Binarize It

filename: `binarize`
(*Difficulty Level:* `Easy`)

Professor Boolando can only think in binary, or more specifically, in powers of 2. He converts any number you give him to the smallest power of 2 that is equal to or greater than your number. For example, if you give him 5, he converts it to 8; if you give him 100, he converts it to 128; if you give him 512, he converts it to 512.

The Problem:

Given an integer, your program should binarize it.

The Input:

The first input line contains a positive integer, *n*, indicating the number of values to binarize. The values are on the following *n* input lines, one per line. Each input will contain an integer between 2 and 100,000 (inclusive).

The Output:

At the beginning of each test case, output "`Input value:` *v*" where *v* is the input value. Then, on the next output line, print the binarized version. Leave a blank line after the output for each test case.

Sample Input:

```
3
900
16
4000
```

Sample Output:

```
Input value: 900
1024

Input value: 16
16

Input value: 4000
4096
```

g2g c u l8r

filename: `texting`
(*Difficulty Level:* `Easy`)

According to the national statistics, a teenager sends/receives 100+ text messages a day. Dr. Orooji's teenage children are no exception but the problem is Dr. O (an old-fashioned, face-to-face communicator) has difficulty reading text messages full of abbreviations (short-hands) sent to him by his children. Dr. O needs your help reading these text messages.

The Problem:

Given the list of abbreviations and a paragraph, you are to expand the text (paragraph) so that Dr. O can read it easily.

The Input:

The first input line contains an integer, n *(*$1 \leq n \leq 20$*)*, indicating the number of abbreviations. These abbreviations are on the following n input lines, one per line. Each input line starts in column 1 and contains an abbreviation (1-5 characters, consisting of only lowercase letters and/or digits). The abbreviation is followed by exactly one space, and this is followed by the expanded version of the abbreviation (1-50 characters, consisting of only lowercase letters and spaces; assume the expanded version does not start or end with a space and contains no multiple consecutive spaces between words). Assume that all abbreviations are distinct, i.e., no duplicates.

The list of abbreviations is followed by a positive integer, p, indicating the number of input lines containing the paragraph to be expanded. The paragraph is on the following p input lines. Assume these input lines do not exceed column 50, do not start or end with a space, and each line contains at least one word. The paragraph will contain only lowercase letters, digits, and spaces. Assume that there will not be multiple consecutive spaces in the input paragraph.

A word is defined as a consecutive sequence of letters/digits. Assume that a word will be entirely on one input line, i.e., a word is not broken over two or more lines.

The Output:

Each line of the input paragraph must be on one line of output. The input line must be printed in the output exactly the same (spacing). The only exception is that each abbreviation must be replaced by its expanded version, i.e., when an abbreviation is found in the input, its expanded version must be output.

Note that an abbreviation must match a word completely and not just part of a word. For example, if u is an abbreviation for "you", then u must appear as a word by itself in the paragraph

in order to be replaced, i.e., if the abbreviation is part of a word in the paragraph (e.g., the paragraph contains the word buy or ugly or you), the u in these words should not be replaced.

Sample Input:

```
8
g2g got to go
g good
c see
l8 late
l8r later
d i am done
u you
r are
6
hi
how r u
you tell me
you are l8
d
c u l8r
```

Sample Output:

```
hi
how are you
you tell me
you are late
i am done
see you later
```

Tip to be Palindrome

filename: `tipit`
(*Difficulty Level:* `Easy`)

One of the cool UCF CS alumni is Dr. Greg, The Palindrome Tipper. A palindrome is a string that reads the same forward and backward, e.g., madam, abba, 3, 44, 525.

One cool thing about Dr. Greg is that he leaves at least 20% tip when he eats out, e.g., if the meal is $30, Dr. Greg leaves $6 (30*.20) for tip. If the tip (20%) is not a whole dollar amount, he rounds up the tip to make it a whole number. For example, if the meal is $12, a 20% tip would be $2.40 (12*0.20) but Dr. Greg would leave $3 for tip.

Another cool thing about Dr. Greg is that he is a palindrome guru. If his total bill (meal plus tip) is not a palindrome, he will increase the total (by adding to the tip) to make the total a palindrome. He will, of course, add the minimum needed to make the total a palindrome.

The Problem:

Given Dr. Greg's meal cost, your program should determine the tip amount for him (according to his rules) and the total bill.

The Input:

The first input line contains a positive integer, *n*, indicating the number of times Dr. Greg ate out. The meal costs are on the following *n* input lines, one per line. Each input will contain an integer between 5 and 10000 (inclusive).

The Output:

At the beginning of each test case, output "`Input cost: c`" where *c* is the input cost. Then, on the next output line, print the tip amount and the total bill, separated by one space. Leave a blank line after the output for each test case.

Sample Input:

```
2
12
84
```

Sample Output:

```
Input cost: 12
10 22

Input cost: 84
17 101
```

Soccer Standings

filename: `soccer`
(*Difficulty Level:* `Medium`)

Soccer fever has gripped the world once again, and millions of people from dozens of countries will be glued to their TV sets for the World Cup. Being an enterprising sort, you've started up your own internet World Cup Soccer Channel for streaming matches online. Recently you came up with the idea of filling up the time between matches by having a couple of 'experts' offer critical analysis of games. For this purpose, you have devised a unique ranking system for soccer teams, which you must now implement.

The Problem:

Given a list of teams and a list of match scores, you must compute several quantities for each team. These are: the total number of goals scored over all their games, the total number of goals scored against them (goals allowed, for short), the number of wins, draws and losses, and the number of points scored so far. Points are to be computed as follows: winning a match nets a team 3 points, losing gets them nothing. In the event of a tie, both teams get 1 point.

In addition to this, you must order the teams correctly according to your new system. Teams are ordered according to points, from highest to lowest. In the event of a tie in points, the team that has a higher *goal difference* comes first. The goal difference is defined as the total number of goals scored by the team minus the total number of goals scored against them.

If there is still a tie (i.e., two or more teams have the same points and the same goal differences), the team with higher total goals scored comes first. If even this is tied, the team whose name comes first in alphabetical order goes first.

The Input:

The first input line contains a positive integer, n, indicating the number of data sets to be processed. The first line of each data set consists of two positive integers T ($T \leq 30$) and G ($G \leq 400$) – the number of teams in this group and the total number of games played by them. The next line contains T unique names separated by single spaces. Each name is a single uppercase word with no more than 15 characters.

Each of the next G input lines will contain the results of a match. Each line is of the form *<country_1> <score_1> <country_2> <score_2>*. For example, "Greece 2 Nigeria 1" indicates that Greece and Nigeria played a game with score 2-1. All four terms will be separated by single spaces.

The Output:

At the beginning of output for each data set, output "`Group g:`" where g is the data set number, starting from 1. Next you should print a single line for each team, ordering teams as mentioned above. For each team, the line you print should be of the form "*<name> <points> <wins> <losses> <draws> <goals scored> <goals allowed>*". These items should be separated by single spaces. Leave a blank line after the output for each data set.

Sample Input:

```
2
2 1
KASNIA LATVERIA
KASNIA 0 LATVERIA 1
4 6
ENGLAND USA ALGERIA SLOVENIA
ENGLAND 1 USA 1
ALGERIA 0 SLOVENIA 1
SLOVENIA 2 USA 2
ENGLAND 0 ALGERIA 0
SLOVENIA 0 ENGLAND 1
USA 1 ALGERIA 0
```

Sample Output:

```
Group 1:
LATVERIA 3 1 0 0 1 0
KASNIA 0 0 1 0 0 1

Group 2:
USA 5 1 0 2 4 3
ENGLAND 5 1 0 2 2 1
SLOVENIA 4 1 1 1 3 3
ALGERIA 1 0 2 1 0 2
```

NIH Budget

filename: `nih`

(*Difficulty Level:* `Medium`)

Recently, a job for an algorithms specialist opened up at NIH. You never thought you'd be using your expertise in algorithms to save lives, but now, here is your chance! While the doctors are very good in carrying out medical research and coming up with better cures for diseases, they are not so good with numbers. This is where you come in.

You have been tasked to allocate money for all disease research at NIH. The interesting thing about disease research is that the number of lives saved doesn't linearly increase with the amount of money spent, in most cases. Instead, there are "break-points". For example, it might be the case that for disease A, we have the following break-points:

Research Funding	Lives Saved
10 million	5
50 million	100
100 million	1000
250 million	1100

If you spend more money than one breakpoint and less than another, the number of lives saved is equal to the amount saved for the previous breakpoint. (In the above example, if you spent $150 million, you'd still only save 1000 lives, and if you spent any amount more than $250 million, you'd still save 1100 lives.)

The doctors have figured out charts just like this one for all the diseases for which they do research. Given these charts, your job will be to maximize the number of lives saved spending no more than a particular budget.

The Problem:

Given several charts with information about how much has to be spent to save a certain number of lives for several diseases and a maximum amount of money you can spend, determine the maximum number of lives that can be saved.

The Input:

The first input line contains a positive integer, n ($n \leq 100$), indicating the number of budgets to consider. The first line of each budget contains two positive integers, d ($d \leq 10$), representing the number of diseases for which there is data and B ($B \leq 100000$), the total budget, in millions of dollars. The following d lines contain information about each of the d diseases. Each of these lines will contain exactly four ordered pairs of positive integers separated by spaces. Each pair will represent a dollar level (in millions) followed by the number of lives saved for that dollar

level of funding. Each of the pairs will be separated by spaces as well. Each of these values will be less than or equal to 100,000. Assume that the dollar levels on an input line are distinct and in increasing order, and that the number of lives saved on an input line are also distinct and in increasing order.

The Output:

For each test case, just output a line with the following format:

`Budget #k: Maximum of x lives saved.`

where k is the number of the budget, starting at 1, and x is the maximum number of lives saved in that budget.

Leave a blank line after the output for each test case.

Sample Input:

```
3
2 2000
10 5 50 100 100 1000 250 1100
100 1 200 2 300 3 1900 1000
3 100
10 100 40 200 70 300 100 500
5 1 25 2 35 3 50 4
200 10000 300 20000 400 30000 500 40000
1 10
100 2 200 3 300 5 400 6
```

Sample Output:

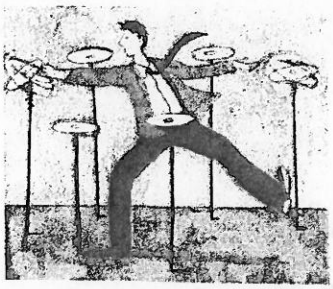
```
Budget #1: Maximum of 2000 lives saved.

Budget #2: Maximum of 500 lives saved.

Budget #3: Maximum of 0 lives saved.
```

Plate Spinning

filename: `plate`

(*Difficulty Level:* `Medium`)

Plate spinning is a circus act where a person spins various objects (usually plates and bowls) on poles without them falling off. It involves spinning an object and then returning back to the object in order to add additional speed to prevent it from falling off the pole.



(Picture from Firefighternation.com)

In this problem you will simulate plate spinning where the plates are placed in a circular arrangement (much like the picture to the right). You must determine whether Chester the Clown will be able to maintain the plates spinning or whether one or more plates will end up falling off poles.

The Problem:

Given the number of poles/plates in a circular arrangement and the speed up to which Chester the Clown spins the plates (in degrees per second), determine if he can maintain the act or if plates will fall. For this problem, we will assume that plates degrade (slow down) at a constant rate of 5-degrees-per-second per second and that Chester can move from one pole to any other pole in 0.5 seconds. In addition, assume that Chester can spin up a plate with zero time cost.

A plate falls off when its rate is zero. However, if Chester arrives at a plate exactly at the same time the rate reaches zero, Chester will spin the plate and prevents it from falling, i.e., the rate must reach zero before Chester arrives for the plate to fall.

The Input:

The first line of the input will be a single positive integer, a, representing the number of acts to evaluate. Each of the following a lines will represent a single act and will contain two positive integers, n and p, separated by a single space, where n represents the number of poles ($1 \leq n \leq 15$) and p represents the speed up to which Chester spins a plate ($0 < p \leq 100$) in degrees per second. At the very beginning of each act, all plates are initially spinning at this speed, and he is currently at a plate in the circle (he can choose which plate to start at in order to maximize his chance of success).

The Output:

For each circus act, output a header "`Circus Act` *i*:" on a line by itself where *i* is the number of the act (starting with 1). Then, on the next line, output "`Chester can do it!`" if Chester can maintain the act, or output "`Chester will fail!`" if one or more plates will fall. Leave a blank line after the output for each test case.

Sample Input:

```
3
2 10
5 7
2 12
```

Sample Output:

```
Circus Act 1:
Chester can do it!

Circus Act 2:
Chester will fail!

Circus Act 3:
Chester can do it!
```