MERCER UNIVERSITY

# Spring Programming Contest

March 1, 2014

| | |
|---|---:|
| Powerful Numbers | (prob1) |
| Sub-Diagonal Paths | (prob2) |
| Social Pairings | (prob3) |
| Problematic Public Keys | (prob4) |
| Say My Number | (prob5) |
| The Disgruntled Professor | (prob6) |
| Trending on Twitter | (prob7) |
| Queen Dido’s New Challenge | (prob8) |
| Ultimate Tic-Tac-Toe | (prob9) |
| Candies To Go | (prob10) |
| Rover | (prob11) |
| What? Where? When? | (prob12) |
| Tree Sales | (prob13) |

The name in parenthesis following eadch problem is the name you must use as your program’s name. You must add the appropriate extension depending upon your choice of programming language (.c .cpp .cs .java .py .rb).

# Powerful Numbers (prob1)

## The Problem

In the July 2006 edition of the Mathematical Gazette, Note 90.32 gives the following definition for powerful number. The n-digit positive integer N = $a_n a_{n-1} \dots a_1$ is defined as a powerful number if f(N) = N, where $f(N) = a_n + a_{n-1}^2 + \dots + a_1^n$.

So given a number N, one may wish to determine if it is powerful.

## Input

Each line will contain a decimal number N whose length is at most 100 digits and its first digit will be nonzero. Input will be terminated by a line containing only a 0. It is not to be processed.

## Output

For each number, print it followed by the string “is/is not a powerful number.” Depending on whether it is or is not a powerful number.

## Sample Input

```
5
89
98
0
```

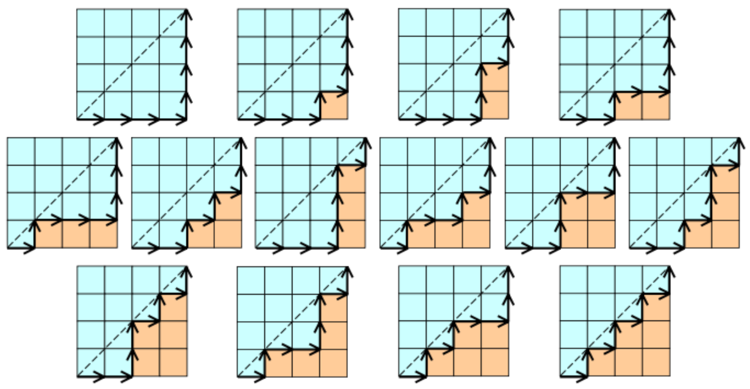
## Sample Output

```
5 is a powerful number.
89 is a powerful number.
98 is not a powerful number.
```

# Sub-Diagonal Paths (prob2)

## The Problem

You are to find all of the paths on the bottom diagonal of a n x n grid.   The path must only go from left-to-right or bottom-to-top. Given a dimension of the grid (ex. n = 4), specify the number of such paths (ex. solution = 14). (Image below is from Wikipedia):



## Input

The input consists of a single integer n, the dimension of the square grid, 1 <= n <= 30, on each line.  A zero will indicate the end of the input and should not be processed.

## Output

For each input n, you should output the number of paths, as described above, that exist in an nxn grid, one per line.

## Sample Input

```
1
2
3
4
0
```

## Sample Output

```
1
2
5
14
```

# Social Pairings (prob3)

## The Problem

John is the CEO of a company. He recently organized a social event for two branches of his company. In order to encourage the attendees to socialize with people from the other branch, John gave each employee a required task at the gathering: have a conversation with exactly 2 people from the other branch. He emphasized exactly 2 people, since if it was more than 2 people, the conversations may not be as meaningful. Thankfully, each of the branches had the same number of employees, otherwise it would be impossible for his requirement to be satisfied.

As John sits and watches his employees socialize, being a curious man, he wonders how many different possible ways there are of the employees conversing given his requirement. John assigns this question to you, his trusted technical advisor. Since John is the CEO, you do not want to disappoint him. Can you figure out how many ways there are and satisfy John's curiosity?

## Input

The first line of input will be a single integer t, representing the number of test cases. Then, t lines follow, each containing a single integer n, 1<=n<=2000, the number of employees at each branch of the company

## Output

For each test case, output the number of possible distinct ways that the two branches can socialize under the restriction that each employee must socialize with exactly 2 employees from the other branch. Since this number may be very large, output it modulo 10^8+7.

## Sample Input

```
5
1
2
3
4
1932
```

## Sample Output

```
0
1
6
90
87032144
```

## Problematic Public Keys (prob4)

### The Problem

On February 15, 2012, the New York Times reported a flaw in the method of generating keys for a public-key encryption system ("Researchers Find a Flaw in a Widely Used Online Encryption Method" by John Markoff). A public key is essentially the product of two unique prime numbers. Unfortunately, the prime numbers being used for generating keys were not always unique. This flaw enables an attacker to determine private keys given a set of flawed public keys simply by finding the shared factors.

Your job is to write a program that takes flawed public keys and determines the corresponding private keys. For the purposes of this problem, a private key consists of a pair of prime numbers, $2 < K_1, K_2 < 2^{31}$, and the corresponding public key consists of their product $K_1 * K_2$.

### Input

The first line of input consists of a single integer P, ($1 \le P \le 100$), which is the number of data sets that follow. Each data set consists of two or more lines. The first line of the data set contains two integer values, N ($1 \le N \le P$) and M ($2 \le M \le 100$), separated by a space. N is the data set number, and M is the number of data values that follow. The following $\lceil M/5 \rceil$ lines contain M "public keys," 5 keys per line (except possibly the last), each key, $k_i$ ($0 \le i < M$, $5 < k_i < 2^{31}$), the product of exactly 2 prime numbers.

### Output

For each data set there are several lines of output. The first line contains only the data set number. The following lines contain the prime factors of the input data values in ascending numeric order, 5 per line, except for the last line. Output values on the same line are separated by a space.

### Sample Input

```
3
1 6
221 391 713 1457 901
299
2 4
13193 18721 31897 18527
3 2
2143650557 2140117121
```

### Sample Output

```
1
13 17 23 31 47
53
2
79 97 167 191 193
3
32717 65413 65521
```

# Say My Number (prob5)

## The Problem

Produce a short, elegant, program that spells out any number between 1 and 1 quintillion ($1 \times 10^{18}$).
Your program should allow the user to input a number in the range from 1 to 1 quintillion (a positive 64-bit number) – the output will be the string consisting of the number spelled out in English words (e.g. 123456 would output "one hundred twenty three thousand four hundred fifty six"). Hyphens are not allowed (e.g. “fifty six” will be accepted, “fifty-six” will not).
*Hint:* The beauty of recursion is that this can be done with just a few if statements and switch/case values (about 35 or so total), thanks to the way we read numbers (the number 123,123,123 is spoken the same as a single 123, but with a few "place" words - million, thousand, etc.).

## Input

The first line of input will be the number of integers to spell out. The remaining lines will each contain one number *n,* $1 \le n \le 10^{18}$, per line.

## Output

The spelled-out English word equivalent of each number, in all-lower-case, separated by line breaks. One space between each word is required.

## Sample Input

```
3
123456789123456789
200000370
987654321
```

## Sample Output

```
one hundred twenty three quadrillion four hundred fifty six trillion
seven hundred eighty nine billion one hundred twenty three million
four hundred fifty six thousand seven hundred eighty nine
two hundred million three hundred seventy
nine hundred eighty seven million six hundred fifty four thousand
three hundred twenty one
```

# The Disgruntled Professor (prob6)

## The Problem

Professor Failemall believes he has discovered a case of cheating on his recent exam, and in his foul mood, decides to exact revenge on his poor students! His N students were all sitting in a row during the exam, and the sequence of scores they earned from left to right is described by an array S[1..N]. Professor Failemall is convinced that a contiguous block of one or more students in the middle of the row cheated, whose scores are a subarray of the form S[i..j] with i>1 and j<N. He is not exactly sure which block of students cheated, but to apply as much punishment to his entire class, he decides that he wants to remove a subarray S[i..j] with i>1 and j<N such that the average of the remaining exam scores is as low as possible. Please compute the smallest possible average he can achieve by doing this. Print your answer rounded to 3 decimal digits.

BOUNDS: 3 <= N <= 100,000; for each i: 1 <= S[i] <= 10,000

## Input

N, followed by S[1..N] each on a separate line

## Output

The answer rounded to 3 decimal places.

## Sample Input

```
5
5
1
7
8
2
```

## Sample Output

```
2.667
```

*Here, the optimal solution is to remove the subarray containing 7 and 8, leaving 5, 1, and 2, whose average is 8/3.*

# Trending on Twitter (prob7)

## The Problem

2013 was the year all news was tweeted. Think back on 2013's news — from sports to technology to politics to fashion — and you'll realize most of it was tweeted. Or, as Twitter puts it: "What happened in the world simultaneously happened on Twitter."

While this alone is not exactly shocking given social media's rise, the top tweet for 2013 may surprise you: *Glee* star Lea Michele's tweet about the death of her costar Cory Monteith. Her tweet was retweeted more than 408,000 times across 133 countries. The second-most-popular tweet was the official announcement of Paul Walker's death, retweeted nearly 400,000 times as news of the *Fast and Furious* star's passing broke around the world.

Other highlights include a peak of 130,000 tweets per minute about the new pope on the appointment of Pope Francis in March, and the use of Twitter by the Boston police department to share news on the bombing.

Your job is to write a program to scan a given number of tweets and keep track of all the hashtags used and their overall frequencies. A hashtag is a word or a phrase prefixed with the symbol #. It is a form of metadata tag. All hashtags are of length one or greater and consist solely of alphabetic characters. A hashtag will always be preceded by white space then the symbol #, and will never be embedded or nested within another hashtag.

Be sure to ignore case when finding your hashtags (i.e., Christmas and christMAS are the same hashtags). Your program should produce a summary of all the hashtags found from all tweets in descending order from those which are most tweeted to those least tweeted.

## Input

The first line of input contains a single integer *n*, ($1 \le n \le 1000$), which is the number of tweets that follow. Each tweet consists of a single line of input containing a string. Each string is of length *n*, ($1 \le n \le 140$). All tweets are separated by a single blank line.

## Output

You should output a summary of all hashtags found from all *n* tweets in descending order by highest frequency. Each hashtag should be printed on one line with its frequency first followed by a single space then the hashtag in all lowercase. Hashtags that have the same overall frequency should be printed in ascending dictionary order by hashtag.

## Sample Input

```
8
Friendly reminder... batteries are not imported from the North Pole.
#Christmas #Shopping

Santa is the man and he's got a plan I'm his biggest fan and he's
coming at #Christmas #WalnutCreekStuff

It's Saturday night...who's watching the #SNL #Christmas episode?

So excited for #Christmas even though it doesn't feel like #Christmas

Baileys #Christmas party was great. @GableBailey
pic.twitter.com/eRUHQazM9G

Guys #Christmas is 3 days away!! Christmas is #soClose

All i want for #christmas is, #food.

Russ Rose would like to wish you all a Merry #Christmas without
smiling. #SNL
```

## Sample Output

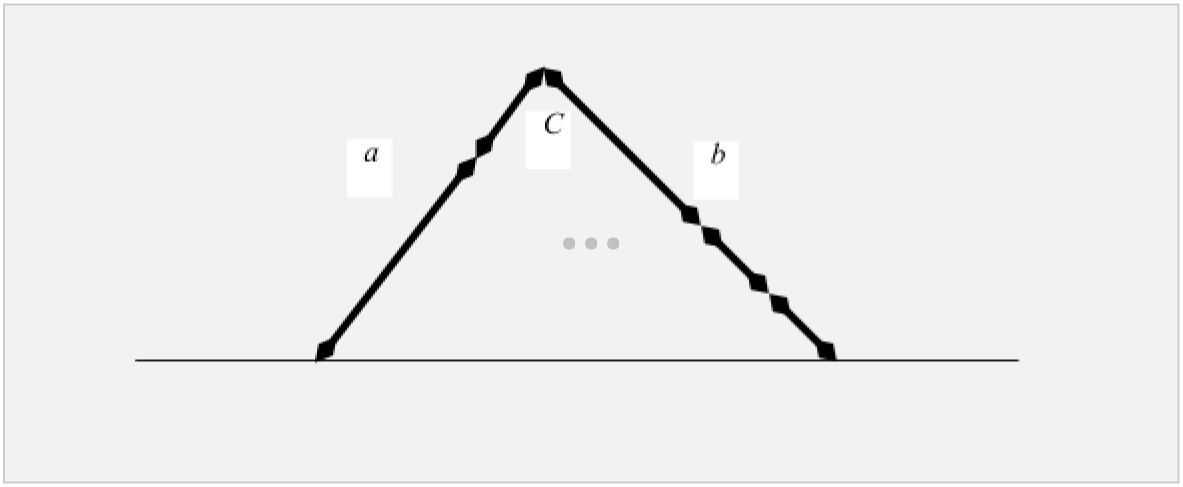
```
9 christmas
2 snl
1 food
1 shopping
1 soclose
1 walnutcreekstuff
```

# Queen Dido’s New Challenge (prob8)

## The Problem

Queen Dido of Carthage (850 BC) solved the first isoperimetric problem. She was promised as much land along the coast as she could enclose with an ox hide. According to the legend, she cut up a bull's hide into very thin strips which she sewed together into one long string. Then she took the seashore as one edge for the piece of land and laid the skin into a semi-circle. Modern calculus of variations would prove that Queen Dido’s solution is optimal and with a fixed string length the semi-circle encloses maximum area along a straight coastline.

Now Queen Dido was facing a new challenge. Given a bunch of twigs of various lengths, she needed to form a triangle along the coast to enclose the largest possible area.



One side of the triangle was the seashore and would not use any twigs. The other two sides should be formed by laying the twigs. The twigs could not be cut down further.

From the area formula Area = (½)ab sin C, it is easy to see that the optimal triangle must be a right triangle. An isosceles triangle (a=b) is clearly optimal, but it may not be possible to form because the twigs are discrete. However, triangles closer to the isosceles triangle will have larger areas, as seen from the formula Area = (½)ab = (½)($m^2$ - $(a-m)^2$), where m = (a + b)/2 is a constant.

## Input

Each input line consists of a positive integer n ( $2 \le n \le 100$ ), the number of twigs, followed by n positive integers representing the lengths of the twigs:

n L1 L2 ... Ln

The sum of the twig lengths is odd and $L1 + L2 + \cdots + Ln < 100000$ .

## Output

For each input line, print the area of the largest triangle, rounded to the nearest integer.

## Sample Input

```
3 4 2 5
4 1 3 7 10
4 3 3 8 11
```

## Sample Output

```
15
55
77
```

# Ultimate Tic-Tac-Toe (prob9)

## The Problem

Tic-tac-toe is a dreadfully boring game. It is partially so boring because it is a solved game: there is an optimal strategy. Two players who know what they are doing can always force a tie. Tic-tac-toe is a great game because of its simplicity, but players quickly grow tired of it for the same reasons.

However, there is a way to keep the same general rules and create a more interesting game. Thus, Ultimate Tic-Tac-Toe was born. The setup of an Ultimate Tic-Tac-Toe board is simple: Create a regular Tic-Tac-Toe board, and in each cell of this board add another Tic-Tac-Toe board. The goal in Ultimate Tic-Tac-Toe is to get 3 in a row, but the player must win the sub-game to earn a given cell.

The rules are generally the same, with one minor nuance: The previous player's move determines which sub-game the next player must play in. For example, if player one chooses to play the upper-left sub-game in the upper-right cell, then player two must play in the upper-right sub-game! Of course, the rules have to be slightly more complex with this new accommodation. Namely, if a player's move chooses a sub-game that is already completed, they can choose any sub-game to play in.

If that wasn't hard enough, a tied (or Cat) sub-game can count towards either player winning! Your task is to write a program that can validate a game of Ultimate Tic-Tac-Toe. It should determine if there were any invalid moves, and if so which move it was.

## Input

The first line of the input will be an integer specifying the number of ultimate tic-tac-toe games to follow.

For each game of ultimate tic-tac-toe, there is a single input line containing at most 250 characters that describes all moves of this game. Each move is encoded in a two digit number representing the row and column the player chose in the entire 9 x 9 grid. For example, 00 encodes the player placing their token in row 0 and column 0 of the entire game board. This means the player placed their token in the upper-left corner of the upper-left sub-game. 82 encodes the player placing their token in row 8 column 2 of the entire game board, meaning they placed their token in the lower-right cell of the lower-left sub-game.

Each move is separated by a single space, the moves on a line alternate between player one and player two, and the turn count is zero based.

## Output

You should have one line of output for each game stating whether it was valid or had an illegal move. If the game had no illegal moves, your program should output the string: “All moves were valid.” If an illegal move was detected your program should output the description of the first illegal move made using the following format:

Player one made an illegal move on turn 6.
Player two made an illegal move on turn 3.

Turns start counting at zero.

## Sample Input

```
3
00 00 00
44 53 71 54 74 55 77 53
44 33 22 78 38 07 23 70 51 83 72 47 43 41 45 57 84 73 50 60 02 17 52 88 67 04 24 63 12 37 05 27 68 80 66
```

## Sample Output

```
Player two made an invalid move on turn 1.
Player two made an invalid move on turn 7.
All moves were valid.
```

# Candies To Go (prob10)

## The Problem

John has N children and they love candies very much. Kids are of different age and eat different types of candies. The i-th child loves candies of i-th type, each candy of this type has weight $w_i$ and gives the child $s_i$ amount of happiness. The family is going for a trip and are packing their luggage. Father has a special box for candies. Unfortunately the box cannot fit all the candies, it can only fit candies of total weight W. John wants to maximize the total happiness of his children by taking some candies of each type. However, since children are of different age they have certain restrictions. Namely an i-th child must have more candies than child 1, child 2, .., child i-1, unless all of them have no candies at all (note, that it is ok if child i+1 have some candies).

In particular, the number of candies of type n should be larger than of any other type or all numbers should be 0. In other words, if an i-th child have some candies, all older children (i+1, i+2, etc) must have more candies.

Can you help John to solve the problem?

## Input

Input starts with a single integer, the number of test cases T (1 ≤ T ≤ 100). Each test case starts with an integer n (1 ≤ n ≤ 100), the number of children in the family and W (1 ≤ W ≤ 100) - the total weight the box can fit. The next 2 lines describes the candies and children satisfaction: first line contains n integers $w_i$ weights of candies of type i, the second line contains n integers $s_i$ - amount of satisfaction i-th child will have from each candy.

## Output

For each test case output a single integer - the total maximum amount of satisfaction of John's children from eating all the candies they will take on a trip.

## Sample Input

```
4
3 10
1 2 4
10 2 1
2 10
11 15
2 6
4 10
1 1 3 2
10 9 8 7
3 13
5 3 5
7 15 1
```

## Sample Output

```
4
0
35
17
```

## Rover (prob11)

### The Problem

You've just been hired onto the JPL's elite team of developers in charge of building the next Mars rover! As exciting as this is, it comes with a great deal of stress simply due to the great costs involved. So, your manager has given you your first task: Build a simulation of the new robot to see how it will behave on the rocky Martian terrain.

Here are the specs of the robot: It has driving wheels on each side, both with a 1 foot radius. The wheel base is also 1 foot (i.e. the wheels are a foot apart from each other). The wheels can spin only in the forward direction, at integral speeds. Furthermore, due to the limited memory of the robot, it can only hold up to 100 seconds worth of driving program at once. For the purposes of the simulation, you can assume the robot starts out with its center (the point directly between the two driving wheels) directly on the origin, and the robot facing down the positive X-axis.

The programs for the robot are given a special format, to enable good compression. The schedule of a single wheel is given in the following format:

v1|d1 v2|d2 ...

This means that the wheel being referred to should rotate at v1 rads/sec for d1 seconds, then v2 rads/sec for d2 seconds and so on.

### Input

Input starts with a single integer N telling how many cases follow. For each case you can assume that the robot resets, and starts back at the origin facing down the positive X-axis. Each case will consist of two lines: The first will have the schedule for the left wheel and the second line a schedule for the right. No schedule will be longer than 100 seconds, and there will never be more than 50 cases.

### Output

Output should consist of only the X and Y position of the robot after the program has ran for each case. Your X and Y positions should be formatted to 3 decimal places.

## Sample Input

```
2
5|1 5|1
5|1 5|1
0|1
3|1
```

## Sample Output

```
10.000 0.000
0.071 0.995
```

# What? Where? When? (prob12)

## The Problem

There is a popular intellectual TV game show called “What? Where? When?” where a team of 6 players tries to answer questions sent in by viewers. The time limit to answer a question is 1 minute. Answering these questions requires logical thinking, intuition, insight, etc. If a team answers a question correctly they get a point, otherwise a “team” of viewers gets a point. The game continues up to 6 points.

Questions are placed on sectors of a round table inside the envelopes. The next question is chosen at random using the arrow on the spinning top. After a question is selected it is removed from the table and the corresponding sector remains empty. If the question from the sector at which the arrow is pointing has already been played the clockwise next unplayed question is chosen. The following picture shows the situation with one question played and arrow pointing to the empty sector. In this case the clockwise next question will be chosen for the round.

Please note, that the size of envelopes doesn’t matter, the question is chosen if the arrow is pointing to the sector with the question, not the envelope itself. The sector with the number 13 on the picture represents a question from the Internet and should be considered a regular question which plays just once (after that the number 13 is taken out from the table).
Some questions are harder and some are easier. You cannot control the arrow, but you can estimate the probabilities of each question to be chosen in round k. Let’s enumerate sectors clockwise from 1 to n ($1 \le n \le 15$) starting at some sector. You will be given a state of the table sector as a sequence of 0s and 1s, where 0 means that question from this sector already played and 1 means that the sector still has a question. For example: 0 0 1 1 1 means that the table has 5 sectors and 2 questions from the 1st and 2nd sectors have already played and if in the next round the arrow will point at sector 1 the 3rd question will play and the state of the table will

be: 0 0 0 1 1. Your task is given a state of the table and number of rounds left k estimate the probability for a question from each sector to be chosen at round k.

## Input

The input consists of several test cases. The first line of the input contains a single integer - the number of test cases. Each test case starts with an integer n (1 ≤ n ≤ 15) - number of sectors on the table and k (1 ≤ k ≤ n) - the number of rounds left. The next line contains n integers - the current state of the table. 0 means that there is no question on this sector and 1 means that the there is a question on the sector.

## Output

For each test case print probabilities for question from each sector to be chosen at round k. The output should have 6 digits of precision.

## Sample Input

```
5
5 3
1 1 1 1 1
5 1
0 0 0 1 1
4 1
1 1 0 0
5 2
0 0 0 1 1
3 1
1 0 1
```

## Sample Output

```
0.200000 0.200000 0.200000 0.200000 0.200000
0.000000 0.000000 0.000000 0.800000 0.200000
0.750000 0.250000 0.000000 0.000000
0.000000 0.000000 0.000000 0.200000 0.800000
0.333333 0.000000 0.666667
```

## Explanation for What? Where? When?

In the first example all questions have equal probabilities to be chosen.
In the second example questions 1-3 have already played and if the arrow chooses these sectors the 4th question will play, the 5th question will be chosen only if arrow points at it directly. So, for questions 1-3 we have probability 0, for 4th question ⅘ and for the last question just ⅕.
Third sample test is similar to the second one. Note that the table is round and the 1st sector is clock-wise next to the last sector.
In the third example, the 4th question has a higher probability to be chosen on the first step. In the next round question 5 will be chosen if question 4 was chosen at the first step, which will happen with probability 0.8.
In the last sample test we have 3 questions and question number 1 will only be chosen if the arrow points at it directly, and otherwise question 3 will play.

# Tree Sales (prob13)

## The Problem

Several well-known companies use a pyramid sales scheme. Being both an entrepreneur AND a computer scientist, however, you prefer to model your new business as a tree sales scheme, where the hierarchical structure of the company can be modeled as a tree.

In particular, the company initiator, or CEO, is the root of the tree structure of the company. From there, any current member of the company can hire a direct subordinate. So, at the beginning, it's up to the CEO to hire other employees who will be directly below the CEO in the tree structure. At any point in time, any employee can make a sale. Total compensation of any employee is calculated based on the sum of sales of all members of the subtree rooted at that employee, so it's important at any point in time to be able to calculate the total sales in any subtree of the company structure.

In order to start your company and allow others to start similarly structured companies, you have decided to write a computer program that will read in a set of operations from the following set:

1) Add an employee (first add is the CEO, rest are made by current employees),
2) An employee makes a sale, and
3) Query for the total sales in an employee's subtree at that point in time,

and execute the appropriate command, in the order given, producing output for all commands of type three.

## Input

The first line of input will contain a single integer, T (T ≤ 10), representing the number of company structures to analyze. The first line of each company structure to analyze will contain a single positive integer, n (n ≤ 100000), representing the number of operations to execute for that company. The following n lines will each contain a single command with one of the following three formats:

```
ADD SPONSOR NEWEMPLOYEE
SALE EMPLOYEE X
QUERY EMPLOYEE
```

All employee names will be strings of 1 to  10 uppercase letters. In the first format, SPONSOR will be an existing employee who is hiring a new employee, and the NEWEMPLOYEE will be the new employee to be added directly below the sponsor. The very first command for each company will be an add with the sponsor "ROOT", indicating that NEWEMPLOYEE is the root of the tree structure for that company. No employee of any company will be named "ROOT". In the second format, EMPLOYEE will be the employee in question and X will be a positive integer less than or equal to  1000 representing the value of the sale made by the given employee. In the third format, EMPLOYEE will be the employee in question for which we must find the total sales of her subtree in the company. All names given for current employees for all three types of commands are guaranteed to be valid current employees. It is guaranteed that all employees added will be identified by distinct strings and that the tree structure produced will not have a height greater than 100. (Note: The height of a tree with two nodes is 1.)

## Output

For each company output a single line header of the form

```
COMPANY K
```

where K is the number of the company, starting with 1. For each query (command of type 3 in the input), output a single line with a positive integer representing the current total sales of the subtree of the employee queried. Note: Each company will have at least one query.

## Sample Input

```
2
14
ADD ROOT BILL
ADD BILL EVELYN
ADD BILL SARAH
SALE BILL 25
SALE EVELYN 75
SALE SARAH 10
QUERY BILL
ADD EVELYN MATT
ADD MATT ANYA
SALE ANYA 1000
QUERY MATT
QUERY EVELYN
QUERY BILL
QUERY SARAH
11
ADD ROOT MARILYN
ADD MARILYN GARY
ADD MARILYN REMY
ADD MARILYN BRIANNE
ADD MARILYN TAJ
SALE TAJ 10
SALE REMY 20
SALE BRIANNE 40
SALE MARILYN 30
QUERY GARY
QUERY MARILYN
```

## Sample Output

```
COMPANY 1
110
1000
1075
1110
10
COMPANY 2
0
100
```