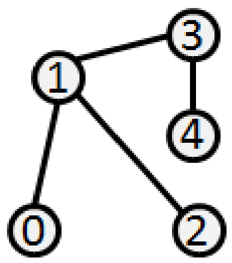
Q1: Take That Train…



A subway system is defined by a set of stations and the train tracks connecting the stations. Each station has a number and is directly connected to at least one other station in the system. A *path* is a sequence of *n* stations s_0, s_1, …, s_{n-1} starting at s_0 and ending at s_{n-1} where each neighboring pair of stations s_i, s_{i+1} in the sequence is directly connected. In the system, there will always be a *shortest path* of unique stations between any two stations, s_{start} and s_{end}, that only visits the stations on the path once.

Given the description of a subway system, your goal is to find the shortest path when traveling between any two different stations, s_{start} and s_{end}. For example, in the above example, the shortest path from station 0 to station 4 is defined by the sequence of stations `0,1,3,4`. Note that `0,1,2,1,3,4` is not the shortest path from station 0 to station 4 because this path visits station 1 twice.

Input

The input will be an integer *N* on a line by itself that determines the number of stations in the subway system. You can assume *N* < 20 and stations are numbered using the consecutive numbers 0 to *N–1*. The remaining *N–1* lines will contain two station numbers "*i j*" separated by a space, representing that station s_i directly connects to s_j.

The final line of input will contain a query string of the form "*i j*" where *i* represents the number of the starting station and *j* represents the number of the ending station.

Output

The output will consist of *K* lines of output, representing the sequence of stations in the shortest path through the subway system from the starting station to the ending station. Each line shall contain the station number on the line by itself. The first output line contains the number of the starting station; the last output line contains the number of the ending station.

Sample Input and Output

Input	Output
5	0
0 1	1
1 2	3
3 1	4
3 4	
0 4	

Input	Output
5	2
0 1	1
1 2	3
1 3	
3 4	
2 3	

Input	Output
6	0
0 1	1
1 2	2
3 2	3
4 3	4
5 4	5
0 5	

Input	Output
6	5
0 1	3
2 3	1
3 1	0
4 1	
5 3	
5 0	

Input	Output
11	1
0 2	2
1 2	4
2 3	5
4 2	8
4 5	9
5 6	
6 7	
5 8	
9 8	
10 8	
1 9	

Q2: Count Those Days…

Write a monthly calendar formatter that takes in three values:

- A name for the month
- A starting day of the week for the first day in the month (1=Sunday, 2 = Monday, …, 7=Saturday)
- The total number of days in the month to print

```
March
 S  M  T  W  T  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

Based on this information, you can print out a formatted monthly calendar as shown above.

Input

The first line of input will be a string on a line by itself. This string (of no more than 20 characters) represents the name of the month being printed. The second line contains an integer *D* representing the day of the week on which the month starts. The third line contains the number of days to print, *N*. You can assume that $1 \leq D \leq 7$ and $28 \leq N \leq 31$.

Output

The first line of output must be the name of the month on a line by itself with no leading or trailing spaces. The second output line must contain the header string "` S  M  T  W  T  F  S`" on a line by itself, which starts with a leading space and has TWO space characters between each of the header characters. The remaining output lines contain the days of the month printed out in order, line by line. There are no trailing spaces on any of the output lines (even when the month ends in mid-week).

Sample Input and Output

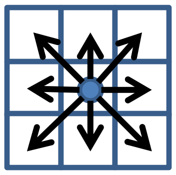
Input	Output
February 6 28	February S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
January 7 31	January S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
September 4 30	September S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Input	Output
February 1 28	February S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

Q3: Sweep Those Mines…

	A	B	C	D
1	M	1	1	M
2	1	1	1	1
3	.	.	1	1
4	1	1	3	M
5	1	M	3	M
6	1	1	2	1

In the classic Minesweeper game, a two-dimensional MxN grid contains M*N cells. The M columns are labeled by letters (A, B, C, …) while the N rows are labeled by numbers (1, 2, 3, …).



Each cell has up to eight direct neighboring cells, as shown on the left. Corner cells in the grid have three neighbors, edge cells have five. If a cell contains a mine, it is labeled "M".

Empty cells are labeled "." (period) if no direct neighboring cell has a mine; otherwise an empty cell is labeled by a number from 1 to 8, reflecting the number of direct neighboring cells that contain a mine.

A grid is initialized with *K* initial mines – in the example above, there are mines in A1, D1, D4, B5, and D5; these cells are all labeled with an "M". Cell B3 is labeled "." because none of its 8 direct neighbors has a cell. Cell C4 is labeled "3" because of the mines in cells B5, D4 and D5. The program must output a representation of each cell corresponding to the full grid.

Input

The first line of input will be two integers separated by a space representing the values of *M* (columns) and *N* (rows). The second line of input contains a single integer *K* representing the number of mines in the grid. The next *K* lines of input will each contain an UPPER CASE LETTER and a number, separated by a space, representing the column and row of a mine in the grid. You can assume that $1 < M < 10$, $1 < N < 10$, and $0 < K < 20$. You can assume that each of the *K* lines represents a valid grid location.

Output

Your output will contain *N* rows of *M* characters each on a line by itself. Each cell containing a mine is represented by an "M" character; each empty cell whose direct neighbors are all empty is represented by a "." (period) character, and other empty cells are labeled by a digit from 1 to 8 representing the number of direct neighboring cells that contain a mine.

Sample Input and Output

Input	Output
4 6	M11M
5	1111
A 1	..11
D 1	113M
D 4	1M3M
B 5	1121
D 5	
2 2	M1
1	11
A 1	

Input	Output
8 2	M2....2M
4	M2....2M
A 1	
A 2	
H 1	
H 2	
4 4	....
4	....
A 4	2332
B 4	MMMM
C 4	
D 4	

Input	Output
8 8	..1M2221
10	22212MM1
A 3	MM1.1221
B 3	33211...
G 2	M12M2.11
H 7	112M2.2M
A 5	..111.2M
D 1	11
D 6	
D 5	
F 2	
H 6	

Q4: Order Those Numbers…

Given a set S_x of unique positive integers, represent that set using a *range notation* R_x of shortest possible size. The set notation R_x for a set S_x is either:

- A string "N", which represents the set containing just the integer {N}
- A string of the form "M–N" where M < N, which represents the set of consecutive integers {M, M+1, M+2, …, N–1, N}
- A string of the form "R_1,R_2,…,R_n", which represents the union of n individual sets S_1, S_2, …, S_n, where the difference between the lowest integer in S_{i+1} and the highest integer in S_i is at least two. Each R_i must be a valid range notation.

The best way to explain these rules is by example. On the right are some sample sets with their corresponding range notation. Note that the only characters that appear in the range notation are: the digits "0" through "9", a "–" (dash), and "," (comma).

Sample Set	Range Notation
{ 8 }	8
{ 2, 6, 3, 5, 4}	2–6
{ 2, 4, 3, 6, 7, 10, 9, 8 }	2–4,6–10
{ 4, 1, 3, 7, 5 }	1,3–5,7
{ 99, 13, 15}	13,15,99
{ 1, 2 }	1–2
{ 103, 972, 18, 19 }	18–19,103,972

To ensure that a range notation is the shortest possible size, you must collapse neighboring ranges. Thus, the range "1–5,6,7–10" must be represented instead as "1–10".

There is a special rule to which you must adhere. When a set contains just two consecutive integers, the range notation must use the dash character rather than a comma. For example, for {1, 2} the output must be "1–2" instead of "1,2"; by implication, for {1, 2, 4, 5}, the output must be "1–2, 4–5".

Input

There will be a single line of input consisting of no more than 140 characters. This line will contain a number of positive integers (in any order) separated by a single space between each integer. Each integer N will be a number $0 < N < 1000$.

Output

Your output will consist of a single line representing the smallest range notation for the input set. This string will only contain the digits "0" through "9", dashes ("–") and commas (","). **There must be no leading or trailing commas in the output.**

Sample Input and Output

Input	Output
`1 6 8 2 3 4 7 13 19 14`	`1-4,6-8,13-14,19`
`1 3 8 2 4 5 9 11 10 13 88 86 76 78 87 59 143 12`	`1-5,8-13,59,76,78,86-88,143`
`99 13 15`	`13,15,99`
`1 2 4 5`	`1-2,4-5`
`1 3 4 6 7 9`	`1,3-4,6-7,9`
`91`	`91`
`92 93`	`92-93`

Q5: Rate That Hand…

A poker hand of five playing cards drawn from a standard 52-card deck has a value, based on the table below. Each card has a rank (Ace, 2 – 9, 10, Jack, Queen, or King) and a suit (♣, ♦, ♥, or ♠). A card is represented as a 2- or 3-character string with a rank of (A, 2–9, 10, J, Q, or K) and a suit of (C, D, H, or S). Thus, "AS" represents the Ace of Spades, while "10C" represents the Ten of Clubs.

Power	Poker Hand Value	Description	Examples
10	ROYAL FLUSH	The 10, Jack, Queen, King and Ace of the same suit. **This hand is the strongest of all poker hands.**	**A♦ K♦ Q♦ J♦ 10♦**
9	STRAIGHT FLUSH	Five cards in sequence. Aces can play low or high. All cards belong to the same suit. Note that Ace may not "wrap around" and play both high and low.	**Q♣ J♣ 10♣ 9♣ 8♣ or 5♦ 4♦ 3♦ 2♦ A♦**
8	FOUR OF A KIND	Contains all four cards of one rank and any other (unmatched) card	**9♣ 9♠ 9♦ 9♥ J♥**
7	FULL HOUSE	Contains three matching cards of one rank and two matching cards of another rank	**7♠ 7♥ 7♦ 4♠ 4♣**
6	FLUSH	All five cards are of the same suit, but not in sequence	**Q♦ 9♦ 7♦ 4♦ 3♦**
5	STRAIGHT	Five cards in sequence using at least two different suits. Aces can play low or high. Note that Ace may not "wrap around" and play both high and low.	**A♣ K♣ Q♦ J♠ 10♠ or 5♠ 4♦ 3♦ 2♠ A♥**
4	THREE OF A KIND	Contains three matching cards of the same rank and two unmatched cards of other ranks	**Q♠ Q♥ Q♦ 7♠ 4♣**
3	TWO PAIR	Contains two cards of the same rank, plus two cards of another rank (that match each other but not the first pair), plus any card not of either rank	**10♠ 10♣ 8♥ 8♣ A♦**
2	ONE PAIR	Contains two cards of one rank, plus three cards which are neither of that rank nor the same as each other.	**4♥ 4♠ K♠ 10♦ 5♠**
1	HIGH CARD	Five cards not meeting any of the above requirements. **This hand is the weakest of all poker hands.**	**K♥ J♥ 8♣ 7♦ 4♠**

Input

There will be a single line of input consisting of a hand of five card representations, separated by a single space between each card. There will be no leading or trailing spaces in the input. All cards will be valid representations with no duplicate cards in a hand.

Output

Your output will consist of a single line of text by itself, consisting of the strongest poker hand value (in CAPITAL LETTERS) as shown exactly in the above table.

Sample Input and Output

Input	Output
9C 9S JH 9D 9H	FOUR OF A KIND
QD 9D 7D 4D 3D	FLUSH
10S 8H 10C 8C AD	TWO PAIR
AD KD 10D QD JD	ROYAL FLUSH
4J 10D 4S KS 5S	ONE PAIR

Input	Output
QS 4C QH 7S QD	THREE OF A KIND
7S 4C 7U 7D 4S	FULL HOUSE
AC JS KC QD 10S	STRAIGHT
5D 3D 2D AD 4D	STRAIGHT FLUSH
KC 3D 2H AS JH	HIGH CARD

Q6: Search That Genome…

A genome is a nucleotide string of four symbols – A, C, G and T – representing the bases of DNA. For example "TACGTGCA" is a sample sequence. What if you wanted to determine whether a given sequence contained either "ACGTG" or "ACGTA" consecutively (the only difference being the last base)? In this case, your *search string* would be "ACGTR" because – using the table on the right – character "R" can represent either a "G" or an "A"; using this search string, a match is found in the sample sequence as shown below.

Search Symbol	Potential Matches
G	G
A	A
T	T
C	C
R	G or A
Y	T or C
M	A or C
K	G or T
S	G or C
W	A or T

```
DNA sequence:   AAAATACGTGCATCGACGTCT
Search String:       ACGTR
```

Write a program that takes a sample DNA sequence and a search string, and determines the first subsequence in the sample (i.e., from the left) that matches the search string. In the above search, the program identifies that subsequence "ACGTG" is a match, and reports it. When there is no match, you should report "NO MATCH".

Input

The first line of input contains a single genome string of no more than 140 "A", "C", "G", and "T" characters on the line by itself. The second line contains a search string (of no more than 10 characters) on the line by itself containing only characters in the first column of the above table.

Output

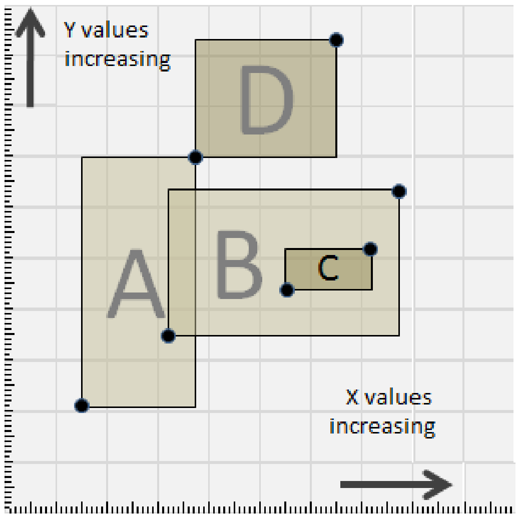
If there is a match, your output will consist of the matched subsequence within the input genome string on a line by itself; if there is no match, then the only line of input will be the string "NO MATCH".

Sample Input and Output

Input	Output
TACGTGCATCGACGTCT ACGTR	ACGTG
TACGTGCATCGACGTCT SSRSKW	CGACGT
TACGTGCATCGACGTCT MKSAT	NO MATCH
AAAACCCC RMWASMYC	AAAACCCC
AAAACCCC MW	AA
ACGTCGAACGTA RYSTR	ACGTA
ACTGA AYKGWR	NO MATCH

Q7: Cover That Area…

You are given a set of *N* rectangles drawn on the Cartesian plane. If the rectangles do not intersect each other, then the total area is simply the sum of the areas of the *N* rectangles. However, your job is to compute the total area covered by the rectangles even when they intersect each other.



Each rectangle is defined by four positive decimal values – x1, y1, x2, y2 – where (x1, y1) is the lower-left-hand corner and (x2, y2) is the upper right corner. Each decimal value is accurate to one digit of precision. In the example on the right, there are four rectangles:

- (1.4,2.1,3.8,7.0) – The vertical rectangle **A**
- (3.2,3.5,7.8,6.3) – The large horizontal rectangle **B**
- (5.5,4.4,7.1,5.1) – The small rectangle **C** wholly contained by the rectangle **B**
- (3.8,7.0,6.6,9.3) – The rectangle **D**

The total area covered by these rectangles is 29.40 units because of the overlapping rectangles. Note that rectangles **A** and **D** do not overlap because they only share a corner point in common.

Input

The first line of input will be an integer *N* ($0 < N \leq 6$) that determines the number of rectangles in the input set. The next *N* lines of input will each contain four positive decimal values "x1 y1 x2 y2" separated by a space representing the lower-left corner (x1, y1) and the upper right corner (x2, y2) of a rectangle in the set. Each decimal value $0 < V < 10.0$ in the input is precise to one digit, and will always include one digit after the decimal point. Thus the smallest allowed value is 0.1 and the highest value is 9.9.

You can assume x1 < x2 and y1 < y2 for each rectangle in the input set.

Output

The output consists of a decimal value accurate to two digits of precision on a line by itself representing the total area of covered by the rectangles. There must be a leading zero for areas smaller than 1.00 and you must always output two digits after the decimal point.

Sample Input and Output

Input	Output

2
1.1 2.8 2.2 7.5
6.2 3.6 9.8 8.0
	21.01

4
1.4 2.1 3.8 7.0
3.2 3.5 7.8 6.3
5.5 4.4 7.1 5.1
3.8 7.0 6.6 9.3
	29.40

Input	Output
1 1.0 1.0 1.1 1.1	0.01

2
1.0 1.0 3.0 3.0
3.0 3.0 7.0 7.0
	20.00

Input	Output

6
1.0 1.0 3.0 8.0
1.0 1.0 5.0 3.0
5.0 3.0 6.0 9.0
6.0 2.0 9.5 3.0
7.0 1.0 9.0 6.0
8.0 4.0 9.5 9.0
	41.00

Q8: Add Those Letters…

```
 SEND          9567
+MORE         +1085
-----          ----
MONEY         10652
```

In an *alphametic* puzzle, a set of words is written down in the form of "long hand" addition. The last word represents the sum and all previous words represent the addends being added together.

0	1	2	3	4	5	6	7	8	9
O	M	Y	.	.	E	N	D	R	S

Each letter in the puzzle can be replaced by a unique decimal digit '0' to '9' (as determined by a mapping) such that the result is a valid sum. The following rules always apply:

1. The mapping of letters to numbers is one-to-one. That is, the same letter always stands for the same digit, and the same digit is always represented by the same letter
2. The digit zero is not allowed to appear as the left-most digit in any of the addends or the sum.

A mapping consists of ten characters read from left to right, representing the digits '0' to '9'. Note how the 'M' character is replaced with the '1' digit everywhere in the computation (and that is the only letter that can represent a '1'). If a digit is unused, its mapping entry is a "." (period).

You are to write a program that validates a *letter-to-digit mapping* for an *alphametic* puzzle. A mapping is valid if the result of applying the mapping produces a valid equation for addition. Above, the proposed mapping below the puzzle results in a correct arithmetic statement, as you can verify.

Input

The first line of input will be an integer *N* ($2 < N \leq 6$) that determines the number of words in the puzzle. The next *N* lines will each contain a word (of no more than 10 characters) of capital letters on a line by itself. The final line will contain ten characters representing a proposed mapping between ten digits '0' to '9' and potentially ten capital letters; if a digit is not used, then a "." (period) appears instead.

Output

The output contains *N+1* lines. The first *N* lines represent the numerical equivalents of the *N* input words, replacing each letter with its mapped digit. The final line is either "VALID" or "INVALID" depending on whether the addition is correct. The output contains no leading or trailing spaces.

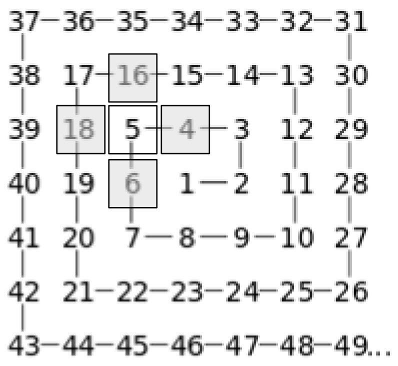
Sample Input and Output

Input	Output
3	9567
SEND	1085
MORE	10652
MONEY	VALID
OMY..ENDRS	
3	734
TWO	734
TWO	1468
FOUR	VALID
.F.WO.UTR.	

Input	Output
4	321
ONE	453
TWO	7389
FOUR	7601
FIVE	INVALID
VENOTWIFUR	
4	285
ITS	438
NOT	8978
THAT	9701
HARD	VALID
RDIONS.ATH	

Input	Output
3	96233
CROSS	62513
ROADS	158746
DANGER	VALID
.DOSEARGNC	
4	61
DO	23
RE	43
ME	510
SOL	INVALID
LOREMSD...	

Q9: Follow Those Numbers…



In 1963, the mathematician Stanislaw Ulam constructed the "Ulam Spiral" as shown on the right. Starting with 1, he created the pattern by spiraling counter-clockwise to add consecutive integers. This structure reveals the apparent tendency of certain quadratic polynomials to generate unusually large numbers of primes.

As you inspect this spiral, observe that every number has two horizontal and two vertical *neighbors*. Define the *neighbors* of a number in the spiral as being these four values. For example, the number 5 has four neighbors – 4, 6, 16 and 18 – as shown in the above graphic. Every number X > 1 clearly has both X – 1 and X + 1 as neighbors, by definition of how the spiral is constructed. The other two neighbors, however, seem to follow no immediate pattern.

Your program must determine whether positive integers X and Y are neighbors in the spiral.

Input

The only line of input will contain two integers X and Y separated by a space. You can assume that 0 < X < 500 and 0 < Y < 500 and that X < Y.

Output

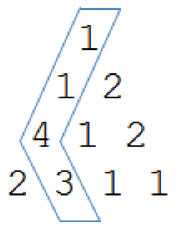
The output consists of the string "NEIGHBOR" on a line by itself if X and Y are neighbors in the Ulam Spiral; if they are not neighbors, then output "STRANGER" on a line by itself.

Sample Input and Output

Input	Output
3 14	NEIGHBOR
5 6	NEIGHBOR
383 464	STRANGER
6 40	STRANGER
310 385	NEIGHBOR

Input	Output
10 27	NEIGHBOR
10 26	STRANGER
324 399	NEIGHBOR
1 8	NEIGHBOR
7 21	STRANGER

Q10: Count Those Paths



You are given a triangle of positive integers, such as appears on the right. The triangle is composed of N rows. The first row contains a single number; the second row contains two numbers; the third row contains three numbers, and so on. Define a *path* in this triangle as starting from the number on the first row and ending at a number on the N^{th} row. The path extends downward by either moving to the number immediately below and to the left, or below and to the right. In the example highlighted on the right, the path starts at 1 on the first row and extends to the 3 on the N^{th} row. The sum of the numbers in this path is 9; no other path in the above triangle has a greater sum, thus 9 is the maximal sum in this triangle. Given a triangle of numbers with N rows, write a program to compute the maximal sum of any such path. Note that each path contains N numbers.

Input

The first line of input is a positive integer N ($1 < n \leq 8$) on a line by itself representing the number of rows in the number triangle. The next N lines contain the positive integers of each successive row of the triangle, with a space between each integer. You can assume that the triangle is properly encoded in the input.

Output

Output consists of a single positive integer on a line by itself representing the greatest sum of all such paths in the triangle.

Sample Input and Output

Input	Output

3
1
2 1
1 2 3
	5

4
1
1 2
4 1 2
2 3 1 1
	9

6
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
	23

6
1
1 1
1 1 2
3 3 1 1
1 1 4 4 6
10 10 5 5 1
	17

Input	Output

5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
	15

5
1
1 1
3 1 3
4 1 2 4
1 1 12 1 1
	19

7
1
1 1
1 1 1
1 1 1 1
1 1 1 1 1
2 1 2 1 4 1
1 5 1 3 1 3 1
	12