# Contents

1. Contest Rules and Scoring  
2. Example Problem Layout  
3. All Roads Lead...  
4. That's English?!?  
5. If the Shoe Fits  
6. A Forest of Words  
7. Wait in Line You Must  
8. Spline This  
9. A Coincidental Note  
10. A Coincidental Note (con’t)
i. Contest Rules and Scoring

The contest will last two and a half hours. Each team may have a maximum of four students. Calculators, notes, outside software, and books other than language references are prohibited. Books are subject to approval by the ACM staff. Team members may not discuss the contest with other teams or with their school sponsor during the competition.

Problem solutions will be graded as the test progresses. Each incorrect submission of a problem will result in a 10 minute time penalty added to your score. Solutions may be submitted as many times as you like. When a team is ready to submit a solution, a representative must raise his/her hand and wait for a grader. Once a hand is raised, a problem attempt will be logged, even if the team decides it is not ready. The grader can only provide the problem’s test input and state whether the solution is correct or incorrect. The graders cannot answer any questions! If a team has any questions about the problems, the grader will summon a judge. The judge’s decision is final.

The following three criteria will be used to determine final placement and prize winners:

- The team with the highest point total will be declared the winner, the second highest second place, and so on.
- Should two or more teams have the same number of points at the end of the competition, the higher rank will be awarded to the team with the lowest overall time, including penalty minutes. If a tie still exists, the team with the fewest attempts will be ranked higher.
- In the unlikely event that two or more teams are still tied after applying the tie-breaking criteria, a tie will be declared.

A note about input and output: Most of the contest problems do not depend upon exact formatting, so embedded spaces in programs’ input and output will be tolerated. Also, the method used for re-running programs for separate tests is left to you. Individual questions may override these general rules in their notes sections. If a team has any questions about how to format a program’s input or output, the team should ask a judge before submitting the problem to be graded.
ii. Example Problem Layout

This block contains a little story. Some might be humorous, and some might be really awful. We make no claims as to the quality of our fiction.

Problem Statement

This is the basic statement of the problem, along with any necessary definitions.

Notes

Typically, the input and output specifications are in the Notes section. Other included tidbits might be constraints on the arguments, hints, etc.

Examples

Example 1:
Obviously, this is the example section.
Computer output is shown like this, and input like this.

Example 2:
Exact input and output formats rarely matter, but it would greatly simplify our grading if your programs appear similar to our examples.
1. All Roads Lead...

To Gainesville? No, but you’re here anyways. Getting to town wasn’t too difficult, but finding an efficient path from the gas station to the Union probably was. If you had a laptop and were awake enough at that ghastly hour, you probably could have helped your poor coach...

Problem Statement

Write a program to find the least expensive path between two points on the map given in figure 1 plus some modifications garnered from locals.

The given map consists of many directional edges, but all the edges in opposite directions between the same nodes have the same cost. This could change, and all new edges will be one-way.

![Figure 1: Basic roadmap with costs in bold](image)

Notes

The first phase of input will consist of a sequence of change and delete commands. A change command will change an existing arc’s weight or add a new arc with the given costs between two nodes. Neither new nodes nor negative costs will be given. A delete command removes a directed edge between two nodes. Only edges which already exist will be deleted.

The exact syntax you type need not match ours, but it must not provide more information than the type of operation, the two nodes, and the cost when appropriate.

The second phase of input will ask for two nodes and output the least expensive path from the first to the second along with its cost. You may assume the destination will always be reachable.

This map is completely imaginary, although many of Gainesville’s roads do lie on a grid.

Examples

Example 1:
(C)hange, (D)elete, (E)nd: c
Enter origin, endpoint, and cost: 4 5 1
(C)hange, (D)elete, (E)nd: d
Enter origin and endpoint: 2 3
(C)hange, (D)elete, (E)nd: e

Find a path from where to where? 1 9
The shortest path from 1 to 9 has length 8 and follows 1 -> 4 -> 5 -> 6 -> 9.
Find a path from where to where? 3 8
The shortest path from 3 to 8 has length 12 and follows 3 -> 6 -> 5 -> 8.

Example 2:
(C)hange, (D)elete, (E)nd: d
Enter origin and endpoint: 1 2
(C)hange, (D)elete, (E)nd: d
Enter origin and endpoint: 2 3
(C)hange, (D)elete, (E)nd: e

Find a path from where to where? 1 3
The shortest path from 1 to 3 has length 18 and follows 1 -> 4 -> 5 -> 6 -> 3.

Find a path from where to where? 3 1
The shortest path from 3 to 1 has length 9 and follows 3 -> 2 -> 1.
2. That’s English?!?

It’s Rennaisance Fair season yet again. The SCA types are making themselves known at school. Ana, the one you can tolerate, has been pestering you. She wants all her computerized documents translated into something more ‘period’. She doesn’t quite seem to understand that they’re stored in a computer, hence are not ‘period’ by construction. Oh well.

**Problem Statement**

Write a program to translate normal English into the garbled language often heard at Rennaisance Fairs.

<table>
<thead>
<tr>
<th>Original</th>
<th>Mangled</th>
<th>Original</th>
<th>Mangled</th>
</tr>
</thead>
<tbody>
<tr>
<td>are/were</td>
<td>art/wert</td>
<td>have</td>
<td>hast</td>
</tr>
<tr>
<td>do</td>
<td>doest</td>
<td>has</td>
<td>hath</td>
</tr>
<tr>
<td>rooting</td>
<td>rootingeth</td>
<td>yes</td>
<td>aye</td>
</tr>
<tr>
<td>you</td>
<td>thou</td>
<td>often</td>
<td>oft</td>
</tr>
<tr>
<td>your</td>
<td>thy</td>
<td>nothing</td>
<td>naught</td>
</tr>
<tr>
<td>yourself</td>
<td>thyself</td>
<td>over</td>
<td>o’er</td>
</tr>
</tbody>
</table>

Table 1: Rules for converting English to Ren-Fair speak

**Notes**

Input will consist of a sentence, which will the program should mangle by the rules in table 1 and then output. You must maintain capitalization.

No input sentence will be longer than 50 characters. We will judge according to the given table, but we will not penalize for creative additions as long as the words given above are properly translated.

**Examples**

**Example 1:**
Text to translate:
*You do your master well.*
Translated:
Thou doest thy master well.

**Example 2:**
Text to translate:
*How are you?*
Translated:
How art thou?
3. If the Shoe Fits

So you heard that the theater folks could use a good hack now and again. You took the class, and by now you’ve managed some pretty spectacular set hacks, but your fingers are itching for a keyboard. The next production is *Into the Woods*, and one of the big effects scenes occurs when the prince comes to Cinderella’s house with the slipper. As everyone knows, the evil step-sisters are going to have their toes chopped off. The teacher has decided that they should pretend to cut the correct toes, but they really can’t figure out which ones must go. Eh, why not?

**Problem Statement**

Write a program to calculate which toes must be removed for the shoe to fit.

Each toe is modeled as an ellipse. The toes’ bottoms are aligned on a baseline which is also an axis of the ellipse modelling the shoe’s toe. The images of the toes along this line do not overlap but do form a continuous line. In other words, the toes are side-by-side but not necessarily flush against each other. The toes are also centered in the shoe. See figure 2 for an illustration.

![Figure 2: A generalized foot and shoe, with toe 5 marked for chopping](image)

**Notes**

The input will be six pairs of real numbers. The first five pairs give the length and width of the five toes. The last pair will give the extent and width of the shoe’s toe. See figure 2 for the generalized foot layout.

The output should state which toes, if any, must be removed.

**Examples**

Example 1:

Height and width of toe #1: 15 10
Height and width of toe #2: 18 12
Height and width of toe #3: 20 15
Height and width of toe #4: 22 18
Height and width of toe #5: 25 20
Extent and width of the shoe’s toe: 24 85
This sister loses toe 5.

Example 2:

Height and width of toe #1: 3 1
Height and width of toe #2: 4 1.5
Height and width of toe #3: 4 2.1
Height and width of toe #4: 5 2.3
Height and width of toe #5: 6 2 3
Extent and width of the shoe’s toe: 7 7
This sister loses toes 1, 2, and 5.
4. A Forest of Words

Ever feel that wading through a Dickens novel is like wandering lost through a very dense forest? Well, considering the number of trees used per book, it’s hardly surprising. And now your English teacher, Penny P. Worde, has asked everyone to write ideas on how to give the books back to the trees.

You’re sure she wants something about recycling, but you have something slightly more unusual in mind: symbolically (hey, it’s English class) replacing the trees with trees made of words. You’ll still be writing, but not an essay...

Problem Statement

Write a program to construct word trees. A word tree is constructed from a sequence by linking the first letter of a word to its first occurrence after the first letter in the word preceeding it, if possible. For example, the ‘a’ from ‘always’ in “analog always” would be linked to the second ‘a’ in ‘analog’.

Word trees are to be displayed as shown in the examples below. The first word of any tree must slant downwards to the right.

Notes

The input will consist of a number followed by a sequence of that many words of lowercase letters.

The output should consist of the word trees formed from that sequence. Separate word trees may be indented any reasonable amount. The program should pause momentarily (or until a keystroke) every ten lines.

At most 20 words will be given, and each word will be at most 20 characters long.

Examples

Example 1:
Number of words: 6
Words: evaluate less seriously inside small layers
Example 2:
Number of words: 5
Words: **word tree patterns are silly**
5. Wait in Line You Must

Wow, they’re back. Wow, what effects. And wow, what lines. It’s the afternoon of opening night, and the line for Jedi is already around the block. You feel very sorry for the poor person selling tickets. Whoever it is probably wants to kill the next person who asks for tickets in a Yoda voice.

They need to open up more ticket booths. You know it can’t hurt, but you start wondering how much it would help. You don’t really have much else to do while in line, so...

Problem Statement

Write a program to simulate a multi-server queue. You should find both the average length of the line and the wait time. The simulation ends when all customers have been served.

When determining the average length of the line, include the empty line while the last customer is being served. The wait time is the length of time one person spends in line before reaching a server.

Notes

Input will consist of the number of servers (at most five) followed by a series of real-valued inter-arrival and service times in minutes. The input will be terminated by two zero values. You will be given no more than twenty pairs.

The output should show the average length of the line and the average wait time to two places after the decimal.

Assume the line is initially empty.

Examples

Example 1:
Number of servers: 1
Inter-arrival and service time: 0 3
Inter-arrival and service time: 1 2.5
Inter-arrival and service time: 3 0.5
Inter-arrival and service time: 0 0

The line’s average length was 0.58, and people generally waited 1.17 minutes before being served.

Example 2:
Number of servers: 2
Inter-arrival and service times: 0 5
Inter-arrival and service times: 1 3
Inter-arrival and service times: 1 4
Inter-arrival and service times: 1 2.5
Inter-arrival and service times: 1 6
Inter-arrival and service times: 1 1
Inter-arrival and service times: 0 0

The line’s average length was 0.78, and people generally waited 1.75 minutes before being served.
6. Spline This

Why did you take shop? The Thal brothers, Nee and Arr, who teach it are neo-Luddites. Everything must be done in the most labor-intensive way possible. Um, excuse me, what’s that the purpose behind computers again?

The current class is about drawing using splines. “These natural curves,” Arr slowly says, “are much finer than anything those modern machines can produce.” Nee’s glaring at you; they know you think they’re anti-technology weenies. To prove a point, you surreptitiously slide out your laptop and begin softly to type.

Problem Statement

Write a program to define a piecewise cubic spline and evaluate it at multiple points.

A piecewise cubic spline is a collection of cubic splines and x intervals where each applies. Given a sequence of points \{ (x_j, y_j) \}_{j=1}^N, the basic cubic spline is defined on each interval by

\[
    y = Ay_j + By_{j+1} + Cy_j'' + Dy_{j+1}'',
\]

with the coefficients

\[
    A = \frac{x_{j+1} - x_j}{x_{j+1} - x_j}, \\
    B = 1 - A, \\
    C = \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2, \\
    D = \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2.
\]

You select the value of \( j \) by determining in which interval the given \( x \) lies.

You are only given the \( y_j \) values, not the \( y_j'' \) values. You can solve for these through the system of equations

\[
\begin{align*}
    \frac{x_j - x_{j-1}}{6}y_j'' - & \frac{x_{j+1} - x_j}{3}y_j'' + \frac{x_{j+1} - x_j}{6}y_j'' = \frac{y_j + y_{j+1} - y_j - y_{j-1}}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}.
\end{align*}
\]

which arise because of the smoothness conditions at each \( (x_j, y_j) \) where \( 1 < j < N \).

Let the first and last \( y'' \) values be zero.

Notes

The first input provided will be the number of points defining the cubic spline and will be at least four. The point coordinates will given next in increasing \( x \), followed by an undetermined number of \( x \) values at which the spline should be evaluated. How you terminate each run is left to you. Only two places beyond the decimal will be required, and no out-of-range data will be given.

Calculating the factors in equation 3 and the spline itself (equation 1) are straight-forward. The interesting bit is solving the system of equations given in 4. Also, the spline passes through each \( (x_j, y_j) \), so you have some simple test data.

Examples

Example 1:

Number of points: 4
Point 1: 0 5
Point 2: 2 7
Point 3: 3 8
Point 4: 7 12

Evaluate at: 1
The curve passes through (1.00, 6.00).
Evaluate at: 4
The curve passes through (4.00, 9.00).
Evaluate at: 6
The curve passes through (6.00, 11.00).

Example 2:
Number of points: 5
Point 1: -1.5 -4
Point 2: 2 3.8
Point 3: 9 11
Point 4: 12 -3
Point 5: 15 0

Evaluate at: 2
The curve passes through (2.00, 3.80).
Evaluate at: 9.8
The curve passes through (9.80, 9.56).
Evaluate at: 11.45
The curve passes through (11.45, 0.03).
Evaluate at: -1
The curve passes through (-1.00, -2.79).
7. A Coincidental Note

Argh. You hate it when you see people cheating. You hate it even more when they try to be overly clever. You’ve found a note left by one of your school’s famous cheating team. Apparently, Henry Gondorf took a test earlier and wants to help his buddies. It looks like random text, but you doubt it. You heard him complaining about funky French names, so you’re guessing it’s a Vigenère cipher. Luckily, you know it’s weakness: analysis through the index of coincidence.

The rest of the story is not necessary for the problem. The Vigenère cipher works by shifting characters by a relative amount. Each letter of the key determines the distance the plaintext will be shifted. For example, if the plaintext were ‘A’ and the appropriate letter of the key were ‘G’, the result would be ‘G’ (‘A’ + ‘G’ - ‘A’). If the next letters of the text and key were ‘C’ and ‘D’, respectively, the result would be ‘F’. Thus, each letter of the key determines a rotation of the alphabet. The key is reused, so if the key were five letters long, every five characters would be from the same rotation.

Problem Statement

Write a program to determine the index of coincidence of a given string with shifts ranging from one to ten.

The index of coincidence of a sample is calculated by rotating the letters to the right and finding the probability of finding the same letter in the same location. For example, the index of coincidence of ackgaknarf with a shift of three is .3. Random letters have an index of coincidence around 3.8%; English text has one around 5.9%. Rotating the alphabet does not change the index of coincidence, so it can be used to identify the length of the key used in a Vigenere cipher. Here, however, the samples will not be long enough to gather useful statistics.

Notes

Input will consist of a sequence of at most 30 letters. Capitalization does not matter.

Your output should be a table of shifts and calculated indices to three places after the decimal point. The output should not be expressed as a percentage.

Examples

Example 1:
Cipher text: ackgaknarf

<table>
<thead>
<tr>
<th>Shift</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
</tr>
<tr>
<td>3</td>
<td>0.300</td>
</tr>
<tr>
<td>4</td>
<td>0.100</td>
</tr>
<tr>
<td>5</td>
<td>0.000</td>
</tr>
<tr>
<td>6</td>
<td>0.100</td>
</tr>
<tr>
<td>7</td>
<td>0.300</td>
</tr>
<tr>
<td>8</td>
<td>0.000</td>
</tr>
<tr>
<td>9</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>1.000</td>
</tr>
</tbody>
</table>
8. A Coincidental Note (con’t)

This problem’s story is a continuation of the previous problem, but the solution is independant.

Ok, now you know how long the key is, but you still don’t know it. Rotating an alphabet isn’t difficult, but you need to run a lot of them. Johnny Hooker, Gondorff’s partner, is probably almost here. You’d love to leave a little, encrypted note in the place of the one you found...

And a little more background on breaking Vigenère ciphers... This simple rotation is just a building block. You’ll need this as a subroutine which will be applied many, many times. The general recipe for breaking Vigenère ciphers isn’t too difficult, actually.

Take each substring with the same rotation (those corresponding to the same letter of the key). For the first substring, guess the rotation by mapping the most common letter to the most common letter of some sample plaintext (normally ‘E’). Later guesses should try less likely matches, typically falling down the string “etaoinshrdlu...”

For each successive substring, guess the rotation by checking the likelihood of possible digrams, pairs of letters. For example, if a letter in the previous substring was ‘T’, the corresponding letter in the current substring is probably ‘H’.

Continue these guesses until you find the plaintext. It could take a while, but not as long as factoring 1024-bit pseudoprimes like those used in public key cryptosystems.

Problem Statement

Write a program to translate a block of text into a rotated alphabet.

A reasonable guess at the correct rotation is to first try mapping the most common letter to ‘E’. Afterwards, try rotating the most common letter to an input character.

Notes

The first input will be a block of at most 30 letters. After displaying the initial attempt, repeatedly prompt for another letter and redisplay the newly transformed string.

In the case of ties, pick the letter which occurs first in the text.

Examples

Example 1:
Ciphertext: qddazqageqjbjofmuaze
Attempt: erroneousexpectations
Rotate ‘q’ to: r
Decoded: reebabhfrkcrpgngvbaf

Ciphertext: yqnekocaknca
Attempt: mebsyecqoybqo
Rotate ‘q’ to: s
Decoded: aspgmsqecmpec
Rotate ‘q’ to: r
Decoded: zroflrpdblodb
Rotate ‘q’ to: u
Decoded: curiousgeorge