

A	Orthogonal Bar Graphs	bar.in	bar.out	2 seconds	64 MiB
B	Binary Conversions	binary.in	binary.out	2 seconds	64 MiB
C	Lost in Translation	lost.in	lost.out	2 seconds	64 MiB
D	Quantum Teleporters	teleport.in	teleport.out	2 seconds	64 MiB
E	Tennis Probability	tennis.in	tennis.out	2 seconds	64 MiB
F	Trophy Case	trophy.in	trophy.out	2 seconds	64 MiB

Problem A- Orthogonal Bar Graphs

Some bar graphs fill the plane such that the empty space (also referred to as the *negative space*) can be filled by an orthogonal bar graph. Some cannot, but would if you permuted its columns.

The problem is this: given a bar graph, find its orthogonal bar graph.

Input Specification:

The input begins with an integer t denoting the number of test cases that follow. Each test case will be a positive integer n , the number of bars, and then n positive integers denoting the lengths of each bar. In all cases, $n \leq 100000$, and the lengths of the bars will not exceed 1000000.

Output Specification:

Your program will output a sequence of integer pairs, $N L$, one per line, separated by a single space. The pair will represent N bars of length L in the orthogonal bar graph. Present them in order of increasing lengths. Terminate the output of each test case (even the last test case) by a single blank line.

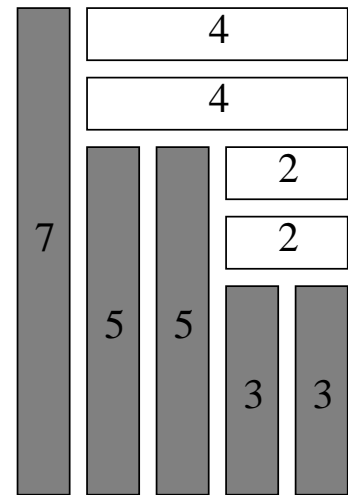
Sample Input:

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

Sample Output:

```
3 1
4 2

2 2
2 4
```



Problem B- Binary Conversions

It is well known that the decimal expansion of $1/3$ is never-ending. Eventually, you have to chop the number, losing some precision.

Binary numbers experience the same problem. Even the decimal number 0.2 gives a repeating decimal that looks like 0.0011001100110011... which must be chopped or rounded at some point.

011011100101110111
10001001101010111100
110111101111100001000110
010100111010010101101101
0111110001100111010110111100
111011110111111000010001.....

Input Specification:

Each line of input presents a single test case: a string of decimal digits, no longer than 120 characters long. There will be no extraneous spaces in the input.

The input ends on EOF.

Output Specification:

For each test case, determine if its binary representation is exact or loses some precision. Output "Exact." or "Some precision lost." as appropriate.

Sample Input:

0.75
0.2
0.625

Sample Output:

Exact.
Some precision lost.
Exact.

Problem C- Lost in Translation

In translating from a word in Argharian to a word in Ackanese, it suffices to make exactly one lexical change. The following are a comprehensive collection of possible single lexical changes:

- An Argharian word ending in ‘e’ has the same meaning as the Ackanese with the exact same spelling, but with the ‘e’ moved to the second last character. E.g. lute → luet.
- Similarly, if there is a ‘rge’ combination in Argharian, you can replace it with ‘che’ in Ackanese. E.g. large → lache.
- Any ‘re’ combination can be replaced by a ‘rec’. E.g. pretty → prectty.
- Any ‘ce’ combination can be replaced by an ‘ec’ combination. E.g. ice → iec.

All words in Argharian are between 3 and 12 characters long, inclusive. If a rule applied in reverse generates a word outside of this range, it is not valid of Argharian and should be discarded.

You can use these rules to find all *synonyms* (words of the same meaning) of an Argharian word. By translating from Argharian to Ackanese and back again, you will have two Argharian words of the same meaning.

You should be reminded that when A is a synonym of B , and B is a synonym of C , then A is a synonym of C .

Write a program that finds a complete list of synonyms for Argharian words.

Input Specification:

You will be presented Argharian words: 3 to 12 lowercase characters, one per line. The input ends with EOF.

Output Specification:

For each word, output its Argharian synonyms, one per line, in alphabetical order. Head each by a case number, as per the sample. Separate successive test cases by a blank line.

Sample Input:

farce
merge
cererge

Sample Output:

Case 1:

farce
fare

Case 2:

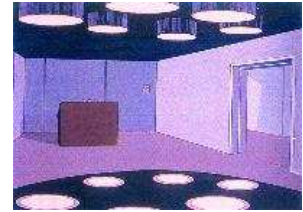
mcehe
merge

Case 3:

cerccehe
cercehe
cercerge
cerehe
cererge

Problem D- Quantum teleporters

It's the future, and all public transit has been replaced with a network of Quantum Teleporterstm! Unfortunately they have a peculiar feature- each teleporter can be in one of two states (let's call them A and B), and you won't know what the states are until you take the entire trip (just like quantum states!).



The states affect the time it takes to be teleported from one place to another. The travel time between two connected teleporters depends on the states they ended up in: AA , AB , BA or BB . Four possible states mean four possible travel times. You need to get to a programming contest and since you absolutely cannot be late, you need to find the path with the smallest guaranteed total time. (That is, once you choose the path, you should assume the states will be in the worst possible configuration.)

Input Specification:

The first line of the input contains a nonnegative integer $T \leq 20$, the number of test cases to follow. Each test case starts with a line containing two integers: N , the number of teleporters, themselves numbered from 0 to $N - 1$ ($2 \leq N \leq 50$), and M , the number of teleporter connections. M lines follow, each containing six integers: U , V , $A_U A_V$, $A_U B_V$, $B_U A_V$, $B_U B_V$; which describe a two-way connection between teleporters U and V , with travel times that depend on the states of U and V . Travel times are integers in the range 0 to 10 inclusive.

In the data, there will be at most one connection between each pair of teleporters and no teleporter will be connected to itself. You start at teleporter 0 and the programming contest is at teleporter 1. There will always be a route from 0 to 1.

Output Specification:

For each test case, output the smallest time you can guarantee the trip will take. Output one case per line.

Sample Input:

```
1
3 2
0 2 1 10 1 10
2 1 9 9 1 1
```

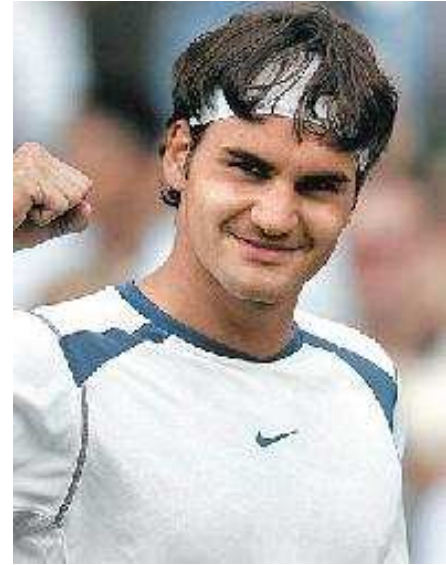
Sample Output:

```
11
```

Problem E- Tennis probability

You know how a *game* is scored in tennis? A game consists of *points*. A point is started with a serve and continues until one of the players fails to return the ball properly. The first player to score four or more points, while leading by at least two, wins the game. (As an aside, a tennis match consists of sets, which consist of games, but we are only concerned with games in this problem).

Suppose you are watching a tennis match and you know the probability that your favorite player will win a point against his current opponent. You assume that all points are independent (the player does not get discouraged after a long losing streak and vice versa). What is the probability that the player will win a game?



Input Specification:

The first line of the input contains a nonnegative integer $T \leq 200$, the number of test cases to follow. Each of the next T lines contains a single floating point number p the probability that the player will win a point ($0 \leq p \leq 1$).

Output Specification:

For each test case, print one line: the probability that the player will win a game, rounded to 5 decimals.

Sample Input:

```
2
0
1.0
```

Sample Output:

```
0.00000
1.00000
```

Problem F- Trophy Case

Igor has won so many ACM trophies that he wants to put them on display for his equally many girlfriends to swoon over. They will appear, side-by-side, on his mantle.

To display the trophies properly, the tallest one must be in the middle somewhere (duh). The trophies that appear on either side must be successively shorter. Put another way, a *proper display* of trophies will be a sequence of trophies of increasing heights, until the tallest trophy, followed by a sequence of trophies of decreasing heights.



Naturally, there are many proper displays, but not all are as *attractive* as others. Measure the attractiveness by the differences in width of adjacent trophies. The lower the total differences are, the more attractive is the display.

Input Specification:

The input will be a series of test cases, presented one line at a time, and terminated by EOF. Each line will be a series of positive integers, the first of which is $n \leq 450$, the number of trophies. Then there will be n pairs of positive integers $h_i w_i$, which describe the height and width of each trophy. The trophies will be input by increasing heights. All heights are distinct.

Output Specification:

For each test case, output the best attractiveness score for Igor's trophies.

Sample Input:

```
3 5 1 6 4 7 2
5 5 1 6 4 7 8 8 2 10 5
```

Sample Output:

```
3
11
```