UCF Programming Team Practice
October 20, 2018
Developmental Team
Individual Problem Set

Problem Name	**Filename**
Balance Point	balance
Blast-a-mon Battle Plan	blastamon
Bunnies	bunnies
Underground Cables	cables
Candy Store	candy
Human Cannonball Run	cannonball
COW	cow
Best Schedule	schedule
Blooscreins of Death	spaceship
Universe in the Cupboard	stars
Air Strike	strike

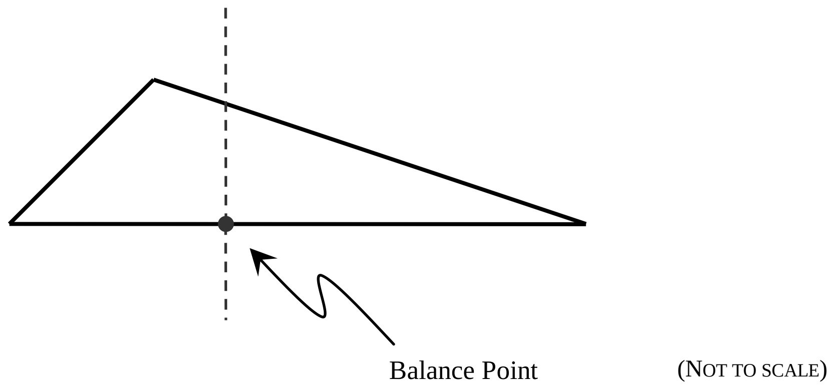
UCF Local Contest — September 5, 2009

Balance Point

filename: `balance`

Given an object, it's often fun to find the point at which you can balance it without it toppling over to one side. This is typically a very difficult problem, but we'd like to at least solve the problem in two dimensions for a simple shape: a scalene triangle. In particular, we will define the balance point of a scalene triangle as follows:

Find the longest side of the triangle. Then, determine the perpendicular line to this side that divides the triangle's area into two equal parts. The point at which this line intersects the longest side is known as the triangle's balance point.



The Problem:

Given the Cartesian coordinates of the three points of a scalene triangle, determine the x and y coordinates of its balance point (to 2 decimal places).

The Input:

There will be multiple triangles in the input file. The first input line contains a positive integer *n*, indicating the number of triangles to be processed. The triangles will be on the following *n* input lines, each on a separate line. Each triangle contains three pairs of x-y coordinates, denoting the three points of the triangle. Each of these six values will be real numbers in between -100 and 100, inclusive, separated by spaces. Assume that all three points are distinct and that they are not collinear, i.e., assume the points form a triangle.

The Output:

At the beginning of each test case, output "`Triangle #`*`t`*` Balance Point: `", where *`t`* is the test case number (starting from 1). Following this header, for each triangle, print the x and y coordinates of that triangle's balance point, rounded to two decimal places, in parentheses, separated by a comma. To clarify “rounded to two decimal places”: the output for 1.274 should be 1.27, for 1.275 should be 1.28, and for 1.276 should be 1.28.

Follow the format illustrated in Sample Output.

Sample Input:

```
2
0.0 0.0 5.0 0.0 1.8 2.4
0.0 0.0 0.0 5.0 -2.4 3.2
```

Sample Output:

```
Triangle #1 Balance Point: (2.17,0.00)
Triangle #2 Balance Point: (0.00,2.83)
```

Blast-a-mon Battle Plan

Filename: BLASTAMON

Your friend is nuts about the latest trading card game known as Blast-a-mon. Like many other card games, Blast-a-mon involves cute little monsters that engage in battles with one another. They use their extraordinary fighting powers to cause massive amounts of damage to their opponents and gravely endanger the population of whatever city they happen to be fighting in.

When your friend isn't boring you with the list of the latest 127 cards that have come out this week, he's bugging you to play a game or two with him. You don't really care about this whole Blast-a-mon thing, but you figure you can be nice to your friend until the fad passes. You decide to write a program to help you plan your Blast-a-mon strategy, so you don't lose horribly every time you play.

The Problem:

Given a list of Blast-a-mon characters and their hit points (amount of damage they can take before falling unconscious), print out the list in order of lowest to highest hit points.

The Input:

There will be multiple Blast-a-mon card decks to analyze. Each deck will begin with a single positive integer, *n,* on the first line, indicating how many Blast-a-mon cards are in the deck. Following this will be *n* sets of two lines, each pair of lines representing a Blast-a-mon card. The first line contains the name of the Blast-a-mon card. The name will be no longer than 20 characters and will consist of letters only, with no leading or trailing spaces. The second line contains a positive integer between 1 and 1000, inclusive, indicating the number of hit points that the Blast-a-mon character has. In any deck, each Blast-a-mon character will have a different number of hit points. Input will be terminated by a value of zero for the number of cards in the deck (this deck should not be processed).

The Output:

For each deck, print all of the cards' names in ascending order according to hit points. Leave one blank line after the output for each deck.

Sample Input:

```
3
Roar
10
Squeak
1
Bark
5
2
Megamon
1000
Minimon
1
0
```

Sample Output:

```
Squeak
Bark
Roar

Minimon
Megamon
```

Bunnies

Filename: bunnies

The Problem

Jill, the new lab assistant has made new cages for the lab rabbits, Peter and Cottontail. Unfortunately, these new cages are more like mazes with walls in every direction. Help Arup determine if the rabbits are in the same section of the cage or different sections. Each rabbit can hop one square directly up, down, right or left. They can hop as many times as they want from one free square to another. Each rabbit must stay on the grid.

The Input

Each input will start with a single integer T ($1 \le T \le 100$) on the first line. The number T will denote the number of test cases that follow. Each test case will begin with two integers, R and C ($1 \le R, C \le 10$) separated by a space. Each of the next R lines will contain C characters of either '_','#', 'P' or 'C' (Quotes for clarity). This will form a grid that represents the cage. A '_' represents a cell free of obstructions, '#' represents a wall, 'P' is Peter's location and 'C' is Cottontail's location. Each grid is guaranteed to have one and only one P and C.

The Output

For each test case output a single line containing "yes" if Peter and Cottontail are in the same section of the cage and "no" if they are not in the same section of the cage.

Sample Input

```
4
2 2
P#
#C
2 2
P_
C_
8 7
__P____
####_##
_____#_
_____#C
##_###_
_____#_
___#_#_
___#___
5 7
__P____
####_##
_____#_
_____#C
##_###_
```

Sample Output

```
no
yes
yes
no
```

J: Underground Cables

A city wants to get rid of their unsightly power poles by moving their power cables underground. They have a list of points that all need to be connected, but they have some limitations. Their tunneling equipment can only move in straight lines between points. They only have room for one underground cable at any location except at the given points, so no two cables can cross.

Given a list of points, what is the least amount of cable necessary to make sure that every pair of points is connected, either directly, or indirectly through other points?

Input

There will be several test cases in the input. Each test case will begin with an integer **N** (2 ≤ **N** ≤ 1,000), which is the number of points in the city. On each of the next **N** lines will be two integers, **X** and **Y** (-1,000 ≤ **X**,**Y** ≤ 1,000), which are the (**X**,**Y**) locations of the **N** points. Within a test case, all points will be distinct. The input will end with a line with a single 0.

Output

For each test case, output a single real number, representing the least amount of cable the city will need to connect all of its points. Print this number with exactly two decimal places, rounded. Print each number on its own line with no spaces. Do not print any blank lines between answers.

Sample Input

```
4
0 0
0 10
10 0
10 10
2
0 0
10 10
0
```

Sample Output

```
30.00
14.14
```

A: Candy Store

You are walking with a friend, when you pass a candy store. You make a comment about how unhealthy their wares are. Your friend issues an interesting challenge: who can be the unhealthiest? Both of you will go into the store with the same amount of money. Whoever buys candy with the most total calories wins!

Since you're a smart computer scientist, and since you have access to the candy store's inventory, you decide not to take any chances. You will write a program to determine the most calories you can buy. The inventory tells you the price and calories of every item. It also tells you that there is so much in stock that you can buy as much of any kind of candy as you want. You can only buy whole pieces of candy.

Input

There will be multiple test cases in the input. Each test case will begin with a line with an integer ***n*** ($1 \le n \le 5{,}000$), and an amount of money ***m*** ($\$0.01 \le m \le \$100.00$), separated by a single space, where ***n*** is the number of different types of candy for sale, and ***m*** is the amount of money you have to spend. The monetary amount ***m*** will be expressed in dollars with exactly two decimal places, and with no leading zeros unless the amount is less than one dollar. There will be no dollar sign. Each of the next ***n*** lines will have an integer ***c*** ($1 \le c \le 5{,}000$) and an amount of money ***p*** ($\$0.01 \le p \le \$100.00$), separated by a single space, where ***c*** is the number of calories in a single piece of candy, and ***p*** is the price of a single piece of candy, in dollars and in the same format as ***m***. The input will end with a line containing '**0 0.00**'.

Output

For each test case, output a single integer, indicating the maximum amount of calories you can buy with up to ***m*** dollars. Output no spaces, and do not separate answers with blank lines.

Sample Input	Sample Output

```
2 8.00
700 7.00
199 2.00
3 8.00
700 7.00
299 3.00
499 5.00
0 0.00
```
	```
	796
	798
	```

Problem E
Human Cannonball Run

You are a world-famous circus performer, a human cannonball. This means that you climb into a big, fake cannon and launch yourself great distances to delight young and old alike. Today, you're not alone. You are at the international human cannonball conference and exposition, where hundreds of similar circus performers have gathered together to share their experiences and practice their craft. While you normally have just one cannon to work with, at the conference there are usually lots of cannons to examine and try out.

The availability of several cannons creates some interesting opportunities for navigating the conference. If you want to travel quickly from point a to point b, you could just run straight from a to b, or, you could run to a nearby cannon and launch yourself somewhere else. From there, you can continue to run toward your destination or you can continue to use cannons in an effort to get to your destination more quickly. With cannons positioned like Figure E.1, you could follow a path like the one in Figure E.2 to get from a to b. The arrows show places where you launched yourself out of a cannon, and the lines show where you ran to the next cannon or to your destination.

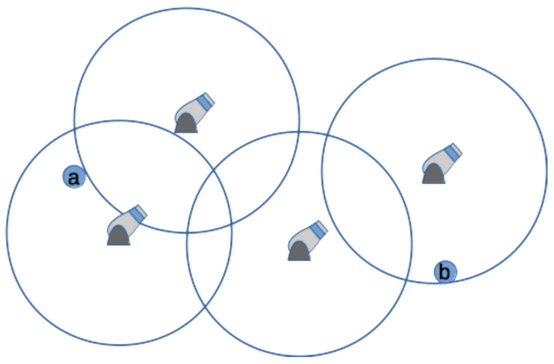


Figure E.1: Illustration of the sample input.

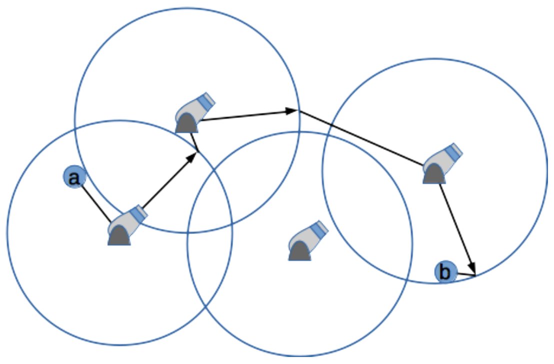


Figure E.2: A suboptimal solution.

You run at a rate of 5 meters per second. All cannons launch you a distance of 50 meters, in any direction you'd like. Climbing into a cannon, launching yourself and landing takes a total of 2 seconds. Cannons are not obstacles; if a cannon is in your way, you can jump over or run around it without it slowing you down. Given your current location, a desired destination and the positions of available cannons, you want to plan how to get to the destination as quickly as possible.

Input

The input describes a single navigation problem. The first line gives a pair of real numbers, the X and Y coordinates where you're currently located. The next line give the real-valued X and Y coordinates of the location you'd like to reach. This is followed by a line with an integer, n, the number of cannons available. The remaining n input lines each contain a pair of real values giving the X and Y coordinates for a cannon. All coordinates are measured in meters, and the value of n will be between 0 and 100, inclusive.

Output

Print a single line of output, the total number of seconds required to reach your destination as quickly as possible. Your answer must be accurate to within 0.001 seconds.

Sample Input 1	Sample Output 1

25.0 100.0
190.0 57.5
4
125.0 67.5
75.0 125.0
45.0 72.5
185.0 102.5
	19.984901

Problem: COW

(Ben Cousins and Brian Dean, 2015)

Bessie the cow has stumbled across an intriguing inscription carved into a large stone in the middle of her favorite grazing field. The text of the inscription appears to be from a cryptic ancient language involving an alphabet with only the three characters C, O, and W. Although Bessie cannot decipher the text, she does appreciate the fact that C, O, and W in sequence forms her favorite word, and she wonders how many times COW appears in the text.

Bessie doesn't mind if there are other characters interspersed within COW, only that the characters appear in the correct order. She also doesn't mind if different occurrences of COW share some letters. For instance, COW appears once in CWOW, twice in CCOW, and eight times in CCOOWW.

Given the text of the inscription, please help Bessie count how many times COW appears.

Input

The first line of input will contain a single positive integer, C, the number if input cases to consider.

The first line of each input case consists of a single integer $N <= 10^5$. The second line contains of a string of N characters, where each character is either a C, O, or W.

Output

For each input case, output on a line by itself the number of times COW appears as a subsequence, not necessarily contiguous, of the input string.

Note that the answer can be very large, so make sure to use 64 bit integers ("long long" in C++, "long" in Java) to do your calculations.

Sample Input	Sample Output
2 6 COOWWW 7 CWOWCOW	6 4

Best Schedule

Filename: schedule

The Problem

Your task is to provide the maximum usage for a single room during a day. You are provided with a list of event requests to use the room. Each request is uniquely labeled with a start time and end time. Each time is given in the number of minutes after midnight. Your job is to schedule the room from 8:00am to 5:00pm in such a manner as to maximize the amount of time the room is used. (*Note: Any number greater than 1020 or less than 480 automatically represents an invalid time for scheduling.*) If an event requests the room outside of these bounds, you can not schedule that event. Your program should determine the maximum amount of time the room can be scheduled. You may schedule two events as long as the first event ends *at or before* the time the second event is scheduled to start.

The Input

The first line of input will contain n $(n < 20)$, the number of test cases in the file. The first line of each test case will contain a positive integer $k (k \le 540)$, the number of events to be considered for that particular schedule. The next k lines will contain two integers each. The first event listed is event 1, the second is event 2, etc. The first of the two integers on each line will be the starting time, s_i, for the i^th^ $(1 \le i \le k)$ event (in minutes past midnight) and the second integer will be the finishing time, e_i, for that event (in minutes past midnight), with $0 < s_i < e_i$.

The Output

For each input case, output the maximum number of minutes the room can be scheduled for that case, on a line by itself.

Sample Input

```
2
5
500   520
520   601
600   700
650   800
800   1000
3
1232321 1240000
200 300
500 520
```

Sample Output

```
451
20
```

Blooscreins of Death

Filename: `SPACESHIP`

The year is 2114. The evil Phil Bates, leader of the Blooscrein Empire, is trying to destroy the human race to make room for his own species. The Earth division of the United Network of Intergalactic Transmissions (UNITs) selected you to deliver a distress signal to neighboring star systems. You *must* get this message through to avoid the extermination of mankind and the further domination of the Blooscreins. They have provided you with an old computer system, much like the one in front of you, and a cloaking spaceship. However, the Blooscrein warships have superior technology. If you fly too close to them, their more advanced radar systems will be able to find you. You have managed to escape Earth, but will you save mankind?

The Problem:

You are currently in the middle of a galaxy called MicroStars, surrounded by Blooscrein Empire's star fleet. Fortunately, your spaceship knows the range of each enemy warship's radar. So you decide to write a program that will tell you how many Blooscrein warships can "see" you. Given your location, and the location and effective range of each enemy's radar, determine how many warships will pick you up. The Blooscrein radar systems will detect you even if you are at the very edge of the radar's range. Assume the radar signals emit from the location of the ship and that the sizes of all ships are negligible.

The Input:

The first line of the file will consist of the number of data sets, *n*. The first line of each data set contains four integers, *x*, *y*, *z*, and *e*; (*x*, *y*, *z*) is your position in 3D space and *e* is the number of Blooscreins in your vicinity. The following *e* lines each contain four integers, x_i, y_i, z_i, d_i. These represent the position and effective radius of each warship's radar.

The Output:

For each data set, print the line "`You will be picked up by` *`x`* `radars.`" where *x* is the number of enemy ships that will detect your spaceship.

Sample Input:

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

Sample Output:

```
You will be picked up by 1 radars.
You will be picked up by 0 radars.
```

Universe in the Cupboard

Input: `stars.in`

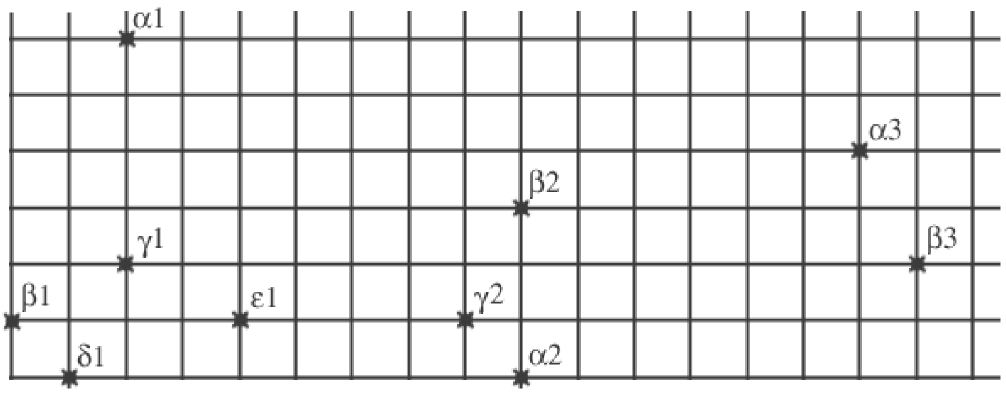
Little Omri has been playing with his magic cupboard again and has accidentally created a parallel universe, complete with stars, planets, people and everything. The people of the planet Parallearth gazed up at the sky on their first night of existence, and noticed that the stars appear to be in groups, which they chose to call *constellations*. Before the big vote to choose names for the constellations, they want to know how many there are, and that's where you come in.

The Parallearthlings have taken a photograph of the night sky, which renders all the bright stars in a rectangular frame. They then put a regular grid on the photograph so they could determine the coordinates of all of the stars. These coordinates have been written down in a list. Your program should process that list of two-dimensional coordinates and print out how many constellations can be made from the visible stars.

There are very simple rules for constructing constellations:

1. Every star is in the same constellation as its closest neighbor.
2. If a star doesn't have a unique closest neighbor—*i.e.* two or more neighbors are equally close and no others are closer—then it is in the same constellation as all of those closest neighbors.
3. If *A* is in the same constellation as *B*, then *B* is in the same constellation as *A*.
4. If *A* is in the same constellation as *B*, and *B* is in the same constellation as *C*, then *A* is in the same constellation as *C*.

For example, if the following were the picture of the night sky:



The first constellation consists of α1, β1, γ1, δ1, and ε1. Stars α1 and γ1 are in the same constellation because γ1 is the closest star to α1 (i.e., by rule 1). Since γ1 has three closest neighbors (β1, δ1, and ε1), all of these are in the same constellation as γ1 (by rule 2). Similarly, α2 is the closest neighbor of γ2, and γ2 is the closest neighbor to β2, so they are all in the same constellation (by rule 4). Finally, α3 and β3 are in a third constellation.

Input

The input to the program will consist of a sequence of universe specifications. Each begins with a single line containing a single integer *n* ($0 < n \le 1000$), the number of stars in that universe. There then follow *n* lines, each containing two integers *x* and *y* ($0 \le x, y \le 9999$), which are the coordinates of a star on the official photograph.

The last universe specification has $n = 0$, and should not be processed.

Output

For each universe specification, your program should print a single line that looks like:

```
Universe n contains c constellations
```

where `n` is replaced by the universe number (the first is `1`), and `c` is replaced by the number of constellations.

Sample Input

```
10
0 1
16 2
1 0
2 6
9 0
4 1
2 2
8 1
9 3
15 4
5
10 10
10 11
20 10
20 11
15 5
0
```

Output Corresponding to Sample Input

```
Universe 1 contains 3 constellations
Universe 2 contains 1 constellations
```

[F] Air Strike

Program:	`strike.(c|cpp|java)`
Input:	`strike.in`
Balloon Color:	`Brown`

Description

General Gee is the commander of a military base. He has just received alarming news from one of his spies: the enemy's preparing an air missile strike. The base contains two magnetic towers. When activated and given sufficient power, each of the magnetic towers creates a powerful horizontal magnetic disk. If any missile passes through this disk it deflects away from the base. Although those towers seem to be an excellent air defense method, there is a problem: The area of the disk generated by a tower is proportional to the amount of energy it receives. The base has enough power plants to generate a certain amount of energy, which has to be divided among those two towers. That means that the total area of the two disks generated from the towers should not exceed the total energy generated by the power plants. Fortunately, the spy was able to know the exact target co-ordinates of the incoming missiles and he reported them to General Gee. The General needs your help in distributing the energy on the two magnetic towers to minimize the number of missiles that will not get deflected by the magnetic towers and therefore will hit the base. You may assume the following:

- The towers have different heights and therefore there are no problems associated with the magnetic disks interfering with each other.
- A missile will deflect if it passes through the magnetic disk of a tower or even if it just touches its boundary.
- A missile hitting a tower (landing exactly on its location) will deflect, even if the tower is not given any energy.
- All incoming missiles will go down simultaneously at the exact instant; therefore, there will not be any time available to redistribute the energy amongst the two towers during the strike.

Input Format

Input consists of several test cases. Each test case is specified on $N+2$ lines. The first line contains an integer $(1 \leq N \leq 1,000)$ representing the number of missiles. The second line contains 5 real numbers X_1, Y_1, X_2, Y_2 and T: (X_1, Y_1) is the coordinates of the first tower, (X_2, Y_2) is the coordinates of the second tower and $(0 \leq T)$ is the total amount of energy generated from the power plants (the total area of the two magnetic disks). Each line of the remaining N lines contains two real numbers representing the landing coordinates of a missile.

The absolute value of all the given real numbers is less than or equal to 100 and may include a decimal point followed by up to 3 digits. Any two consecutive numbers on the same line are separated by one or more white-space characters. Zero or more blank lines may appear between test cases.

The last line of the input file is made of a single zero.

Output Format

For each test case, print the following line:

```
k.␣M
```

Where `k` is the test case number (starting at one,) and `M` is the minimum number of missiles that will NOT be deflected in the best distribution of energy among the two towers. Use $\pi = 3.141$.

Sample Input/Output

strike.in
```
6
-3 0  3 0  40.833
-1 4
-2 2.5
1 2
5 2
-4 0
-3 -1

2
0 0  1 1  0
0 0
1 1

0
```

OUTPUT
```
1. 2
2. 0
```