CCC '01 S1 - Keeping Score

Time limit: 2.0s **Memory limit:** 256M

Canadian Computing Competition: 2001 Stage 1, Junior #3, Senior #1

In a card game, each player's hand is made up of 13 cards. Each hand has a total point value determined by the number of cards that have a point value. The cards which are worth points are the `Ace` (4 points), `King` (3 points), `Queen` (2 points) and `Jack` (1 point). The other cards `(2, 3, 4, 5, 6, 7, 8, 9, 10)` have no point value.

There are four of each type of card, one in each of the four suits. The suits are called clubs (`C`), diamonds (`D`), hearts (`H`), and spades (`S`). As well, points are assigned for each suit which has a `void` (3 points), a `singleton` (2 points), or a `doubleton` (1 point). A void in a suit means that there are no cards of that suit (e.g. a hand with no spades). A singleton in a suit means that there is only one card in that suit (e.g. a hand with only one diamond). A doubleton in a suit means that there are only two cards in that suit.

Write a program to read a set of thirteen cards in the form of a string, then evaluate the number of points in the hand. The suits will appear in increasing alphabetical order. Within each suit there will be no duplicate cards.

The output is to be the hand and the point value shown in a table form as below. Your output should list the cards in the same order as the input. Note that `10` is represented by the character `T` in both the input and the output.

Sample Input 1

```
C258TJKD69QAHSTJA
```

Sample Output 1

```
Cards Dealt              Points
Clubs 2 5 8 T J K             4
Diamonds 6 9 Q A              6
Hearts                        3
Spades T J A                  5
                     Total 18
```

```
separate each suite like
C258TJK
D69QA
H
STJA

now for each suite, add points
if its a void singleton or
doubleton

then add points for A, K, Q, J

ignore numbered cards
```

Sample Input 2

```
CAD578KAHAS47TQKA
```

Sample Output 2

```
Cards Dealt              Points
Clubs A                       6
Diamonds 5 7 8 K A            7
Hearts A                      6
Spades 4 7 T Q K A            9
                        Total 28
```

Note: your output does not need to match exactly. The spacing is up to you.

CCC '02 S1 - The Students' Council Breakfast

Time limit: 2.0s **Memory limit:** 256M

Canadian Computing Competition: 2002 Stage 1, Junior #3, Senior #1

The students council in your school wants to organize a charity breakfast, and since older students are both wiser and richer, the members of the council decide that the price of each ticket will be based on how many years you have been in the school. A first year student will buy a PINK ticket, a second year student will buy a GREEN ticket, a third year student will buy a RED ticket, and a fourth year student will buy an ORANGE ticket.

Assume that all tickets are sold. Each colour of ticket is uniquely priced.

Input Specification

Input the cost of a PINK, GREEN, RED, and ORANGE ticket (in that exact order), followed by the exact amount of money to be raised by selling tickets.

Output Specification

Output all combinations of tickets that produce exactly the desired amount to be raised. The combinations may appear in any order. Output the total number of combinations found. Output the smallest number of tickets to print to raise the desired amount so that printing cost is minimized.

Sample Input

```
1
2
3
4
3
```

```
used backtracking to check all possible ways of selecting tickets
and saving the combinations that lead to money needed to be raised

total combinations is the length of the list thats store the tuples

minimum is the tuple with the smallest sum
```

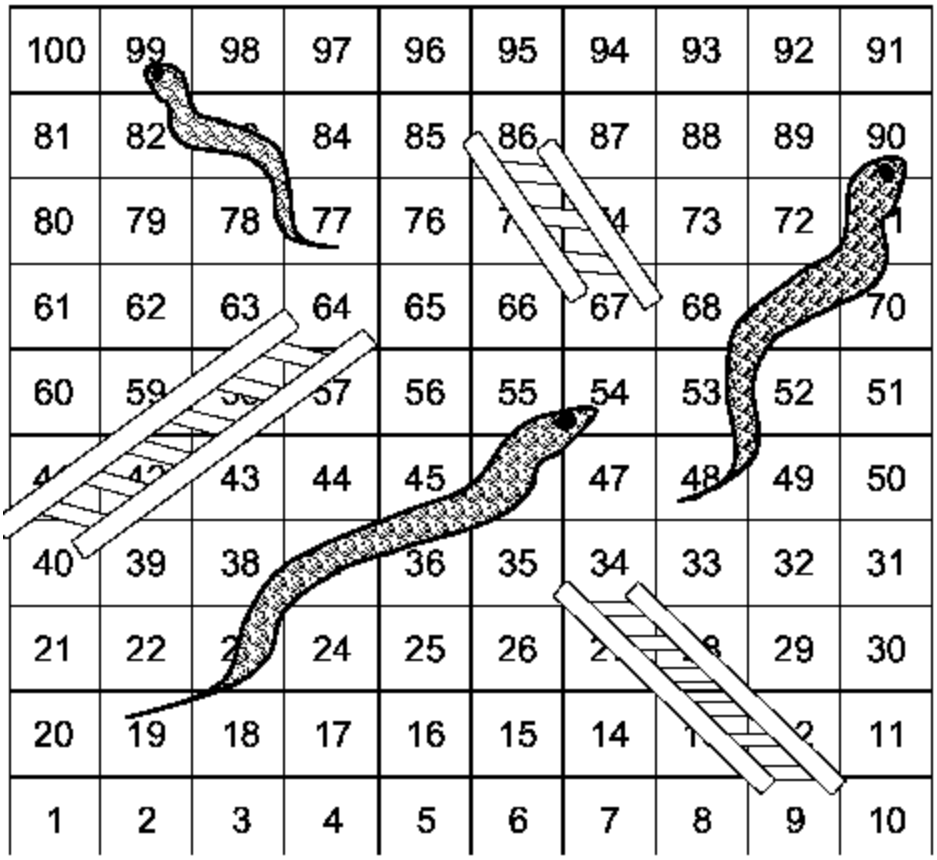
Sample Output

```
# of PINK is 0 # of GREEN is 0 # of RED is 1 # of ORANGE is 0
# of PINK is 1 # of GREEN is 1 # of RED is 0 # of ORANGE is 0
# of PINK is 3 # of GREEN is 0 # of RED is 0 # of ORANGE is 0
Total combinations is 3.
Minimum number of tickets to print is 1.
```

CCC '03 S1 - Snakes and Ladders

Time limit: 2.0s **Memory limit:** 256M

Canadian Computing Competition: 2003 Stage 1, Junior #3, Senior #1



Here (see illustration) is a game board for the game Snakes and Ladders. Each player throws a pair of dice to determine how many squares his/her game piece will advance. If the piece lands on the bottom of a ladder, the piece moves up to the square at the top of the ladder. If the piece lands on the top of a snake, the piece "slides" down to the square at the bottom of the snake. If the piece lands on the last square, the player wins. If the piece cannot advance the number of squares indicated by the dice, the piece is not moved at all.

In order to help you play this game via a cell phone while travelling, you will write a program that simulates your moves on the board shown and, of course, runs on your handheld computer. You will repeatedly throw the dice and enter the result into the program. After each throw, the program will report the number of the square where your piece lands.

When the program starts it should assume the piece is on square 1. It should repeatedly read input from the user (a number between 2 and 12) and report the number of the square where the piece lands. In addition, if the piece moves to the last square, the program should print `You Win!` and terminate. If the user enters 0 instead of a number between 2 and 12, the program should print `You Quit!` and terminate.

For clarity, you are to use the board pictured above and you should note that the board has 3 snakes (from 54 to 19, from 90 to 48 and from 99 to 77) and 3 ladders (from 9 to 34, from 40 to 64 and from 67 to 86).

Sample Input

```
use a dict to store snake and ladder pairs, eg
if at 54 check dicts and go to 19 since 54:19
exists in snake dict

shouldnt go out of board <= 100 after rolling
dice
```

```
9
11
12
7
3
5
10
9
```

Sample Output

```
You are now on square 10
You are now on square 21
You are now on square 33
You are now on square 64
You are now on square 86
You are now on square 91
You are now on square 91
You are now on square 100
You Win!
```

CCC '00 S2 - Babbling Brooks

Time limit: 1.0s **Memory limit:** 256M

Canadian Computing Competition: 2000 Stage 1, Junior #4, Senior #2

A series of streams run down the side of a mountain. The mountainside is very rocky so the streams split and rejoin many times. At the foot of the mountain, several streams emerge as rivers. Your job is to compute how much water flows in each river.

At any given elevation there are m streams, labelled 1 to m from left-to-right. As we proceed down the mountainside, one of the streams may split into a left fork and a right fork, increasing the total number of streams by 1, or two streams may rejoin, reducing the total number of streams by 1. After a split or a rejoining occurs, the streams are renumbered consecutively from left-to-right. There is always at least one stream and there are never more than 100 streams.

Input Specification

The first line of input contains n, the initial number of streams at some high altitude. The next n lines give the flow in each of the streams from left-to-right. Proceeding down the mountainside, several split or rejoin locations are encountered. For each **split** location, there will be three lines of input.

- a line containing `99` (to indicate a split)
- a line containing an integer, the number of the stream that is split
- a line containing an integer between 0 and 100, the percentage of flow from the split stream that flows to the left fork. (The rest flows to the right fork)

For each **join** location, there will be two lines of input:

a line containing `88` (to indicate a join)
a line containing an integer, the number of the stream that is rejoined with the stream to its right

The flow from both joined streams is combined. After the last split or join location will be:

a single line containing `77` (to indicate end of input)

Output Specification

Your job is to determine how many streams emerge at the foot of the mountain and what the flow is in each. Your output is a sequence of real numbers, rounded to the nearest integer, giving the flow in rivers 1 through m.

Sample Input

```
make a list of given water in streams
[None,10,20,30] add dummy element at start to
make it 1 indexed

then processor each join and split
eg:
99
1
50

means split first stream in half
so overwrite 10 with the new right stream 5

and insert the new left stream 5 at this same
index (right will move over 1 index and will
remain to the right of this)

eg:
88
3
88

list is now [None,5,5,20,30]
so join 3rd stream with the one on its right
so 20 with 30

delete the right stream 30 and overwrite 20
with the new joined stream value 50
```

```
3
10
20
30
99
1
50
88
3
88
2
77
```

Sample Output

```
5 55
```

CCC '01 S2 - Spirals

Time limit: 2.0s **Memory limit:** 256M

Canadian Computing Competition: 2001 Stage 1, Junior #4, Senior #2

A spiral of numbers can start and end with any positive integers less than 100. Write a program which will accept two positive integers x and y as input, and output a list of numbers from x to y inclusive, shown in a spiral. You may assume that the end value is greater than or equal to the start value.

A spiral starts with the first number in the centre. The next number appears immediately below the first number. The spiral continues with the numbers increasing in a counter-clockwise direction until the last number is printed.

Sample Input 1

```
10
27
```

Sample Output 1

```
      27 26
16 15 14 25
17 10 13 24
18 11 12 23
19 20 21 22
```

Sample Input 2

```
7
12
```

Sample Output 2

```
12 11
 7 10
 8  9
```

notice that the number of steps in a direction
increases by 1 after completing 2 directions,

1 step down to 11
1 step right to 12

increment steps by 1

2 steps up to 14
2 steps left to 16

and so on

assume position of x as (0,0)

so 10 is at (0,0)

now simulate the down, right, up, left spiral,
moving 'steps' number in each direction at a time

after placing them all, we would have negative
indexes for the numbers placed above 10

we can fix those by finding the largest negative row
and column, and adding abs()
back to each row and col

so 27 at (-2,1) would instead become (0,2)

-2 + abs(-2)
 1 + abs(-1)

CCC '02 S2 - Fraction Action

Time limit: 0.5s **Memory limit:** 256M

Canadian Computing Competition: 2002 Stage 1, Junior #4, Senior #2

Many advanced calculators have a fraction feature that will simplify fractions for you.

You are to write a program that will accept for input a non-negative integer as a numerator and a positive integer as a denominator, and output the fraction in simplest form. That is, the fraction cannot be reduced any further, and the numerator will be less than the denominator. You can assume that all input numerators and denominators will produce valid fractions.

Sample Input 1

```
28
7
```

Sample Output 1

```
4
```

Sample Input 2

```
13
5
```

Sample Output 2

```
2 3/5
```

Sample Input 3

```
0
7
```

```
check if divisible, R == 0 then just print quotient.
x = 55
y = 10

Q = 5
R = 5

otherwise the fraction may be proper or improper.

if it were improper, Q would be > 0

if proper, Q would be == 0

For the improper case, we need to make it a mixed
fraction. So R and D would change (Q + R/D)

Now for both cases, we need ensure that the proper
fraction R/D is in simplified form

so use math.gcd(R,D) to find the largest common
divisor and divide both R and D by it

Now print as mixed (Q + R/D) if original was
improper (x>y)

Or as proper (R/D)
```

Sample Output 3

```
0
```

Sample Input 4

```
55
10
```

Sample Output 4

```
5 1/2
```

CCC '03 S2 - Poetry

Time limit: 2.0s **Memory limit:** 256M

Canadian Computing Competition: 2003 Stage 1, Junior #4, Senior #2

A simple poem consists of one or more four-line verses. Each line consists of one or more words consisting of upper or lowercase letters, or a combination of both upper and lowercase letters. Adjacent words on a line are separated by a single space.

We define the last syllable of a word to be the sequence of letters from the last vowel (`a`, `e`, `i`, `o`, or `u`, but not `y`) to the end of the word. If a word has no vowel, then the last syllable is the word itself. We say that two lines rhyme if their last syllables are the same, ignoring case.

You are to classify the form of rhyme in each verse. The form of rhyme can be *perfect, even, cross, shell*, or *free*:

- perfect rhyme: the four lines in the verse all rhyme
- even rhyme: the first two lines rhyme and the last two lines rhyme
- cross rhyme: the first and third lines rhyme, as do the second and fourth
- shell rhyme: the first and fourth lines rhyme, as do the second and third
- free rhyme: any form that is not perfect, even, cross, or shell.

The first line of the input file contains an integer N, the number of verses in the poem, $1 \leq N \leq 5$. The following $4N$ lines of the input file contain the lines of the poem. Each line contains at most 80 letters of the alphabet and spaces as described above.

The output should have N lines. For each verse of the poem there should be a single line containing one of the words `perfect`, `even`, `cross`, `shell`, or `free` describing the form of rhyme in that verse.

Sample Input 1

```
1
One plus one is small
one hundred plus one is not
you might be very tall
but summer is not
```

Output for Sample Input 1

```
cross
```

Sample Input 2

process 4 lines at a time since that makes 1 verse and repeat the following for each verse:

find last syllable of the 4 lines

then do the rhyme if conditions in given order

eg

```
elif o == s and t == f:
    form = 'even'
```

```
2
I say to you boo
You say boohoo
I cry too
It is too much foo
Your teacher has to mark
and mark and mark and teach
To do well on this contest you have to reach
for everything with a lark
```

Output for Sample Input 2

```
perfect
shell
```

Sample Input 3

```
2
It seems though
that without some dough
creating such a bash
is a weighty in terms of cash
But how I see
the problem so fair
is to write subtle verse
with hardly a rhyme
```

Output for Sample Input 3

```
even
free
```

CCC '02 S3 - Blindfold

Time limit: 1.0s **Memory limit:** 256M

Canadian Computing Competition: 2002 Stage 1, Junior #5, Senior #3

Rose and Colin are playing a game in their backyard. Since the backyard is rectangular, we can think of it as a grid with r rows and c columns. Rose and Colin place obstacles on some of the squares.

The game is played as follows:

Colin covers his eyes with a blindfold then Rose carries him to some grid square in the backyard. She sets him down so that he is facing north, south, east, or west. Colin does not know this initial position or direction. Rose then instructs Colin to make a series of m moves through the backyard. Each move is one of:

- `F` - moves forward one grid square in the direction that he is facing, or
- `L` - turns 90 degrees counter-clockwise, remaining on the same square, or
- `R` - turns 90 degrees clockwise, remaining on the same square.

After making these moves, Colin is standing at some final position. He must now figure out where he is standing. You will help him by writing a program to determine all possible final positions. Assume that Colin's initial position, final position, and all intermediate positions lie within the backyard but never in a square that contains an obstacle. You may also assume that Colin is always facing a direction that is parallel to the sides of the backyard (north, south, east, or west).

Input Specification

The input begins with r and c $(1 \leq r \leq 375; 1 \leq c \leq 80)$, each on a separate line. Next are r lines of c characters describing the backyard: a `.` denotes a grid square that Colin may walk through; an `X` denotes a grid square with an obstacle. Below the grid is the number m $(0 \leq m \leq 30\,000)$ followed by m lines describing Colin's moves. Each line has a single character: `F`, `L`, or `R`.

Output Specification

Your program should output the backyard grid, indicating all possible final positions with `*`.

Sample Input

```
Assume each empty cell as starting position and run the simulation for given moves, if at any point we cannot perform the move because next position is blocked, then it means the starting position was not valid. Otherwise just mark the ending postion.

Also, since colin may be facing any direction NSEW in the start, we need to run the simulation for all 4 directions for each starting position.
```

```
2
4
....
.XX.
3
F
R
F
```

Sample Output

```
.*..
.XX*
```