
10 100 1440
4  55 1380
7
500 0 1
17  11  8  40   80
700 8 3
24  11  8  45    0
18  11  6  40  120
23  11 10  15  600
720 8 1
16   3  8 100    0
750 8 2
4  15  2  50  180
7  15  6  50  120
760 8 4
14   3  4  20  300
27   3  2  20  180
33   3 10  35  600
16   3  6  25  120
780 8 2
12   9  2  25  180
15   9  4  35  300
800 8 1
19  18 10  50  600
```

Output for the Sample Input:

```
There is no wait for a stripping door at ICPC 0.
The average wait for a stripping door at ICPC 8 is 63.3 minutes.

The late shipments are:
Id Origin Destination Volume
17     11           8     40
23     11          10     15
33      3          10     35
19     18          10     50
```