Eighth Annual
University of Central Florida

High School Programming Tournament: Online Edition

Problems

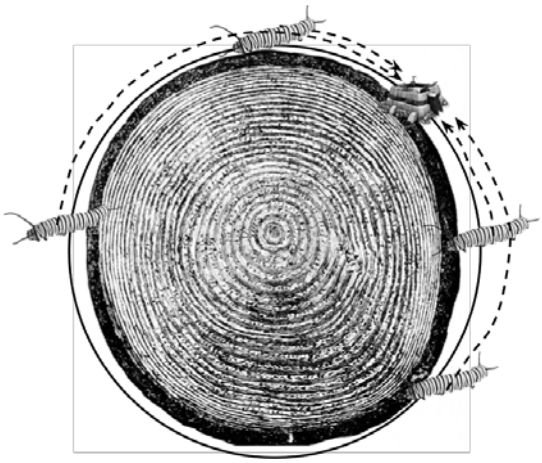
Problem Name	Filename
Caterpillar Bunker	bunker
Bag of Coins	coins
Team Dinner	dinner
Elemental Phrases	elemental
Endangered Species	lerps
Ben's Multiplication Tables	multiply
Optimal Prime	optimal
Unite the Universe	unite
Wipeout!	wipeout

Call your program file:
filename.c, *filename*.cpp, *filename*.java, or *filename*.py

For example, if you are solving Caterpillar Bunker,
Call your program file:
bunker.c, bunker.cpp, bunker.java, or bunker.py
Call your Java class: bunker

Caterpillar Bunker

Filename: `bunker`



In the swamp area near the back of the UCF Student Union, caterpillars have made neighborhoods around the bases of the trees. This area was prime real estate since the local squirrels always knock down enough fresh leaves to feast on, but unfortunately now the area is surveyed by a family of great crested flycatchers. These birds have eaten far too many caterpillars to be ignored. The clever caterpillar council devised a plan: a bunker will be built at the base of every tree that housed a caterpillar neighborhood. When the call of the flycatcher is heard, every caterpillar crawls around their tree to take refuge in their bunker until the coast is clear. There is debate over where to build the bunkers for each tree as all the caterpillars want it to be as close to their own house as possible. To resolve the bickering, the caterpillar council decreed that all bunkers should be built on the base of the tree at the location that minimizes the sum of the distance all caterpillars must crawl. A caterpillar neighborhood around a tree can be modeled by caterpillars moving around a circle, since caterpillars will not crawl up or down the tree, nor crawl away from the tree.

The Problem:

Given the radius of a tree the caterpillar neighborhood is built around and the location of the caterpillars' homes in the form of angles, find the minimum sum of distances all caterpillars must walk when the bunker is placed optimally.

The Input:

The first line of the input contains a single, positive integer, *t*, representing the number of trees. This is followed by *t* descriptions of trees. Each description starts with two integers, *r* and *c* ($1 \le r \le 100$; $1 \le c \le 100$), representing the radius of the tree and the number of caterpillar homes around the tree, respectively. The next line contains *c* real numbers, θ_i ($0 \le \theta_i < 360$), which are the angular locations of the caterpillar homes in degrees to at most 6 decimal places.

The Output:

For each tree, output "`Tree #i: d`" where *i* is the number of the tree in the input (starting from 1) and *d* is the minimum sum of distance all caterpillars must crawl to enter the optimally placed bunker. Output the distance rounded to three decimal places. You must always print to three places after the decimal so you may have to include trailing zeroes such as "4.000" instead of "4.0".

Sample Input:

```
2
1 4
0.0 180.0 90.0 315.0
10 3
33.96 63.96 93.96
```

Sample Output:

```
Tree #1: 5.498
Tree #2: 10.472
```

Bag of Coins

Filename: `coins`

One Saturday after programming practice, Matthew and Bill found a bag of coins on the sidewalk while they were walking back to their dorms. They were excited at first, but then they discovered that the bag was filled only with pennies and a single one-dollar coin. Neither of them cared about the pennies, but they both wanted to keep the one-dollar coin. In order to determine who got to keep the one-dollar coin, they decided to play a game. They stacked all of the coins on top of each other, with the one-dollar coin on the very bottom, and counted that there were n coins in total. Each of them would take turns removing at least 1 coin and at most k coins from the top of the stack. They would continue to do this until one of them picked up the one-dollar coin, and that player would get to keep it.

The Problem:

Given the number of coins in the bag, the maximum number of coins Matthew and Bill can remove from the top of the stack each turn, and which player goes first, determine which player will win and receive the one-dollar coin, assuming that both players play optimally. Playing optimally means that if it is possible for a player to guarantee victory for themselves, then they will always win.

The Input:

The first line will contain a single, positive integer, t, representing the number of games to process. Each game will consist of a single line containing two single integers, n ($1 \leq n \leq 10^9$) and k ($1 \leq k \leq 10^9$), representing the number of coins in the bag and the maximum number of coins a player can remove on their turn, respectively, followed by the name of the player that will go first (either "`Matthew`" or "`Bill`"), each separated by a single space.

The Output:

For each game, output a single line of the form "`Game #`*`i`*`: `*`w`*" where i represents the number of the game in the input (starting from 1) and w represents the name of the player that will win and receive the one-dollar coin.

Sample Input:

```
3
12 3 Matthew
14 3 Bill
21 6 Bill
```

Sample Output:

```
Game #1: Bill
Game #2: Bill
Game #3: Matthew
```

Team Dinner

Filename: `dinner`

After every Saturday practice for the UCF Programming Team, Timothy always implores his fellow teammates to go get dinner with him. Obviously, he always suggests that the team go to Chili's as it is the best restaurant in the world (in Timothy's opinion). Other people suggest other inferior options, but no one can ever seem to agree. Tyler said he will only go get dinner if the potential restaurant's name is of prime length and has an odd integer amount of vowels. The team decided to count y's as ½ of a vowel. Thus, if there are an odd number of y's, the restaurant is immediately ruled out. Furthermore, while discussing restaurants, at one point Michael mentioned that a certain restaurant's name had a perfect square number of vowels and a perfect square length. Thus, it was determined that any restaurant with this property is automatically valid.

The Problem:

Given a list of potential restaurants that the team is considering, print which restaurants are valid options. A restaurant is valid if both of the following hold:

- The length of its name (including spaces and punctuation) is a prime number (an integer greater than 1, which is divisible only by itself and 1)
- The number of vowels (with y's counted as ½ of a vowel) is an odd integer

or if the following holds:

- The length (including spaces and punctuation) and the number of vowels (with y's counted as ½ of a vowel) are both perfect squares (a perfect square is any number x^2 where x is any integer)

The Input:

The first line of input will contain a positive integer, p, representing the number of days of practice to consider. Each practice in the input will begin with a line containing an integer, n ($1 \leq n \leq 1{,}000$), representing the number of restaurants to consider for that practice. The next n lines will each contain a distinct string of the name of i^{th} restaurant. These strings will only contain uppercase and lowercase letters, numbers, spaces, apostrophes ('), and ampersands (&). No name will begin or end with a space. There will never be two adjacent spaces in a restaurant's name. Each restaurant's name will be no longer than 50 characters.

The Output:

For each practice, first output "`Practice #i:` " where *i* is the practice number in the order given in the input starting with 1. Then, on the same line if there are 1 or more valid options, output "*x* `valid options`" where *x* is the number of restaurant options for the i^{th} practice that are valid. However, if there are no valid restaurant options, output "`It's hopeless!`" Next, output the name of each valid restaurant option for that practice, each on a new line. Output these restaurant names in the same order given in the input. Output a blank line after each practice.

Sample Input:

```
2
7
Chili's Bar & Grill
Tijuana Flats
Four Rivers
Mellow Mushroom
Texas Roadhouse
Steak 'n Shake
Cool Tofu
2
Wendy's
McDonald's
```

Sample Output:

```
Practice #1: 2 valid options
Tijuana Flats
Cool Tofu

Practice #2: It's hopeless!
```

Elemental Phrases

Filename: `elemental`

Sir Kohl is a chemist who studies the elements of the periodic table. He also likes to goof around and combine different atomic symbols to create phrases so that he can impress his friends or post something silly onto his bulletin board—he calls such phrases "elemental".

For example, the word "cat" can be represented with the sequence of elements *carbon astatine* in that order, because the symbol for carbon is "C" and the symbol for astatine is "At", which together makes [C][At]; thus, "cat" is an elemental phrase. Likewise, "bass cannon" can be represented with the series of elements *barium sulfur sulfur californium nitrogen nobelium nitrogen*, which makes [Ba][S][S] and [Ca][N][No][N] and is therefore an elemental phrase.

6	84
C	At
Carbon	Astatine

However, Kohl realized that not all phrases are elemental after trying to make a "cool" sign using element symbols and then failing to find any element combinations which can represent the word "cool". Thus, "cool" is not elemental. Also, the placement of spaces in between words can affect whether or not a phrase is elemental. "Papyrus" is an elemental phrase because the sequence *protactinium phosphorous yttrium ruthenium sulfur* can be used to represent it ([Pa][P][Y][Ru][S]), but "Papyr Us" is not elemental because there is at least one word in that phrase that isn't elemental (in this case, "Papyr" is not elemental while "Us" can be represented with *uranium sulfur* ([U][S]).)

The Problem:

Given a list of all known elements and a phrase, determine if any combination of the elements can combine to form the phrase.

The Input:

The input will consist of several queries from galaxies consisting of different elements. The first line of each query contains a single integer, e ($1 \leq e \leq 200$)**,** representing the number of known elements. The second line of each query will consist of e space-separated, unique strings of element symbols, denoting the symbols of each of the e elements. Each element symbol will consist of lowercase letters only, and have a minimum length of 1 character and a maximum length of 3 characters. The element symbols do not necessarily need to represent real elements, and do not necessarily need to include all real-life elements.

The third line of each query consists of the string, *s*, representing the phrase which Kohl wants to know whether it's elemental or not. All strings will be at least one character in length and at most 1,000 characters in length. Each string, *s*, can consist of both lowercase letters and spaces, but will always start with a lowercase letter and end with a lowercase letter.

End of input is denoted by a single 0.

The Output:

For each query, output a line containing "`Phrase #`*k*`: `*a*" where *k* is the query number (starting at 1) and *a* is a string consisting of "`Elemental`" if *s* can be constructed using the elements given in the input, and "`Not Elemental`" if otherwise. Leave a blank line after each line of output.

Sample Input:

```
3
c ca at
cat
6
ba s sc ca n no
bass cannon
6
i am c co o la
cool
6
u pa p s y ru
papyrus
6
pa p y ru u s
papyr us
3
n a y
nanny
1
p
papyrus
0
```

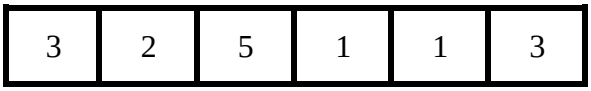
Sample Output:

```
Phrase #1: Elemental

Phrase #2: Elemental

Phrase #3: Not Elemental

Phrase #4: Elemental

Phrase #5: Not Elemental

Phrase #6: Elemental

Phrase #7: Not Elemental
```

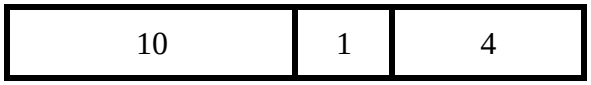
Endangered Species

Filename: `lerps`

Lerps are alien organisms which are very emotionally delicate. They cry when they are lonely, and when they do, they temporarily forget to breath and faint. This can start a chain reaction as lerps who see their acquaintance faint will start crying too. The Galactic Organism Conservatory (GOC) has their hands full trying to keep some lerps happy in captivity. The GOC has built several artificial environments which are divided into rooms. The rooms are arranged in a line, and it is possible to remove the walls dividing adjacent rooms. The GOC has discovered that the only way to keep lerps happy is for every lerp to make a *mutual* best friend. However, this is impossible if there isn't an even number of lerps as the odd one out without a best friend will inevitably cry and soon all the lerps in the room will be crying. Since it is risky to move lerps from their current room, the GOC will merge some rooms by joining some adjacent rooms.



An example artificial environment layout with 6 rooms holding various quantities of lerps and 5 walls capable of being removed. If left as it is only the 2 lerps in the second room will be happy while other 13 lerps in rooms with odd numbers of lerps will faint.



After removing 3 walls the first 3 and last 2 rooms can be merged, making 14 lerps happy and leaving only 1 crying.

The Problem:

Given the quantity of lerps in each room of an artificial environment, find the maximum amount of lerps that can be kept happy by removing zero or more walls between the rooms.

The Input:

The first line of the input contains a single, positive integer, *e*, representing the number of artificial environments. This is followed by *2e* lines. For each environment the first line contains a single, positive integer, *n*, ($1 \leq n \leq 500$), representing the number of rooms in the environment. The next line contains *n* integers, q_0 through q_{n-1} ($1 \leq q_i \leq 1{,}000{,}000$), the quantity of lerps in each room in order.

The Output:

For each environment, output "`Environment #i: x lerps`" where *i* is the environment's number in the input (starting from 1) and *x* is the maximum amount of lerps that can be kept happy.

Sample Input:

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

Sample Output:

```
Environment #1: 14 lerps
Environment #2: 14 lerps
Environment #3: 20 lerps
```

Ben’s Multiplication Tables

Filename: `multiply`

Rachel's little brother, Ben, has just started learning his multiplication tables. Unfortunately for Rachel, Ben has been asking her to check his homework every night for the past week. Rachel, being a good older sister, wants to help her brother, but she is worried that the time she spends checking his homework could be better spent completing her own homework. That's why she has asked you to write a program to check Ben's homework for her!

The Problem:

Write a program that will output the result of two integers multiplied together.

The Input:

The first line of input will contain a positive integer, *p*, indicating the number of problems that Ben was given for homework. The following *p* lines will contain two integers each, *m* and *n* ($0 \leq m, n \leq 1000$), the two numbers that Ben is being asked to multiply.

The Output:

For each pair of numbers given, output the result of *m* multiplied by *n* on a line by itself.

Sample Input:

```
3
2 3
1 5
6 8
```

Sample Output:

```
6
5
48
```

Optimal Prime

Filename: `optimal`

A war has erupted amongst the numbers! The irrational decepticons are waging war against rational autobots. A special team of these rational autobots are fighting back against the irrationals. These are the prime integers. All of the prime integers break into different squads in order to maximize efficiency. However, each of these squads must choose a leader. Their leader, of course, must be an optimal prime.

An optimal prime is a positive integer which is prime and can transform into another prime integer. Like all autobots, an optimal prime cannot simply change into whatever number it pleases. Rather, it can only change into different numbers by rearranging its digits. All the digits from the original number must be in the new number and leading zeroes are not allowed. If a prime integer can be arranged in such a way to produce a different prime integer, it is considered to be an optimal prime and can thus lead its squad to victory. The autobots need your help to save the world and defeat those irrational decepticons! Remember that a prime number is a positive integer greater than one which is only divisible by itself and one.

The Problem:

Given a positive integer, determine whether or not it is an optimal prime. (An optimal prime is a prime number whose digits can be rearranged to produce a different prime number.)

The Input:

The first line of input will consist of a single positive integer, *b*, representing the number of battles to consider. The next *b* lines each contain a single positive integer, x_i ($1 \leq x \leq 10{,}000{,}000$), representing the number of the leader for the i^{th} battle.

The Output:

For each battle first output "`Battle #i: `" where *i* is the battle number in the order given in the input (starting with 1). Then output either "`Autobots, roll out!`" if the given number for the leader of that battle is an optimal prime. Otherwise, if it is not an optimal prime, output "`Oh no, the autobots are doomed!`"

(Sample Input and Sample Output follow on next page)

Sample Input:

```
4
37
521
41
24
```

Sample Output:

```
Battle #1: Autobots, roll out!
Battle #2: Autobots, roll out!
Battle #3: Oh no, the autobots are doomed!
Battle #4: Oh no, the autobots are doomed!
```

Unite the Universe

Filename: `unite`

In the distant future, teleportation has finally been invented by peerless genius Dr. Tel Elport. The concept of long distance could be effectively removed if every bus station in the universe was replaced by a teleportation station. He has tried to give instructions for the construction of these teleportation stations, but too many engineers made errors in construction and instead made cloning stations. After many headaches, Tel was commissioned to travel the universe to teach the engineers at every major star system how to create a teleportation station. Each star system's location (as well as Tel's initial position) is identified by an (x, y, z) location. The teleportation panel wishes to minimize the cost of travel. Travel can currently be done between any two star systems with the cost of d^2, where d is the Euclidean distance between the star systems in 3D space. Naturally, after a teleportation station is built at a star system, he can use it to travel back there instantly from another star system with a teleportation station, and by the time Tel finishes teaching at any star system a teleportation station will be accessible there. The cost of teleportation is entirely free, so the cost equation is not used when teleporting.

The Problem:

Given the locations of the star systems, find the minimum cost for Tel to visit all n star systems at least once starting from his initial position.

The Input:

The first line will contain a single, positive integer, t, representing the number of universes to be processed. Following this, for each universe the first line will contain a single, positive integer, n ($1 \leq n \leq 500$), representing the number of locations within that universe's star systems. The next n lines contains 3 integers, x, y and z ($-500 \leq x \leq 500$; $-500 \leq y \leq 500$; $-500 \leq z \leq 500$), representing a coordinate within the universe. The first given coordinate is the originating location of Tel, and the remaining $n-1$ lines represent the locations of the star systems Tel needs to visit.

The Output:

For each universe, output "`Universe #i: x`" where i is the universe's number in the input (starting from 1) and x is the integer cost to travel the optimal (minimum cost) route to visit all star systems.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
2
3
0 0 0
0 0 2
0 0 -2
7
0 0 5
0 3 0
0 3 5
2 0 0
2 0 5
2 3 0
2 3 5
```

Sample Output:

```
Universe #1: 8
Universe #2: 55
```

Wipeout!

Filename: `wipeout`

Welcome to Wipeout! On today's show, you are tasked with navigating an obstacle course as quickly as you can. The producers have provided you with a map of the course (with the start at platform 1 and the end at platform *n*, and platforms connected to each other by obstacles) and you must quickly plan your path to victory. Of course, some obstacles are harder than others, so you must take into account both how time consuming and how energy consuming each obstacle is. The producers have informed you that, while there may be multiple paths to the finish, there's no bonus for completing all obstacles. All that matters is making it to the finish as fast as possible. Will you walk away with the prize, or will you Wipeout?

The Problem:

Given a description of an obstacle course, find the minimum time required to complete the course without running out of stamina. Note that collapsing at the finish line is perfectly acceptable (if not the healthiest decision); that is you may finish with 0 stamina.

The Input:

The input will begin with a single, positive integer, *h*, representing the number of shows to consider.

For each show, the first line will consist of three integers, n ($2 \leq n \leq 500$), m ($1 \leq m \leq 1000$) and s ($1 \leq s \leq 1000$), representing the number of platforms, the number of obstacles, and your initial stamina, respectively.

This will be followed by m lines, each representing an obstacle that connects two platforms (note that obstacles may only be traversed in one single direction). Each line will consist of four integers, a ($1 \leq a \leq n$), b ($1 \leq b \leq n$), d ($1 \leq d \leq 1000$) and t ($1 \leq t \leq 100$), representing the platform the obstacle begins at, the platform the obstacle ends at, the difficulty of this obstacle (i.e. the stamina cost), and the amount of time completing this obstacle will take, respectively. Again, note that obstacles are one-directional, you cannot go back through an obstacle. However, there may be multiple obstacles between two platforms, in any combination of directions.

The Output:

Output a single line in the form "`Episode #i: c`" where i is the show being considered, and c is a single integer, representing the minimum time to complete the obstacle course.

If it is not possible to complete the obstacle course, output "`Episode #i: Wipeout!`" instead.

Sample Input:

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

Sample Output:

```
Episode #1: 4
Episode #2: Wipeout!
```