MERCER UNIVERSITY

Spring Programming Contest

February 25, 2017

Ant Clans	(prob1)
Bang for your Buck	(prob2)
Election	(prob3)
Fit the Word	(prob4)
Graze the Rainbow	(prob5)
Planted Motifs	(prob6)
Quick, Find an Oasis	(prob7)
Serious Sudoku	(prob8)
Spelling Bee	(prob9)
Two Suns	(prob10)
Word Gaps	(prob11)

The name in parenthesis following each 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).

Ant Clans (prob1)

The Problem

An ant dynasty has decided to move into a giant ant hill consisting of of *n* burrows. The dynasty has a complicated social structure formed by a coalition of *k* ant clans, where each clan doesn't get along well with the others. To keep the peace, the ant emperor wants to partition the burrows into *k* equally-sized districts (one per clan), so that no ant from any clan should be able to reach any ant of an opposing clan. After deciding on these districts, the ants will drill tunnels between certain carefully-chosen pairs of burrows so ants in each clan can travel among all the burrows allocated to the district for that clan.

The ant emperor would like to know the cheapest cost possible for forming his districts and building the resulting tunnels.

Input

The first line of input is a single integer, a ($1 \le a \le 100$), representing the number of anthills to analyze.

For each anthill, the first line contains three space separated values: n ($1 \le n \le 20$), m ($1 \le m \le n(n-1)/2$), and k ($1 \le k \le n$), representing the number of burrows, number of possible tunnels that can be drilled, and the number of districts to form, respectively. Burrows are labeled with identifiers in the range 1..n .

This is followed by m lines each containing three space separated integers: i ($1 \le i \le n$), j ($1 \le j \le n$, $i \neq j$), and c ($1 \le c \le 1000$) representing that burrows labeled i and j can be connected by a tunnel with cost c . No pair (i, j) will be listed more than once and if a tunnel is dug between burrows i and j, then ants can move freely from burrow i to burrow j and also from burrow j to burrow i.

Output

For each anthill, on a line by itself, print the minimum cost possible of the desired partition or -1 if such a district plan is impossible.

Sample Input

```
3
4 4 2
1 2 300
2 3 200
3 4 100
4 1 8
6 4 3
1 2 3
2 3 4
4 5 6
5 6 7
12 18 4
1 2 3
2 3 4
3 4 5
5 6 6
6 7 8
7 8 11
9 10 5
11 12 200
1 5 7
2 6 4
3 7 2
4 8 3
6 9 1
7 10 4
9 11 100
10 12 300
5 11 8
8 12 4
```

Sample Output

```
208
-1
33
```

Bang for your Buck (prob2)

The Problem

Congratulations! You never thought you'd actually win when you mailed in that cereal box top, but you're the grand prize winner of the Best Buy shopping spree. You get to spend $500 on whatever you want, but since it's Best Buy, there's a catch. You must select three unique items (not one, not two, and no duplicates), and the total price of those three items must be exactly $500.00 -- not a penny more or less.

Luckily your friend was able to find a way to access a list of prices for every single item in Best Buy stores and online via the bestbuy.com website. The question you want to answer is whether or not it is possible to satisfy Best Buy's contest conditions and get some free stuff! Write a program that reads in the list of prices your friend found (from standard input) and reports to standard output the exact number of 3-item sets with prices totaling $500.00.

One more thing: even though your friend's script removes any duplicates, it still finds a LOT of prices. In order to avoid affecting the Best Buy servers, make sure your algorithm runs in less than 1 second.

Input

A single integer N representing the number of problem instances to follow (N<=100)

-N repetitions of the following:

--A single integer P representing the number of prices to follow (P<=5000) on its own line

--P unique prices, one per line, in the form: $X.YY (0 < X <= 10000, YY always two digits in the range 00 to 99, never omitted)

Output

N integers, one per line, representing the counts of item triples summing to exactly $500.

Sample Input

```
2
4
$100.00
$5.00
$210.00
$190.00
8
$100.00
$5.00
$200.01
$499.99
$199.99
$251.00
$248.00
$1.00
```

Sample Output

```
1
2
```

Election (prob3)

The Problem

You try to win an election in an electoral college system similar to the U.S. presidential election. There are *n* states and each state has a certain number of electoral votes *vi*. A candidate winning a state will take all the electoral votes of the state. You win the election if you receive strictly more than half of the total number of electoral votes of all states. However, elections do cost money. To win the state *i*, you need to spend *di* dollars. The problem is find the cheapest way to win the election.

Input

The input consists of a number of test cases. Each case is described by three input lines.

```
n
v1 v2 … vn
d1 d2 … dn
```

The first line of a case contains a single integer *n* representing the number of states (0<*n*<100). The second line contains a list of *n* integers delimited by a space. Each integer represents the number of electoral votes of a state (0<*vi* <100). The third line contains a list of *n* integers representing the dollars (in millions) you need to spend in each of the states in order to win the state (0<*di* <1000). The input is terminated by a line consisting of the number 0.

Output

For each of the test cases, print the minimal number of millions of dollars you need to spend to win the election.

Sample Input

```
3
4 3 2
5 6 4
4
4 3 2 3
6 2 1 5
0
```

Sample Output

```
9
8
```

Fit the Word (prob4)

The Problem

Have you ever seen these images composed of words of different sizes? They are called word clouds and are built to visualize the most important words from some text. The algorithms to generate such word clouds usually start by sorting words from a text by frequency, and most frequent terms will be displayed with larger font sizes.

The next step is to actually put words on the image. A word printed in a certain font will take some rectangular space. Of course, we don't want words to overlap, therefore before printing the word we need to make sure there is enough space for it, otherwise the font size may have to be decreased.

Let's suppose we have a rectangular canvas of size **H*W**, which we are using to print a word cloud. Some pixels on this canvas may still be blank, while the others are already used. You will be given a word **S** to be printed using some font size **F**. When a word is printed, each letter takes a square space of size **F*F**, and letters are placed next to each other horizontally. Your task is to determine the maximum possible font size, which can be used to print a given word, so that it doesn't overlap with any other words on the canvas.

Input

The input starts with an integer T ($T \leq 100$), the number of test cases.

For each test case you are given the height and width of the canvas ($0 < H, W \leq 500$), followed by H lines of W characters, which represent the current state of the canvas. '.' means a blank pixel, and 'X' means that this pixel is already used. Next, you are given a word to print S ($1 \leq |S| \leq 50$), which consists of lowercase letters only.

Output

For each test case output a single integer, the largest possible font size to print the word, or 0 if it's impossible.

Sample Input

```
4
2 6
......
......
yes
2 6
XXXXXX
X....X
java
4 8
X......X
X.......
........
..X...XX
ab
2 5
.X.X.
X.X.X
no
```

Sample Output

```
2
1
3
0
```

Graze the Rainbow (prob5)

The Problem

In the latest economic crisis, the cost of corn has skyrocketed! Farmer Joanna is having a harder and harder time buying enough corn for all her cows to eat and has turned to alternative sources. Luckily, she has friends in the candy-making industry who are willing to sell her their surplus candies at a reduced price.

Farmer Joanna has just purchased her first shipment of candy – a truck-load of small disc-shaped candies in a variety of colors – and she has tried to convince her cows to eat the candies. Sadly, many of the cows are unwilling to accept candy as a substitute for their beloved corn, but Farmer Joanna has a plan. Unlike the rest of the cows, Bennie the cow loves the candies even more than corn, so Farmer Joanna plans to feed candies to him in front of the rest of the cows so they will be tempted to eat some themselves.

However, the cows are only going to want to eat candies which they consider interesting. When a shipment comes, the cows will notice any candies that are the only instance of their color in the entire shipment and see those candies as interesting. Then, as Bennie eats the candies, if the last candy left is one of the interesting candies, the other cows will form a cowpile on top of the candy in an effort to be the cow that gets to eat it. If Farmer Joanna can get this to happen enough times, the cows will eventually learn from experience that candy is just as delicious as corn, and she can start feeding it to all of them.

Also unlike the other cows, Bennie the cow does not care too much about the color of the candies, so when he is given a shipment of candies to eat, he simply eats them in a random order, so all of the candies in the pile are equally likely to be the last one eaten, making Farmer Joanna's plan more difficult to implement.

Nevertheless, Farmer Joanna is determined to convince her entire herd of cows to eat candy, so she will continue ordering shipments of candy she succeeds. For each shipment, she is told how many candies she will receive, and how many colors of candies will be included. Since there is a large supply of defective candies, it is assumed that each candy has an equal probability of being each color, which is independent of the colors of other candies. For example, if the colors included are blue, purple, and red, then each candy has a 1/3 chance of being blue, a 1/3 chance of being purple, and a 1/3 chance of being red, regardless of the colors of other candies in the shipment.

Every time a shipment comes, Farmer Joanna wants to know how likely it is that the cows will form a cowpile on the last candy in the shipment. Since this is way too much math for somebody without a computer to calculate on their own, she would like you to write a program to figure out this probability for her.

Given the number of candies in a shipment, as well as the number of colors of candies, determine the probability that the last candy eaten is of a color that appears only once in the entire shipment.

Input

The first line of input will consist of a single positive integer, t *($t \leq 100$)*, representing the number of input cases. The first and only line of each case will contain two space separated positive integers n ($n \leq 500$) and k *($k \leq 500$)*, where n is the number of candies in the shipment and k is the number of colors the candies can be..

Output

For each input case, output a single decimal number representing the probability that the cows will form a cowpile on top of the last candy, rounded to three decimal places. It is guaranteed that small changes to the answer (caused by loss of precision) up to 10^{-6} will not be enough to change the rounded answer.

Sample Input

```
3
1 1
5 3
3 5
```

Sample Output

```
1.000
0.198
0.640
```

Planted Motifs (prob6)

The Problem

Consider a set, S, of n strings, all of equal length. Given integers l and d, find all l-length common (shared amongst all strings in S) substrings in the alphabet (A, C, G, T) in all strings of S which are at a Hamming distance of at most d. The Hamming distance between two inputs of equal lengths is the number of corresponding positions where the inputs differ. That is, there is at most d mismatches between the substrings. For example, million and billion have a Hamming distance of 1 and the names Brendan and Brandon have a Hamming distance of 2.

Input

The first line of input contains a single integer, c ($1 \leq c \leq 100$), the number of test sets.

Each test case will consist of the following, each on a separate line:

- A single integer, n ($2 \leq n \leq 10$), the number of strings in the set
- n lines containing a string whose length is between 4 and 100 characters, inclusive. Each of the strings on the n lines will have equal length.
- A single integer, l ($2 \leq l \leq 100$), the length of the substring to test
- A single integer, d ($2 \leq d \leq l$), the Hamming distance.

The input for each test case is separated with a blank line.

Output

For each set, output each substring of length l that are at most d Hamming distance from the common substrings in the set; each substring on a separate line. The substrings should be sorted so that each substring is printed only once for each set.

Separate the output for each test case with a blank line.

Sample Input

```
2
3
ACTGACGCAG
TCACAACGGG
GAGTCCAGTT
4
1

5
TTAC
CCAA
GTAC
GACT
TCAC
2
1
```

Sample Output

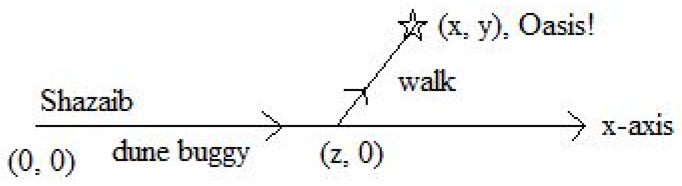
```
ACAG
CAGA
CCCA
TCAG

AA
AC
AG
AT
CA
CC
CT
GA
GC
TA
TC
```

Quick, Find an Oasis! (prob7)

The Problem

Shazaib is in the desert and needs to find some water quick. He happens to be on a dune buggy, but that dune buggy can only travel on the main road, which can be modeled by the positive x-axis. His starting point is (0, 0). The nearest oasis with water is at a point (x, y) in the first quadrant. His dune buggy travels at a known constant speed and he walks at a different known constant speed (slower than the speed of the dune buggy). His optimal strategy will be to travel on the dune buggy from (0, 0) to a point (z, 0), for some non-negative value z, followed by walking a straight line from (z, 0) to (x, y). For the purposes of this problem, treat the oasis as a single point.



Given the position of the nearest oasis, Shazaib's speed on the dune buggy, and Shazaib's walking speed, help Shazaib calculate the minimum amount of time it will take him to get to the water.

Input

The first line of input will consist of a single positive integer, *n (n ≤ 10000)*, representing the number of input cases. Each test case follows, one per line. Each test case will contain four space separated positive integers: *x*(*x* ≤ 100000), *y*(*y* ≤ 100000), *d* (2 ≤ *d* ≤ 1000), and *w* (*w* < *d*), representing the x coordinate of the nearest oasis in feet, the y coordinate of the nearest oasis in feet, the speed of the dune buggy in feet/minute, and the Shazaib's walking speed in feet/minute, respectively.

Output

For each input case, output the minimum time, in minutes, it will take Shazaib to get to the oasis, rounded to 3 decimal places.

Sample Input

```
2
100 1000 600 100
1550 427 200 175
300 400 101 100
```

Sample Output

```
10.050
8.931
5.000
```

Serious Sudoku (prob8)

The Problem

Sudoku is a popular recreational puzzle game. The rules are simple: a square is divided into 9 rows and 9 columns, containing 81 squares total. Some squares are filled in with numbers already; some remain blank. The goal is to find a way to fill in all blank squares such that the following rules hold:

- Each row contains every number 1-9 and no duplicate numbers.
- Each column contains every number 1-9 and no duplicate numbers.
- Each of the following 3x3 sub-squares of the 9x9 square contains every number 1-9 and no duplicates:
  - The square containing rows 1-3 and columns 1-3.
  - The square containing rows 1-3 and columns 4-6.
  - The square containing rows 1-3 and columns 7-9.
  - The square containing rows 4-6 and columns 1-3.
  - The square containing rows 4-6 and columns 4-6.
  - The square containing rows 4-6 and columns 7-9.
  - The square containing rows 7-9 and columns 1-3.
  - The square containing rows 7-9 and columns 4-6.
  - The square containing rows 7-9 and columns 7-9.

These rules are trivially extendable to any *n***n* square where *n* is itself a perfect square. For instance, a 16x16 Sudoku puzzle would have each row and column containing each of the numbers 1-16 once and the squares containing rows/columns 1-4/1-4, 1-4/5-8, 1-4/9-12, 1-4/13-16, 5-8/1-4, 5-8/5-8, 5-8/9-12, 5-8/13-16, 9-12/1-4, 9-12/5-8, 9-12/9-12, 9-12/13-16, 13-16/1-4, 13-16/5-8, 13-16/9-12, and 13-16/13-16, each containing each of the numbers 1-16 once.

Write a program to solve the generalized Sudoku problem which reads in a set of Sudoku puzzles to solve and outputs the solution for each.

Input

Your input will consist of multiple Sudoku puzzles to solve. Each Sudoku puzzle will be prefaced by *n*, which specifies the length of each row and column, followed by *n* rows of *n* numbers, each number separated by a single space, describing the Sudoku puzzle to solve. Each number read in will either be a number *k* 1≤*k*≤*n* or 0, meaning that it is the computer's job to determine a valid value for that location in the puzzle. *n* will be one of 1, 4, 9, or 16. The list of Sudoku puzzles to solve is terminated by a single "0". Do not process this case. Every Sudoku problem in the input will have either one solution or zero solutions.

Output

Your Sudoku solver should output the solution to the puzzle if one exists, in the same format as the input.. If the puzzle as given is impossible to solve, you are to output instead a single line containing the text “ILLEGAL BOARD”.

Sample Input

```
4
4 0 0 2
0 2 3 4
0 0 2 1
0 1 0 3
9
7 0 0 0 0 0 2 4 9
1 9 5 0 0 0 3 8 0
0 3 0 7 0 0 0 0 0
5 8 0 1 7 4 0 0 3
0 7 9 2 6 0 0 1 8
0 2 1 3 9 0 0 0 5
0 0 2 9 0 3 8 7 4
0 4 0 8 0 6 0 0 0
8 1 3 4 0 7 0 0 2
4
1 2 2 4
3 4 1 2
2 3 4 1
4 1 2 3
0
```

Sample Output

```
4 3 1 2
1 2 3 4
3 4 2 1
2 1 4 3
7 6 8 5 3 1 2 4 9
1 9 5 6 4 2 3 8 7
2 3 4 7 8 9 1 5 6
5 8 6 1 7 4 9 2 3
3 7 9 2 6 5 4 1 8
4 2 1 3 9 8 7 6 5
6 5 2 9 1 3 8 7 4
9 4 7 8 2 6 5 3 1
8 1 3 4 5 7 6 9 2
ILLEGAL BOARD
```

Spelling Bee (prob9)

The Problem

Come on down! You've been chosen to compete on the Price Is Right, and the first game you are selected to play is Spelling Bee. In Spelling Bee, there are face down cards that you must draw from to spell out a word. You get *k* tries, and to win you must draw one of each type of card at least once.

For instance, to win a car you might have to spell the word CAR, but you only get five draws. Moreover, some cards show up more frequently, such as C of which there are five. However, A only shows up once, making this game pretty difficult to win. S

ince you want to be prepared, you first want to think through all the scenarios and see what your chances are of winning. More importantly, for a generic Spelling Bee game where you get *k* flips and a word of *L* unique letters with various frequencies $\{f_1, f_2, \dots f_L\}$, what are the chances of winning the game?

Input

The first line of input contains a single integer *n*, ($1 \le n \le 1000$), which is the number of test cases that follow. Each case will start with two integers, the first being an integer *k* ($1 \le k \le 10$), the number of flips, and the second an integer *L* ($1 \le L \le 6$), the number of letters in the word.

The next line will consist of *L* integers indicating how many of each letter there are.
There will be at most four of each type of card.

You may also assume :

- Order of selection does not matter.
- Selection is performed without replacement.
- Words will either never have two of the same letter, or you only need to draw one letter to satisfy all occurrences of a letter in a word.

Output

For your output, you should print the probability of winning for that case (rounded to five decimal places).

Sample Input

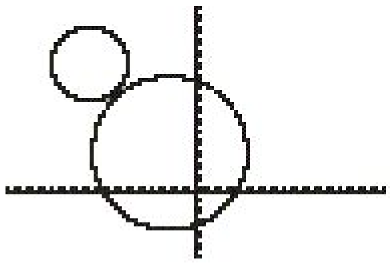
```
2
3 3
1 1 1
3 3
1 2 1
```

Sample Output

```
1.00000
0.50000
```

Two Suns (prob10)

The Problem



In science fiction, one encounters planets that have binary suns. When bringing these stories to the big screen, one is faced with the prospect of drawing a pairs of suns on the screen. To get the full visual impact of two suns, one should draw them so that they do not overlap. Here is the conundrum, the graphics guy wants non-overlapping suns, but the person modeling the lighting does not want them to be any larger than necessary. Note, for the purposes of this problem, if one circle is contained inside the other one, they are overlapping.

Here is where you come in. Given four points *P*, *Q*, *R*, and *S*, you are to find two circles *C* and *K* such that *P* and *Q* are on the first circle, while *R* and *S* are on the second circle. To keep the graphics person happy, the circles cannot overlap nor intersect (but they may meet/intersect at one point). To keep the modeling guy happy, you are to pick the circles so that the combined area is minimized.

For the first sample input problem, one can use the midpoints of the two segments as the centers of the circles. In that case, the radii are 12.33 and 6.01. The circles do not intersect, so this configuration must be the minimum. The area is $\pi(12.33^2 + 6.01^2) = 591.088$ (when rounded).

For the second sample input case, one circle/sun is centered at (26.7285,-5.6260) and has radius 14.439948, while the other circle/sun is centered at (26.0285, 40.6979) and has radius 31.889238. The area is $\pi(14.439948^2 + 31.889238^2) = 3849.819$ (when rounded).

Input

The first line of input is a positive integer *N* which is the number of cases that will follow.

Each of the following *N* lines will contain 6 doubles *u*, *v*, *w*, *x*, *y*, *z*, each value is in the range of [-100, 100]. The points (*u*, *v*) and (*w*, *v*) are on the boundary of one sun, while the points (*x*, *y*) and (*z*, *y*) are both on the boundary of other sun. You may assume that $u \leq w$ and $x \leq z$.

Output

For each input case, output a single number, rounded to 3 decimal places, that is the minimum sum of the areas of the two suns (which must not intersect). (Technically in some cases, it will be the area of the limiting case where the two suns meet at one point.)

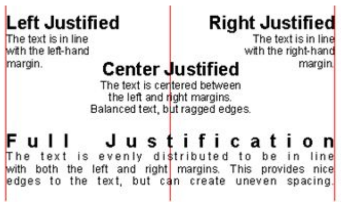
Sample Input

```
2
-16.49 6.63 8.17 -22.91 20.97 -10.89
18.26 6.07 35.197 2.037 19.69 50.02
```

Sample Output

```
591.088
3849.819
```

Word Gaps (prob11)


Left Justified
The text is in line with the left-hand margin.
Right Justified
The text is in line with the right-hand margin.
Center Justified
The text is centered between the left and right margins. Balanced text, but ragged edges.
Full Justification
The text is evenly distributed to be in line with both the left and right margins. This provides nice edges to the text, but can create uneven spacing.

The Problem

Sometimes you want your text to look flush. That is, to have full justification so that it is aligned along both the left and right margins. To make this happen, extra spaces have to be added between words as necessary. We want to write a program to calculate the number of words on a line, the total characters in those words, and the basic gap width between the last two words that would produce flush text. Any leftover spaces are inserted one at a time into the gaps from left to right until there are no more leftover spaces. Assume each formatted string will be of length 80 after all those gaps are filled in with spaces. The gap width between the last two words is equal to :

(80 – Total Characters in all Words on Line) / (Total Words on Line – 1)

Input

The first line of input contains a single integer n, (1 ≤ n ≤ 1000), the number of test cases that follow. Each test case consists of one line of input containing a string of length n, (1 ≤ n ≤ 80). This string will consist of two or more words as input. A word is defined as one or more contiguous characters followed by whitespace (space, tab, or newline).

Output

For each test case, output it exactly as formatted below showing the line number, total words on the line, total characters in all words on the line, and the gap width. Use integer division for all gap width calculations, and note the denominator will never be 0 since each line will always have at least two words. There is exactly one space between all words on each output line.

Sample Input

```
5
When April with his
sweet showers has
pierced the drought of March
to the root.
The End
```

Sample Output

```
Line 1: words=4 chars=16 gap width=21
Line 2: words=3 chars=15 gap width=32
Line 3: words=5 chars=24 gap width=14
Line 4: words=3 chars=10 gap width=35
Line 5: words=2 chars=6 gap width=74
```