UCF Local Contest — September 5, 2015

Find the Twins
filename: `twins`
(*Difficulty Level:* `Easy`)

Dr. Orooji's twins (Mack and Zack) play soccer. We will assume Mack wears jersey number 18 and Zack wears 17. So, Dr. O has to look for these two numbers when trying to find the twins.

The Problem:

Given a list of 10 numbers, determine if the twins are there.

The Input:

The first input line contains a positive integer, n, indicating the number of data sets to check. The sets are on the following n input lines, one set per line. Each set consists of exactly 10 single-space-separated distinct integers (each integer between 11 and 99 inclusive) giving the jersey numbers for the players.

The Output:

Print each input set. Then, on the next output line, print one of four messages (`mack, zack, both, none`), indicating how many of the twins are in the set. Leave a blank line after the output for each test case.

Sample Input:

```
4
11 99 88 17 19 20 12 13 33 44
11 12 13 14 15 16 66 88 19 20
20 18 55 66 77 88 17 33 44 11
12 23 34 45 56 67 78 89 91 18
```

Sample Output:

```
11 99 88 17 19 20 12 13 33 44
zack

11 12 13 14 15 16 66 88 19 20
none

20 18 55 66 77 88 17 33 44 11
both

12 23 34 45 56 67 78 89 91 18
mack
```

UCF Local Contest — September 5, 2015

Medal Ranking
filename: `medal`
(*Difficulty Level:* `Easy`)

When different countries compete against each other (e.g., in the Olympics), they receive gold/silver/bronze medals. The countries can then be ranked in one of two ways: by "count" which is based on the total number of medals (regardless of the medal colors) or by "color" which is based on the number of gold medals (and silver medals if tied in gold medals, and bronze medals if tied in gold and silver).

The Problem:

Given the gold/silver/bronze medal counts for USA and Russia, you are to determine if USA wins in these two ranking methods.

The Input:

The first input line contains a positive integer, *n*, indicating the number of data sets to check. The sets are on the following *n* input lines, one set per line. Each set consists of 6 integers (each integer between 0 and 500 inclusive); the first three integers represent (respectively) the gold, silver, and bronze medal counts for USA; the last three integers provide this info for Russia (in same order).

The Output:

Print each input set. Then, on the next output line, print one of four messages (`count`, `color`, `both`, `none`), indicating how USA can win. USA will win by `count` if its total medal count is higher than the total for Russia. USA will win by `color` if it has more gold medals than Russia (if tied in gold, then USA must have more silver; if tied in gold and silver, then USA must have more bronze). Leave a blank line after the output for each test case.

Sample Input:

```
5
10 5 15 10 1 0
10 5 15 10 6 10
12 5 10 5 20 30
10 0 15 10 5 30
10 5 15 10 5 15
```

Sample Output:

```
10 5 15 10 1 0
both

10 5 15 10 6 10
count

12 5 10 5 20 30
color

10 0 15 10 5 30
none

10 5 15 10 5 15
none
```

UCF Local Contest — September 5, 2015

Brownies vs. Candies vs. Cookies

filename: `brownie`

(*Difficulty Level:* `Easy`)

Everyone is welcome to the UCF Programming Team practices, and many students take advantage of this opportunity. The main benefit is that these students improve their problem solving and programming skills. Another benefit is that the students enjoy the refreshments Dr. Orooji brings every week! Dr. O usually brings candies but sometimes he brings cookies or brownies. Brownies are very popular and don't usually last long, so Dr. O has to come up with some clever trick to make the brownies last longer (so that the students stay for the entire practice!). Well, the easiest solution is to cut the brownies in half; that will double the number of brownies.

The Problem:

Given the original number of brownies and the students wanting brownies, you are to keep track of the brownie count as Dr. O cuts them in half.

The Input:

The first input line contains a positive integer, n, indicating the number of programming team practices. This is followed by the data for these practices. The first input line for each practice contains two integers (separated by a space): the number of students (between 1 and 30 inclusive) in the practice and the number of brownies (between 60 and 600 inclusive) Dr. O has brought that day. The next input line for the practice contains a positive integer, m, indicating how many groups of students approach the refreshment table to take brownies. This is followed by the number of students in each group, one number per line. Assume that the input values are valid, e.g., the number of students in a group will be at least 1 and it will not be greater than the number of students in the practice.

If a group of students is approaching the refreshment table and Dr. O notices that the number of remaining brownies is less than or equal to the number of students in the group, Dr. O cuts the brownies in half to be sure they won't be all gone after each student in the group grabs one brownie. Note that, if needed, Dr. O will cut the brownies more than once (as many times as needed). For example, if there are 3 brownies left and 24 students are approaching the table, Dr. O has to cut the brownies four times ($3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48$) to be sure the brownies won't be all gone after each student in the group grabs one.

The Output:

At the beginning of each practice, output "`Practice #p: s b`" where p is the practice number (starting with 1), s is the number of students in this practice, and b is the number of brownies. Then, on each of the following output lines, print the number of students in a group approaching the refreshment table and the number of brownies left after each of these students

has grabbed one brownie (note that cutting in halves may occur before grabbing). Leave a blank line after the output for each practice.

Sample Input:

```
2
20 60
8
15
10
20
18
9
12
2
10
15 100
4
1
2
3
5
```

Sample Output:

```
Practice #1: 20 60
15 45
10 35
20 15
18 12
9 3
12 12
2 10
10 10

Practice #2: 15 100
1 99
2 97
3 94
5 89
```

Lemonade Stand
filename: `lemonade`
(*Difficulty Level:* `Medium`)

You are running a lemonade stand and have the good fortune of knowing exactly how many cups of lemonade customers are going to want to buy on each day that you run the lemonade stand. You hate to turn any customer away, so you would like to make sure that you always have enough lemons and sugar to make the appropriate number of cups of lemonade on each day. Unfortunately, the cost of lemons and sugar change daily, so you have to choose on which days you buy each, and how much of each to buy. You can buy individual lemons and five pound bags of sugar. (Note that there are 16 ounces in one pound.) On the days you choose to buy ingredients, you buy them in the morning, before any sales are made. (You're an early riser, so you can always get to the store and back before any customers would come.) Note that you can buy as little or as much as you wish on any day to minimize your overall cost, i.e., you have enough startup money (capital) to buy as much as you wish on any day.

The Problem:

Given that you always want to have enough lemons and sugar to serve each customer, determine the minimum cost of buying those lemons and sugar.

The Input:

The first input line will have a single integer, n $(1 \le n \le 100)$, the number of cases to process. The first line of each test case will have three space-separated positive integers: d $(1 \le d \le 1000)$, the number of days you'll run the lemonade stand, x $(1 \le x \le 10)$, the number of lemons needed to make a single cup of lemonade, and s $(1 \le s \le 10)$, the number of ounces of sugar needed to make a single cup of lemonade. The following d lines will contain data for days 1 through d, respectively. Each of these lines will have three integers separated by spaces: c $(1 \le c \le 1000)$, the number of cups sold for that day, p_l $(1 \le p_l \le 50)$, the price of a single lemon in cents for that day, and p_s $(1 \le p_s \le 500)$, the price of a five pound bag of sugar in cents for that day. Note that the extra sugar and lemon from each day carry over to the next day.

The Output:

For each test case, print the minimum cost of supplies (in cents) necessary to make sure that no customer who wants a cup of lemonade gets turned away.

Sample Input:

```
2
3 3 2
200 10 399
300 8 499
400 12 499
2 5 10
9 10 199
8 20 99
```

Sample Output:

```
31977
1347
```

Rain Gauge

filename: `rain`

(*Difficulty Level:* `Medium`)

When going to your internship you got a nice apartment with a skylight. However, one crazy party later and you now have a square-shaped hole where your skylight used to be. Rather than telling the landlord, you decided you would "fix" it by putting a circular pot to collect the water but, as the saying goes, round peg square hole. You need to determine how much of the square the circle covers to help you determine if you should buy a larger pot. Don't worry about the area not covered; you can do multiplication and subtraction easily in your head.
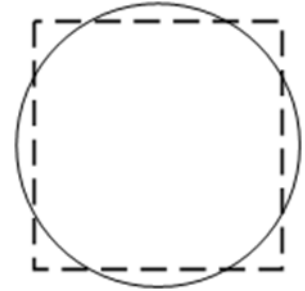
The Problem:

Given the radius of a circular pot and the length of the square skylight, calculate the amount of skylight rain area covered by the pot assuming the two shapes have the same center (i.e., are coaxial) with respect to the direction rain falls from (up). In other words, the center of the square will be directly above the center of the circle. See the picture for an example; let up be the direction from above the page, while the dotted square is the skylight and the solid circle is the pot to collect water.

The Input:

The first input line contains a positive integer, *n*, indicating the number of scenarios to check. Each of the following *n* lines contains a pair of integers, *s*, *r* ($1 \leq s \leq 100$, $1 \leq r \leq 100$), which represents the length of the side of the skylight and the radius of the pot, respectively.

The Output:

For each scenario, output a single decimal representing the area under the skylight that is covered by the pot. Round the answers to two decimal places (e.g., 1.234 rounds to 1.23 and 1.235 rounds to 1.24). For this problem, use 3.14159265358979 as the value of pi.

Sample Input: | **Sample Output:**

```
3                      1.00
1 1                    62.19
8 5                    50.27
10 4
```

UCF Local Contest — September 5, 2015

Balanced Strings
filename: `balance`
(*Difficulty Level:* `Medium`)

Being an individual obsessed with balancing and trying to find strings that meet that criteria, you have begun a new endeavor to study the curious structure of what we will call balanced strings.

Balanced strings are strings that maintain an equal ratio of consonants to vowels in all of their substrings. What? You don't know the difference between a consonant and a vowel? Vowels are the characters 'a', 'e', 'i', 'o', 'u' and sometimes 'y'. Actually, you don't like the character 'y' because this makes the problem much harder. What was the difficulty level of this problem again? Oh yeah Medium! We can't have a problem about consonants and vowels that makes 'y' sometimes a vowel! That would make the problem a Hard and too many hard problems is a very bad thing. Being obsessed with balance, you have decided that 'y' will be included with the vowels. That way, there are 6 vowels and 20 consonants. A nice balanced even number of both! Oh! I almost forgot! A consonant is any letter that is not a vowel. Also to simplify things (this is a medium problem after all!), we will consider strings composed of lowercase letters only.

Now you are ready to understand balanced strings! A balanced string is a string that has an equal number of consonants and vowels in all of its substrings. Well… not all of its substrings. Just the substrings of even length! For example, the string "orooji" has the following set of even-length substrings: {"or", "ro", "oo", "oj", "ji", "oroo", "rooj", "ooji", "orooji"}. In that set the following substrings are unbalanced: {"oo", "oroo", "ooji", "orooji"}. That is, the substrings do not contain an equal number of vowels and consonants. So, the string "orooji" is not balanced. But a string like "arup" is balanced. The requisite even-length substrings are: {"ar", "ru", "up", "arup"} and all these substrings are balanced (have the same number of vowels and consonants), thus the string "arup" is balanced. Note that a balanced string may contain an odd number of characters, e.g., the string "ali" is balanced since all its even-length substrings ({"al", "li"}) are balanced.

Now you want to take words and erase some of the letters and replace them with new letters to form balanced strings. Since this is a medium problem, we've taken the liberty of erasing some of the letters for you and replaced them with '?' characters. See! The problem is already halfway solved!

The Problem:

Given a string of lowercase letters and '?' characters, count the number of ways to replace all the '?' with lowercase letters such that the string results in a balanced string. Two ways are considered different if there exists some i such that the ith character in one string differs from the ith character of the other string. Note that all the '?' do not have to be replaced by the same letter.

The Input:

The first input line contains a positive integer, n, indicating the number of strings to balance. The strings are on the following n input lines, one string per line. Each string contains only lowercase letters and/or '?' characters. Assume each input string has at least one character and at most 100 characters.

The Output:

For each string, output "String #d: v" where v is the number of ways to replace the questions marks with lower case letters. It is guaranteed that v will fit in a signed 64-bit integer for the strings we provide. Leave a blank line after the output for each string.

Sample Input:

```
7
orooji
al?
a?i
g?ha
a?u?
????????????????
arup
```

Sample Output:

```
String #1: 0

String #2: 6

String #3: 20

String #4: 6

String #5: 400

String #6: 1117952409600000000

String #7: 1
```

Note: The first and last strings in Sample Input do not have any '?'. The first input is not balanced so there is no way of replacing all '?' to make it balanced, thus the output is zero. However, the last input is balanced so there is one way of replacing all '?' to make it balanced, thus the output is 1.

UCF Local Contest — September 5, 2015

Towers of Hanoi Grid

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

Towers of Hanoi is a rather famous problem for computer scientists as it is an excellent exercise in recursion. For those of you unfamiliar, here is the classic problem. You are given three pegs. On the first peg, there are d disks placed in decreasing order of size (as placed on the peg). The objective of the game is to move the entire tower from the first peg to the last peg. In each move, you are only allowed to move a single disk from the top of one stack to another stack. For the entire game, no disk of larger size is ever allowed to be placed on top of a disk of smaller size. The goal of the puzzle is to move the tower in the minimum number of moves.
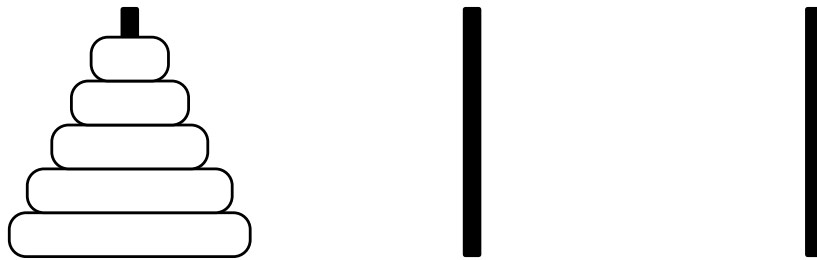
In our problem, we will instead have an $n \times n$ grid of pegs. The rows are numbered top to bottom from 1 to n, while the columns are similarly labeled, from left to right, 1 to n. The original tower is placed on the top left peg $(1, 1)$. The goal is to move the tower to the bottom right peg (n, n) in the minimum number of moves possible.
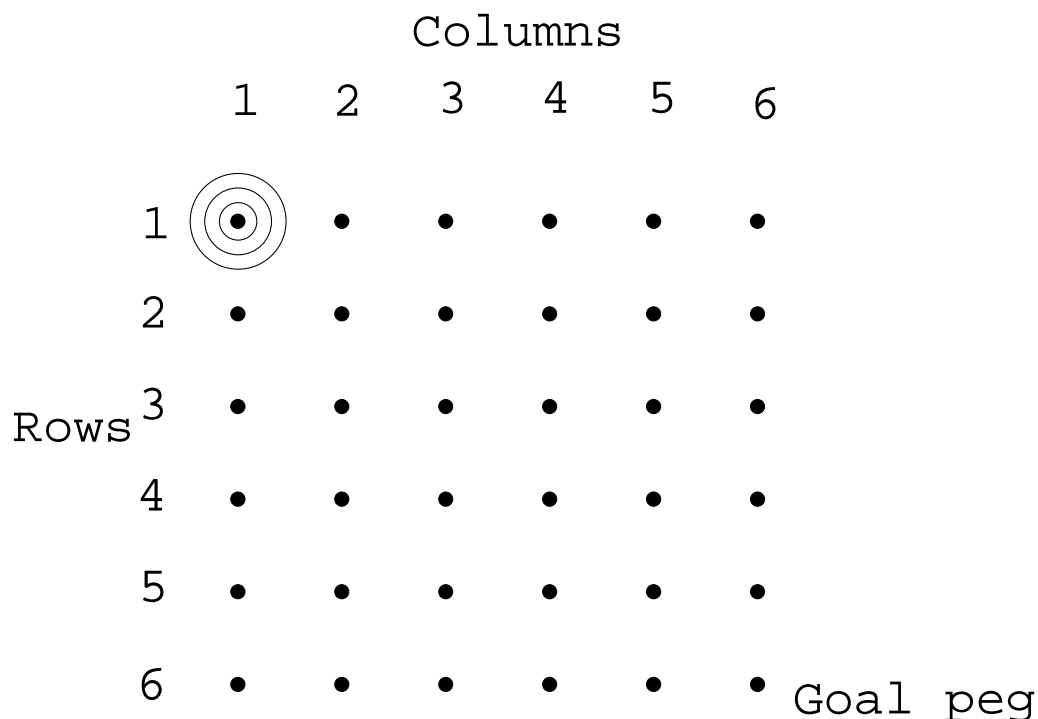
Our game will have some different but related rules:
- For a peg (r, c) at row r and column c, you may only move the top-most disk from peg (r, c) to peg $(r + 1, c)$ or peg $(r, c + 1)$, in a single move and only if such a pair of pegs exists.
- Only pegs $(1, 1)$ and/or (n, n) may have more than one disk at any time; all other pegs may contain at most one disk.
- You can choose any peg for each move.
- You still may never place a larger disk on a smaller disk.

The Problem:

Given the number of disks on the starting peg and the number n described above, determine the minimum number of moves to solve our Tower of Hanoi Grid puzzle.

The Input:

The first input line contains a positive integer, g, indicating the number of grids to solve. The grids are on the following g input lines, one grid per line. Each grid is described by two integers d and n $(2 \leq d, n \leq 100)$, representing the number of disks and the dimensions of the grid, respectively.

The Output:

For each grid, output "Grid #d: v" where v is the minimum number of moves to solve the Tower of Hanoi Grid puzzle. If it is not possible to move the disks from peg (1,1) to peg (n, n), output "impossible" (without quotes) for v. Leave a blank line after the output for each grid.

Sample Input:

```
3
2 2
100 8
3 100
```

Sample Output:

```
Grid #1: 4

Grid #2: impossible

Grid #3: 594
```

UCF Local Contest — September 5, 2015

Reach for the Stars
filename: `stars`
(*Difficulty Level:* `Hard`)

When you were a little kid, you would stamp papers randomly but your parents were still so impressed by your art. Now you are dealing with the UCF Programming Team Coaches and you need to be more creative to impress them. Good luck!

You are given a star shaped stamp like the one below. The black area is covered in ink and the white area is not. When the stamp hits the paper, it leaves a mark for each cell of ink that hits the paper.
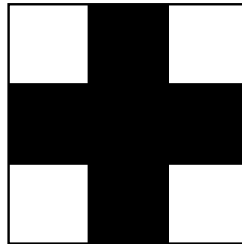
For example, the image below can be made with two stampings. Notice the stamp must always remain axis-aligned when hitting the paper. We also require that the stamp be completely contained within the paper. Note a cell of paper stamped once with black ink is indistinguishable from a cell of paper stamped multiple times with black ink. Note also that cells and stamp line up properly, i.e., a cell is either covered completely by the stamp or not covered at all, i.e., the stamp will not cover part of a cell.
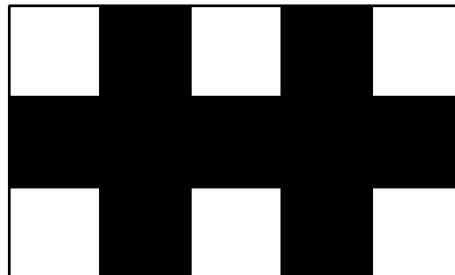
The Problem:

Given a black and white image, determine the minimum number of times, if possible, you would need to stamp the paper with the star stamp to end up with the design specified.

The Input:

The first input line contains a positive integer, n, indicating the number of images to evaluate. Each image starts with a line containing two integers, r and c, $(1 \leq r \leq 9, 1 \leq c \leq 9)$, representing the number of rows and columns, respectively. The next r input lines contains c characters each. The characters are either '.', representing a blank cell of the image and '#', representing a cell of the image covered in ink.

The Output:

For each image, output "Image #d: v" where v is the minimum number of stampings required to make the image. Replace v with "impossible" (without quotes) if it is not possible to form the image using the star shaped stamp. Leave a blank line after the output for each test case.

Sample Input:

```
4
1 1
.
1 1
#
3 3
.#.
###
.#.
3 5
.#.#.
#####
.#.#.
4 7
.##.#..
######.
.######
..#..#.
```

Sample Output:

```
Image #1: 0

Image #2: impossible

Image #3: 1

Image #4: 2

Image #5: 5
```

Longest Path
filename: `longpath`
(*Difficulty Level:* `Hard`)

You are visiting a lovely garden, with several points of interest. Every pair of points of interest have a single, one way path (edge) connecting them. (The paths are one way so that guests don't get upset having to deal with other guests walking in the opposite direction.) Luckily, you get to decide which point of interest to begin from. Naturally, to get your money's worth, you'd like to visit as many of the points of interest as possible without revisiting any point of interest, since that would be a waste of time. Determine a walking path that allows you to visit the maximal number of points of interest without repeating any of them.

The Problem:

Given a directed graph containing exactly one edge (path) between every pair of vertices (points of interest), determine a walking path of maximal length that doesn't revisit any vertex.

The Input:

The first input line contains a positive integer, *n (n ≤ 100)*, indicating the number of gardens to check. The input for each garden follows. Each set starts with a line with a single integer, *v (1 ≤ v ≤ 500)*, the number of points of interest for that case. The following *v* lines will contain *v* space separated integers each. The jth *(1 ≤ j ≤ v)* of these integers on the ith *(1 ≤ i ≤ v)* line is 1 if there is a path from point *i* to point *j*, or 0 if there is a directed path from point *j* to point *i* instead. The ith integer on the ith line will always be 0. It is guaranteed that the input information is consistent - if the jth integer on the ith line is 1 then the ith integer on the jth line is 0, and vice versa.

The Output:

For each garden, on a line by itself, output a space separated list of any maximal length sequence of unique points of interest to visit.

Sample Input:

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

Sample Output:

```
1 2 4 3
1 2 3
```

UCF Local Contest — September 5, 2015

You Shall Pass
filename: `passing`
(*Difficulty Level:* `Hard`)

Students failing COT 3100 (Introduction to Discrete Structures) has become a large concern for the professors at UCF. Matt and Sean have compiled a large amount of data, and after extensively analyzing it, they have generated some interesting probabilities. For each student, there are two basic probabilities known: the probability they will pass Matt's class and the probability they will pass Sean's class. Also known is a large table of probabilities that a student will pass based on being in the same class as another student. When a pair of students are in the same class, they may get together to form a study group, increasing their probability of passing the class. For each pair of students, i, j, a value a_{ij} is known. If students i and j are in the same class then student i is a_{ij} more likely to pass due to a study group. That is, i's probability of passing increases by a_{ij}. For example, if $i = 0.4$ and $a_{ij} = 0.2$, then i's probability becomes 0.6 (0.4 + 0.2 = 0.6).

Now Matt and Sean are trying to find a way to split the students into two classes such that the expected number of people passing is maximized.

The Problem:

Given the initial passing probabilities of the students and the passing probabilities from study groups, you are to determine the maximum expected number of passing students obtainable by splitting the students into two classes. Note that every student will be in exactly one class but one class may be empty.

The Input:

The first input line contains a positive integer, g, indicating the number of semesters to check. The data for each semester starts with an integer n $(2 \leq n \leq 50)$, which is the number of students who sign up for Discrete (COT 3100) in that semester. The following two input lines contain n decimal values each. The i-th value of the first line represents the probability of the i-th student passing Matt's class, and the i-th value of the second line represent the probability of the i-th student passing Sean's class. This is followed by n lines. The i-th line contains n non-negative decimal numbers. The j-th number on each line is a_{ij} as specified above.

For any given class configuration, assume that at no time a student will have a probability of passing higher than 1 or less than 0 (even after considering study groups). Each decimal number in input will be of the form "`#.##`", where "`#`" denotes a digit from 0 to 9, inclusive (i.e., each input value will be given to exactly two decimal places).

The Output:

For each semester, output a single decimal value representing the maximum expected number of students passing. Round the answers to two decimal places (e.g., 1.234 rounds to 1.23 and 1.235 rounds to 1.24).

Sample Input:

```
2
2
0.75 0.25
0.25 0.75
0.00 0.20
0.20 0.00
3
0.20 0.60 0.95
0.40 0.40 0.95
0.00 0.00 0.55
0.00 0.00 0.35
0.00 0.00 0.00
```

Sample Output:

```
1.50
2.65
```

UCF Local Contest — September 5, 2015

Turing's Challenge
filename: `turing`
(*Difficulty Level:* `Hard`)

Knuth was looking through some of Turing's memoirs and found a rather interesting challenge that Turing had left for one of his successors. Naturally, Knuth has slyly decided to ask you, his best student, to write a computer program to solve the challenge, but plans on taking credit for the work. Since you know that co-authoring a paper with Knuth is to computer scientists what co-authoring a paper with Erdos is to mathematicians, you've decided to take the bait. Help Knuth solve Turing's problem!

The Problem:

The challenge is as follows:

Given positive integer values for X and N, define the set T as follows:

$$T = \{T_i | 1 \le i \le N + 1\}, \text{ where } T_i = \binom{N}{i-1} X^{i-1}$$

The goal of the challenge is to pick a set S of maximal sum with $S \subseteq \{i | 1 \le i \le N + 1\}$, such that $\prod_{i \in S} T_i \equiv 2 \pmod{4}$.

In other words, we seek a subset of terms in the binomial expansion of $(1 + X)^N$ such that the product of the terms leaves a remainder of 2 when divided by 4 and the sum of the *indices* of those terms is maximal.

The goal of Turing's challenge is to determine this maximal sum.

As an example, consider X = 3 and N = 5. The corresponding terms are $T_1 = 1, T_2 = 15, T_3 = 90, T_4 = 270, T_5 = 405$, and $T_6 = 243$.

The product, $T_1 T_2 T_4 T_5 T_6 = 1 \times 15 \times 270 \times 405 \times 243 = 398580750 \equiv 2 \pmod{4}$, thus the solution to this specific challenge is $1 + 2 + 4 + 5 + 6 = 18$, since no other product of terms with a higher sum of indices is equivalent to 2 (mod 4).

The Input:

The first input line contains a positive integer, *q* (1 ≤ *q* ≤ 500), indicating the number of queries. Each of the next *q* lines will contain a pair of space-separated integers, where the first integer is X (1 ≤ X < 2^{31}), and the second integer is N (1 ≤ N < 2^{31}), for that query.

The Output:

For each query, output on a line by itself, the desired maximal sum of indices. If no such subset of terms exists, output 0 instead.

Sample Input:

```
4
3 5
1 4
4 6
```

Sample Output:

```
18
9
0
```