Tenth Annual
University of Central Florida

High School Programming Tournament: Online Edition

Problems – Division 2

Problem Name	**Filename**
Calculator Game	calc
Dictionary Ordering	dictionary
Exponential Escape Escapade	escape
Election Fraud	fraud
Hanukkah Graphs	hanukkah
Pangramic Judges	judges
Super Scrabble	scrabble
Slices	slices
Textiles & Triforces	triforce

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

For example, if you are solving Exponential Escape Escapade,
Call your program file:
escape.c, escape.cpp, escape.java, or escape.py
Call your Java class: escape

Calculator Game

Filename: `calc`

Lauren recently downloaded a game on her phone, and she's been playing it for hours! Here are the rules: At the beginning of each level, a starting number and a target number are displayed on the screen. Additionally, there are several buttons that vary between levels. Each of these buttons performs a specific operation. For example, one button might add 4 to the current number, another might multiply it by 10, and a third button might turn all of the 1's in the current number into 2's (so 4131 would become 4232). A full list of possible operations is given below. Lastly, an integer, *b*, is displayed, indicating the number of button presses allowed.

Beginning with the starting number, the goal of the game is to attain the target number after exactly *b* button presses. Buttons may be pressed in any order, and any button may be used any number of times (even 0). Since the calculator only supports 8 digits, intermediate values can't be greater than 99,999,999 or less than -99,999,999. The calculator only supports integers, so division may only be used if the result is an integer.

For example, suppose the starting number is 4, the target number is 224, *b* is 5, and that there are three buttons like the ones explained above. One solution is to first press the add 4 button twice ($4 \rightarrow 8 \rightarrow 12$), then the multiply by 10 button ($12 \rightarrow 120$), then add 4 again ($120 \rightarrow 124$), and finally turn all 1's into 2's ($124 \rightarrow 224$).

Here is a list of all possible operations that can be used in a button:

Operation	Input Form	Bounds	Notes
Add *x*	A *x*	$1 \le x \le 100$	
Subtract *x*	S *x*	$1 \le x \le 100$	
Multiply by *x*	M *x*	$1 \le x \le 100$	
Divide by *x*	D *x*	$1 \le x \le 100$	This button may only be pressed if the number is divisible by *x*.
Turn all *x* digits into *y*'s	T *x y*	$1 \le x \le 9$ $1 \le y \le 9$	Does nothing if there are no *x*'s in the number.
Append digit *y*	P *y*	$1 \le y \le 9$	Appends *y* to the end of the number. If the current number is 0, the number changes to *y*.

The Problem:

Determine if it's possible to achieve the target number with exactly *b* button presses. If not, tell Lauren to give up.

The Input:

The first line contains a single, positive integer, *v*, representing the number of levels to process. Each level starts with four integers, *s*, *t*, *b*, and *n* ($1 \leq s \leq 100000$; $1 \leq t \leq 100000$; $1 \leq b \leq 6$; $1 \leq n \leq 8$), representing the starting number, the target number, the number of button presses, and the total number of buttons, respectively. The following *n* lines each describe a button in one of the 6 forms given in the chart on the previous page. The bounds for each operation are shown in the chart.

The Output:

For each level, output a single line, starting with "`Level #x:` ", where *x* is the level number in the input (starting with 1). Following this, output "`It can be done`" if it is possible to reach the target number or "`You should give up`" if not.

Sample Input:

```
2
4 224 5 3
A 4
M 10
T 1 2
7 6 2 2
A 1
A 2
```

Sample Output:

```
Level #1: It can be done
Level #2: You should give up
```

Dictionary Ordering

Filename: `dictionary`

Back when the first dictionary was being written, Dr. Dinkledorf suggested that words should be ordered by the *count* of their letters in alphabetical order. For example, if a word *i* has more occurences of the letter ‘a’ than the word *j*, then *i* should come before *j*. If they had the same quantity (of the letter ‘a’), then if it has more occurences of the letter ‘b’, ‘c’, ‘d’... and so on.

“But what if the words are anagrams[1]?” asked his assistant, Mr. Binkley. “No worries,” Dinkledorf responded, “we will just sort them in lexicographical order[2] in this case.”

Of course, this was the worst idea ever, which is why you’ve never heard of it. Since you would like to demonstrate how bad this idea is to your friends, you plan to write a program that will sort words in Dr. Dinkledorf’s ordering scheme.

The Problem:

Sort a list of unique words using Dr. Dinkledorf’s ordering scheme.

The Input:

The first line contains a single, positive integer, *t*, representing the number of dictionaries. For each dictionary, multiple lines follow. The first line contains a single integer, n ($1 \le n \le 10^5$), representing the number of words. Then, *n* lines follow, each consisting of a string of only lowercase letters of positive length up to 20. It is guaranteed that each word is unique.

The Output:

For each dictionary, output n+1 lines. The first line to be printed is “`Dictionary #`*`i`*`:`” where *i* is the number of the dictionary in the order of the input (starting with 1). The next *n* lines should each contain a word given in the input, presented in sorted order. Output a blank line after each dictionary.

[1] Anagram – a word formed by rearranging the letters of another, such as *earth* formed from *heart*.
[2] Lexicographical order – also known as alphabetical order. This is the way words are ordered in modern dictionaries.

Sample Input:

```
3
3
david
lionel
natasha
4
cabbbbb
baaaaaaaa
sarahsara
bbbbbccccc
1
heyguys
```

Sample Output:

```
Dictionary #1:
natasha
david
lionel

Dictionary #2:
baaaaaaaa
sarahsara
cabbbbb
bbbbbccccc

Dictionary #3:
heyguys
```

Exponential Escape Escapade

Filename: `escape`

Greg and Steve love doing escape rooms together! They are doing the latest room offered in town and are down to their last lock! Despite Greg's insistence that the pair always solves the actual puzzles, Steve likes to hack the locks by trying various combinations.

Their friend, Travis, completed this escape room just yesterday (and set the record time of 34 minutes!). Travis (being a super-duper math guy) gave Steve a spoiler on the final lock combination. He told Steve that the final combination is the sum of $a^3+b^3+c^3$ for some values of a, b, and c. Steve knows the various possible clues, and wants to check various combinations to see if they will open the final lock! He needs your help so Greg and Steve can beat Travis' record!

The Problem:

Given values for a, b, and c, determine what the combination that Steve should try.

The Input:

The first line of the input will contain a single, positive integer, n, representing the number of possible lock combinations to compute. Each of the following n lines will contain three positive integers, a, b and c, representing the values of three clues from the escape room. It is guaranteed that all sets of three integers will compute at most a four-digit combination.

The Output:

For the each possible lock combination, output what the lock combination is as a single integer given the spoiler that Travis gave Steve. Steve knows to zero-pad the front of the number, if necessary, so you should not.

Sample Input:

```
3
9 8 11
1 3 9
5 11 13
```

Sample Output:

```
2572
757
3653
```

Election Fraud

Filename: `fraud`

Elections for student council are coming up and you really want to be class president. You have just received a much-needed briefing of how elections at the University of Fraud work. You're not the best listener, but you think you got the gist. Basically, the winner of the popular vote doesn't really matter too much. All you have to do to be president is get more votes in the Electrical College.

Since bribery was never explicitly outlawed (or if it was, you weren't paying attention during that part) you decide to win the election by choosing which electrical engineers to bribe optimally. Each electrical engineer has two bribe amounts which depend on how much they dislike each candidate: one indicating how much they would need to be paid to vote for you, the other indicating how much they would need to be paid to vote for your opponent. Today, you may pick any subset of engineers that you have enough money to bribe and bribe them. Tomorrow, after learning of your foul play, your opponent may choose any subset of unbribed engineers to bribe (as long as he has enough money to do so). Each engineer can only be bribed once.

You have *d* dollars that you are willing to spend bribing members of the Electrical College. You want to know the minimum amount of money your opponent would need to bribe at least as many electrical engineers as you, assuming you both make optimal choices whenever possible. Both candidates know each other's budget and the bribe costs of every electrical student before bribing starts. If it is possible for you to win a majority regardless of the amount of money your opponent has, print -1.

The Problem:

Given the bribe amounts for all *n* members of the Electrical College and the amount of money you are willing to spend *d*, determine the minimum amount of money your opponent would need to bribe at least as many members as you if you choose who to bribe optimally, or determine that you can force a victory regardless of your opponent's budget.

The Input:

The first line will contain a single, positive integer, *y*, representing the number of years you are considering running for class president (thankfully, you expect to graduate in fewer than 100 years, but it's better that your program handles this just in case you get suspended for rigging an election or something crazy). The description of each year contains two integers, n ($1 \le n \le 70$) and d ($1 \le d \le 1{,}000$), representing the number of electrical engineers that can be bribed, and the amount of money you have at your disposal, respectively. This is followed by *n* lines, each containing two integers, c_{you} and $c_{opponent}$ ($1 \le c_{you} \le 1{,}000$; $1 \le c_{opponent} \le 1{,}000$), representing the amount this engineer would have to be paid to vote for you and to vote for your opponent respectively.

The Output:

Output *y* lines each of the form "`Election #`*`x`*`: `*`m`*" where *x* is the election number and *m* is the minimum amount of money your opponent would need to secure at least as many votes as you, or `-1` if you can force a victory.

Sample Input:

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

Sample Output:

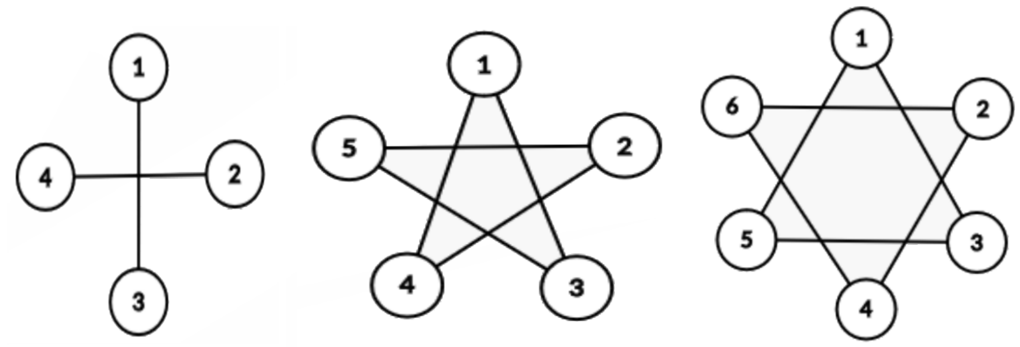
```
Election #1: 9
Election #2: -1
```

Hanukkah Graphs

Filename: `hanukkah`

The judges hope you enjoy your upcoming winter break and have a wonderful holiday season! As you probably know, Christmas *trees* are very popular during the holiday season, but we thought we would instead ask you about Hanukkah *graphs*!

We are defining a Hanukkah graph of size *n* to be *n* nodes arranged in a circle, labeled 1 to *n* in clockwise order, with each node having a bidirectional edge to the node two clockwise and two counterclockwise from itself. The Hanukkah graph of size 6 resembles the Star of David, and is shown below. We would like you to calculate the shortest distance between two given nodes in a Hanukkah graph of size *n*, or determine that it is impossible to reach the second node from the first traveling only along existing edges.



Hanukkah graphs of sizes 4, 5, and 6

The Problem:

For a Hanukkah graph of size *n*, calculate the shortest distance from node *a* to node *b*, or determine that there is no way to reach one node from the other.

The Input:

The first line of input contains a single, positive integer, *g*, representing the number of Hanukkah graphs to consider. This is followed by *g* lines, each containing three integers: *n* ($3 \le n \le 10^{18}$), *a* ($1 \le a \le n$), and *b* ($1 \le b \le n$), representing the number of nodes in the Hanukkah graph, and the two nodes to calculate the distance between, respectively.

The Output:

Output *g* lines each of the form "`Graph #`*`x`*`: `*`d`*" where *x* is the graph number in the input (starting with 1), and *d* is the shortest distance between the two given nodes or "`Impossible`" if it is impossible to reach one node from the other.

Sample Input:

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

Sample Output:

```
Graph #1: 1
Graph #2: Impossible
Graph #3: 2
Graph #4: 2
```

Pangramic Judges

Filename: `judges`

Sharon noticed that the judges of today's contest all have first names starting with a letter from A-J, except for him. This made him feel very lonely. Sharon started looking more carefully and noticed that every letter in the alphabet appears in some judge's name. Then, he quickly realized that some judges' names don't need to be included in order for this property to exist either!

The Problem:

Given a list of names, determine the minimum size of subset of these names such that every letter in the alphabet (A-Z) appears as either an uppercase or a lowercase letter in somebody's name.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of lists. For each list, several lines follow. The first contains a single integer, $1 \leq n \leq 16$, representing the number of judges. Then, *n* lines follow, each containing either spaces, uppercase or lowercase letters, representing a judge's name. Each name will contain at least one letter and will contain no more than 40 characters.

The Output:

For each list, output a single line saying "`Judge List #`*`i`*`: `*`c`*" where *i* is the number of the list in the input (starting with 1), and *c* is the minimum size subset of judges that includes all letters in some name. If Sharon is actually a liar and it is not possible to find a subset that includes all letters, output *`c`* as -1.

Sample Input:

```
2
16
Glenn Martin
Andy Phan
Atharva Nagarkar
Billy Quirogax
Brett Fazio
Charles Bailey
Christopher Stephens
David Harmeyer
Dylan Lyon
Ellie Kozlowski
Eric Hoyer
Jacob Magnuson
Jim Geist
John Edwards
Josh Wozniak
Sharon Barak
3
Natasha
David
Lionel
```

Sample Output:

```
Judge List #1: 5
Judge List #2: -1
```

Super Scrabble

Filename: `scrabble`

Alice and Bob are playing a variant of Scrabble called Super Scrabble. In this game, there is a pool of some nonempty subset of the lowercase alphabet that is considered "usable" and a list of *n* valid words. Alice and Bob take turns picking one of the usable letters and putting a tile with that letter next in the sequence. If at any point, the sequence of placed letters is equal to a valid word, then the last player who played a letter wins and the game is over. If the sequence is longer than every valid word, then the game ends in a draw. There are enough tiles of each letter that they don't need to worry about running out of tiles for any of the usable letters.

Alice will go first. Obviously, both Alice and Bob want to win, and if they cannot win, would rather draw than lose. As usual, Alice and Bob are perfect logicians and will play optimally.

The Problem:

Given the list of usable letters and the valid words, determine the result of the game, assuming Alice plays first and both players play optimally.

The Input:

The first line of input contains a single, positive integer, *g*, representing the number of games Alice and Bob will play. The description of each game follows. The first line of each description contains two integers, *n* and *m* ($1 \leq n \leq 26$; $1 \leq m \leq 10^5$), representing the number of usable letters and the number of valid words, respectively. The next line contains *n* unique lowercase letters in increasing alphabetic order representing the usable letters. The next *m* lines each contain a single string of lowercase letters representing a valid word. Each word is at most 10^5 characters long composed out of usable letters, and the sum of all word lengths in a game is at most $2*10^5$.

The Output:

Output *g* lines each of the form "`Game #`*x*`:` *r*" where *x* is the game number and *r* is either the message "`Alice wins`", "`Bob wins`", or "`Draw`" depending on the result of the game.

Sample Input:

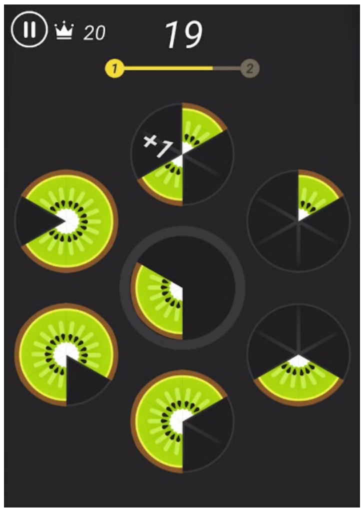
```
3
2 3
ab
aaa
ab
ba
3 4
abc
aaa
ab
cccc
ba
3 3
def
de
dedede
d
```

Sample Output:

```
Game #1: Bob wins
Game #2: Draw
Game #3: Alice wins
```

Slices

Filename: `slices`



Kelly and Jim are obsessed with a new mobile game called Slices! In Slices, the goal of the game is to fill up different circles with slices of fruit and pies while scoring points by placing pieces and completing full circles. After each level, the score required to move on increases.

Kelly and Jim may finish the levels at different times. Sometimes one player gets much further ahead than the other. When this happens, the other player becomes very angry and can't focus on anything in life other than catching up. (In extreme cases of differences between the two friends' progress, one player may forever hate the other.)

Specifically, if there ever is a time that Kelly is more than k levels ahead of Jim, Jim will forever hate Kelly. Similarly, if Jim is ever greater than k levels ahead of Kelly, Kelly will forever hate Jim.

The Problem:

Given the order which Kelly and Jim complete levels in Slices and the maximum number of levels one friend can be ahead of the other without anyone getting angry, determine whether Kelly hates Jim, Jim hates Kelly, they both hate each other, or if their friendship can last another day.

The Input:

The first line contains a single, positive integer, s, representing the number of scenarios to analyze. The first line of each scenario contains two positive integers, n and k ($1 \le n \le 1{,}000$; $1 \le k \le 1{,}000$), representing the number of events to follow, and the max number of levels one friend can be ahead of the other without putting a strain on their friendship, respectively. The next n lines each contain a single string (either "Kelly" or "Jim") representing who solved a level next and an integer, x ($1 \le x \le 1{,}000$), representing which level they just completed. It is guaranteed that the players solve the levels in order starting at 1.

The Output:

For each scenario, if neither friend hates the other, output a single line containing "`Everything is good`". If both players hate each other, output a single line containing "`Their friendship is doomed`". Otherwise, output either "`Kelly hates Jim`" or "`Jim hates Kelly`" depending on which friend hates the other.

Sample Input:

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

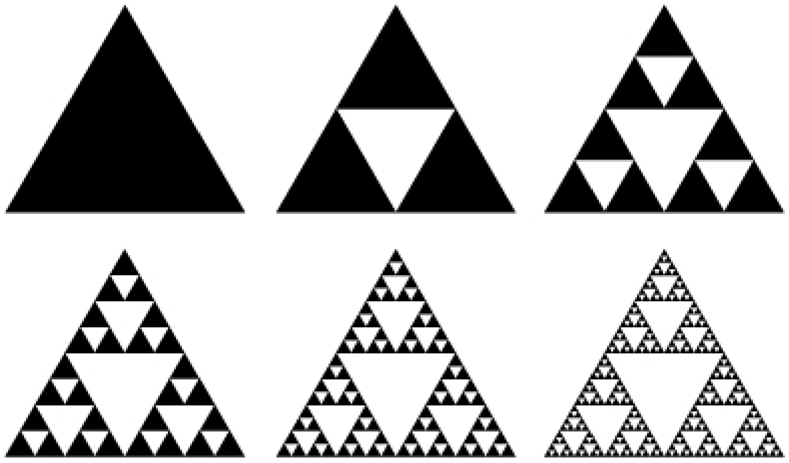
Sample Output:

```
Kelly hates Jim
Everything is good
Their friendship is doomed
```

Textiles & Triforces

Filename: `triforce`

The computer science team is primarily made up of nerds, and as nerds we all love Zelda. In tribute to our favorite game, we think it'd be an excellent use of funds to make a big fabric shrine for the Zelda Triforce symbol and hang it on the wall. However, since we're also a curious bunch, we want to make a Triforce with more than one iteration! The Triforce (seen below) is a shape that starts with a solid-colored equilateral triangle. The first iteration (creating the popular Zelda Triforce) takes one triangle out of the center, splitting the larger triangle into three smaller triangles. Each iteration repeats the process on the three smaller triangles, making mini Triforces. But arts and crafts isn't cheap! The nice, golden fabric we want to weave our Triforce out of costs *p* dollars per square foot. Since we're too scared to ask the coaches about the cost of our golden shrine, we're leaving it up to you!



The Problem:

Given the number of custom Triforces we are considering, the side length of each original Triforce triangle in feet, the price per square foot of the golden textile, and the number of iterations on each custom Triforce, tell us the cost of our custom Triforce.

The Input:

The first line of input will consist of a single, positive integer, *n*, representing the number of custom Triforces to consider. Following will be *n* lines of input, each consisting of three integers, *s*, *t* and *p* ($0 \le s \le 100$; $0 \le t \le 10^8$; $0 < p \le 50$), where *s* is the side length of the original Triforce triangle in feet, *t* is the number of Triforce iterations, and *p* is the cost of the golden fabric in dollars per square foot, respectively.

The Output:

Output *n* lines of the form "`Triforce #`*`i`*`:` *d*" where *i* is the number of the Triforce in the input (starting with 1), and *d* is the cost of the *i*-th Triforce rounded to the nearest cent and output to two decimal places. For example, 2.171 rounds to 2.17, 2.175 rounds to 2.18, and 2.178 rounds to 12.18.

Sample Input:

```
3
5 0 1
5 1 1
90 10000 30
```

Sample Output:

```
Triforce #1: 10.83
Triforce #2: 8.12
Triforce #3: 0.00
```