ACM Hong Kong
Scholastic Programming Contest 1995

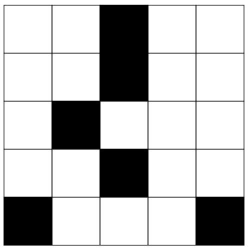
Total number of questions: 6

Total number of pages: 13

Contest time: 4 hours

1 Crossword Puzzles

A board is an $m \times n$ array of squares. For this problem, we consider boards with some of the squares missing. The following is an empty 5×5 board.



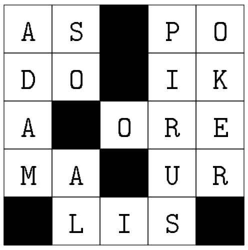
Your job is to fill up the remaining squares of a board using a set of given words, each of which is a sequence of at least 2 letters. Given a continuous block of squares, we number the squares (1) from top to bottom if the block is vertical and (2) from left to right if the block is horizontal. The following constraints must be satisfied in filling up a board.

- Each word must be used *exactly* once.
- Each continuous block of squares, either horizontal or vertical, must be filled using *one* word. A block B can be filled by word W if the number of squares in B is equal to the number of letters in W. We fill B with W by placing letter i of W in square i of B.
- If two continuous block of squares B_1 and B_2 intersect at square i of B_1 and square j of B_2, then we can fill B_1 and B_2 using words W_1 and W_2 respectively if and only if
  - W_1 can fill B_1 and W_2 can fill B_2,
  - letter i of W_1 and letter j of W_2 are the same.

Filling the empty board shown above using the following set of words

```
DO ORE MA LIS UR AS PO SO PIRUS OKER AL ADAM IK
```

gives



Write a program which reads in a series of crossword puzzle descriptions. For each description, determine if it is possible to fill the specified board with the associated set of words. If so, your program should print out the filled board. If not, your program should print "This crossword puzzle cannot be solved."

Input

When the program starts, it should prompt the user to enter the name of the input file with the display string "Enter input file name: ". The input file contains at least one crossword puzzle description and the descriptions are separated by one blank line. Each puzzle description consists of a board specification, followed by a list of words for filling the board. The format for a puzzle description is:

- The first line contains two positive integer separated by spaces, specifying the number of rows m and columns n of the board respectively.
- The next m lines describe the board, with each line containing n characters of either "0" or "1." A "0" character indicates a missing square; while a "1" character indicates the presence of a square.
- Subsequent lines list the words used in filling the board, with one word per line.
- A crossword puzzle description ends with a blank line or the end-of-file character.

Output

The output for the N-th puzzle should be led with a line:

```
Puzzle N:
```

Then your program should display the board as it appears when filled using the given words if the puzzle is solvable. Positions of the board where there are no squares should appear as blanks in the output. If there is more than one possible solution for a given puzzle, print all of them. If the puzzle is not solvable, output the message "This crossword puzzle cannot be solved." Output for different puzzles and output for different solutions of the same board should be separated by one blank line.

You can assume that the maximum size of the boards is 30×30. Efficiency is an important consideration for this question. Your program should return a solution within 5 seconds.

Example

`Input file:`

```
5 5
11011
11011
10111
11011
01110
DO
ORE
MA
LIS
UR
```

```
AS
PO
SO
PIRUS
OKER
AL
ADAM
IK

3 4
1100
0011
1100
JIM
MARY
```

Output on screen:

```
Puzzle 1:
AS PO
DO IK
A ORE
MA UR
 LIS

Puzzle 2:
This crossword puzzle cannot be solved.
```

2 Permutations and Inversions

A permutation of a finite set is an arrangement of its elements into a row. Permutations are of special importance in the study of sorting algorithms since they represent the unsorted input data. In this question, we consider permutations of $\{1, 2, \ldots, n\}$ for some positive integer n. The concept of inversions of permutations was introduced by G. Gramer in 1750 in connection with his famous rule for solving linear equations. Let $a_1, a_2, \ldots, a_n$ be a permutation of the set $\{1, 2, \ldots, n\}$. If $i < j$ and $a_i > a_j$, then the pair (a_i, a_j) is called an *inversion* of the permutation. For example, the permutation $3, 1, 4, 2$ has three inversions: $(3, 1)$, $(3, 2)$, and $(4, 2)$. Each inversion is a pair of elements that is "out of sort" so that the only permutation with no inversions is the sorted permutation $1, 2, \ldots, n$.

The *inversion table* $b_1, b_2, \ldots, b_n$ of the permutation $a_1, a_2, \ldots, a_n$ is obtained by letting b_j be the number of elements to the *left* of j that are *greater than* j. In other words, b_j is the number of inversions whose second component is j. For example, the permutation

$$5, 9, 1, 8, 2, 6, 4, 7, 3$$

has the following inversion table

$$2, 3, 6, 4, 0, 2, 2, 1, 0$$

since 5 and 9 are to the left of 1; 5, 9, 8 are to the left of 2; *etc.*, 20 inversions in all. Thus inversion table is a measure of the randomness of a permutation. Perhaps the most important fact about inversions is Marshall Hall's observation that *an inversion table uniquely determines the corresponding permutation.*

Write a program to read in a series of records, each containing either a permutation or an inversion table. If the input record is a permutation, then your program should output its associated inversion table. You must also check the validity of an input permutation. If the permutation is invalid, output "Invalid permutation." If the input record is an inversion table, then your program should output its associated permutation. It is possible that an inversion table is invalid, meaning that no permutations can correspond to the specified inversion table. In that case, output "Invalid inversion table."

Input

When the program starts, it should prompt the user to enter the name of the input file with the display string "Enter input file name: ". Each line in the input file consists of a sequence of positive integers separated by commas. Immediately following the last integer in the sequence is a blank space followed by either the letter "`P`" or the letter "`I`." The letter "`P`" indicates that the sequence of numbers is a permutation; while the letter "`I`" indicates an inversion table. You can assume that the number of integers for each permutation, n, *is less than or equal to* 20.

Output

For each input line, your program should print the type of input, followed by a colon, followed by the sequence of numbers, separated by commas. On the next line, if the input is valid, your program should output the type of the converted sequence, followed by a colon, followed by the corresponding sequence of numbers. Otherwise, print an appropriate error message. Outputs for each input line should be separated by exactly one blank line.

Example

Input file:

```
5,9,1,8,2,6,4,7,3 P
2,3,6,4,0,2,2,1,0 I
1,3,4,5 P
```

Output on screen:

```
Permutation: 5,9,1,8,2,6,4,7,3
Inversion table: 2,3,6,4,0,2,2,1,0

Inversion table: 2,3,6,4,0,2,2,1,0
Permutation: 5,9,1,8,2,6,4,7,3

Permutation: 1,3,4,5
Invalid permutation.
```

3 Common Stored-Value Ticket

A Common Stored-Value Ticket (CSVT) is a stored-value ticket that can be used in train, underground railway and Kowloon buses in Hong Kong. It is a thin plastic card with a short magnetic tape on it to store the current value of the ticket. Every time when the ticket is used, it is inserted into a computer-controlled gate and the computer updates the stored value of the ticket by deducting the correct fare from it. However, the issuing companies of the ticket adopt the 'last-ride-bonus' policy, that is, no matter how little stored value is left on a ticket, a passenger can always use it for his last ride. Therefore, if a passenger uses a ticket with only \$0.10 of stored value on it for a ride of \$5, he can effectively save \$4.90. When there is no value left, the ticket will be retained by the computer and eventually returned to the issuing companies.

An interesting question is that, given a ticket with initial stored values X and a sequence of n trips that cost $s_1, s_2, \ldots, s_n$, respectively, how the ticket should be used in these trips in order to save as much as possible. Suppose the ticket is used for trips $i_1, i_2, \ldots, i_m$, $m \leq n$, then the total saving is *defined* to be

$$\max(0, (s_{i_1} + s_{i_2} + \cdots + s_{i_m}) - X)$$

Write a program that, given X and $s_1, s_2, \ldots, s_n$, computes the maximum possible total saving. You should also indicate in which trip(s) the ticket is to be used in order to achieve this maximum saving.

Input

When the program starts, it should prompt the user to enter the name of the input file with the display string "Enter input file name: ". The file contains one or more ticket descriptions, and two adjacent ticket descriptions are separated by a line with the character '`#`' only. For each ticket description, the first line contains a positive integer that is the initial stored value X, in cents, of the ticket. You may assume that the initial stored value is in the range of \$1 to \$50 inclusive. On the second line are n integers showing the fares of the n trips in which the tickets are to be used, where the ith integer is the fare, in cents, of the ith trip. You may assume that all these n integers are in the range of \$0.1 to \$10 inclusive. Note that the $(i+1)$st trip must be made after the ith trip. That is, you *cannot* use the ticket for the $(i+1)$st trip *and then* the ith one.

Output

For the N-th ticket description, you should first output a line:

```
Ticket description N:
```

And then, if there are l different ways to achieve the maximum saving, the output should consist of $l+1$ lins. On the first line is a single integer that shows the maximum saving obtainable. On each of the following l lines, you should show one possible way to achieve this saving. For example, if a way to achieve the maximum saving is to take the trips $i_1, i_2, \ldots, i_m$, $m \leq n$ (note that $i_1 < i_2 < \ldots < i_m$), then you should print all these m numbers on one line.

The numbers should be separated by a single space. Note that you are required to print the last l lines in sorted lexicographical order. The outputs of two adjacent descriptions are separated by one blank line.

Example

Input file:

```
100
50 50 50 100
#
100
50 25 24 50 100 25
#
```

Output on screen:

```
Ticket description 1:
 50
 1 4
 2 4
 3 4

Ticket description 2:
 99
 1 2 3 5
 2 3 4 5
```

4 Numeric Puzzle

A numeric puzzle is an $m \times n$ matrix. Each element is either a blank or an uppercase letter chosen from 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', and 'J'. The following is a 3×4 numeric puzzle:

```
A  D
B C
B BA
```

In a numeric puzzle, each of the 10 letters 'A' to 'J' stands for a *different* digit. Your task is to find the mapping from the characters to the digits so that all the digits in each row and column add up to the *same* number N. If there is more than one mapping that satisfies this requirement, you should find all the mappings that produce the largest N. For example, for the puzzle above, the largest N is 13 (when A=5, B=4, C=9 and D=8).

Input

When the program starts, it should prompt the user to enter the name of the input file with the display string "Enter input filename: ". The input file contains one or more puzzle descriptions. Two adjacent descriptions are separated by a blank line. For each description, the first line contains two positive integers m and n which define the dimension of the puzzle. Each of the next m line contains n characters which are either space (' ') or a letter chosen from 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', and 'J'. You can assume that $2 \leq m, n \leq 20$.

Output

Your program should find all the solutions of a numeric puzzle within 1 minute, and output the solutions in the format as follows. For the M-th puzzle, it should first print out a line:

```
Puzzle M:
```

Suppose the there are k possible ways to assign values to the letters to produce the maximum N. If $k = 0$, then your program should print the message: `NO ANSWER`. Otherwise, the output should consists of $k + 2$ lines. On the first line are the names of the letters (in ASCII order) separated by single spaces. On the second to the $k+1$st lines are the corresponding mapping values of the solutions, separated by single spaces. On the $k + 2$nd line the value of N is printed. The outputs of two adjacent puzzles are separated by one blank line.

Example

`Input file:`

```
3 4
A  D
B C
B BA
```

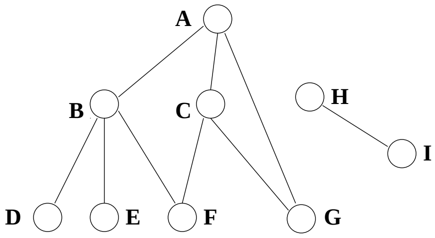
```
3 4
AJ D
BBC
B BA
```

```
Output on screen:

  Puzzle 1:
   A B C D
   5 4 9 8
   The largest sum = 13

  Puzzle 2:
   NO ANSWER
```

5 Path Checking In Graphs

Graphs have been widely used in all kinds of computational methods to help solving problems. The nodes in a graph usually denote problem states, situations, or instances while the arcs connecting the nodes usually represent some kinds of relations or operations that mark the transformation amongst the nodes. A path in a graph is a sequence of nodes (or sometime denoted as arcs) that are inter-connected by a series of arcs where the terminating node of an arc is the starting node of the next arc.

An example of a finite graph with nodes labelled with capital English alphabets is shown below:



In this figure, some of the valid paths are: `ABE`, `FCAG`, `ACGABFC`, `HI`, ..., etc., and some of the invalid paths are: `AH`, `DBC`, ..., etc.

You are required to write a program that accepts a number of graphs and paths specified in an input file, and check whether the paths are valid in the corresponding graphs.

Input

When the program starts, it should prompt the user to enter the name of the input file with the display string "Enter input file name: ". In the input file, different graphs with the paths to be checked are separated into groups by "?". The first line of each group denotes the graph, and the other lines of the group are path specifications.

In the representation of graphs for this problem, each node label, which is a capital English alphabet, is followed by a single digit numerical value to indicate the number of subsequent nodes that are connected to it. In cases where there are preceding nodes such as `B3` that is preceded by `A3` in the graph example above, the three nodes that are connected to node `B` will only be given after all the nodes connected by node `A` have been listed. Sometimes, not all connectivities of a node are given in a single description and repetitions may occur. Note that spaces may also appear amongst the graph descriptors. Therefore, the graph example shown above will have many possible input forms and two of them are:

```
A3B3C 2G1 F0E0D0 F0G0 C0 A1C0   H1I0
```

and

```
A2B3C1D0E0F0F0C1G0I1H0A1G0G1A0A0A0B0
```

Output

For the N-th group, it should first display a line with `Graph N:` and the graph specification with no space characters in it. Then, it should display messages on screen if the specified paths are valid or not in the correpsonding graph. If the input specifications contain some error, you should display an appropriate error message. See the example output below for the detail of the messages.

Example

Input file:

```
A3B3C2G0F0E0D0F0 G0H1I0
ABE
ACGABFC
AH
DBC
AxY
?
A2s0 T0
sAT
?
B1A1
AB
```

Note that this file contains three graphs: the first one is the example with the corresponding paths shown above and a path containing an invalid character "x"; the second one contains an invalid node label character "s"; the third one is an incomplete description of a graph.

Output on screen:

```
Graph 1: A3B3C2G0F0E0D0F0G0H1I0
  Path ABE is a valid path.

  Path ACGABFC is a valid path.

  Path AH is NOT a valid path.

  Path DBC is NOT a valid path.

  Invalid character: x; path AxY is discarded.

Graph 2: A2s0T0
  Invalid character: s

Graph 3: B1A1
  Graph description is incomplete.
```

6 List Processing

List a data structure widely used in computing. A list is commonly represented as follows:

$$(\textit{list-element}\ [\textit{list-element}])$$

where a list element can either be an atom or a list. For examples, in the list `(a b c)`, `a`, `b` and `c` are atoms, and in the list `(a b c (d e))`, `a`-`e` are atoms and the forth element of the list is itself a list.

In list processing the function head (or h for short) accepts a list as input and returns the first element of the list. For examples,

head of `(a b c)` = `a`
head of `((a b) c)` = `(a b)`
head of `((a b c))` = `(a b c)`
head of `a` = `ERROR` because a is not a list

Similarly, the function tail (or t for short) accepts a list as input and returns a list with all elements of the input list excluding its head, for examples,

tail of `(a b c)` = `(b c)`
tail of `((a b) c)` = `(c)`
tail of `((a b c))` = `()`, *i.e.*, an empty list
tail of `a` = `ERROR` becasue a is not a list

Furthermore, head and tail functions can be nested, for examples,

head of tail of `(a b c)` = `b`
tail of head of `(a b c)` = `ERROR` because the input to tail is not a list
head of tail of tail of `(a b c)` = `()`, *i.e.*, an empty list

you are required to write a program which performs head and tail list operations, and to give the final results of the operations.

Input

When the program starts, it should prompt the user to enter the name of the input file with the display string "Enter input file name: ". The input file contains a number of list operation descriptions. The format of each description is specified as follows:

- The first line specifies the list operation to be executed. It consists of a string of h and t, representing the head and tail operations, respectively. The order of execution should be from right to left.
- The next line is the input list to the list operation specified in preceding line. An atom is an alphabet between a-z and the same alphabet cannot occur twice in the list. Furthermore, the list can contain arbitrary number of levels of nesting.
- The next line is a blank line (optional). This separates the next list operation.

Output

The results of the list operations specified in the input file should be printed on the screen. The display contains multiple lines, each separated by a blank line. Each lines shows the result of the list operation for the corresponding line of the input file. Each output is numbered.

Your program should detect all possible errors and print out appropriate error messages. In the case of operation error, you should also point out in the error message which h or t operation causes the error (*e.g.*, sample error message: "the 3 operation which is a tail operation fails because the input is not a list").

Example

Input file:

```
h
(a b c d e)

ht
(a b c d e)

thh
(((a b c d e)))

th
(a b c d e)
t
(a
```

Output on screen:

```
1. a

2. b

3. (b c d e)

4. ERROR: the 2 operation which is a tail operation fails
   because the input to it is not a list.

5. ERROR: unbalanced bracket in the input list.
```

***** End of All Problems *****