UW-Madison ICPC
Graph Practice Problems

October 20, 2004

In addition to the problems on the following pages, you may also attempt the following problems on the Valladolid programming contest site (http://acm.uva.es/).

- 157: Route finding (http://acm.uva.es/p/v1/157.html)
- 315: Network (http://acm.uva.es/p/v3/315.html)
- 341: Non-Stop Travel (http://acm.uva.es/p/v3/341.html)
- 459: Graph Connectivity (http://acm.uva.es/p/v4/459.html)

.

The Twentieth Annual ACM International Collegiate

Programming Contest Finals

sponsored by

Microsoft®

Problem

Cutting Corners

Input file: corner.in

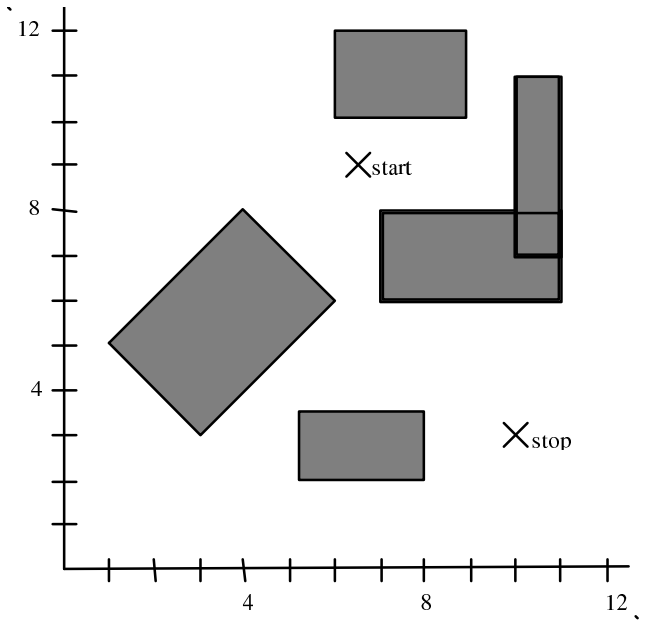
Bicycle messengers who deliver documents and small items to businesses have long been part of the guerrilla transportation services in several major U.S. cities. The cyclists of Boston are a rare breed of riders. They are notorious for their speed, their disrespect for one-way streets and traffic signals, 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 modelled 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.)
- For a given route, the starting and stopping points are never inside buildings.
- Their is always some route from a starting point to a stopping point.

Your program must be able to process several scenarios. Each scenario is a bird's-eye snapshot showing the buildings and the starting and ending points for a delivery route. The picture below is 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. There are from 0 to 100 such rectangles inclusive.
Second line:	$x_1\ y_1\ x_2\ y_2$	The x- and y-coordinates of the starting and stopping points of the route.
Remaining n lines:	$x_1\ y_1\ \ x_2\ y_2\ \ x_3\ y_3$	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.

To avoid problems with real precision, the input data set restricts all coordinates to be between 0 and 1000 inclusive. The interior enclosed by any two intersecting rectangles will be at least large enough to contain a square of .01 unit on a side. In addition, two buildings that do not intersect will be at least .01 unit apart.

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.

The following scenario is described by the Sample Input below.

The 2000 24th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem A

Abbott's Revenge

Input File: abbott.in

The 1999 World Finals Contest included a problem based on a "dice maze." At the time the problem was written, the judges were unable to discover the original source of the dice maze concept. Shortly after the contest, however, Mr. Robert Abbott, the creator of numerous mazes and an author on the subject, contacted the contest judges and identified himself as the originator of dice mazes. We regret that we did not credit Mr. Abbott for his original concept in last year's problem statement. But we are happy to report that Mr. Abbott has offered his expertise to this year's contest with his original and unpublished "walk-through arrow mazes."

As are most mazes, a walk-through arrow maze is traversed by moving from intersection to intersection until the goal intersection is reached. As each intersection is approached from a given direction, a sign near the entry to the intersection indicates in which directions the intersection can be exited. These directions are always left, forward or right, or any combination of these.

Figure 1 illustrates a walk-through arrow maze. The intersections are identified as "(row, column)" pairs, with the upper left being (1,1). The "Entrance" intersection for Figure 1 is (3,1), and the "Goal" intersection is (3,3). You begin the maze by moving north from (3,1). As you walk from (3,1) to (2,1), the sign at (2,1) indicates that as you approach (2,1) from the south (traveling north) you may continue to go only forward. Continuing forward takes you toward (1,1). The sign at (1,1) as you approach from the south indicates that you may exit (1,1) only by making a right. This turns you to the east now walking from (1,1) toward (1,2). So far there have been no choices to be made. This is also the case as you continue to move from (1,2) to (2,2) to (2,3) to (1,3). Now, however, as you move west from (1,3) toward (1,2), you have the option of continuing straight or turning left. Continuing straight would take you on toward (1,1), while turning left would take you south to (2,2). The actual (unique) solution to this maze is the following sequence of intersections: (3,1) (2,1) (1,1) (1,2) (2,2) (2,3) (1,3) (1,2) (1,1) (2,1) (2,2) (1,2) (1,3) (2,3) (3,3).

You must write a program to solve valid walk-through arrow mazes. Solving a maze means (if possible) finding a route through the maze that leaves the Entrance in the prescribed direction, and ends in the Goal. This route should not be longer than necessary, of course.

Input

The input file will consist of one or more arrow mazes. The first line of each maze description contains the name of the maze, which is an alphanumeric string of no more than 20 characters. The next line contains, in the following order, the starting row, the starting column, the starting direction, the goal row, and finally the goal column. All are delimited by a single space. The maximum dimensions of a maze for this problem are 9 by 9, so all row and column numbers are single digits from 1 to 9. The starting direction is one of the characters N, S, E or W, indicating north, south, east and west, respectively.

All remaining input lines for a maze have this format: two integers, one or more groups of characters, and a sentinel asterisk, again all delimited by a single space. The integers represent the row and column, respectively, of a maze intersection. Each character group represents a sign at that intersection. The first character in the group is N, S, E or W to indicate in what direction of travel the sign would be seen. For example, S indicates that this is the sign that is seen when travelling south. (This is the sign posted at the north entrance to the intersection.) Following this first direction character are one to three arrow characters. These can be L, F or R indicating left, forward, and right, respectively.

The list of intersections is concluded by a line containing a single zero in the first column. The next line of the input starts the next maze, and so on. The end of input is the word END on a single line by itself.

Output

For each maze, the output file should contain a line with the name of the maze, followed by one or more lines with either a solution to the maze or the phrase "No Solution Possible". Maze names should start in column 1, and all other lines should start in column 3, i.e., indented two spaces. Solutions should be output as a list of intersections in the format "(R,C)" in the order they are visited from the start to the goal, should be delimited by a single space, and all but the last line of the solution should contain exactly 10 intersections.

The first maze in the following sample input is the maze in Figure 1.

Sample Input	Output for the Sample Input

SAMPLE
3 1 N 3 3
1 1 WL NR *
1 2 WLF NR ER *
1 3 NL ER *
2 1 SL WR NF *
2 2 SL WF ELF *
2 3 SFR EL *
0
NOSOLUTION
3 1 N 3 2
1 1 WL NR *
1 2 NL ER *
2 1 SL WR NFR *
2 2 SR EL *
0
END
	SAMPLE
	  (3,1) (2,1) (1,1) (1,2) (2,2) (2,3) (1,3) (1,2) (1,1) (2,1)
	  (2,2) (1,2) (1,3) (2,3) (3,3)
	NOSOLUTION
	  No Solution Possible

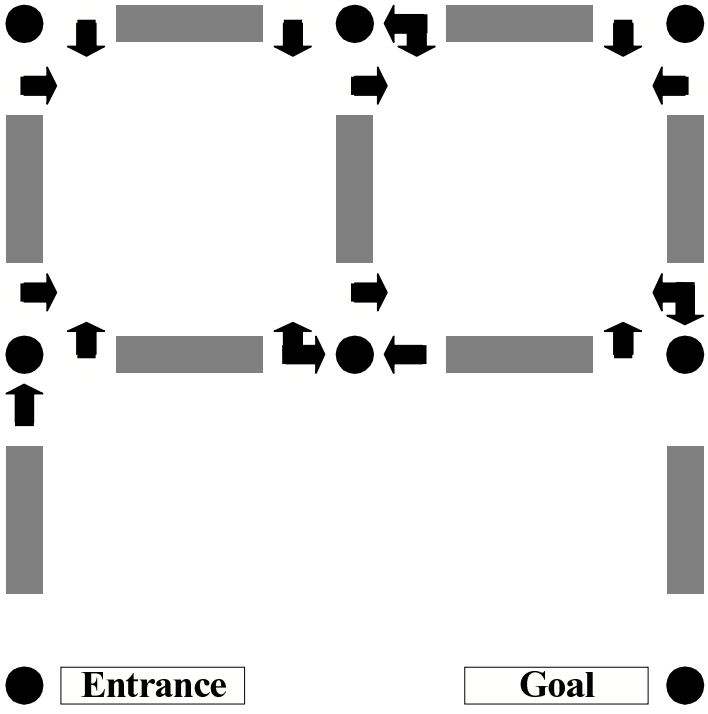


Figure 1: An Example Walk-Through Arrow Maze

The 2002 26th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem B

Undecodable Codes

Input: codes.in

Phil Oracle has a unique ability that makes him indispensable at the National Spying Agency. His colleagues can bring him any new binary code and he can tell them immediately whether the code is uniquely decodable or not. A *code* is the assignment of a unique sequence of characters (a *codeword*) to each character in an *alphabet*. A binary code is one in which the codewords contain only zeroes and ones. For example, here are two possible binary codes for the alphabet {**a**,**c**,**j**,**l**,**p**,**s**,**v**}.

	Code 1	Code 2
a	1	010
c	01	01
j	001	001
l	0001	10
p	00001	0
s	000001	1
v	0000001	101

The *encoding* of a string of characters from an alphabet (the *cleartext*) is the concatenation of the codewords corresponding to the characters of the cleartext, in order, from left to right. A code is *uniquely decodable* if the encoding of every possible cleartext using that code is unique. In the example above, Code 1 is uniquely decodable, but Code 2 is not. For example, the encodings of the cleartexts "**pascal**" and "**java**" are both **001010101010**. Even shorter encodings that are not uniquely decodable are **01** and **10**.

While the agency is very proud of Phil, he unfortunately gives only "yes" or "no" answers. Some members of the agency would prefer more tangible proof, especially in the case of codes that are not uniquely decodable. For this problem you will deal only with codes that are *not* uniquely decodable. For each of these codes you must determine the single encoding having the minimum length (measured in bits) that is ambiguous because it can result from encoding each of two or more different cleartexts. In the case of a tie, choose the encoding which comes first lexicographically.

Input

One or more codes are to be tested. The input for each code begins with an integer m, $1 \le m \le 20$, on a line by itself, where m is the number of binary codewords in the code. This is followed by m lines each containing one binary codeword string, with optional leading and trailing whitespace. No codeword will contain more than 20 bits.

The input is terminated by the number zero on a line by itself.

Output

For each code, display the sequential code number (starting with 1), the length of the shortest encoding that is not uniquely decodable, and the shortest encoding itself, with ties broken as previously described. The encoding must be displayed with 20 bits on each line except the last, which may contain fewer than 20 bits. Place a blank line after the output for each code. Use the format shown in the samples below.

Sample Input	Output for the Sample Input

3
  0
  01
  10
5
 0110
  00
  111
  001100
  110
5
  1
  001
  0001
  00000000000000000001
  10000000000000000000
0
	Code 1: 3 bits
	010
	
	Code 2: 9 bits
	001100110
	
	Code 3: 21 bits
	10000000000000000000
	1

The 2002 26th Annual **acm** International Collegiate

Programming Contest World Finals

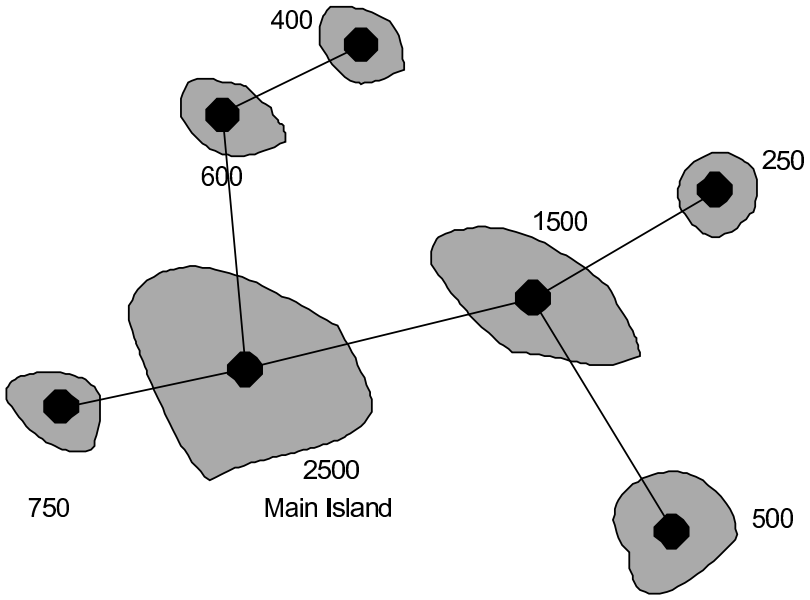
sponsored by IBM

Problem E

Island Hopping

Input: islands.in

The company Pacific Island Net (PIN) has identified several small island groups in the Pacific that do not have a fast internet connection. PIN plans to tap this potential market by offering internet service to the island inhabitants. Each groups of islands already has a deep-sea cable that connects the main island to the closest internet hub on the mainland (be it America, Australia or Asia). All that remains to be done is to connect the islands in a group to each other. You must write a program to help them determine a connection procedure.



For each island, you are given the position of its router and the number of island inhabitants. In the figure, the dark dots are the routers and the numbers are the numbers of inhabitants. PIN will build connections between pairs of routers such that every router has a path to the main island. PIN has decided to build the network such that the total amount of cable used is minimal. Under this restriction, there may be several optimal networks. However, it does not matter to PIN which of the optimal networks is built.

PIN is interested in the average time required for new customers to access the internet, based on the assumption that construction on all cable links in the network begins at the same time. Cable links can be constructed at a rate of one kilometer of cable per day. As a result, shorter cable links are completed before the longer links. An island will have internet access as soon as there is a path from the island to the main island along completed cable links. If m_i is the number of inhabitants of the i^{th} island and t_i is the time when the island is connected to the internet, then the average connection time is:

$$\frac{\sum t_i * m_i}{\sum m_i}$$

Input

The input consists of several descriptions of groups of islands. The first line of each description contains a single positive integer n, the number of islands in the group ($n \leq 50$). Each of the next n lines has three integers x_i, y_i, m_i, giving the position of the router (x_i, y_i) and number of inhabitants m_i ($m_i > 0$) of the islands. Coordinates are measured in kilometers. The first island in this sequence is the main island.

The input is terminated by the number zero on a line by itself.

Output

For each group of islands in the input, output the sequence number of the group and the average number of days until the inhabitants are connected to the internet. The number of days should have two digits to the right of the decimal point. Use the output format in the sample given below.

Place a blank line after the output of each test case.

Sample Input	Output for the Sample Input
7 11 12 2500 14 17 1500 9 9 750 7 15 600 19 16 500 8 18 400 15 21 250 0	Island Group: 1 Average 3.20

The 2003 27th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem A

Building Bridges

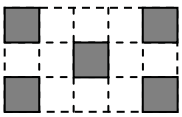
Input File: bridges.in

The City Council of New Altonville plans to build a system of bridges connecting all of its downtown buildings together so people can walk from one building to another without going outside. You must write a program to help determine an optimal bridge configuration.

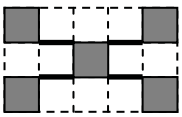
New Altonville is laid out as a grid of squares. Each building occupies a connected set of one or more squares. Two occupied squares whose corners touch are considered to be a single building and do not need a bridge. Bridges may be built only on the grid lines that form the edges of the squares. Each bridge must be built in a straight line and must connect exactly two buildings.

For a given set of buildings, you must find the minimum number of bridges needed to connect all the buildings. If this is impossible, find a solution that minimizes the number of disconnected groups of buildings. Among possible solutions with the same number of bridges, choose the one that minimizes the sum of the lengths of the bridges, measured in multiples of the grid size. Two bridges may cross, but in this case they are considered to be on separate levels and do not provide a connection from one bridge to the other.

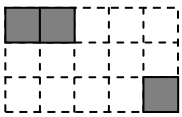
The figure below illustrates four possible city configurations. City 1 consists of five buildings that can be connected by four bridges with a total length of 4. In City 2, no bridges are possible, since no buildings share a common grid line. In City 3, no bridges are needed because there is only one building. In City 4, the best solution uses a single bridge of length 1 to connect two buildings, leaving two disconnected groups (one containing two buildings and one containing a single building).



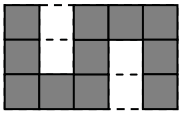
City 1



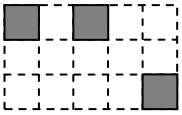
City 1
with bridges



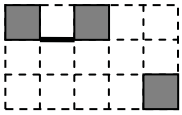
City 2
No bridges are possible



City 3
No bridges are needed



City 4



City 4
with bridges

Input

The input data set describes several rectangular cities. Each city description begins with a line containing two integers r and c, representing the size of the city on the north-south and east-west axes measured in grid lengths ($1 \le r \le 50$ and $1 \le c \le 50$). These numbers are followed by exactly r lines, each consisting of c hash ("#") and dot (".") characters. Each character corresponds to one square of the grid. A hash character corresponds to a square that is occupied by a building, and a dot character corresponds to a square that is not occupied by a building.

The input data for the last city will be followed by a line containing two zeros.

Output

For each city description, print two or three lines of output as shown below. The first line consists of the city number. If the city has fewer than two buildings, the second line is the sentence "No bridges are needed." If the city has two or more buildings but none of them can be connected by bridges, the second line is the sentence "No bridges are possible." Otherwise, the second line is "*N* bridges of total length *L*" where *N* is the number of bridges and *L* is the sum of the lengths of the bridges of the best solution. (If *N* is 1, use the word "bridge" rather than "bridges.") If the solution leaves two or more disconnected groups of buildings, print a third line containing the number of disconnected groups.

Print a blank line between cases. Use the output format shown in the example.

Sample Input	Output for the Sample Input
3 5 #...# ..#.. #...# 3 5 ##... ....# 3 5 #.### #.#.# ###.# 3 5 #.#.. ....# 0 0	City 1 4 bridges of total length 4 City 2 No bridges are possible. 2 disconnected groups City 3 No bridges are needed. City 4 1 bridge of total length 1 2 disconnected groups

Problem J
Toll
Input File: toll.in

Sindbad the Sailor sold 66 silver spoons to the Sultan of Samarkand. The selling was quite easy; but delivering was complicated. The items were transported over land, passing through several towns and villages. Each town and village demanded an entry toll. There were no tolls for leaving. The toll for entering a *village* was simply one item. The toll for entering a *town* was one piece per 20 items carried. For example, to enter a town carrying 70 items, you had to pay 4 items as toll. The towns and villages were situated strategically between rocks, swamps and rivers, so you could not avoid them.

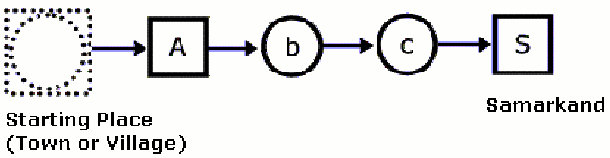


Figure 1: To reach Samarkand with 66 spoons, traveling through a town followed by two villages, you must start with 76 spoons.

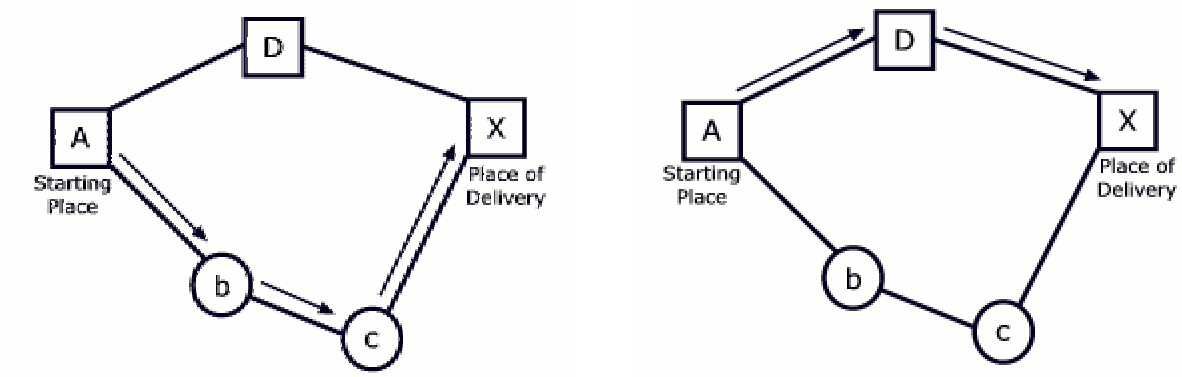


Figure 2: The best route to reach X with 39 spoons, starting from A, is A→b→c→X, shown with arrows in the figure on the left. The best route to reach X with 10 spoons is A→D→X, shown in the figure on the right. The figures display towns as squares and villages as circles.

Predicting the tolls charged in each village or town is quite simple, but finding the best route (the cheapest route) is a real challenge. The best route depends upon the number of items carried. For numbers up to 20, villages and towns charge the same. For large numbers of items, it makes sense to avoid towns and travel through more villages, as illustrated in Figure 2.

You must write a program to solve Sindbad's problem. Given the number of items to be delivered to a certain town or village and a road map, your program must determine the total number of items required at the beginning of the journey that uses a cheapest route.

Input

The input consists of several test cases. Each test case consists of two parts: the roadmap followed by the delivery details.

The first line of the roadmap contains an integer n, which is the number of roads in the map ($0 \le n$). Each of the next n lines contains exactly two letters representing the two endpoints of a road. A capital letter represents a town; a lower case letter represents a village. Roads can be traveled in either direction.

Following the roadmap is a single line for the delivery details. This line consists of three things: an integer p ($0 < p \le 1000$) for the number of items that must be delivered, a letter for the starting place, and a letter for the place of delivery. The roadmap is always such that the items can be delivered.

The last test case is followed by a line containing the number -1.

Output

The output consists of a single line for each test case. Each line displays the case number and the number of items required at the beginning of the journey. Follow the output format in the example given below.

Sample Input	Output for the Sample Input
1 a Z 19 a Z 5 A D D X A b b c c X 39 A X -1	Case 1: 20 Case 2: 44