# ACM International Collegiate Programming Contest
# 2004 East Central Regional Contest
# Ashland University
# Carnegie Mellon University
# Sheridan University
# University of Cincinnati
# November 6, 2004

**Sponsored by IBM**

Rules:

1. There are **eight** questions to be completed in **five hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contest officials (e.g., that might generate a security violation).
6. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
7. All communication with the judges will be handled by the $PC^2$ environment.
8. Judges' decisions are to be considered final. No cheating will be tolerated.

# Problem A: Alphacode

Alice and Bob need to send secret messages to each other and are discussing ways to encode their messages:

> Alice: "Let's just use a very simple code: We'll assign 'A' the code word 1, 'B' will be 2, and so on down to 'Z' being assigned 26."
> Bob: "That's a stupid code, Alice. Suppose I send you the word 'BEAN' encoded as 25114. You could decode that in many different ways!"
> Alice: "Sure you could, but what words would you get? Other than 'BEAN', you'd get 'BEAAD', 'YAAD', 'YAN', 'YKD' and 'BEKD'. I think you would be able to figure out the correct decoding. And why would you send me the word 'BEAN' anyway?"
> Bob: "OK, maybe that's a bad example, but I bet you that if you got a string of length 500 there would be tons of different decodings and with that many you would find at least two different ones that would make sense."
> Alice: "How many different decodings?"
> Bob: "Jillions!"

For some reason, Alice is still unconvinced by Bob's argument, so she requires a program that will determine how many decodings there can be for a given string using her code.

## Input

Input will consist of multiple input sets. Each set will consist of a single line of digits representing a valid encryption (for example, no line will begin with a 0). There will be no spaces between the digits. An input line of '0' will terminate the input and should not be processed

## Output

For each input set, output the number of possible decodings for the input string. All answers will be within the range of a **long** variable.

## Sample Input

```
25114
1111111111
3333333333
0
```

## Sample Output

```
6
89
1
```

## Problem B: Anti-prime Sequences

Given a sequence of consecutive integers $n, n+1, n+2, \ldots, m$, an *anti-prime sequence* is a rearrangement of these integers so that each adjacent pair of integers sums to a composite (non-prime) number. For example, if $n = 1$ and $m = 10$, one such anti-prime sequence is $1, 3, 5, 4, 2, 6, 9, 7, 8, 10$. This is also the lexicographically first such sequence.

We can extend the definition by defining a degree $d$ anti-prime sequence as one where all consecutive subsequences of length $2, 3, \ldots, d$ sum to a composite number. The sequence above is a degree 2 anti-prime sequence, but not a degree 3, since the subsequence 5, 4, 2 sums to 11. The lexicographically first degree 3 anti-prime sequence for these numbers is $1, 3, 5, 4, 6, 2, 10, 8, 7, 9$.

### Input

Input will consist of multiple input sets. Each set will consist of three integers, $n$, $m$, and $d$ on a single line. The values of $n$, $m$ and $d$ will satisfy $1 \leq n < m \leq 1000$, and $2 \leq d \leq 10$. The line 0 0 0 will indicate end of input and should not be processed.

### Output

For each input set, output a single line consisting of a comma-separated list of integers forming a degree $d$ anti-prime sequence (do not insert any spaces and do not split the output over multiple lines). In the case where more than one anti-prime sequence exists, print the lexicographically first one (i.e., output the one with the lowest first value; in case of a tie, the lowest second value, etc.). In the case where no anti-prime sequence exists, output

```
No anti-prime sequence exists.
```

### Sample Input

```
1 10 2
1 10 3
1 10 5
40 60 7
0 0 0
```

### Sample Output

```
1,3,5,4,2,6,9,7,8,10
1,3,5,4,6,2,10,8,7,9
No anti-prime sequence exists.
40,41,43,42,44,46,45,47,48,50,55,53,52,60,56,49,51,59,58,57,54
```

## Problem C: Hit or Miss

One very simple type of solitaire game known as "Hit or Miss" (also known as "Frustration," "Harvest," "Roll-Call," "Talkative", and "Treize") is played as follows: take a standard deck of 52 playing cards — four sets of cards numbered 1 through 13 (suits do not matter in this game) which have been shuffled — and start counting through the deck 1, 2, 3, . . . , and so on. When your count reaches 13, start over at 1. Each time you count, look at the top card of the deck and do one of two things: if the number you count matches the value of the top card, discard it from the deck; if it does not match it, move that card to the bottom of the deck. You win the game if you are able to remove all cards from the deck (which may take a very long time).

A version of this game can be devised for two or more players. The first player starts as before with a 52 card deck, while the other players have no cards initially. As the first player removes cards from her deck, she gives them to the second player, who then starts playing the same game, starting at count 1. When that player gets a match, he passes his card to the third player, and so on. The last player discards matches rather than passing them to player 1. All players who have cards to play with perform the following 2-step cycle of moves in lockstep:

1. Each player says his or her current count value and checks for a match. If there is no match, the top card is moved to the bottom of the deck; otherwise it is passed to the next player (or discarded if this is the last player).
2. Each player except the first takes a passed card (if there is one) and places it at the bottom of his or her deck.

These rules are repeated over and over until either the game is won (all the cards are discarded by the last player) or an unwinnable position is reached. If any player ever runs out of cards, he waits until he is passed a card and resumes his count from where he left off (e.g., if player 3 passes his last card on a count of 7, he waits until he receives a card from player 2 and resumes his count with 8 at the beginning of the next 2-step cycle).

### Input

Input will consist of multiple input sets. The first line of the file will contain a single positive integer $n$ indicating the number of input sets in the file. Each input set will be a single line containing 53 integers: the first integer will indicate the number of players in the game and the remaining 52 values will be the initial layout of the cards in the deck, topmost card first. These values will all lie in the range $1 \ldots 13$, and the number of players will lie in the range $1 \ldots 10$.

### Output

For each input set, output the input set number (as shown below, starting with 1) and either the phrase "`unwinnable`" or a list showing the last card discarded by each player. Use a single blank to separate all outputs.

## Sample Input

NOTE: Each sample input set below is split across multiple lines in order to fit on the page – in the actual file each set will be on a single line.

```
2
4 1 2 3 4 5 6 7 8 9 10 11 12 13 1 2 3 4 5 6 7 8 9 10 11 12 13
  1 2 3 4 5 6 7 8 9 10 11 12 13 1 2 3 4 5 6 7 8 9 10 11 12 13
4 2 3 4 5 6 7 8 9 10 11 12 13 1 2 3 4 5 6 7 8 9 10 11 12 13 1
  2 3 4 5 6 7 8 9 10 11 12 13 1 2 3 4 5 6 7 8 9 10 11 12 13 1
```

## Sample Output

```
Case 1: 13 13 13 13
Case 2: unwinnable
```

# Problem D: I Conduit!

Irv Kenneth Diggit works for a company that excavates trenches, digs holes and generally tears up people's yards. Irv's job is to make sure that no underground pipe or cable is underneath where excavation is planned. He has several different maps, one for each utility company, showing where their conduits lie, and he needs to draw one large, consolidated map combining them all. One approach would be to simply draw each of the smaller maps one at a time onto the large map. However, this often wastes time, not to mention ink for the pen-plotter in the office, since in many cases portions of the conduits overlap with each other (albeit at different depths underground). What Irv wants is a way to determine the minimum number of line segments to draw given all the line segments from the separate maps.

## Input

Input will consist of multiple input sets. Each set will start with a single line containing a positive integer $n$ indicating the total number of line segments from all the smaller maps. Each of the next $n$ lines will contain a description of one segment in the format

$x_1$ $y_1$ $x_2$ $y_2$

where $(x_1, y_1)$ are the coordinates of one endpoint and $(x_2, y_2)$ are the coordinates of the other. Coordinate values are floating point values in the range $0 \ldots 1000$ specified to at most two decimal places. The maximum number of line segments will be 10000 and all segments will have non-zero length. Following the last input set there will be a line containing a 0 indicating end of input; it should not be processed.

## Output

For each input set, output on a single line the minimum number of line segments that need to be drawn on the larger, consolidated map.

## Sample Input

```
3
1.0 10.0 3.0 14.0
0.0 0.0 20.0 20.0
10.0 28.0 2.0 12.0
2
0.0 0.0 1.0 1.0
1.0 1.0 2.15 2.15
2
0.0 0.0 1.0 1.0
1.0 1.0 2.15 2.16
0
```

## Sample Output

```
2
1
2
```

# Problem E: Roll Playing Games

Phil Kropotnik is a game maker, and one common problem he runs into is determining the set of dice to use in a game. In many current games, non-traditional dice are often required, that is, dice with more or fewer sides than the traditional 6-sided cube. Typically, Phil will pick random values for all but the last die, then try to determine specific values to put on the last die so that certain sums can be rolled with certain probabilities (actually, instead of dealing with probabilities, Phil just deals with the total number of different ways a given sum can be obtained by rolling all the dice). Currently he makes this determination by hand, but needless to say he would love to see this process automated. That is your task.

For example, suppose Phil starts with a 4-sided die with face values 1, 10, 15, and 20 and he wishes to determine how to label a 5-sided die so that there are a) 3 ways to obtain a sum of 2, b) 1 way to obtain a sum of 3, c) 3 ways to obtain 11, d) 4 ways to obtain 16, and e)1 way to obtain 26. To get these results he should label the faces of his 5-sided die with the values 1, 1, 1, 2, and 6. (For instance, the sum 16 may be obtained as $10+6$ or as $15+1$, with three different "1" faces to choose from on the second die, for a total of 4 different ways.)

## Input

Input will consist of multiple input sets. Each input set will start with a single line containing an integer $n$ indicating the number of dice that are already specified. Each of the next $n$ lines describes one of these dice. Each of these lines will start with an integer $f$ (indicating the number of faces on the die) followed by $f$ integers indicating the value of each face. The last line of each problem instance will have the form

$$r\ m\ v_1\ c_1\ v_2\ c_2\ v_3\ c_3\ \cdots\ v_m\ c_m$$

where $r$ is the number of faces required on the unspecified die, $m$ is the number of sums of interest, $v_1, \ldots, v_m$ are these sums, and $c_1, \ldots, c_m$ are the counts of the desired number of different ways in which to achieve each of the respective sums.

Input values will satisfy the following constraints: $1 \leq n \leq 20$, $3 \leq f \leq 20$, $1 \leq m \leq 10$, and $4 \leq r \leq 6$. Values on the faces of all dice, both the specified ones and the unknown die, will be integers in the range $1 \ldots 50$, and values for the $v_i$'s and $c_i$'s are all non-negative and are strictly less than the maximum value of a 32-bit signed integer.

The last input set is followed by a line containing a single 0; it should not be processed.

## Output

For each input set, output a single line containing either the phrase "`Final die face values are`" followed by the $r$ face values in non-descending order, or the phrase "`Impossible`" if no die can be found meeting the specifications of the problem. If there are multiple dice which will solve the problem, choose the one whose lowest face value is the smallest; if there is still a tie, choose the one whose second-lowest face value is smallest, etc.

### Sample Input

```
1
4 1 10 15 20
5 5 2 3 3 1 11 3 16 4 26 1
1
6 1 2 3 4 5 6
6 3 7 6 2 1 13 1
4
6 1 2 3 4 5 6
4 1 2 2 3
3 3 7 9
8 1 4 5 9 23 24 30 38
4 4 48 57 51 37 56 31 63 11
0
```

### Sample Output

```
Final die face values are 1 1 1 2 6
Impossible
Final die face values are 3 7 9 9
```

## Problem F: Team Rankings

It's preseason and the local newspaper wants to publish a preseason ranking of the teams in the local amateur basketball league. The teams are the Ants, the Buckets, the Cats, the Dribblers, and the Elephants. When Scoop McGee, sports editor of the paper, gets the rankings from the selected local experts down at the hardware store, he's dismayed to find that there doesn't appear to be total agreement and so he's wondering what ranking to publish that would most accurately reflect the rankings he got from the experts. He's found that finding the *median ranking* from among all possible rankings is one way to go.

The median ranking is computed as follows: Given any two rankings, for instance **ACDBE** and **ABCDE**, the *distance* between the two rankings is defined as the total number of pairs of teams that are given different relative orderings. In our example, the pair **B**, **C** is given a different ordering by the two rankings. (The first ranking has **C** above **B** while the second ranking has the opposite.) The only other pair that the two rankings disagree on is **B**, **D**; thus, the distance between these two rankings is 2. The median ranking of a set of rankings is that ranking whose sum of distances to all the given rankings is minimal. (Note we could have more than one median ranking.) The median ranking may or may not be one of the given rankings.

Suppose there are 4 voters that have given the rankings: **ABDCE**, **BACDE**, **ABCED** and **ACBDE**. Consider two candidate median rankings **ABCDE** and **CDEAB**. The sum of distances from the ranking **ABCDE** to the four voted rankings is $1 + 1 + 1 + 1 = 4$. We'll call this sum the *value* of the ranking **ABCDE**. The value of the ranking **CDEAB** is $7 + 7 + 7 + 5 = 26$.

It turns out that **ABCDE** is in fact the median ranking with a value of 4.

### Input

There will be multiple input sets. Input for each set is a positive integer $n$ on a line by itself, followed by $n$ lines ($n$ no more than 100), each containing a permutation of the letters **A**, **B**, **C**, **D** and **E**, left-justified with no spaces. The final input set is followed by a line containing a 0, indicating end of input.

### Output

Output for each input set should be one line of the form:

*ranking* `is the median ranking with value` *value*.

Of course *ranking* should be replaced by the correct ranking and *value* with the correct value. If there is more than one median ranking, you should output the one which comes first alphabetically.

### Sample Input

```
4
ABDCE
BACDE
ABCED
ACBDE
0
```

### Sample Output

```
ABCDE is the median ranking with value 4.
```

# Problem G: To and Fro

Mo and Larry have devised a way of encrypting messages. They first decide secretly on the number of columns and write the message (letters only) down the columns, padding with extra random letters so as to make a rectangular array of letters. For example, if the message is "There's no place like home on a snowy night" and there are five columns, Mo would write down

```
t o i o y
h p k n n
e l e a i
r a h s g
e c o n h
s e m o t
n l e w x
```

Note that Mo includes only letters and writes them all in lower case. In this example, Mo used the character '`x`' to pad the message out to make a rectangle, although he could have used any letter.

Mo then sends the message to Larry by writing the letters in each row, alternating left-to-right and right-to-left. So, the above would be encrypted as

```
toioynnkpheleaigshareconhtomesnlewx
```

Your job is to recover for Larry the original message (along with any extra padding letters) from the encrypted one.

## Input

There will be multiple input sets. Input for each set will consist of two lines. The first line will contain an integer in the range $2 \ldots 20$ indicating the number of columns used. The next line is a string of up to 200 lower case letters. The last input set is followed by a line containing a single 0, indicating end of input.

## Output

Each input set should generate one line of output, giving the original plaintext message, with no spaces.

## Sample Input

```
5
toioynnkpheleaigshareconhtomesnlewx
3
ttyohhieneesiaabss
0
```

## Sample Output

```
theresnoplacelikehomeonasnowynightx
thisistheeasyoneab
```

# Problem H: Translations

Bob Roberts is in charge of performing translations of documents between various languages. To aid him in this endeavor his bosses have provided him with translation files. These files come in twos — one containing sample phrases in one of the languages and the other containing their translations into the other language. However, some over-zealous underling, attempting to curry favor with the higher-ups with his initiative, decided to alphabetically sort the contents of all of the files, losing the connections between the phrases and their translations. Fortunately, the lists are comprehensive enough that the original translations can be reconstructed from these sorted lists. Bob has found this is most usually the case when the phrases all consist of two words. For example, given the following two lists:

| **Language 1 Phrases** | **Language 2 Phrases** |
|---|---|
| *arlo zym* | *bus seat* |
| *flub pleve* | *bus stop* |
| *pleve dourm* | *hot seat* |
| *pleve zym* | *school bus* |

Bob is able to determine that *arlo* means *hot*, *zym* means *seat*, *flub* means *school*, *pleve* means *bus*, and *dourm* means *stop*. After doing several of these reconstructions by hand, Bob has decided to automate the process. And if Bob can do it, then so can you.

## Input

Input will consist of multiple input sets. Each input set starts with a positive integer $n$, $n \leq 250$, indicating the number of two-word phrases in each language. This is followed by $2n$ lines, each containing one two-word phrase: the first $n$ lines are an alphabetical list of phrases in the first language, and the remaining $n$ lines are an alphabetical list of their translations into the second language. Only upper and lower case alphabetic characters are used in the words. No input set will involve more than 25 distinct words. No word appears as the first word in more than 10 phrases for any given language; likewise, no word appears as the last word in more than 10 phrases. A line containing a single 0 follows the last problem instance, indicating end of input.

## Output

For each input set, output lines of the form

*word1/word2*

where *word1* is a word in the first language and *word2* is the translation of *word1* into the second language, and a slash separates the two. The output lines should be sorted according to the first language words, and every first language word should occur exactly once. There should be no white space in the output, apart from a single blank line separating the outputs from different input sets. Imitate the format of the sample output, below. There is guaranteed to be a unique correct translation corresponding to each input instance.

## Sample Input

```
4
arlo zym
flub pleve
pleve dourm
pleve zym
bus seat
bus stop
hot seat
school bus
2
iv otas
otas re
ec t
eg ec
0
```

## Sample Output

```
arlo/hot
dourm/stop
flub/school
pleve/bus
zym/seat

iv/eg
otas/ec
re/t
```