

ACM Intercollegiate Programming Contest

Asia Regional Contest, Singapore

Nanyang Auditorium

Nanyang Technological University

Singapore

29–30 November 2001

Contents

Problem 1.....	1
Problem 2	3
Problem 3	6
Problem 4	9
Problem 5	16
Problem 6	18
Problem 7.....	20
Problem 8.....	22

Problem 1

Encryption

Input file: data1.txt

Say you are the founder and sole employee of your own e-business start-up. Hours before delivering an e-commerce solution to your very first client, you realize to your horror that you have forgotten to include encryption features for sensitive information.

Faced with a very tight deadline, you decide to incorporate a simple encryption routine and later sell a security upgrade to your client at a tidy sum. After looking through possible methods that you can implement quickly, you finally settle on the following scheme known as a Vigenere cipher.

First, you determine a small repeated *key* that is the length of your plaintext (unencrypted) message. Next, you align the plaintext message such that the first letter of the message is aligned to the first letter of the key. Finally, each key letter index is added to its corresponding plaintext letter index to produce the *ciphertext* (encrypted) letter index. These combined ciphertext letter indices will form the final encrypted message. Your task is to write such a program.

Input

The input file consists of pairs of lines. The first line of each pair will contain the key while the second line will contain the plaintext message. Only letters of the alphabet (A-Z) plus a space will be used. These may be in upper- or lower-case and you will have to convert them into upper-case if necessary.

Output

The output file consists of one line of upper-case text for each pair of lines in the input file. Each line will contain the encrypted message.

Sample Input

ABCABCABCABCAB

Attack at dawn
shineshines
How are you

Sample Output

BVWBENACWAFDXP
AWFNFKMIMTN

Problem 2

Tiling Polygons

Input file: data2.txt

In a polygon tiling game, you are required to fill a polygon with a set of basic shapes called “tiles”. This is equivalent to cutting up the entire polygon into a number of small pieces; each piece must have exactly the same shape as one of the provided tiles. Let us consider a simple version of the game. Two types of rectangular tiles are given as shown in Figure 2.1. Their width/height are $1/3$ and $3/1$ respectively. You are asked to use these two types of tiles to tile a rectilinear polygon (that is, a polygon with only vertical or horizontal edges). For instance, Figure 2.2(a) shows such a polygon and Figure 2.2(b) shows how it can be properly tiled with three $3?1$ tiles and one $1?3$ tile.

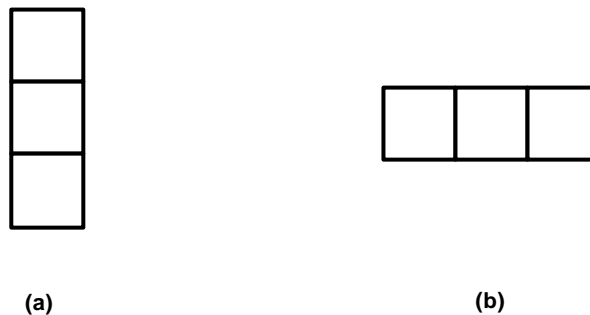


Figure 2.1: $1?3$ tile and $3?1$ tile

The coordinate system used in this game is described by the following rules:

- (1) A polygon is always positioned in such a way that its lowest edge is on the x-axis and its leftmost edge is on the y-axis, as shown in Figure 2.2.
- (2) A polygon is represented by an ordered list of points that enumerates all of its vertices. The enumeration starts from the lower-left vertex and proceeds counter-clockwise. For instance, the polygon in Figure 2.2(a) is represented by the set of points $\{A, B, C, D, E, F, G, H\}$.
- (3) A grid (a rectangle whose width and height are both 1) is represented by a pair (x, y) , where x and y are the x-coordinate and y-coordinate of its top-right vertex. In Figure 2.2(a), the coordinate of every grid is indicated inside the grid.

- (4) A tile is designated by a triple (x, y, z) , where x and y are the coordinates of the middle grid of the tile and z represents the orientation of the tile. If the tile is horizontal, then $z=0$; otherwise, $z=1$. Figure 2.3 gives two examples.

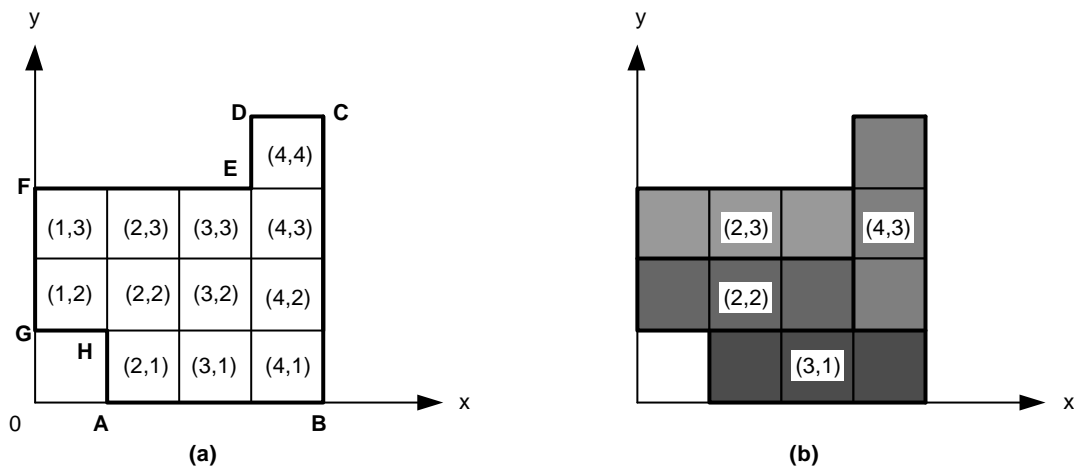


Figure 2.2 An example of tiling

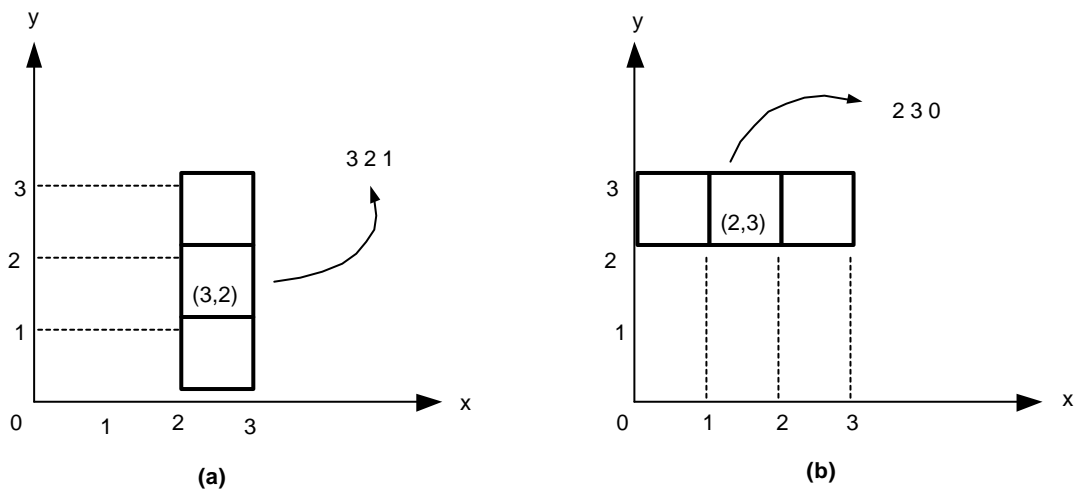


Figure 2.3 Representing a tile

Your task is to write a program that finds a way to tile a given polygon. The polygon is known to be tillable. If there is more than one ways of tiling; you are only required to find one of them.

Input

The input file describes a polygon to be tiled. The first line is the number of vertices of the polygon. The subsequent lines specify the vertices of the polygon in the order stated above; one line per vertex. A vertex is specified as an x -coordinate followed by its y -coordinate, separated by a white space.

Output

The output produces a list of tiles that tiles the polygon specified in the input file. A tile is specified as a triple $x y z$ as described above, separated by a white space; one tile per line. The tiles may appear in any order.

Sample Input

The following shows what the input file looks like if the polygon in Figure 2.2(a) is to be tiled:

```
8
1 0
4 0
4 4
3 4
3 3
0 3
0 1
1 1
```

Sample Output

The following shows what your output may look like if you want to represent the solution as shown in Figure 2.2(b):

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

Problem 3

Searching Sequence Database in Molecular Biology

Input file: data3.txt

Molecular biologists frequently compare *bio-sequences* to see if any similarities can be found in the hope that what is true of one sequence is also true of its analogue. In this problem, we focus on nucleic acid sequences that are composed of four symbols 'A', 'C', 'G' or 'T'. Generally, such comparisons involve aligning sections of the two sequences in a way that exposes the similarities between them. Given a query sequence and a set of sequences stored in a database, you are asked to write a program that searches the database and finds the sequence having the largest similarity score with the query sequence.

The similarity score between the query sequence and a database sequence is the sum of the alignment scores of the aligned pairs of symbols from an alignment of the two sequences. Two identical symbols that are aligned are given a score of 5 while a mismatched pair of symbols is assigned a score of 4. A gap is introduced into an alignment if one symbol in one sequence is not aligned with symbols in the other. The penalty for a gap is a score of 7. For example, given a query sequence $m = \text{"GAAGGCA"}$ and a database sequence $n = \text{"GCAGAGCA"}$, the following alignment between them (aligned pairs of symbols are written one above the other) has a similarity score of $5 + 4 + 5 + 5 + 7 + 5 + 5 = 21$.

Sequence m: G A A G ? G C A

Sequence n: G C A G A G C A

Note that a gap in the above alignment is represented by the symbol '?'. The dynamic programming algorithm provides a rigorous mathematical approach towards this alignment problem.

Input

The input consists a query sequence (on the first two lines) and a set of data sequences (each occupying two lines). Each sequence has the following format:

>sequence name

sequence data

There is a blank line between two adjacent sequences.

Output

The output indicates the sequence having the highest similarity score with the query sequence. The result should be printed out in the following format:

The query sequence is:

(the query sequence data)

The most similar sequences are:

(sequence 1 data)

The similarity score is: (similarity score)

(sequence 2 data)

The similarity score is: (similarity score)

Sample Input

>query

ACGGG

>seq1

ACGGT

>seq2

ACGGGG

>seq3

TCCGGTT

>seq4

TCGGG

>seq5

AACGGG

Sample Output

The query sequence is:

ACGGG

The most similar sequences are:

ACGGGG

The similarity score is: 18

AACGGG

The similarity score is: 18

Problem 4

Comparing Trees

Input: data4.txt

You work on a submarine and have just received an updated version of the submarine's manual, which is several gigabytes in size. The captain does not want the crew to waste time leafing through the new version to figure out how it differs from the one they read in training. On the other hand, she would hate to accidentally launch a missile when trying to turn on the microwave. As the expert programmer on board, you are assigned the task of comparing the old and new versions and marking how they differ from each other. For this task, the captain graciously provides you with a highlighter and lots of coffee.

You have a better idea. Since both versions are available in electronic form, you think of writing a program to compare them automatically. The manual is organized hierarchically in volumes, chapters, sections, subsections, paragraphs, etc., so you model it using a tree structure. All you need to do now is to write a program that takes two trees and produces a description of how they differ.

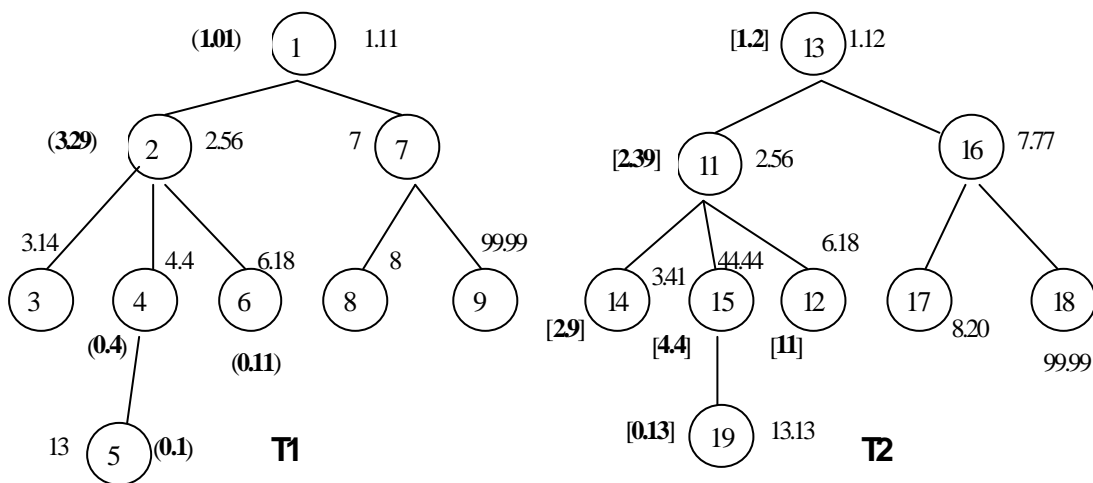


Figure 4a: Comparing Trees

We use the term *tree* to mean a rooted, ordered, labeled tree, such as those depicted in Figure 4a. The node number in the figure is used to identify a node and is indicated within the circle representing the node. Each node in a tree also has a label, which is indicated next to that node in the figure (numbers which are not bold and enclosed in parentheses or square brackets). For this problem, we assume that labels are real numbers. For example, the label of node 4 is 4.4. Note that the numbers, which are bold and enclosed in parentheses and square brackets in T1 and T2, are *deletion* and *insertion costs* of the nodes respectively and shall be explained later. Each node except the root also has a unique parent node. In the figure, we connect a node to its parent (depicted above the node) using a line. For example, the parent of node 7 is node 1. The children of each interior node are ordered, and the figure depicts the children of each node left-to-right in this order. For example, the children of node 2, in order, are 3, 4, and 6.

All the nodes that lie on the path (of length zero or more) from a node to the root are called the node's *ancestors*. Note that every node is its own ancestor. If a node n_1 is the ancestor of node n_2 , then n_2 is called a *descendant* of n_1 . The tree consists of all the descendants of a node n is called its *subtree*. The length of the path from a node to the root is called the node's *depth*. Thus, the depth of the root is always 0. In our example, the depth of node 4 is 2.

The *preorder list* of a tree consists of the root followed by the preorder list of the subtrees rooted at its children, in order. For example, the preorder list of tree T1 is (1, 2, 3, ..., 9). The preorder list of tree T2 is (13, 11, 14, 15, 19, 12, 16, 17, 18).

Given such a tree, we can *edit* (modify) it using the following three kinds of *edit operations*:

- ? ? An *update operation* can be used to change the label of a node. For example, we can change the label of node 3 to 3.41 by using the operation $upd(3, 3.41)$. That is, the operation $upd(n, l)$ changes the label of node n to l .
- ? ? A *delete operation* can be used to remove any subtree from a given tree. For example, we can remove the subtree rooted at node 4 by using the operation $del(4)$. That is, the operation $del(n)$ applied to any tree containing all of n 's descendants (including n) from the tree.
- ? ? An *insert operation* can be used to attach one tree to another tree. Refer to the trees T* and T3 in Figure 4b. T* can be attached as the second child of the node 2 in T3 by using the $ins(T^*, 2, 2)$. That is, the operation $ins(T, n, i)$ makes T the i th child of node n .

An *edit script* is a sequence of such edit operations. An edit script is applied to a tree by applying the operations one after another in the order they are listed. Figure 4b

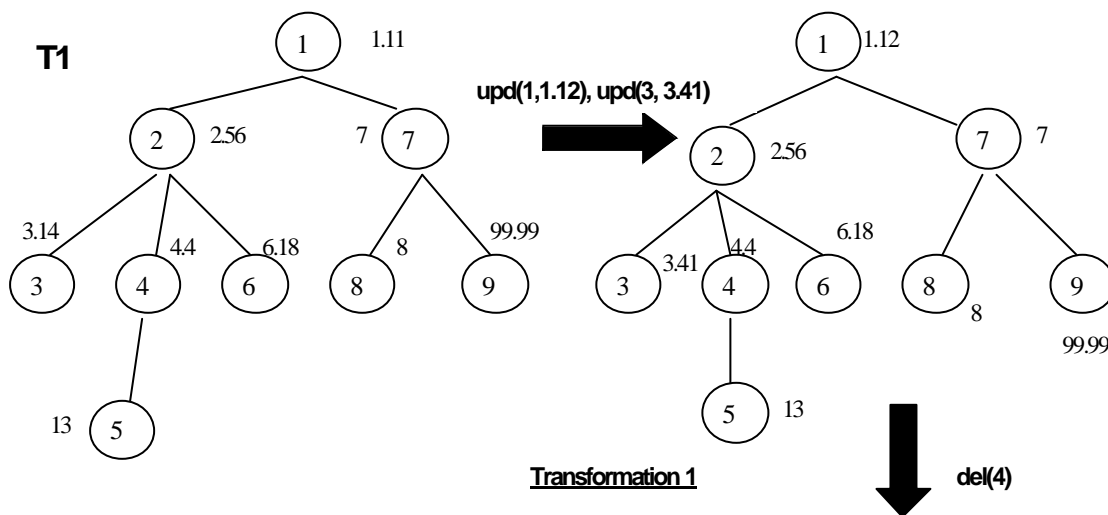
depicts the edit script ($upd(1, 1.12)$, $upd(3, 3.41)$, $del(4)$, $ins(T^*, 2, 2)$, $upd(7, 7.77)$, $upd(8, 8.20)$) applied to tree T1 from Figure 4a. Note that the resulting tree is identical to tree T2 (except for node identifiers). We say that our *edit script transforms* T1 to T2.

Each edit operation has a cost associated with it as described below:

? ? The *cost of an update operation* is simply the absolute value of the difference between the old and new labels. Thus, updating a label 1.11 to 1.12 costs 0.01 units and updating a label 9.99 to 99.9 costs 89.91 units.

? ? Each node has a *deletion cost* that depends on its label and that is specified as part of the input. In Figure 4a (T1), the deletion cost of a node is indicated in parentheses next to the node. If no such number appears next to a node, its deletion cost is 1 unit by default. The *cost of a delete operation* is the sum of the costs of deleting all the nodes in the deleted subtree. For example, the operation $del(4)$ in our example costs $0.4 + 0.1 = 0.5$ units.

? ? Similarly, each node has an *insertion cost* that depends on its label and that is specified as part of the input. In Figure 4a (T2), the insertion cost of a node is indicated in square brackets next to the node. If no such number appears next to a node, its insertion cost is 1 unit by default. The *cost of an insert operation* is the sum of the insertion costs of all the nodes in the inserted subtree. For example, using the costs indicated in Figure 4a, the cost of the operation $ins(T^* 2, 2)$ in earlier example is $4.4 + 0.13 = 4.53$ units.



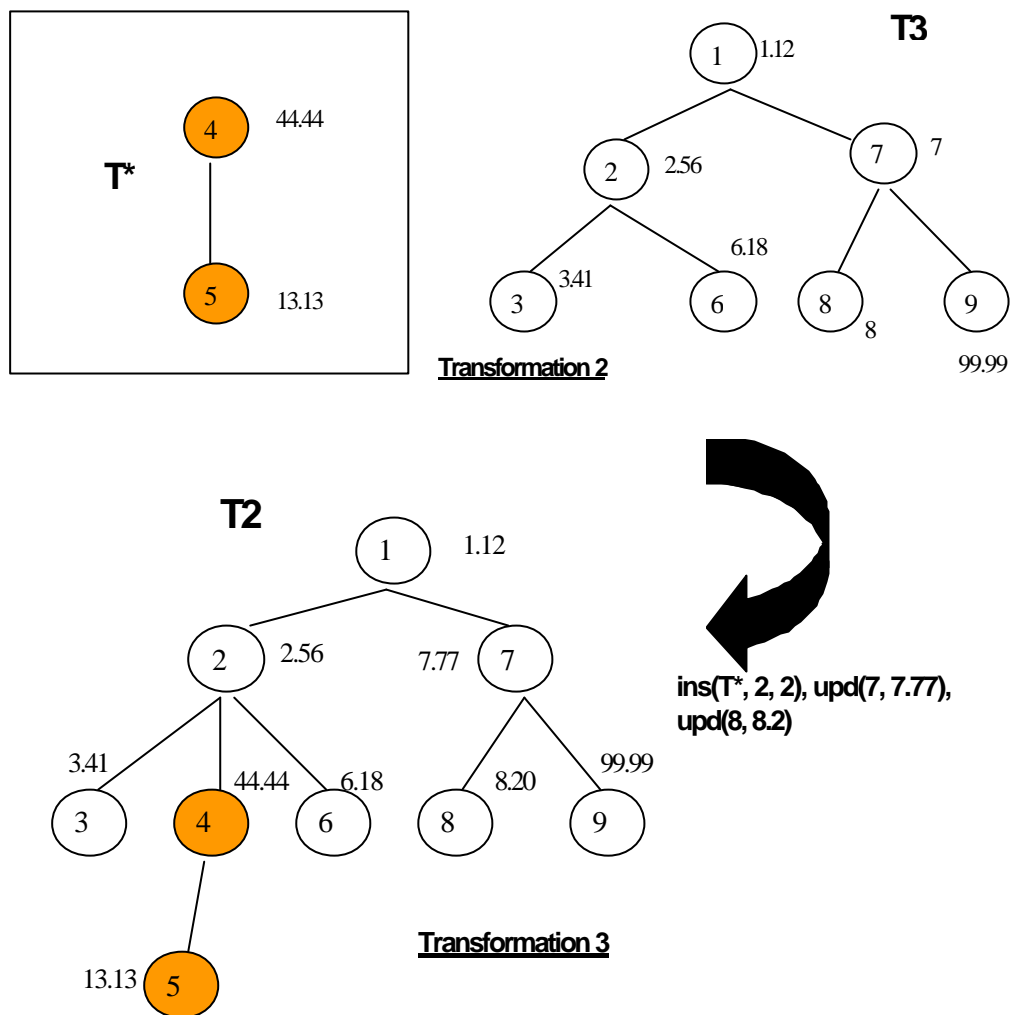


Figure 4b: Transforming a tree using edit script.

?? The cost of inserting or deleting a node whose label does not appear in any of the input trees is 1 unit by default. Note that to avoid cluttering the insertion and deletion costs of the nodes are not displayed in Figure 4b.

?? The *cost of an edit script* is the sum of the costs of the operations it contains.

Your goal is to write a program that takes as input two trees (and the node insertion and deletion costs) and produces as output the cost of a minimum-cost edit script that transforms one tree to another. (You need not compute such a minimum-cost edit script; you only need its cost.) As was the case for the trees in Figure 4a, there is no correspondence between the node identifiers in the two trees; thus they are not specified in the input. You are also required to pretty-print the input trees as described below.

Input

The input file first lists the nodes in the first tree in preorder, one per line. For each node, the input file contains a line with three numbers that are separated using spaces. The first number is the depth of the node. The second is the node's label. The third is that node's deletion cost. After all the nodes in the first tree are listed in this manner, the input file contains a line `"-1 0 0"` to denote the end of the tree's listing.

Next, the input contains a blank line followed by a listing of the second tree. The second tree is listed using the method described above, the only difference being that the third number on each line is now the node's insertion cost. This listing of the nodes in the second tree is also followed by a line `"-1 0 0"` to denote the end of the listing.

Assume that all real numbers in the input are in the range $[0, 99.99]$ (inclusive).

Output

The output consists of two parts. First, you pretty-print the two input trees. Second, you report the cost of a minimum-cost edit script that transforms the first tree to the second.

You must first output the first tree by listing one node per line, in preorder. Each such line must begin with 4d space characters, where d is the depth of the node. Next, it contains the node's label, with two digits before and two digits after the decimal point. If the label is less than 10, include a leading space character. Thus, 9.87 is output as `"? 9.87"` where ? denotes the ASCII space character. This label is followed by a single space character, followed by the node's deletion cost listed in the same format as that used for the label.

Next, you must output a blank line, followed by the listing of the second tree in the above format (with the deletion cost replaced by the insertion cost).

Finally, you output a blank line followed by a line containing the string `"Distance:"` followed by a space character followed by the cost of the minimum-cost edit script. For outputting this cost, use the same format you used for the tree labels. (You can assume that the cost will always be a real number in the range $[0, 99.99]$.)

Do not include anything else in your output, not even blank lines.

Sample Input

For the tree depicted in Figure 4a, the input and output are shown below. (For clarity, we use the ? symbol to denote the ASCII space character.) Note that the space characters at the beginning of lines in the input are optional; your program should

work whether or not they occur in the input. As it turns out, the edit script depicted in Figure 4b is a minimum-cost edit script for this input, and its cost is 6.28, as reported in the output.

```
0?1.11?1.01
??1?2.56?3.29
????2?3.14?1
????2?4.4?0.4
??????3?13?0.1
????2?6.18?0.11
??1?7?1
????2?8?1
????2?99.99?1
-1?0?0
```

```
0?1.12?1.20
??1?2.56?2.39
????2?3.41?2.9
????2?44.44?4.4
??????3?13.13?0.13
????2?6.18?11.0
??1?7.77?1
????2?8.2?1
????2?99.99?1
-1?0?0
```

Sample Output

```
?1.11??1.01
?????2.56??3.29
?????????3.14??1.00
?????????4.40??0.40
????????????13.00??0.10
?????????6.18??0.11
?????7.00??1.00
?????????8.00??1.00
?????????99.99??1.00
```

```
?1.12??1.20
?????2.56??2.39
```

?????????3.41??2.90
?????????44.44??4.40
?????????????13.13??0.13
?????????6.18??11.00
?????7.77??1.00
?????????8.20??1.00
?????????99.99??1.00

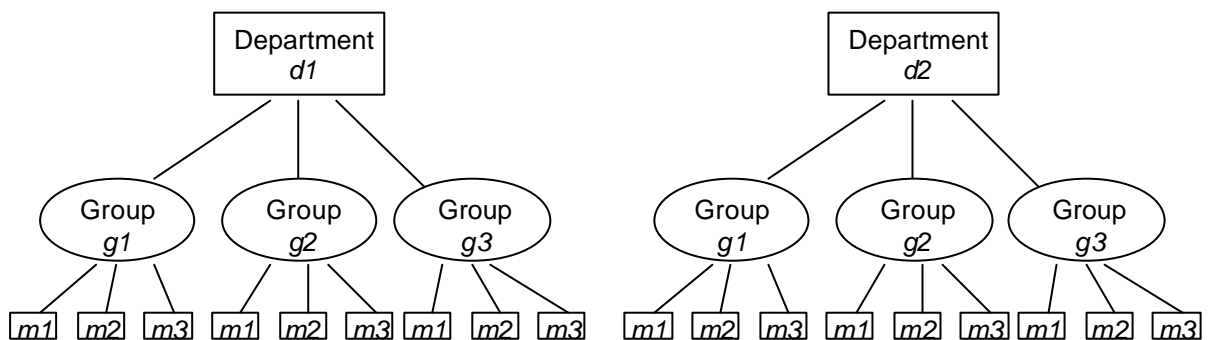
Distance:??6.28

Problem 5

Enumeration of Cross-Department Teams

Input file: data5.txt

A company consists of D departments. Each department consists of groups of members as illustrated in the figure below. As part of his strategic planning, the company's CEO would like to select a cross-department team consisting of x members from a group in a department and y members from a group in another department to work directly with him on new business strategies.



In this problem, you are requested to develop a program that generates a listing of all possible cross-department teams. A cross-department team is represented by its $(x+y)$ members, each globally identified by a concatenation of his/her department id, group id and member id. For example, the member m_3 from group g_2 in department d_1 is identified by $d_1g_2m_3$. For simplicity, we assume that every department consists of G groups, and every group consists of N members ($N < 30$).

Input

The input consists of the number of departments, the number of groups per department, the number of members per group, x and y .

Output

The output consists of a listing of unique cross-department teams that can be formed. The global ids of the members of each team are listed on an output line. The teams should be sorted by the department and group ids of their members.

Sample Input

4 3 3 2 1

Sample Output

d0g0m0 d0g0m1 d1g0m0

d0g0m0 d0g0m1 d1g0m1

d0g0m0 d0g0m1 d1g0m2

d0g0m0 d0g0m2 d1g0m0

...

d2g2m2 d3g2m0 d3g2m2

d2g2m2 d3g2m1 d3g2m2

Problem 6

Container Set Selection

Input file: data6.txt

To facilitate the shipping of certain hazardous materials, a shipping company has come up with the idea of introducing variable sized containers. The volume of each container is an integer in some unit and because of the nature of the content, each container must be completely filled before it can be used to transport the goods. The company has determined that since the handling cost of each container is the same irrespective of the size, it will charge the customers according to the number of containers hired. This policy then provides an incentive for the customer to choose the minimum possible number of containers for a given volume of goods.

To make the task of choosing the minimum number easier, the shipping company is looking into selecting a set of container sizes such that the minimum number of containers for any given volume can be obtained by choosing the largest possible sizes first. Any set of container sizes that satisfies this property will be designated as acceptable; otherwise, it is unacceptable. For example, the set 10, 6, 3, 1 is unacceptable because for a volume of 12 units, choosing the largest possible sizes first will result in the combination $10+1+1$ with a count of 3 containers; whereas, the combination $6+6$ requires 2 containers only. Your task is to write a program that will determine whether a given set of container sizes is acceptable or not acceptable.

Input

The input format will be one line per container set. Each line will contain the number of container sizes in the set, followed by each size value in descending order. The maximum number of container sizes will be 20, and the minimum 1. The smallest size will be 1 and the largest is 200. Your algorithm should be valid for any integral volume of goods without any upper limit. The end of file is denoted by a container size set with zero sizes.

Output

The output will be a single line listing the sizes in the set with a single space between the sizes and ending with the word "Acceptable" or the words "Not Acceptable".

Sample Input

```
6 6 5 4 3 2 1
6 100 40 20 9 5 1
5 20 10 5 2 1
0
```

Sample Output

```
6 5 4 3 2 1 Acceptable
100 40 20 9 5 1 Not Acceptable
20 10 5 2 1 Acceptable
```

Problem 7

Separating Golden and Silver Balls

Input file: data7.txt

In a treasure house, there are $2N$ caskets in a row. All the caskets are filled with either golden or silver balls except two adjacent caskets, which are empty. There are altogether $(N + 1)$ golden balls and $(N - 1)$ silver balls; there is only one ball per casket. An example of a row configuration is shown below where $N=5$, G represents a golden ball and S represents a silver ball:

G	S	S	G			G	S	G	S
---	---	---	---	--	--	---	---	---	---

Any two adjacent caskets can be moved into the two empty caskets without changing the order of the two adjacent caskets. Your task is to write a program that makes all gold balls located to the left of all silver balls using a minimal sequence of moves.

Input

The input consists a number of initial row configurations as indicated by a number at the beginning of the input file. Each configuration consists of a number representing N on one line, and an initial state of $2N$ caskets on the second line. Each empty casket is represented by a space character. Each configuration is separated by a blank line from the next configuration.

Output

The output consists of a set of row configurations illustrating the sequence of moves; there is a set of row configurations per input row configuration. Each set of row configurations is preceded by the initial row configuration; each configuration is preceded by an ordinal number on the same line represent the step count; the initial configuration is preceded by 0.

Sample Input

2

5

abba abab

6

ababab baba

Sample Output

0 abba abab

1 abbabaa b

2 a abaabbb

3 aaaab bbb

0 a babbababa

1 aabbabbab a

2 aabbab bbaa

3 aa abbbbaa

4 aaaaabbbbb

Problem 8

Neighbourhood of a point set

Input file: data8.txt

Given a set of m points on a $N \times N$ grid, we want to find its neighbourhood size, which is the number of grid points that are at a distance at most t from any of the m input points. Let $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \dots, p_m = (x_m, y_m)$ be the set of m input points. We treat the input points as points from a $N \times N$ grid. This grid is labeled in such a way that the bottom left corner is $(0, 0)$ and the upper right corner is $(N - 1, N - 1)$. Thus, x_i and y_i are integers satisfying $0 \leq x_i < N$ and $0 \leq y_i < N$, for each $i = 1, 2, \dots, m$. The input points are already sorted so that for any $i < j$, we have either $x_i < x_j$ or $(x_i = x_j)$ and $(y_i < y_j)$.

In this problem, the distance between two points is measured by the *infinity norm*; that is, the distance between two points (x, y) and (a, b) is $\|(x, y) - (a, b)\|_\infty = \max\{|x - a|, |y - b|\}$. We want to find the total number of grid points, whose distance from the nearest input point is less than or equal to t . To be precise, let us define the function D on the grid points as $D(x, y) = \min_{i=1, 2, \dots, m} \|(x, y) - (x_i, y_i)\|_\infty$. The size of the neighbourhood is the number of grid points (x, y) such that $D(x, y) \leq t$ for each grid configuration. (Note that because (x, y) is a grid point, we have $0 \leq x < N$ and $0 \leq y < N$.)

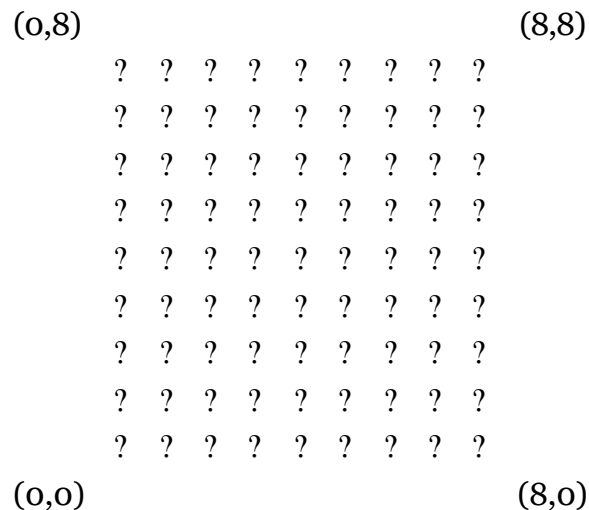


Figure 8: Illustration of the sample input.

As an example, consider the grid in Figure 8. In this figure, a “?” is a grid point, “?” is an input point and “?” is a grid point that is within a distance of 2 from the nearest input point. The output is the number of “?” and “?”, which is 40.

Your program should be efficient enough to handle large point set. In particular, your program should take less than 5 minutes system time on the provided machine.

Input

The input file consists of two configurations of points. Each configuration is represented by 3 integers which correspond to N , t , and m , and follows by a sequences of m pairs which correspond to the m input points $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \dots, p_m = (x_m, y_m)$. The second configuration follows immediately after the first configuration in the input file.

Output

The output is the neighbourhood size of the two configurations.

Sample Input

```
9 2 3
4 3
4 4
7 1
5 1 1
2 2
```

Sample Output

```
the first configuration has 40 neighbouring points.
the second configuration has 9 neighbouring points.
```