Computer Science Competition Region 2024 Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.
2. All problems have a value of 60 points.
3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.
5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Audrey
Problem 2	Bob
Problem 3	Casandra
Problem 4	Clarabelle
Problem 5	Harmony
Problem 6	Jared
Problem 7	Jimothy
Problem 8	Paola
Problem 9	Riley
Problem 10	Sasha
Problem 11	Svetlana
Problem 12	Wesley

1. Audrey

Program Name: Audrey.java **Input File: None**

UIL CompSci has decided it is time to celebrate. This year is the 33rd year of the UIL CompSci Event. In honor of the event, there will be a parade, a chili cook-off, a formal dance, a picnic, and a karaoke contest. Of course, there will be merchandise for sale to cover all the costs. There will be hats, t-shirts, hoodies, and special socks for sale.

Audrey, who is in charge of organizing the event, needs your help. With all of the merchandise being sold, there needs to be a special logo for the celebration. She wants you to write a program to display the logo shown below.

Do this for Audrey, and do this for UIL CompSci.

Input: None

Output: Output the picture you see below.

Sample input:
None

Sample output:
```
U.........U.........U
.I........I........I.
..L.......L.......L..
...C......C......C...
....O.....O.....O....
.....M....M....M.....
......P...P...P......
.......S..S..S.......
........C.C.C........
.........III.........
UILCOMPSCI*ICSPMOCLIU
.........III.........
........C.C.C........
.......S..S..S.......
......P...P...P......
.....M....M....M.....
....O.....O.....O....
...C......C......C...
..L.......L.......L..
.I........I........I.
U.........U.........U
```

2. Bob

Program Name: Bob.java **Input File: bob.dat**

Little Bob is a fifth grader who is learning about order of operations in arithmetic. His teacher is emphasizing the fact that one must do multiplication before addition.

Little Bob's class has just completed a unit on putting numbers in order. Little Bob's older brother Big Bob wants to help his sibling and at the same time practice his Java programming. So, he has given his brother a challenge to do. While the younger Bob is solving the problem by hand, Big Bob will write a program to check the work.

Read in five integers, no one number is equal to any of the other four. Sort the integers from largest to smallest creating a list we will call A, B, C, D, and E where A is the largest value and E is the smallest. Plug the numbers into the following math expression and find the value.

$$A \times B + C + D \times E$$

Go ahead and write the program, for all the Bobs out there. They are counting on you.

Input: First line contains a single integer **T** representing the number of test cases that follow with **T** ≤ 10. Each test case is a line of exactly 5 integers separated by whitespace, each with a value in range [0, 100]. There will be no repeats on a line – that is, the five values will all be different.

Output: For each test case, output 1 integer representing: **A x B + C + D x E** where the five integers are sorted from largest to smallest.

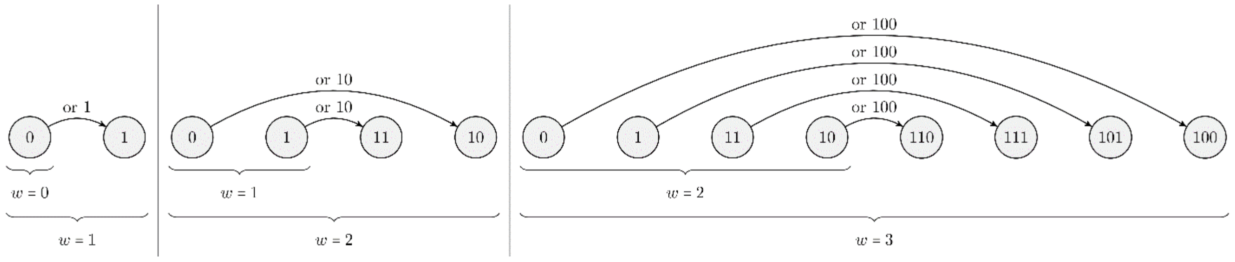
Sample input:
```
5
1 2 3 4 5
11 43 9 22 17
4 8 5 7 6
15 51 20 63 12
3 8 0 9 2
```

Sample output:
```
25
1062
82
3413
75
```

3. Casandra

Program Name: Casandra.java **Input File: casandra.dat**

Casandra's friend, Leah, recently discovered a way to efficiently determine the i^{th} Gray Code number. Leah observed that, for an arbitrary bit width w, that the Gray Code ordering for those 2^w numbers overlapped with the first 2^w numbers of the Gray Code ordering for width $w + 1$. The following is a visualization of this process starting from $w = 0$, to $w = 3$:



While this provides a nice visual understanding of this process, Casandra realized that the calculations shown above provides a decently slow process, as, to generate the i^{th} Gray Code, you must first generate the 0th through $i - 1^{\text{th}}$ Gray Codes. As a result, Casandra decided to investigate into methods of generating an arbitrary Gray Code number without the need to compute prior Gray Code numbers. In her research, Casandra found a process that allows her to further simplify this process using an XOR operation as follows, where g_i is the i^{th} Gray Code:

$$g_i = i \oplus (i \gg 1)$$

Having now discovered an efficient and independent way of generating Gray Code numbers, Casandra began to investigate some of the many applications of Gray Code numbers, one of which she found particularly useful: Karnaugh Maps. Karnaugh Maps can be used to efficiently and completely simplify Boolean expressions. For a given Boolean expression with V unique variables, first you must create a truth table for that Boolean expression, where you iterate over all 2^V permutations of true and false values. Suppose we had the Boolean expression:

$$Y = A \mid (B \,\&\, !C) \mid (C \,\&\, !B)$$

Then, the truth table for this expression would look like the following:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Next, a Karnaugh Map can be generated by simply re-arranging the values from the table above:

Y C \ AB	00	01	11	10
0	0	1	1	1
1	1	0	1	1

~ Casandra continued… ~

The process of re-arranging this table is as follows: First, given the number of unique variables, V, create a table of size $U \times D$, where $U = \left\lceil \frac{V}{2} \right\rceil$ ($V/2$ Rounded Up) and $D = \left\lfloor \frac{V}{2} \right\rfloor$ ($V/2$ Rounded Down). From there, the first U variables will be represented by the column headers, and the remaining D variables will be represented by the row headers (note that the variables in the truth table are sorted alphabetically, and thus, the order in the Karnaugh Map maintains that sorting). Next, generate the Gray Codes g_0 through g_{U-1}, and generate the Gray Codes g_0 through g_{D-1}. Take the Gray Codes g_0 through g_{U-1} and replace the column headers. Similarly, take the Gray Codes g_0 through g_{D-1} and replace the row headers. Lastly, fill in the table for values of Y corresponding to the values of the V unique variables from the original truth table. See below for how the Karnaugh Map above was derived.

Y	AB	g_0	...		g_{U-1}
C		00	01	11	10
g_0	0	$ABC = 000$ $\Rightarrow Y = 0$	$ABC = 010$ $\Rightarrow Y = 1$	$ABC = 110$ $\Rightarrow Y = 1$	$ABC = 100$ $\Rightarrow Y = 1$
g_{D-1}	1	$ABC = 001$ $\Rightarrow Y = 1$	$ABC = 011$ $\Rightarrow Y = 0$	$ABC = 111$ $\Rightarrow Y = 1$	$ABC = 101$ $\Rightarrow Y = 1$

Before learning how to use Karnaugh Maps to simplify Boolean expressions, Casandra wants to ensure that she understands the process of generating them perfectly. As a result, she decides to test her understanding by writing a program that, given a Boolean expression, generates an equivalent Karnaugh Map for that Boolean expression.

Input: The first line of input will consist of a single integer n ($1 \le n \le 50$) denoting the number of testcases to follow. The next n lines will each contain a single string denoting the Boolean expression that needs to be converted. Each of the n lines will consist solely of the characters ['A'-'Z'] (uppercase 'A' through uppercase 'Z'), '!' (the logical NOT operator), '|' (the logical OR operator), '&' (the logical AND operator), '^' (the logical XOR operator), '(' (the open parenthesis), and ')' (the close parenthesis). It is guaranteed that no two variables are directly adjacent to one another in the string, and are separated by, at least, 1 non-variable character. Additionally, it is guaranteed that if character c appears in the list of unique variables, then all characters starting from 'A' leading up until c also appear. Lastly, $2 \le V \le 10$, as to prevent issues where the Karnaugh Map matrix is only a vector, or grows too large.

Output: For each of Casandra's n requests, print out the $U \times D$ Karnaugh Map matrix of the Y-values, where each row is separated by a newline character, each column is separated with a single space, and each Karnaugh Map is followed by a newline containing the string "--------" (a string of eight hyphens).

Sample input:
```
4
A|(B&!C)|(C&!B)
A|B
A&(B|(C^!D))
!(A)|!(B)
```

Sample output:
```
0 1 1 1
1 0 1 1
--------
```
***Sample output continues next column*:**

***Sample output continued*:**
```
0 1
1 1s
--------
0 0 1 1
0 0 1 0
0 0 1 1
0 0 1 0
--------
1 1
1 0
--------
```

4. Clarabelle

Program Name: Clarabelle.java **Input File: clarabelle.dat**

Clarabelle Ono has always been disappointed with her first name. Her Mom and Dad wanted their youngest child to have a very fancy name, so they gave her that 10-letter name. Their older children, Asa, Nan, Ava, Eve, Bob, and Ana all had three-letter palindrome names, but not Clarabelle.

Clarabelle owns a guinea pig farm and she is determined to name every one of her animals with a three-letter palindrome like her family. She is getting desperate. She now is looking at every work she encounters to see if somewhere inside, a three-letter palindrome is hidden. Please help her by writing a program.

Read in a word written in all lowercase letters which contains [a,z]. The words you will test have lengths in the range of [3,12] letters. Your job is to print a list of three-letter palindromes that are hidden in the input word. These palindromes will consist of three consecutive letters that appear in the word.

You will list the palindromes in alphabetical order, making sure to list each palindrome only one time.

Input: First line contains a single integer **`T`** representing the number of test cases that follow with **`T`** ≤ 10. Each test case will consist of one word with at least 3 letters and not more than 12 letters. The words may not contain any other characters except for lowercase letters.

Output: For each test case, output an alphabetical list of three-letter palindromes using a single space to separate the words. If there are no palindromes, output **`NONE`**.

Sample input:
```
5
qwqtquqoqhqh
university
tttrrraaappp
bob
ututututututut
```

Sample output:
```
hqh qhq qoq qtq quq qwq
NONE
aaa ppp rrr ttt
bob
tut utu
```

5. Harmony

Program Name: Harmony.java **Input File: harmony.dat**

Harmony is fascinated with Web sites that display a password strength for new passwords. She has developed a new scoring algorithm of her own and needs help to code a solution. She found a list of most commonly used passwords for 2023 (https://nordpass.com/most-common-passwords-list/) and a string of letters from Wikipedia with the most frequently used to least frequently used letters based on words from the Concise Oxford English Dictionary. That list of letters, **`EARIOTNSLCUDPMHGBFYWKVXZJQ`**, identifies the letter **`E`** as the most often occurring letter in English words, with letter **`A`** next, and letter **`Q`** as the least often occurring letter.

The scoring rules of her algorithm are as follows:

- Match any of the top 100 common passwords, results in score of 0
- Length less than 8 characters results in score of 0
- Does not contain at least 3 of the 4 character categories: lowercase letters, uppercase letters, digits, and special characters, results in score of 0
- Passwords not excluded by the above criteria are scored based on character content
- Letters, regardless of case, found in the **first** half of the highest-to-lowest used letters list earn 1 point each
- Letters, regardless of case, found in the **last** half of the highest-to-lowest used letters list earn 3 points each
- Digits earn 3 points each
- Special characters earn 5 points each
- Using all 4 categories (uppercase, lowercase, digits, special characters) earns +5 bonus points
- Lengths greater than 10 characters earn +3 bonus points for each extra character
- Category change between sequential characters (e.g., letter to digit, uppercase to lowercase, etc.) earns +3 bonus points for each change. "UiL" as part of a password would earn +3 bonus points each for "Ui" and "iL" case changes and "u-n" would earn +3 bonus points each for "u-" and "-n".
- Sequentially repeated characters, of same case for letters, earn a −2 point penalty for each repetition. For example, "sss" as part of a password would earn points for the individual characters but get penalized −2 points each for the second and third sequential occurrence of the same character 's'; same would apply for repeated digits or special characters.

Strength ratings based on total scores:

- 0 points and lower → UNACCEPTABLE
- 1 – 20 points → WEAK
- 21 – 35 points → FAIR
- 36 – 50 points → GOOD
- 51 points and higher → STRONG

Examples:

- "`password`" and "`qwerty123`" score 0: in list of most common passwords
- "`pass1234`" scores 0: contains only 2 categories of characters: lowercase and digits
- "`fgh123$`" scores 0: length less than 8 characters
- "`Qwerty123`" scores 27 points: 21 character points with +6 bonus for "`Qw`" and "`y1`" category changes
- "`0circleS`" scores 16 points: 10 character points and +6 bonus for "`0c`" and "`eS`" category changes
- "`UiL+2024`" scores 37 points: 20 character points with +5 bonus points for using all 4 categories and +12 bonus points for category changes "`Ui`", "`iL`", "`L+`", and "`+2`"
- "`u20-IL_24`" scores 45 points: 25 character points with +5 bonus points for using all 4 categories and +15 bonus points for 5 category changes "`u2`", "`0-`", "`-I`", "`L_`", and "`_1`"
- "`Sxxxxxxx0`" and "`W0000000s`" both score 19 points: 25 character points with +6 bonus points for 2 category changes and −12 penalty points for 6 repeated '`x`' and '`0`' after the initial characters
- "`Pass12345!`" scores 36 points: 24 character points with +5 bonus points for using all 4 categories and +9 bonus points for 3 category changes and −2 penalty points for repeated '`s`'
- "`ABZYcdx[3-2-1-0]`" scores 105 points: 52 character points with +5 bonus points for using all 4 categories and +30 bonus points for 10 category changes and +18 bonus points for 6 extra characters

~ Harmony continued… ~

Input: First line contains 26 uppercase letters with no spacing, the highest-to-lowest use ranking of letters. Second line contains allowable special characters. The next ten lines contain the 100 most commonly used passwords in the U.S. for 2023 in decreasing order (https://nordpass.com/most-common-passwords-list/). There are ten passwords per line separated by commas. The remaining lines, maximum of 25, contain one password per line to be rated for strength. Passwords will not contain spaces or commas and will not exceed 20 characters in length.

Output: For each password, display one line of output with the password followed by a colon ':' followed by the calculated score and another colon ':' followed by the strength rating. There must be no additional spacing.

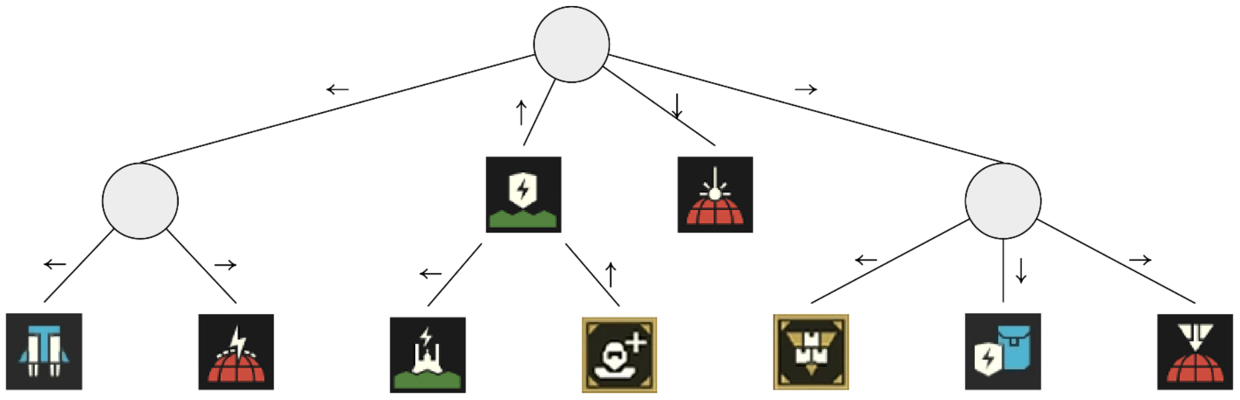
Sample input:
```
EARIOTNSLCUDPMHGBFYWKVXZJQ
`~!@#$%^&*()-_=+[{]}\|;:'"<.>/?
123456,123456789,picture1,password,12345678,111111,123123,12345,1234567890,senha
1234567,qwerty,abc123,Million2,0,1234,iloveyou,aaron431,password1,qqww1122,
123,omgpop,123321,654321,qwertyuiop,qwer123456,123456a,a123456,666666,asdfghjkl
ashley,987654321,unknown,zxcvbnm,112233,chatbooks,20100728,123123123,princess,jacket025
evite,123abc,123qwe,sunshine,121212,dragon,1q2w3e4r,5201314,159753,123456789
pokemon,qwerty123,Bangbang123,jobandtalent,monkey,1qaz2wsx,abcd1234,default,aaaaaa,soccer
123654,ohmnamah23,12345678910,zing,shadow,102030,11111111,asdfgh,147258369,qazwsx
qwe123,michael,football,baseball,1q2w3e4r5t,party,daniel,asdasd,222222,myspace1
asd123,555555,a123456789,888888,7777777,dockery,1234qwer,superman,147258,999999
159357,love123,tigger,purple,samantha,charlie,babygirl,88888888,jordan23,789456123
password
qwerty123
pass1234
fgh123$
Qwerty123
0circleS
UiL+2024
u20-IL_24
Sxxxxxxx0
B0000000s
Pass12345!
ABZYcdx[3-2-1-0]
jfm2amj0^jas1ond9
```

Sample output:
```
password:0:UNACCEPTABLE
qwerty123:0:UNACCEPTABLE
pass1234:0:UNACCEPTABLE
fgh123$:0:UNACCEPTABLE
Qwerty123:27:FAIR
0circleS:16:WEAK
UiL+2024:37:GOOD
u20-IL_24:45:GOOD
Sxxxxxxx0:19:WEAK
B0000000s:19:WEAK
Pass12345!:36:GOOD
ABZYcdx[3-2-1-0]:105:STRONG
jfm2amj0^jas1ond9:86:STRONG
```

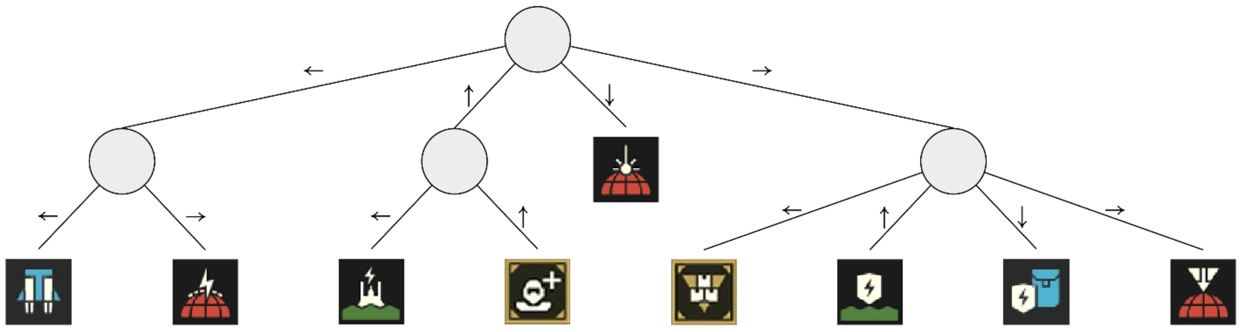
6. Jared

Program Name: Jared.java Input File: jared.dat

Jared recently discovered a new up-and-coming game with a key mechanic known as "Stratagems." These Stratagems allow the player to input a unique code using the left, up, down, and right arrow keys, which allows them to trigger a special ability. However, through his own playing of the game, he found himself oftentimes struggling to input the codes for these various Stratagems in the heat of the moment. However, after looking into why he was struggling so much, he discovered a fatal flaw with the input codes for the Stratagems: some of the codes are prefixes of other codes!



The tree above represents the current state of Jared's Stratagem inputs. Note that the input "↑" is a prefix for the input "↑←" and "↑↑".



The tree above represents Jared's modified set of inputs where no Stratagem input is the prefix of any other Stratagem input.

Before Jared decides to convince his friends to buy the game and play it with him, he wants to make sure that no two Stratagem input codes are prefixes of one another. Help Jared write a program that, given a set of Stratagem input codes, determines if no Stratagem input code is the prefix of any other Stratagem input, and, if there does exist such a pair of input codes, how many there are.

Input: The first line of input will consist of a single integer, $1 \leq n \leq 50$, denoting the number of testcases to follow. Each of the next n lines will consist of a comma-separated list of Stratagem inputs, which are denoted with the letters 'U' for up, 'D' for down, 'L' for left, and 'R' for right. It is guaranteed that the length of any given stratagem input will be between 1 and 10 characters in length, and that there will be between 1 and 100 stratagem inputs for any arbitrary request.

Output: For each of Jared's n requests, on their own line, if there are no Stratagem inputs which are a prefix of any other Stratagem inputs, output the string "`Democracy Prevails!`" If, however, there exists some set of invalid Stratagem inputs, output the string "`There are <#> misinputs...`" where `<#>` is the number of pairs of inputs in which one input is the prefix for the other input.

~ Jared continued… ~

Sample input:
```
4
LL,LR,U,UL,UU,D,RL,RD,RR
LL,LR,UL,UU,D,RL,RU,RD,RR
LLL,LLDR,LLDU,LLDD,LLD,LL,LUD,LU,LR,R
LLL,LLDR,LLDU,LLDD,LLD,LL,LUD,LU,LR,R,L
```

Sample output:
```
There are 2 misinputs...
Democracy Prevails!
There are 9 misinputs...
There are 18 misinputs...
```

7. Jimothy

Program Name: Jimothy.java **Input File: jimothy.dat**

You and Jimothy have a paper on Binary Search tree traversals due tomorrow. Write a program that will, given a list of strings, generate a binary search tree, and run a pre-order, in-order, post-order, and reverse-order traversal on it. You should construct the binary search tree as normal, and the traversals should be as normal as well.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of an unknown number of space-separated strings to be put into the binary search tree. They should be inserted into the tree in the order given, duplicates should be inserted into the left side of the tree.

Output: First output the string `"TEST CASE #n:"`, where `n` is replaced with the number of the test case, starting at 1. Then, on the following line, output the string `"PRE-ORDER TRAVERSAL: "`, the elements of the tree after being traversed in pre-order, all separated by spaces. Then, on the following line, output the string `"IN-ORDER TRAVERSAL: "`, the elements of the tree after being traversed in in-order, all separated by spaces. Then, on the following line, output the string `"POST-ORDER TRAVERSAL: "`, the elements of the tree after being traversed in post-order, all separated by spaces. Then, on the following line, output the string `"REVERSE-ORDER TRAVERSAL: "`, the elements of the tree after being traversed in reverse-order, all separated by spaces.

Sample input:
```
3
Never Gonna Give You Up
Somewhere Over The Rainbow Way Up High
Dont Stop Believing
```

Sample output:
```
TEST CASE #1:
PRE-ORDER TRAVERSAL: Never Gonna Give You Up
IN-ORDER TRAVERSAL: Give Gonna Never Up You
POST-ORDER TRAVERSAL: Give Gonna Up You Never
REVERSE-ORDER TRAVERSAL: You Up Never Gonna Give
TEST CASE #2:
PRE-ORDER TRAVERSAL: Somewhere Over High Rainbow The Way Up
IN-ORDER TRAVERSAL: High Over Rainbow Somewhere The Up Way
POST-ORDER TRAVERSAL: High Rainbow Over Up Way The Somewhere
REVERSE-ORDER TRAVERSAL: Way Up The Somewhere Rainbow Over High
TEST CASE #3:
PRE-ORDER TRAVERSAL: Dont Believing Stop
IN-ORDER TRAVERSAL: Believing Dont Stop
POST-ORDER TRAVERSAL: Believing Stop Dont
REVERSE-ORDER TRAVERSAL: Stop Dont Believing
```

8. Paola

Program Name: Paola.java Input File: paola.dat

You have been hired by Professor Paola to handle a new method of storing data he calls "matrixizationing". You will be given a matrix of integers, and a number of commands to use to alter the matrix. The commands will all be one of the following:

- `"ROTROW I N"` – Rotate row `I` of the matrix `N` spaces to the right. If `N` is a negative value, then row `I` should be rotated `|N|` spaces to the left (`|N|` is absolute value of `N`).
- `"ROTCOL I N"` – Rotate column `I` of the matrix `N` spaces downward. If `N` is a negative value, then column `I` should be rotated `|N|` spaces to the left (`|N|` is absolute value of `N`).
- `"ADDROW I [A, B, …]"` – Add the given row `[A, B, …]` to the matrix at index `I`. If the size of the given row is not the same size as the other rows currently in the matrix, do not add it, and output the string `"ROW [A, B, …] OF INVALID LENGTH"`. This output should be on its own line, before the output for the data set. If the index is out of bounds, and the given row is of incorrect length, print both error messages, with the index out of bounds message coming first.
- `"ADDCOL I [A, B, …]"` – Add the given column `[A, B, …]` to the matrix at index `I`. If the size of the given column is not the same size as the other columns currently in the matrix, do not add it, and output the string `"COLUMN [A, B, …] OF INVALID LENGTH"`. This output should be on its own line, before the output for the data set. If the index is out of bounds, and the given column is of incorrect length, print both error messages, with the index out of bounds message coming first.
- `"DELROW I"` – Delete the row of the matrix at index `I`.
- `"DELCOL I"` – Delete the column of the matrix at index `I`.
- `"GETCOL I"` – Print out the column at index `I` in the format `[A, B, …]`. This output should be on its own line, before the output for the data set.
- `"GETROW I"` – Print out the row at index `I` in the format `[A, B, …]`. This output should be on its own line, before the output for the data set.
- `"SETVAL I J N"` – Set the value at row `I` column `J` equal to `N`.
- If for any of the commands, one or more of the given indices is out of bounds, ignore the command and output the string `"INDEX OUT OF BOUNDS."`. This includes adding a row or column at index = size, i.e., if the matrix is 3x3, adding a row or column at index 3 will be out of bounds.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will begin with 3 integers, `r`, `c`, and `m`, denoting the number of rows and columns in the starting matrix, and the number of commands to be done on said matrix, respectively. Each of the following `r` lines will each contain `c` integers, denoting a row of the matrix. Following these `r` lines, there will be `m` lines, each containing a command in the format shown above in the problem description.

Output: Output, after any outputs from the commands as specified, the matrix of integers after all commands have been run, with each row appearing on its own line with every element separated by spaces. Separating each test case, output the following string `"----------"` containing 10 hyphens.

~ Sample input and output on next page…~

~ Paola continued …

Sample input:
```
2
3 3 4
1 2 3
4 5 6
7 8 9
ADDROW 1 [10, 11, 12]
GETCOL 1
DELCOL 0
SETVAL 1 4 0
2 4 3
1 2 3 4
5 6 7 8
ADDCOL 2 [10, 11, 12]
SETVAL 0 1 9
ROTCOL 2 1
```

Sample output:
```
[2, 11, 5, 8]
INDEX OUT OF BOUNDS.
2 3
11 12
5 6
8 9
----------
COLUMN [10, 11, 12] OF INVALID LENGTH
1 9 7 4
5 6 3 8
```

9. Riley

Program Name: Riley.java Input File: riley.dat

Riley's friend Priscella warned her about past programming challenges that required working with 2-D arrays in a non-standard pattern instead of straight rows and columns. Priscella suggested Riley practice by walking around the outer edge of the array and working inward.

The following example is NOT meant to show any pattern in the way that random data will actually exist in the table, the data in the table is specifically organized to only show the pattern of processing the cells in the table. Teams are NOT to infer any meaning; it is just data in a table that will be processed in a unique pattern.

Start at the top left corner and walk straight down the first column, visiting cells with 1 … 7. Movement then changes direction and walks across the bottom row, visiting cells with 8 … 13. Movement again changes direction and walks up the right column, visiting cells with 14 … 19. Movement again changes direction and across the top row, visiting cells with 20 … 24. The top cell of that column was previously visited and is not included. That same pattern is repeated with the remaining portion of the array, visiting cells with 25 … 29, then 30 … 33, then 34 … 37, and 38 … 40. The process continues, spiraling inward, until all cells are visited: cells with 41 … 43, then 44 … 45, then 46 … 47, then 48, and finally 49.

Col → ↓ Row	0	1	2	3	4	5	6
0	1	24	23	22	21	20	19
1	2	25	40	39	38	37	18
2	3	26	41	48	47	36	17
3	4	27	42	49	46	35	16
4	5	28	43	44	45	34	15
5	6	29	30	31	32	33	14
6	7	8	9	10	11	12	13

For each path around the array as described above, calculate and output the average of the cell values visited during that trip around the array as shown below in the first line of sample output.

Input: First line contains a single integer **T** the number of test cases that follow with **T** ≤ 10. Each test case starts with a line containing 2 integers separated by whitespace: **R**, the number of rows, and **C**, the number of columns, with both 2 ≤ **R**, **C** ≤ 15. That line will be followed by **R** lines of data with each containing **C** integers separated by whitespace containing integers in [-100, 100].

Output: For each test case, output 1 row of averages produced from the segments visited, separated by single space. A trailing space at end of each line is permitted. Display the averages with 2 digits after the decimal point.

Sample input:

```
3
7 7
1    24   23   22   21   20   19
2    25   40   39   38   37   18
3    26   41   48   47   36   17
4    27   42   49   46   35   16
5    28   43   44   45   34   15
6    29   30   31   32   33   14
7    8    9    10   11   12   13
```

~ Sample input continues on right ~

Sample input continued

```
3 6
40   90   -42  21   97   31
21   -28  -84  67   -85  -67
-30  -55  -36  -99  35   -22
5 2
11   -81
26   86
71   -23
-68  6
-62  48
```

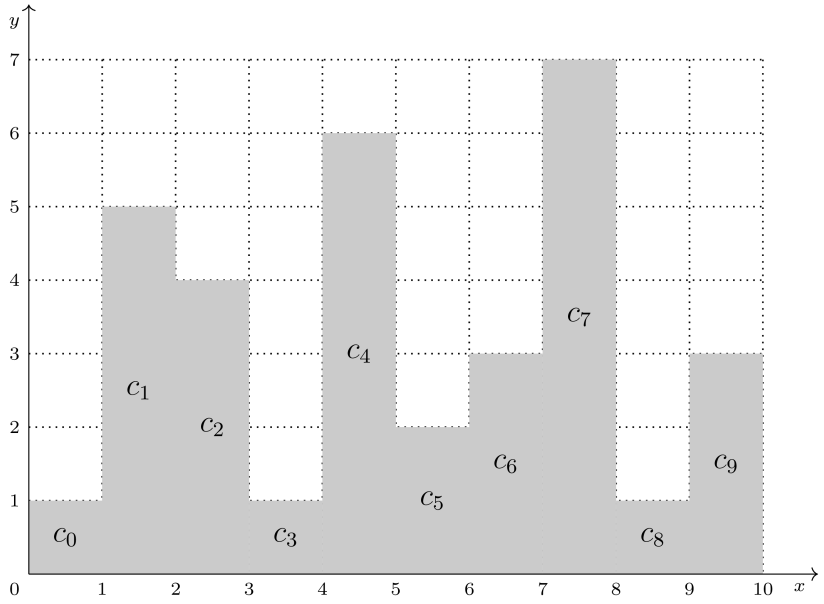
Sample output:

```
12.50 32.50 44.50 49.00
-1.14 -32.50
1.40
```

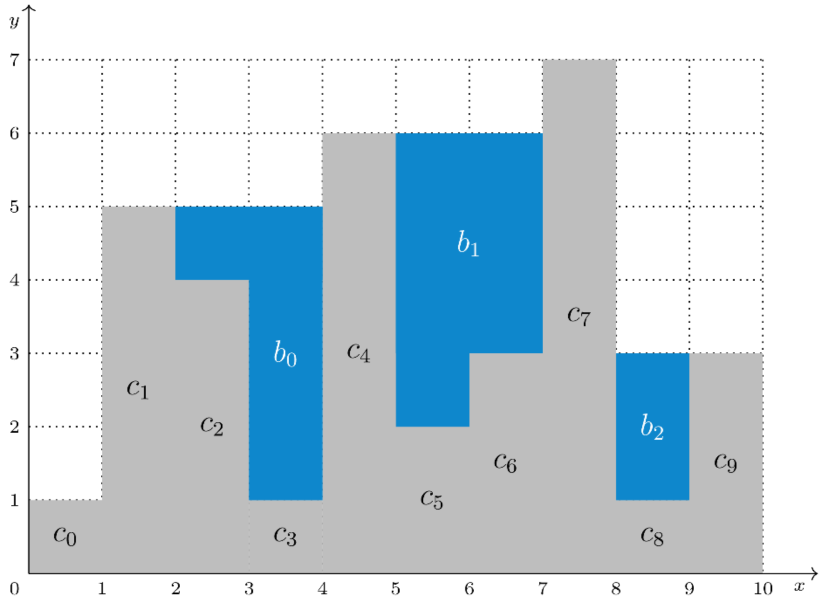
10. Sasha

Program Name: Sasha.java **Input File: sasha.dat**

Sasha has recently taken an interest in the physics of rainwater in a 2D space and wants to be able to visualize what would happen to a 2D space with a series of concrete, nonabsorbent columns, should it rain for an arbitrarily long period of time. The following markup is an example of what such a 2D space of columns might look like (assume that the only columns that exist in this 2D space are those which are explicitly stated):



Sasha discovered that rainwater would collect into certain "buckets" which lie between the different concrete, nonabsorbent columns. Assume that water tension does not exist in this 2D space, as well as any excess water which would result from overflowing from a bucket gets absorbed into the ground surrounding the left and right most columns, effectively voiding any runoff. Sasha manually determined that the following buckets would form from the example above in such a scenario:



However, Sasha wishes to automate this process. Help Sasha write a program that, given a list of column heights, determines the total amount of water that is contained within each bucket, as well as the entire 2D plane.

~ *Input and Output on next page…* ~

~ Sasha continued… ~

Input: The first line of input will consist of a single integer, $1 \leq n \leq 100$, denoting the number of testcases to follow. The next n pairs of lines will start with a line consisting of a single integer, $1 \leq m \leq 200$, denoting the number of columns for the current test case, which are labeled $c_0, c_1, \ldots, c_{m-1}$. The second line consists of m space-separated integers which denote the height of the columns, labeled $h_0, h_1, \ldots, h_{m-1}$. It should be noted that any arbitrary column, c_i, has a width spanning from $x = i$ to $x = i + 1$ and a height spanning from $y = 0$ to $y = h_i$, with $0 \leq h_i \leq 10^4$.

Output: For each of Sasha's n requests, first determine if there are any buckets that would form if it were to rain for an arbitrarily long period of time to fill any given bucket. If there does not exist any such bucket, then print the string "`Dry as a Bone...`". If there does exist such a bucket, then, on the same line, for each bucket $b_0, b_1, \ldots, b_k$, with b_0 being the left-most bucket and b_k being the right-most bucket, print the total "volume" (surface area since we are dealing in a 2D plane) of water that each of those buckets can hold, each separated by a space, followed by the total volume of all buckets combined.

Sample input:
```
3
10
1 5 4 1 6 2 3 7 1 3
11
0 1 2 3 4 5 4 3 2 1 0
11
5 4 3 2 1 0 1 2 3 4 5
```

Sample output:
```
5 7 2 14
Dry as a Bone...
25 25
```

11. Svetlana

Program Name: Svetlana.java **Input File: svetlana.dat**

Svetlana's favorite toy growing up was the 8 puzzle game. For those unfamiliar with the 8 puzzle game, the goal of the 8 puzzle game is to get a random ordering of the numbers 1 thru 8, into acceding order, leaving a blank at the bottom, right most corner. The only legal move is to slide a number into a blank. No two numbers, can be swapped. For example, take the below board. This board would take 6 moves to optimally get all blocks in the correct ascending order 1 thru 8, with the blank at the bottom right most position.

2	4	3
1		6
7	5	8

Start

2		3
1	4	6
7	5	8

	2	3
1	4	6
7	5	8

1	2	3
	4	6
7	5	8

1	2	3
4		6
7	5	8

1	2	3
4	5	6
7		8

1	2	3
4	5	6
7	8	

Goal

Move 1	Move 2	Move 3	Move 4	Move 5	Move 6
Slide 4 into BLANK	Slide 2 into BLANK	Slide 1 into BLANK	Slide 4 into BLANK	Slide 5 into BLANK	Slide 8 into BLANK

Although Svetlana, can solve this puzzle, she wants to know if she's solving it in the best, optimal way. Can you help Svetlana write a program that given a starting board, will output the number of moves it would take to optimally solve the puzzle?

Input: Input will begin with an integer N, the number of test cases. N will be in range [1,10]. Each starting board is preceded by eight dashes. This serves to break up each test case, making it easier to read. Following the eight dashes, will be the 3 x 3 board, representing the starting position of the tiles. The value of `-1` will be used to indicate the BLANK position in the board. It is guaranteed that the numbers 1-8 will be given. There will be no duplicates, and no numbers outside of the range [1,8].

Output: For each test case, you are to output: "`Number of steps needed to solve: #`", where # is the number of moves to optimally solve the puzzle. If the puzzle has no solution, output: "`No solution exists.`"

~ Sample input and output are on next page… ~

~ Svetlana continued… ~

Sample input:

```
7
--------
2 4 3
1 -1 6
7 5 8
--------
1 2 3
4 5 6
7 8 -1
--------
1 2 3
8 -1 4
7 6 5
--------
1 2 3
4 5 6
7 -1 8
--------
2 7 3
8 1 5
4 6 -1
--------
2 1 -1
3 6 5
7 8 4
--------
8 6 7
2 5 4
3 -1 1
```

Sample output:

```
Number of steps needed to solve: 6
Number of steps needed to solve: 0
No solution exists.
Number of steps needed to solve: 1
Number of steps needed to solve: 14
Number of steps needed to solve: 20
Number of steps needed to solve: 31
```

12. Wesley

Program Name: Wesley.java Input File: wesley.dat

You and your friend Wesley have been trapped in a dimensional maze by some loser whose name doesn't even matter for the problem. You need to escape the maze, that's it, but plot twist, this isn't just any normal maze, because that would be too easy for you. This maze shifts dimensions. The maze runs on a 5 second cycle, beginning in second 1, and as every step takes 1 second, the maze will shift for every step you take. When stepping, you will need to make sure your step is into a space that is valid in the next dimension, as the dimensions will change as you step. You can only take steps in the 4 cardinal directions (up, down, left, right), no diagonals. The different dimensions will be explained in the Dimensions section. The maze will be made up of only the following characters:

- `'S'` – This character denotes your starting point in the maze. This space will always be passable, except when in dimension 4.
- `'E'` – This character denotes the exit from the maze. This space will always be passable, except in dimension 4 (in dimension 4, you cannot escape).
- `'1'` – This character represents a wall that disappears when in dimension 1. When it comes to any other dimension, it should be treated as a wall.
- `'3'` – This character represents a wall that disappears when in dimension 3. When it comes to any other dimension, it should be treated as a wall.
- `'5'` – This character represents a wall that disappears when in dimension 5. When it comes to any other dimension, it should be treated as a wall.
- `'#'` – This character represents a wall. This object is impassable, unless you are in dimensions 2 or 4.
- `'.'` – This character represents an empty space. This is always passable, except in dimension 4.
- Dimensional Walls: In the following section `'1'`, `'3'`, and `'5'` will be referred to as dimensional walls.
- There is not guaranteed to be an exit from the maze.

Dimensions:

- Dimension 1: You begin in dimension 1. All characters are as listed above, except the walls marked '1' are now empty spaces. When you take a step (1 second passes), you move to dimension 2.
- Dimension 2: This is the rest dimension. All the spaces in the maze turn into empty space, walls and dimensional walls included. When you take a step (1 second passes), you move to dimension 3.
- Dimension 3: This is similar to dimension 1. All characters are as listed above, except the walls marked '3' are now empty spaces. When you take a step (1 second passes), you move to dimension 4.
- Dimension 4: This is the backwards dimension. All walls are now open spaces (including dimensional walls), and all open spaces are now walls. When you take a step (1 second passes), you move to dimension 5.
- Dimension 5: This is similar to dimension 1. All characters are as listed above, except the walls marked '5' are now empty spaces. When you take a step (1 second passes), you move back to dimension 1.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will begin with 2 space-separated integers, `r` and `c`, denoting the number of rows and columns in the maze. Each of the following `r` lines will contain `c` characters denoting a row the maze.

Output: For each test case, first output the string `"Test Case #n: "`, where `n` should be replaced by the number of the test case, starting with 1. If it is possible to escape the maze, output `"The Great Escape."` If escape is impossible, output `"Guess I won't be home in time for dinner."`.

~ Sample input and output are on next page ~

~ Wesley continued ~

Sample input:
```
2
8 8
S..1.5.#
#.3.33.5
1.####35
3355E###
...###..
..3..15.
##.3...1
5....53.
6 6
S35#.5
11..5.
5..33.
##135#
##513#
E.351#
```

Sample output:
```
The Great Escape.
Guess I won't be home in time for dinner.
```

UIL Computer Science Competition

Region 2024

<u>JUDGES PACKET - CONFIDENTIAL</u>

I. Instructions

1. The attached printouts of the judge test data are provided for the reference of the contest director and programming judges. Additional copies may be made if needed for this purpose.

2. This packet must remain CONFIDENTIAL. Additional copies may be made and returned to schools when other confidential contest material is returned.

II. Table of Contents

Problem #1
60 Points

1. Audrey

Program Name: Audrey.java **Input File: None**

Test Input File: None

Test Output To Screen:
```
U.........U.........U
.I........I........I.
..L.......L.......L..
...C......C......C...
....O.....O.....O....
.....M....M....M.....
......P...P...P......
.......S..S..S.......
........C.C.C........
.........III.........
UILCOMPSCI*ICSPMOCLIU
.........III.........
........C.C.C........
.......S..S..S.......
......P...P...P......
.....M....M....M.....
....O.....O.....O....
...C......C......C...
..L.......L.......L..
.I........I........I.
U.........U.........U
```

Problem #2
60 Points

2. Bob

Program Name: Bob.java **Input File: bob.dat**

Test Input File:
```
10
1 2 3 4 5
11 43 9 22 17
4 8 5 7 6
15 51 20 63 12
3 8 0 9 2
44 33 22 11 55
30 31 32 33 34
0 1 2 3 4
10 5 55 50 90
45 1 10 100 35
```

Test Output To Screen:
```
25
1062
82
3413
75
2695
2084
14
5050
4545
```

Problem #3
60 Points

3. Casandra

Program Name: Casandra.java **Input File: casandra.dat**

Test Input File (*indented lines are continuation of previous line*):

```
50
A^B&A^B^A^A^A
B^B^E^E&C^E|C^B&D&A|A&C^D&B^C^E^B|D^C&D^C&B^E|D^D^E^C^E^D|E^D
!(!!!!(!(I&D&D|H&I|G^C|E&H^B^F|F^E)&D|E|I^E&D^G)|I^B^D|A^D&I^H|C&E|B^B^B|F&F^H&D
     &A&I|D&F^B|H|A^C|I|A|F)&B&H&A
!F^H|H^B|H|C^F^A^C&F|B^D^H^D^F&A^G|B&E^D&E&E^H^E|H|B
(!(A^E|C^F^A^C&E&F&D^B&D^E^C&F^D&D^A)&F^F|F)^D&A^A&E&A^A^A&D|B
!A^B^B^B&D|A|B&A&D|C&B|A^B&A|A&B
!(G|C^F&F^D|B^D^B|D^B|D&B^C)|B|E|C^E&F&C|A^D|E|B|D|E|G|D^C^F
!(((!(F^B&E|C^E|A|I|H)|B^I^H^F|H))|H^B^A^I^F|B^F|A&A&J|I&F^D&F|H&F&A^B^F&J^D^J^E
     )^G&J&I|D&A^H&I^B^A
(B^B&C|C)&B|C&A^B^A&C|A|C^B&C^A^C|B
!!!D&F&H&H|A^B^G|H^C^D&B&C^G|B&E^D&H^D&A|B&A|H&E&D&G^B^E&H&H^A&A&D
!!(!G&G|B|C^C^C^E&C&D^B^B^B^B^C^B|F&A&E&A&A^C)&B&E&F&A&G|C^C|G&B
(C^C&B|D&B|D&C|D|A^D&C)&D^B&C^C&C^D|B|A^D|A
(!D^C^E|F&A|B^C|G^F|F^D^D^F^F^A)^D&F|D
!!((((G^G&A&F^A^C|H|G)&B^F^D&G|E^C|B&E)|F&F^B|F|G)^G^E|G)&F^F|F^E&G|H^B|A^D^C|A^
     D^A
!!!A|I^D^H^B&E|G|I&B&H&H|F^B&I&E&F|H|E&E&H&E|H|E^I^F^B|A^C&A&D^A&D&A^I^I|H^H^F
((!(!F^A^H|E^D^B^F^E^B^A^B|I^A^E|A&B^F^D|C)&H|F^B|A^C&H|D^A|D|G&E|H^H|H|G&E|D&G^
     D&F)|J&B^J|J|C&H^A)&D^G|E^A&I^F^H
!G^G^E|D^E&H^A|E&F^B^C^D|C|D&A&B&F^A&A^D|E^A^A|E|C|F|H^A|D|B|G^G&A
(H^H^A&H^A^D|H^H^A|B&B^G^G|G)|C&F^G&G|A^B|G|H|B|E^D^E^D&D^C&C^E&H|A^D^A^D^E
!!(((E&A)|B&A&B|C^D|F^D&B|E^C^B^E&E|F^F^B)&B&C|F|D^B&E)&C&D^F^F^A&F^F&B|D^D&E|E|
     D|A
(B|C^D|C&C|D|C|C^D|A|E|E^A^E^D)^D&D|C^A|E|D&C|D^E
!!C^A&C&A^D|A|A&C^C|F|B^F|F^A&E|D^A^C^A^F^C^C^D&A
!!!!B^C^B|E|E^E|E|D&F&D^C^E^D^E^C^E&F^C&A&D^C^D&D&F^B|E^C&A
(((((!!!!!A^J|G|J^E^F|A^A|F^A|J^E&E&I^H&C|B&E|J|F|H|D&B|I|F|C^E^I|E)^G)^J^H&H|A&
     G&J^I&D|H^A&B|E)^B&G|A&I)^G&G^G&G)|C^J|F
!D&D^B^C^D&C^A&C&A^D&B^D&B|C&C&A&C&C&C
(!(E^D^D|C)&B&C^D|C)|A^E&B^A^C^A|B^C|D^B^C&C
!B|A^B^D^A^A^D|C^A^B^A
!!B^B^C^D&A|D^C&D&D&D^B^D
!!E|C&D^E|D|A&D^D|D&E&C|D^D^E|D&B^C^E
!B^B&A|A^A^A|C&B&A&C^B|A&B^B^A|B|C
!!(!((A&C&I|C^E)&B)|D&I|E|E)&G^G^I^D&E|H|A^B&B^A|I|A|B^D^F&H&E|D&D&B^E&A^I^A|I|E
     ^G&A|E|C|B^B&I^C|G^B&D^F
!!!(A&C|D^G&G|C^D|C&C|I&I^I^H^A^G|I^E&C^F^C|B|G|E&D)^B^I&G|E&F^I^B&C^C^F^E&G
!((!D&E^H^D^C)|D^E&E&A)^E|B^C|E^B&G^F^D&D^C&A&B&H&B^A&H^H&D^C^C&A^D^E&B^H^D|H^H
!B^B^A&B^A^B&B&B
!!G^H^A|H^I^I|G|B|B^B&G&I|E|B^A^A^G|D^H&B^F&A|F|E^E^G|H^B^C|A^B|G|G|I^H|G^A&D|A
A&A|A^B^B&A^B^B^A|A|B|A&B&B
!(C|G^B&A)^G^E^E^D|G&G^C|B&G|C|C&D|D^A|G&B^B^G&F&C|D|B^E
```

~ Input continues next page… ~

~ Casandra continued ~

***~ Input continued from previous page… ~* (*indented lines are continuation of previous line*)**

```
!((!(G|D|A)&E|G&E|A|A|G^E|A^E&A^D)&F&B|B&B^A^F|D)|D&H|G&I&C&F^E|F|H^E&H^F|H^E^C^
     B&C|C^H|E|H^I&H&A|D
((!!(B^D)^D&D^C&D&B^C&B|B&B^D|A^D^D&B))^C^C&D
!(B|C|F|D|D^E^F|B|C^B^B)^D^C^B|B&F^F&A^B|C&F^A|B|A&A^F
!!(E|A&A)|G^H&B^A&A&C&C^C|D&D^B&F^B&B|B^C|A|F|H^H|C^C|B
(!((((((F^J|C^D&F&D&C^A|G^E)^A|A^H^A&G&F&F&H^D|G)&C&J^H^H|D|A^D^F^B|C|E|G^B|A^G|
     H|B^D|E))&B&I)|H^D&E|F&G)^A&B&C|H^F^E)&J|G&C
!(!B&C^C&A&G&C|D|B&B^A&E)&A^C&E&F&E&G|C&A^A|E^D
!!!!(!!!B&B&G^D|F)&A|A&F|G&G^C|B|A|B^G|D|B&D^G^G|F&B|B^G|B^F|F^F|E|F&E&D|G|A
!(!F|G|D&E&E&A^G^A&D^C^B^F^F^F|A^A^G|A&C&E|A|A^B)^F|G&C|B&B^F|B^G^C|F|E|C|A&C&B^
     F|E^E&G
!(!!(!(!!(J^A&D^D)&H|D|F^D&H|B^A^A|E&E|E&A^G|B^I|B)^I^D&E|D&G^H)&H|G^B|C|B^B^I^I
     ^G|J^A^H&A|E^J^G&H^D|A|E)|D^H^H|J&B^A
!(!B^G&B&A|F&F&C&F|C|D&F&D^C|A^A^B|G|C^A|B|D^G|D|E|A|A|F^F&F&G|C|G)&G&G^D&B|C^G|
     B^C^D
!((!(H^E)|H|I^D&D)|D&B)&E&E|F&D&G^C|G^I^E^B^C|H&G|F&E^H^I&D&F^F^E^C^H^D|C^F|G^G^
     D&G^F|E&E&F^F^G^A
(((!!!!G^C|G^F|E^E^D^F^G&A|F&B&E^D^G&G^D|E^G^D&C|C))|C|B|G&F)&B^E^G^D^D|E^D&C|C&
     B&E^E
((!C|C&C|E|A&B&C|E|B))&E|D|B^B|C&C|B
C^E|B|D^F^B|F^C^D&F^E^E|F|D^E^B^C|F|E&B|B^B^F^C|D&D&D^C^D^E|F^F&E&B|B&A
```

Test Output To Screen: (first 3 cases only)

```
0 0
1 0
--------
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
--------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
--------
```

~ Program produces 608 lines of output – see casandra.out file for complete output ~

Problem #4
60 Points

4. Clarabelle

Program Name: Clarabelle.java **Input File: clarabelle.dat**

Test Input File:
```
10
qwqtquqoqhqh
university
tttrrraaappp
bob
utututututut
mama
uuuuuuuu
abcbabcdcbcd
yabbadabbado
popupbadadud
```

Test Output To Screen:
```
hqh qhq qoq qtq quq qwq
NONE
aaa ppp rrr ttt
bob
tut utu
ama mam
uuu
bab bcb cbc cdc
ada
ada dad dud pop pup
```

Problem #5
60 Points

5. Harmony

Program Name: Harmony.java Input File: harmony.dat

Test Input File:
```
EARIOTNSLCUDPMHGBFYWKVXZJQ
`~!@#$%^&*()-_=+[{]}\|;:'"<.>/?
123456,123456789,picture1,password,12345678,111111,123123,12345,1234567890,senha
1234567,qwerty,abc123,Million2,0,1234,iloveyou,aaron431,password1,qqww1122,
123,omgpop,123321,654321,qwertyuiop,qwer123456,123456a,a123456,666666,asdfghjkl
ashley,987654321,unknown,zxcvbnm,112233,chatbooks,20100728,123123123,princess,jacket025
evite,123abc,123qwe,sunshine,121212,dragon,1q2w3e4r,5201314,159753,123456789
pokemon,qwerty123,Bangbang123,jobandtalent,monkey,1qaz2wsx,abcd1234,default,aaaaaa,soccer
123654,ohmnamah23,12345678910,zing,shadow,102030,11111111,asdfgh,147258369,qazwsx
qwe123,michael,football,baseball,1q2w3e4r5t,party,daniel,asdasd,222222,myspace1
asd123,555555,a123456789,888888,7777777,dockery,1234qwer,superman,147258,999999
159357,love123,tigger,purple,samantha,charlie,babygirl,88888888,jordan23,789456123
password
qwerty123
pass1234
fgh123$
Qwerty123
0circleS
UiL+2024
u20-IL_24
Sxxxxxxx0
B0000000s
Pass12345!
ABZYcdx[3-2-1-0]
jfm2amj0^jas1ond9
UIL+2024
B000000s
RS01st23
aBCDEFG0
_it'sHard4me2
biteme.19?
biteme
DoU@L0veMe?
123456
123456.7x
192837.4x
asdf-098765
(uil2024)
!abc/123
PASSWORD
8a>Z&B<Q3#
UrS0{VrY*WrOnG}
```

~ Output continues on next page ~

~ Harmony continued ~

Test Output To Screen:
```
password:0:UNACCEPTABLE
qwerty123:0:UNACCEPTABLE
pass1234:0:UNACCEPTABLE
fgh123$:0:UNACCEPTABLE
Qwerty123:27:FAIR
0circleS:16:WEAK
UiL+2024:37:GOOD
u20-IL_24:45:GOOD
Sxxxxxxx0:19:WEAK
B0000000s:19:WEAK
Pass12345!:36:GOOD
ABZYcdx[3-2-1-0]:105:STRONG
jfm2amj0^jas1ond9:86:STRONG
UIL+2024:26:FAIR
B000000s:18:WEAK
RS01st23:25:FAIR
aBCDEFG0:22:FAIR
_it'sHard4me2:67:STRONG
biteme.19?:35:FAIR
biteme:0:UNACCEPTABLE
DoU@L0veMe?:60:STRONG
123456:0:UNACCEPTABLE
123456.7x:38:GOOD
192837.4x:38:GOOD
asdf-098765:38:GOOD
(uil2024):34:FAIR
!abc/123:33:FAIR
PASSWORD:0:UNACCEPTABLE
8a>Z&B<Q3#:68:STRONG
UrS0{VrY*WrOnG}:99:STRONG
```

Problem #6
60 Points

6. Jared

Program Name: Jared.java **Input File: jared.dat**

Test Input File (*indented lines are continuation of previous line*):

```
50
L,LLL,LLLLL,LLLLLLL,LLLLLLLL,LLLLLLLR,LLLLLLLD
L,R,D,LLL,LLR,LLD,LLLLL,LLLLR,LLLLD,LLLLLLL,LLLLLLD,LLLLLLR,LLLLLLLL,LLLLLLLR,LLLLLLLD
UDRLU,UDRU,DDUR,DLDUR,DLRUD,DLDUUL,DDLUR,DLRRL,DLUDU,DLDUUR,DRDULR,DDUDD,RDLUU,RRR,RRDLRD
    ,RDUULDD,RDRDRD,RDURD,RUDDR,URR,URDR,URDDR,URDU,DUUDU,URUD,URUL,URDDD,RRU,RRDR,RRLD,R
    RDU,DULRRL,DDLRLR,DURULR,DLUR,DLDUUD,DLULD,DLDUL,DLLD,DULURR,DLDDUL,DRDULL,DULRLR,DUR
    RU,DURL,DURRD,DULURD,DURULU,DURRL,DURDR
DD,DLRUUUR,RRDULRU,LRR,RRL,DDDUD,DR,UD,RURLUDUUDR,DU,RRR,URLDUUR,LRLRR,URRRLDR,UL,DRRULDD
    ,ULRRDLDRDL,UU,DDLDRDLRU,UDDLLRDL,ULLUU,LLLR,DLLLR,DUDLRDDL,DLRDR,RL,UUUDDUDUDU,RUDDR
    U,RRRDLURUR,RLRRDL,DUURRRDL,LURDLRLRUU,RU,ULLRR,RLLULR,L,DDLDDLRRL,URL,DRRDDUUDR,R,UR
    UL,UDLUDLLUL,U,LDL,RDUURD,RLURDLDU,LUUUU,LRLLU,DLRRRL,RUD,RDDULRLD,DRDDRLLDLL,ULLUURD
    LR,UDUDDURLU,UDLDDD,RDDULUL,DLDRLDUR,DULLUR,ULLDRRUDD,ULDDDLUU,LLLUDDD,LLDUURLL,RLLUR
    DDRU
LL,UDDULU,UURLLURDDL,LR,DDDLUULLU,LRL,DULRLLDRR,RURULU,UD,DU,LDRR,LUDULU,DRDRUL,DDLRD,DRD
    LLRDRD,DLLUUURUU,DLDRUULLL,DUUDUL,UDRL,DLDL,DLUD,LURLRULDU,RL,UDRULDDLLD,LDUUDDLRDL,L
    LLLDLRULL,URLUDLDUUD,RDLDURUL,URLU,ULDLLDRUU,LRLDU,RU,RDRDD,D,URDLU,DDULR,RRLLDU,UUUU
    DUL,DUDRLL,LDULDR,L,RDRUD,UUURDULDR,R,RRRDDLULU,RLDLDDUDDU,LUDRURRDR,DDR,RDD,URR,U,LD
    L,DLDULRRR,DRDUUDRU,LLULDRL,LRRDURDUU,UULULLRDR,RDDU,DLLRU,LULLRLRRUL,LRLLDLL,RUDUU,R
    DDRULLL,RLDU,UURDL,RUU,RDURLRRL,DDLLRRDRR,DUDL,RLRLDDRLUL,LRD
LLLLR,RDRLRDL,URUDUDRL,RDRRLUUUDD,RRLUULRR,DLUR,LR,URDDURDUD,DRDLDRDLUL,DLRLUL,LU,DLDRLRR
    ,LDDLD,DURRLR,RRL,LRU,RRUULLRLLU,ULDRUUDDR,RLLLLULDR,DUUDR,DLUULUL,LRUDLD,URLLDR,ULDL
    LDDR,UDDDUDUDLD,RLLUUDULUU,ULU,URRRDDLDRD,RRRLD,RUDDUDLDRL,DRDUUUDDU,UDRLU,UURDRLDDU,
    URDLDRRLL,LRDRULL,UUDRDDDLR,ULULLRR,UUL,DLUDLD,UUDRR,DLRDRD,DDRRLRDURR,UUULDRUDLL,RL,
    RRLLLRDLLD,DRRUU,RUDLDLRLLD,D,LLD,UDDU,DRDLUDD,L,RUULLD,DUD,LLLUDL,DDR,LLU,U,UDDURRDD
    UL,RLR,DRLDDRL,LDLR,RDLULRDLR,DDRR,UDLRDDLU,RRLLURDRL,RUDUU,RUDUDLDLL,LLRDDLDLDR,RDDL
    LLDU,DLRRLLLR,RDRURULL,RRULU,ULULU,RRDU,LD,RLUULUUDR,DULU,DLDUULLR,DRL,LLRDUDUDUL
ULLRRUUL,ULLLUUL,UDDULU,RUURUUULUD,ULURL,DULRRL,LDLDDURRDU,DULRDLRUDD,URLLULRLU,LLLRDDUD,
    ULL,RRURUUDU,LLRL,LRUULUU,ULR,DRRUURL,ULU,UU,DRUDDRR,RLRDLD,UDU,LUDRRRRR,UUDDULDU,LDD
    LURRLUU,LUUUD,DDDLRR,DRDUUDRRLL,RDRURLU,DRLU,RL,URUUL,RRULUURUUL,DDDRDDUDL,ULRDLRUR,U
    LLRRUURDU,UUUUU,RU,RDDLLUL,DUUL,UDUU,RLLRLLLDL,RURDRDRDRL,URUD,UUDRURURU,L,RURDUL,DDR
    RDDLR,UDRLRLDRLD,DLUUU,DRLRDDL,R,LLLULDU,U,UDLURRUD,RRRL,UDLRLUDD,LRLDRDRR,UDDRDRL,LU
    UUR,ULLLDDDDUL,RRLR,LRRL,LRLUDDLLRL,UUUU,DRRRLR,RULLUD,LUDUU,UUDDUL,RRLDRU,ULLL,DRLUU
    DRDRU,DDLURURLDL,RLRUURRUDR,RRRDLD,LUDDR
LL,RUURUU,RDUUUUL,DULRUDLUR,UDULRDRDR,DLUU,DL,LRLLRLR,ULRUULR,UULU,DR,UD,RUDULDLLDU,DU,RR
    R,RURULUR,DUUURURR,URLRDURDD,DDRRRD,UR,UU,RULDRRU,LLUULR,DLRRUURRDL,LRDD,UUUDUDUUL,LD
    RLU,DLLLU,DDUULULD,UULULLLRR,LRUUR,DDUDLR,DDLUL,UULD,DLUULLRULR,ULLLUDLRU,DRULUD,DRUU
    RRRR,UDDDRLLRR,RU,DDD,DULRDRR,RRLLURUD,L,LDD,DDURRRR,LRDRLLUDL,LDLRDDLLR,R,UURRDUD,UU
    UUDDD,RLU,DDUUUDUDDR,RDL,URUURRLL,LUDLLU,LDLDLR,DDLDLU,ULDDLLDR,ULDUDDRD,ULLRULL,DDDD
    DL,DRULDUL,RRLDRDLULL,RLUR,RDUDLLD,LRRURULDU,LUURUL,RLLUUUUL,LULLU,LDUDLRLR,LLDDUDDLL
    ,DURRRLUDU,LRRUURURR,DUUDRR
D,LRL,LDRUDDDD,LLDDRRULD,RURLDR,URUD,URUUDURUUU,R,RLRURLRDUD,LDLD,LDRDD,RUD,LUL,LUDDUDDDR
    D,DDDUUUD,RRLUDLU,UUUU,RUL,DLDD,ULLD,RUDUU,ULDDUDRR,LLUULLDD,UUULULL,RRLRUU,RUDLRU,LU
    DDL
```

~ Input file contains 52 lines of data, some lines long – see jared.dat file for complete input ~

~ Jared continued ~

Test Output To Screen:

```
There are 18 misinputs...
There are 30 misinputs...
Democracy Prevails!
There are 74 misinputs...
There are 92 misinputs...
There are 75 misinputs...
There are 82 misinputs...
There are 63 misinputs...
There are 12 misinputs...
There are 51 misinputs...
There are 97 misinputs...
There are 114 misinputs...
There are 93 misinputs...
There are 94 misinputs...
There are 82 misinputs...
There are 116 misinputs...
There are 2 misinputs...
There are 35 misinputs...
There are 24 misinputs...
Democracy Prevails!
There are 100 misinputs...
There are 112 misinputs...
There are 83 misinputs...
Democracy Prevails!
There are 133 misinputs...
There are 65 misinputs...
There are 99 misinputs...
There are 130 misinputs...
There are 126 misinputs...
Democracy Prevails!
There are 117 misinputs...
There are 4 misinputs...
There are 29 misinputs...
There are 43 misinputs...
Democracy Prevails!
There are 58 misinputs...
There are 84 misinputs...
There are 120 misinputs...
There are 119 misinputs...
There are 2 misinputs...
There are 38 misinputs...
There are 83 misinputs...
There are 106 misinputs...
There are 34 misinputs...
There are 119 misinputs...
There are 19 misinputs...
There are 67 misinputs...
There are 48 misinputs...
There are 21 misinputs...
There are 67 misinputs...
```

Problem #7
60 Points

7. Jimothy

Program Name: Jimothy.java   **Input File: jimothy.dat**

Test Input File: (***indented lines are continuation of previous line*** **)**

```
18
Never Gonna Give You Up
Somewhere Over The Rainbow Way Up High
Dont Stop Believing
Hello Its Me
I Was Wondering If After All These Years Youd Like To Meet
Somebody Once Told Me The World Was Macaroni
K A S D E W Q R G H Y U I O P M N B V C X Z L J F T
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
eij weofi wefnn fsverter pwqeork qmler xcvm poiuy lkjh fds rew bvcx gfds mnbv
     lkjh
M N O P Q R S F G H I J K L T U V W X Y Z A B C D E
YUI CVBN RINIM UNVME RIMV UIKMNBG IKMNGTRD ASDFGHJK ZXCVBN QWERTYUI MNBVCXRTYUIK
     OKJGRDCVHJI LKJUIOPOIUH JNBVCFRTREDSXCV
Somebody Once Told Me The World Was Macaroni I Aint The Sharpest Tool In The
     Shed
E F G H I J K L M N O P Q R A B C D S T U V W X Y Z
L M N O P Q R S T U V W X Y Z A B C D E F G H I J K
A B U V W X Y C N O P Q E F G H I R S T Z D J K L M
qwer wert rt rty tyu tyui yui uio iop asd sdf sdf gdfg hfg hghj kjk ljk zxc xcv
     cvb vbn bnm jkl hjk ghj fgh dfg sdf asd qwe wer ert rty tyu yui uio iop
A
B C A
```

Test Output To Screen: (***indented lines are continuation of previous line*** **)**

```
TEST CASE #1:
PRE-ORDER TRAVERSAL: Never Gonna Give You Up
IN-ORDER TRAVERSAL: Give Gonna Never Up You
POST-ORDER TRAVERSAL: Give Gonna Up You Never
REVERSE-ORDER TRAVERSAL: You Up Never Gonna Give
TEST CASE #2:
PRE-ORDER TRAVERSAL: Somewhere Over High Rainbow The Way Up
IN-ORDER TRAVERSAL: High Over Rainbow Somewhere The Up Way
POST-ORDER TRAVERSAL: High Rainbow Over Up Way The Somewhere
REVERSE-ORDER TRAVERSAL: Way Up The Somewhere Rainbow Over High
TEST CASE #3:
PRE-ORDER TRAVERSAL: Dont Believing Stop
IN-ORDER TRAVERSAL: Believing Dont Stop
POST-ORDER TRAVERSAL: Believing Stop Dont
REVERSE-ORDER TRAVERSAL: Stop Dont Believing
```

~ ***Output continues next page…*** ~

~ Jimothy continued ~

~ Output continued from previous page… ~ **(*indented lines are continuation of previous line*)**

```
TEST CASE #4:
PRE-ORDER TRAVERSAL: Hello Its Me
IN-ORDER TRAVERSAL: Hello Its Me
POST-ORDER TRAVERSAL: Me Its Hello
REVERSE-ORDER TRAVERSAL: Me Its Hello
TEST CASE #5:
PRE-ORDER TRAVERSAL: I After All Was If These Like Meet To Wondering Years Youd
IN-ORDER TRAVERSAL: After All I If Like Meet These To Was Wondering Years Youd
POST-ORDER TRAVERSAL: All After Meet Like To These If Youd Years Wondering Was I
REVERSE-ORDER TRAVERSAL: Youd Years Wondering Was To These Meet Like If I All
    After
TEST CASE #6:
PRE-ORDER TRAVERSAL: Somebody Once Me Macaroni Told The World Was
IN-ORDER TRAVERSAL: Macaroni Me Once Somebody The Told Was World
POST-ORDER TRAVERSAL: Macaroni Me Once The Was World Told Somebody
REVERSE-ORDER TRAVERSAL: World Was Told The Somebody Once Me Macaroni
TEST CASE #7:
PRE-ORDER TRAVERSAL: K A D B C E G F H I J S Q O M L N P R W U T V Y X Z
IN-ORDER TRAVERSAL: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
POST-ORDER TRAVERSAL: C B F J I H G E D A L N M P O R Q T V U X Z Y W S K
REVERSE-ORDER TRAVERSAL: Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
TEST CASE #8:
PRE-ORDER TRAVERSAL: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
IN-ORDER TRAVERSAL: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
POST-ORDER TRAVERSAL: Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
REVERSE-ORDER TRAVERSAL: Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
TEST CASE #9:
PRE-ORDER TRAVERSAL: eij bvcx weofi wefnn fsverter fds pwqeork poiuy lkjh gfds
    lkjh mnbv qmler rew xcvm
IN-ORDER TRAVERSAL: bvcx eij fds fsverter gfds lkjh lkjh mnbv poiuy pwqeork
    qmler rew wefnn weofi xcvm
POST-ORDER TRAVERSAL: bvcx fds lkjh gfds mnbv lkjh poiuy rew qmler pwqeork
    fsverter wefnn xcvm weofi eij
REVERSE-ORDER TRAVERSAL: xcvm weofi wefnn rew qmler pwqeork poiuy mnbv lkjh lkjh
    gfds fsverter fds eij bvcx
TEST CASE #10:
PRE-ORDER TRAVERSAL: M F A B C D E G H I J K L N O P Q R S T U V W X Y Z
IN-ORDER TRAVERSAL: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
POST-ORDER TRAVERSAL: E D C B A L K J I H G F Z Y X W V U T S R Q P O N M
REVERSE-ORDER TRAVERSAL: Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
TEST CASE #11:
PRE-ORDER TRAVERSAL: YUI CVBN ASDFGHJK RINIM RIMV IKMNGTRD QWERTYUI MNBVCXRTYUIK
    LKJUIOPOIUH JNBVCFRTREDSXCV OKJGRDCVHJI UNVME UIKMNBG ZXCVBN
IN-ORDER TRAVERSAL: ASDFGHJK CVBN IKMNGTRD JNBVCFRTREDSXCV LKJUIOPOIUH
    MNBVCXRTYUIK OKJGRDCVHJI QWERTYUI RIMV RINIM UIKMNBG UNVME YUI ZXCVBN
```

~ Output continues next page… ~

~ Jimothy continued ~

~ Output continued from previous page… ~ (indented lines are continuation of previous line)

```
POST-ORDER TRAVERSAL: ASDFGHJK JNBVCFRTREDSXCV LKJUIOPOIUH OKJGRDCVHJI
     MNBVCXRTYUIK QWERTYUI IKMNGTRD RIMV UIKMNBG UNVME RINIM CVBN ZXCVBN YUI
REVERSE-ORDER TRAVERSAL: ZXCVBN YUI UNVME UIKMNBG RINIM RIMV QWERTYUI
     OKJGRDCVHJI MNBVCXRTYUIK LKJUIOPOIUH JNBVCFRTREDSXCV IKMNGTRD CVBN ASDFGHJK
TEST CASE #12:
PRE-ORDER TRAVERSAL: Somebody Once Me Macaroni I Aint In Sharpest Shed Told The
     The The World Was Tool
IN-ORDER TRAVERSAL: Aint I In Macaroni Me Once Sharpest Shed Somebody The The
     The Told Tool Was World
POST-ORDER TRAVERSAL: Aint In I Macaroni Me Shed Sharpest Once The The The Tool
     Was World Told Somebody
REVERSE-ORDER TRAVERSAL: World Was Tool Told The The The Somebody Shed Sharpest
     Once Me Macaroni In I Aint
TEST CASE #13:
PRE-ORDER TRAVERSAL: E A B C D F G H I J K L M N O P Q R S T U V W X Y Z
IN-ORDER TRAVERSAL: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
POST-ORDER TRAVERSAL: D C B A Z Y X W V U T S R Q P O N M L K J I H G F E
REVERSE-ORDER TRAVERSAL: Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
TEST CASE #14:
PRE-ORDER TRAVERSAL: L A B C D E F G H I J K M N O P Q R S T U V W X Y Z
IN-ORDER TRAVERSAL: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
POST-ORDER TRAVERSAL: K J I H G F E D C B A Z Y X W V U T S R Q P O N M L
REVERSE-ORDER TRAVERSAL: Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
TEST CASE #15:
PRE-ORDER TRAVERSAL: A B U C N E D F G H I J K L M O P Q R S T V W X Y Z
IN-ORDER TRAVERSAL: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
POST-ORDER TRAVERSAL: D M L K J I H G F E T S R Q P O N C Z Y X W V U B A
REVERSE-ORDER TRAVERSAL: Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
TEST CASE #16:
PRE-ORDER TRAVERSAL: qwer iop asd asd gdfg cvb bnm fgh dfg ert hfg ghj hghj hjk
     iop kjk jkl ljk qwe wert rt rty rty tyu sdf sdf sdf tyu tyui uio uio vbn
     wer yui xcv yui zxc
IN-ORDER TRAVERSAL: asd asd bnm cvb dfg ert fgh gdfg ghj hfg hghj hjk iop iop
     jkl kjk ljk qwe qwer rt rty rty sdf sdf sdf tyu tyu tyui uio uio vbn wer
     wert xcv yui yui zxc
POST-ORDER TRAVERSAL: asd bnm ert dfg fgh cvb ghj iop hjk hghj hfg gdfg asd jkl
     qwe ljk kjk iop rty sdf sdf tyu sdf uio wer vbn uio tyui tyu rty rt yui xcv
     zxc yui wert qwer
REVERSE-ORDER TRAVERSAL: zxc yui yui xcv wert wer vbn uio uio tyui tyu tyu sdf
     sdf sdf rty rty rt qwer qwe ljk kjk jkl iop iop hjk hghj hfg ghj gdfg fgh
     ert dfg cvb bnm asd asd
TEST CASE #17:
PRE-ORDER TRAVERSAL: A
IN-ORDER TRAVERSAL: A
POST-ORDER TRAVERSAL: A
REVERSE-ORDER TRAVERSAL: A
TEST CASE #18:
PRE-ORDER TRAVERSAL: B A C
IN-ORDER TRAVERSAL: A B C
POST-ORDER TRAVERSAL: A C B
REVERSE-ORDER TRAVERSAL: C B A
```

Problem #8
60 Points

8. Paola

Program Name: Paola.java **Input File: paola.dat**

Test Input File:

```
12
3 3 4
1 2 3
4 5 6
7 8 9
ADDROW 1 [10, 11, 12]
GETCOL 1
DELCOL 0
SETVAL 1 4 0
2 4 3
1 2 3 4
5 6 7 8
ADDCOL 2 [10, 11, 12]
SETVAL 0 1 9
ROTCOL 2 1
5 5 5
0 9 10 19 20
1 8 11 18 21
2 7 12 17 22
3 6 13 16 23
4 5 14 15 24
DELROW 3
DELCOL 1
ROTROW 2 3
GETROW 0
ROTCOL 0 -2
5 5 5
0 9 10 19 20
1 8 11 18 21
2 7 12 17 22
3 6 13 16 23
4 5 14 15 24
GETCOL 4
ROTROW 3 12
ROTCOL 1 -8
ROTROW 0 -4
DELCOL 2
5 5 5
0 9 10 19 20
1 8 11 18 21
2 7 12 17 22
3 6 13 16 23
4 5 14 15 24
DELROW 3
DELROW 2
DELROW 1
DELROW 0
DELCOL 2
```

~ Input continues next column… ~

~ Input continued from previous column… ~

```
2 2 8
1 2
3 4
ADDCOL 1 [5, 6]
ADDROW 0 [7, 8, 9]
ADDROW 0 [10, 11, 12, 13]
ADDCOL 0 [14, 15]
GETROW 5
GETCOL -2
DELROW -4
DELCOL 9
2 2 8
1 2
3 4
ADDCOL 1 [5, 6, 7]
ADDROW 0 [8]
ADDROW 0 [10, 11]
ADDCOL 0 [14, 15, 16]
SETVAL 0 6 9
SETVAL 6 0 7
SETVAL -3 -4 2
SETVAL 1 1 212
5 5 2
0 9 10 19 20
1 8 11 18 21
2 7 12 17 22
3 6 13 16 23
4 5 14 15 24
ADDCOL 9 [212]
ADDROW -5 [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
5 5 5
0 9 10 19 20
1 8 11 18 21
2 7 12 17 22
3 6 13 16 23
4 5 14 15 24
DELCOL 0
DELCOL 0
DELCOL 0
DELCOL 0
DELROW 2
```

~ Input continues next page… ~

~ Paola continued… ~

~ Input continues from previous page… ~

```
10 10 15
0 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 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
DELROW 5
DELCOL 4
DELROW 7
DELCOL 2
ROTROW 5 -3
ROTCOL 1 104
DELROW 2
DELCOL 4
SETVAL 0 0 212
ROTCOL 2 -76
ROTROW 3 345
DELROW 1
DELCOL 1
GETROW 2
ADDCOL 3 [100, 101, 102, 103, 104, 105]
```

~ Input continues next column… ~

~ Input continued from previous column… ~

```
10 10 10
0 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 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
ROTROW 1 2
ROTROW 2 3
ROTROW 3 4
ROTROW 4 5
ROTROW 5 6
ROTROW 6 7
ROTROW 7 8
ROTROW 8 9
ROTROW 9 10
ROTROW 0 11
5 5 6
2 54 765 43 877
342 32 1 234 5
87 987 9876 65 4
123 234 3444 3434 44
7 4 5 6 7
GETROW 4
ADDCOL 3 [1, 2, 3, 4, 5]
DELROW 2
SETVAL 2 2 39
ROTCOL 1 2
DELCOL 0
```

~ Output shown on next page… ~

~ Paola output continued… ~

Test Output To Screen:

```
[2, 11, 5, 8]
INDEX OUT OF BOUNDS.
2 3
11 12
5 6
8 9
----------
COLUMN [10, 11, 12] OF INVALID LENGTH
1 9 7 4
5 6 3 8
----------
[0, 10, 19, 20]
12 10 19 20
4 11 18 21
0 17 22 2
1 14 15 24
----------
[20, 21, 22, 23, 24]
20 0 10 19
1 5 18 21
2 9 17 22
16 8 6 13
4 7 15 24
----------
4 5 15 24
----------
ROW [10, 11, 12, 13] OF INVALID LENGTH
COLUMN [14, 15] OF INVALID LENGTH
INDEX OUT OF BOUNDS.
INDEX OUT OF BOUNDS.
INDEX OUT OF BOUNDS.
INDEX OUT OF BOUNDS.
7 8 9
1 5 2
3 6 4
----------
COLUMN [5, 6, 7] OF INVALID LENGTH
ROW [8] OF INVALID LENGTH
INDEX OUT OF BOUNDS.
INDEX OUT OF BOUNDS.
INDEX OUT OF BOUNDS.
14 10 11
15 212 2
16 3 4
----------
```

~ Output continues next column… ~

~ Output continued from previous column … ~

```
INDEX OUT OF BOUNDS.
COLUMN [212] OF INVALID LENGTH
INDEX OUT OF BOUNDS.
ROW [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] OF
     INVALID LENGTH
```
~ this line is continuation ~

```
0 9 10 19 20
1 8 11 18 21
2 7 12 17 22
3 6 13 16 23
4 5 14 15 24
----------
20
21
23
24
----------
[48, 40, 41, 33, 45, 47]
212 93 5 100 7 8 9
30 13 35 101 37 38 39
48 40 41 102 33 45 47
65 43 68 103 60 61 63
70 67 75 104 77 78 79
90 73 95 105 97 98 99
----------
9 0 1 2 3 4 5 6 7 8
18 19 10 11 12 13 14 15 16 17
27 28 29 20 21 22 23 24 25 26
36 37 38 39 30 31 32 33 34 35
45 46 47 48 49 40 41 42 43 44
54 55 56 57 58 59 50 51 52 53
63 64 65 66 67 68 69 60 61 62
72 73 74 75 76 77 78 79 70 71
81 82 83 84 85 86 87 88 89 80
90 91 92 93 94 95 96 97 98 99
----------
[7, 4, 5, 6, 7]
234 765 1 43 877
4 1 2 234 5
54 39 4 3434 44
32 5 5 6 7
```

Problem #10
60 Points

10. Riley

Program Name: Riley.java **Input File: riley.dat**

Test Input File:

```
7
7 7
     1    24    23    22    21    20    19
     2    25    40    39    38    37    18
     3    26    41    48    47    36    17
     4    27    42    49    46    35    16
     5    28    43    44    45    34    15
     6    29    30    31    32    33    14
     7     8     9    10    11    12    13
3 6
    40    90   -42    21    97    31
    21   -28   -84    67   -85   -67
   -30   -55   -36   -99    35   -22
5 2
    11   -81
    26    86
    71   -23
   -68     6
   -62    48
2 2
   -29    10
    84    61
15 15
   -73   -75    26    -4   -62   -73    48   -50    23    39   -46   -70     8   -55   -50
    46    96   -96   -51   -32    -3   -99    81    50    60   -92   -90   -83   -39   -99
   -90    -7   -46   -25   -65   -15   -90    68    91    50    19   100    35    42    -6
    57   -23    98    93   -45    88    -7     7   -64   -54   -70    19   -63    64   -32
    -3   -58   -77   -57    69    38    -2    78    20    20   -48     1   -84     3    97
    65  -100   -97   -59   -66    49   -79    49   -49    -1   -54    55    67    50    71
    64   -65   -51   -52   -70    27    91   -94   -84   -29   -96    79   -63   -28    77
    18    19   -25   -69   -99   -34   100   -54   -50   -51    77     5    61   -94   -93
    41   -93   -48   -90    26   -20   -39    93   -92    86   -62   -54     8    62    70
   -13    39    17    25     0    -5    44    88     4    13    -8    -5    -2    71    27
    15    86   -54   -97   -88   -53   -77    -1    44    10   -31    73   -45   -74   -24
   -21   -47    22    73    57    97    21   -87    77    16   -66   -33    37   -84   -65
   -23   -13    96    35   -73     3    25     6   -70    73    67    83    58    31    35
    36    74   -38   -24   -19    49    84    10    97   -78    57   -30    64    54   -57
   -86  -100    50    -4    34   -56   -37    -9   -40   -90    15    94   -40    79   -53
2 7
     5    -2   -51   -11     4   -65   -88
   -53    81    86    90    46     1   -87
3 3
   -25    26    13
    81    91   -53
   -48   -95   -26
```

Test Output To Screen:

```
12.50 32.50 44.50 49.00
-1.14 -32.50
1.40
31.50
-8.29 -4.52 3.15 -3.84 -15.54 5.75 -9.38 -54.00
-3.14
-15.88 91.00
```

Problem #9
60 Points

9. Sasha

Program Name: Sasha.java **Input File: sasha.dat**

Test Input File (*indented lines are continuation of previous line*):

```
100
57
21 61 10 496 578 4 530 0 9618 1992 3437 89 162 77 9 31 893 6 6631 8 85 999 0 1 6
     2416 339 5 814 549 9 6358 39 54 8216 20 4 67 0 16 7 1110 7 208 8 14 14 8608
     267 9175 1 7 613 34 2874 3889 8307
49
476 94 302 671 1865 65 78 2012 5 6919 8506 50 1 408 80 7132 44 2620 62 94 8 779
     97 26 8 3242 872 5 8882 756 8 4402 3070 9711 70 6 914 574 0 4390 152 88
     2281 88 0 2 82 865 648
188
35 590 1439 63 4 44 1 72 3216 5244 400 8 215 397 8368 76 2642 61 7 8019 352 791
     37 60 8858 63 1256 0 149 9093 775 4 477 4664 3 163 5345 7 5 863 1 56 598
     6025 1 38 5 98 0 33 9 8 6 94 3 554 93 0 59 1 77 9610 4226 16 7 329 64 2 71
     5 910 373 933 5 6842 85 268 754 82 267 60 0 6706 7 8899 9984 9 90 61 87 565
     76 4 5630 5 757 1 172 1 5302 729 588 2 8784 505 0 719 6937 7 5066 6546 7
     671 5210 13 8795 882 930 6 0 707 55 132 6458 28 9 9 77 414 13 7468 5753
     1820 0 1644 6 117 239 9 22 957 1 748 3 4084 2 1 0 843 5 598 74 3858 12 42
     5368 50 34 21 7679 4 56 1144 50 67 6 3200 64 9 88 529 1 368 3502 9 160 4942
     620 49 93 8586 507 8 806 32 7 2 65
127
7253 942 3 913 486 1 0 7 8549 602 4 80 63 77 31 3017 17 572 7 76 2 4270 327 603
     5 48 2 5 7118 87 4722 5211 7 58 55 45 92 478 84 4 29 665 90 7 72 548 374
     943 6 6765 793 1685 28 96 3 366 309 6020 256 1040 3064 928 6429 5582 8729
     9301 7 32 9151 54 708 35 1 3711 5966 4804 1 621 69 59 24 1 160 71 4 665 52
     2 9 0 0 64 58 684 55 844 45 51 113 20 199 8610 5109 3002 7263 7 7530 4175
     79 257 769 22 654 376 234 7863 6 6551 7 259 9275 888 794 85 26 3126 5580
35
10 7 26 18 183 30 1823 58 6 6 3 18 9019 631 3002 3175 28 6 8 3319 36 352 1243
     847 9 9046 0 5772 3844 2 5626 95 8162 766 206
72
2 166 590 220 1 83 7 38 6 5 26 53 5842 6303 9 709 188 9 5 2 79 991 1989 37 457
     1750 9 427 291 621 1173 9 584 3 640 16 81 3973 5 14 0 6 3058 8645 4 0 459
     6819 265 9 441 5654 3567 6 67 190 889 6 9262 728 303 0 33 516 0 7 912 6407
     271 59 5 517
122
179 7504 26 0 0 4 377 2 5 7558 66 3030 43 28 5719 1059 268 359 1026 9547 470 4 9
     751 8007 9 1 305 21 3427 6999 31 772 6 7104 752 52 42 8 4 8 1 0 938 5392
     864 57 2179 8 5 266 9 2 533 5744 0 551 867 354 5 9 17 171 6123 1 50 1 220
     970 8 5843 31 6427 6 2 1295 166 0 2215 94 20 3 0 385 8 0 4 601 85 23 2589 4
     9 68 406 97 7 1 482 2 9385 232 2516 4 531 267 13 505 79 551 4269 26 8560 97
     5885 5 9 654 62 3899 5 1176
```

~ Input file contains 201 lines of data, some lines extremely long – see sasha.dat file for complete input ~

~ Sasha continued ~

Test Output To Screen:
```
51 1200 323425 42424 367100
556 3587 2007 129074 27292 20386 4322 3288 190512
7011 19956 63267 33964 261818 190119 206511 483365 1097 154 1267262
48419 406328 419737 22981 897465
3 8 153 9024 95572 33633 138393
4871 165652 102654 48757 1216 323150
52114 56424 675800 85167 5788 14866 1171 891330
1644 175907 98554 120915 437447 247371 25110 80508 5750 13 1193219
105 61755 297195 463234 116700 51238 4934 995161
45013 619835 551226 37 1216111
93 23406 15873 889 40261
6480 97416 384202 297424 315987 8549 1110058
261 3522 2303 21468 48298 422066 10454 372 83 7 508834
86 86871 420152 226767 78886 22700 608 836070
30755 1913 2611 405 35684
62157 386438 448595
Dry as a Bone...
7021 36664 96144 17376 17031 8216 240 182692
15986 454104 913 39 1 471043
1422 16510 5058 1021403 162931 22048 245 902 1230519
23310 322512 89699 29447 29595 638 495201
15438 20780 2308 6501 87 63 45177
1857 6755 8612
87 5037 2983 19762 27869
4956 146999 125848 21791 25907 38028 363529
376658 1074133 66929 240 6 1517966
3104 2308 125722 34475 454127 380382 142480 968 1143566
27762 500525 102401 184788 13801 132859 74627 1036763
1755 14491 7070 1155247 163286 1341849
134776 1033103 37758 54 1205691
1439 18804 41829 286054 391983 519078 79639 14240 21303 1374369
1877 4073 340657 919121 23737 7356 57001 49153 1402975
10657 4562 355138 361 9000 379718
9 2 42 634816 292089 23832 95 75 950960
28524 149519 229733 131965 674 540415
30 262 477 115723 78096 337536 359851 362902 2253 32 1257162
Dry as a Bone...
1781 84 1865
228458 647 229105
117 10887 6144 17128 337539 618494 76417 6179 1072905
1055 26572 26769 128172 18334 200902
180 609 3607 4396
7111 65176 176042 17108 917651 1183088
71001 115681 81709 217152 52169 631 538343
6506 161405 361667 584704 775 1115057
115 1093 79 1287
177 928 138323 25706 246663 109104 188247 6064 15857 731069
126148 1028384 8012 551 475 1163570
1932 358519 46998 257582 35436 6181 706648
12118 45154 5628 34264 60036 775571 1937 934708
8072 33314 3016 55639 61264 46894 21990 527305 12554 790 102 770940
```

~ Output continues next page… ~

~ Sasha continued ~

~ Output continued from previous page… ~

```
6345 65397 524184 71066 60 667052
80 107 22165 184132 74107 54054 334645
194822 158639 124560 33675 1959 513655
23 2778 85486 37737 46269 657 259 173209
3811 2158 2790 310746 107346 50127 53704 11119 541801
39810 137353 109159 73169 82789 73404 515684
911 59244 8105 24836 287776 7698 87759 476329
48305 13237 23820 203219 112313 48431 11217 460542
418 9055 11671 1454 22598
3248 7200 3147 9450 7250 5623 59 35977
141188 44727 117840 242652 27568 573975
3 6704 108279 133923 351151 79080 2027 681167
1 40954 36810 354822 121974 1458 556019
38147 42555 80702
83729 535843 78786 34892 49 46 733345
236 10510 44430 114166 59437 228779
14 3280 68687 606889 109666 1337 123 789996
76 91386 77229 182094 255156 209588 103555 57216 976300
20010 528733 611075 63909 30814 7634 2829 1265004
195677 429782 625459
242631 943314 81257 49982 6089 43875 1208 1368356
18780 31039 20785 86912 27015 25073 32388 77574 21373 2869 343808
2092 28244 18236 48572
6798 91088 43992 15930 1068797 1226605
54573 5379 26025 9388 48819 252152 120288 97481 1227 615332
1750 728 25349 266454 300500 563823 144659 1966 1305229
59513 126862 4156 190531
52844 59115 102701 214660
275752 407180 18232 594907 51764 82 1347917
428 2692 138708 145502 194271 105548 19336 23268 629753
3 304 36123 153725 201 3 190359
117415 445514 85412 5116 663 159 81 654360
1445 5469 56016 445198 407638 3309 919075
585 1656 70619 144921 124372 26472 527 369152
4566 145815 93326 289425 84207 672202 23174 1312715
677411 392040 149094 76269 21833 5234 825 75 1322781
Dry as a Bone...
193004 27459 444795 199578 39910 48 904794
5 84704 125280 22943 232932
45 29275 70485 240593 492216 18080 8344 859038
299 468 767
703 66757 34303 126249 897682 111348 864 1664 1239570
3674 87391 136979 412671 28211 155796 50350 26676 901748
504 1426 6017 358864 193865 27665 10069 598410
1975 86990 473150 284572 35925 46805 67662 1582 998661
188 311896 987289 7154 39986 1354 1347867
20924 167479 23048 12489 38202 262142
21018 98647 468111 116813 10133 261 714983
895 9700 3543 6233 4 20375
```

Problem #11
60 Points

11. Svetlana

Program Name: Svetlana.java **Input File: svetlana.dat**

Test Input File:
```
8
--------
6 4 7
8 5 -1
3 2 1
--------
8 6 7
2 5 4
3 -1 1
--------
-1 8 7
6 5 4
3 2 1
--------
2 3 -1
1 4 6
8 7 5
--------
8 1 2
4 5 3
7 -1 6
--------
3 1 2
6 5 -1
8 7 4
--------
2 -1 3
1 4 6
8 7 5
--------
4 2 3
1 -1 6
8 7 5
```

Test Output To Screen:
```
Number of steps needed to solve: 31
Number of steps needed to solve: 31
Number of steps needed to solve: 28
Number of steps needed to solve: 20
Number of steps needed to solve: 13
Number of steps needed to solve: 25
Number of steps needed to solve: 19
No solution exists.
```

Problem #12
60 Points

12. Wesley

Program Name: Wesley.java **Input File: wesley.dat**

Test Input File:

```
10
8 8
S..1.5.#
#.3.33.5
1.####35
3355E###
...###..
..3..15.
##.3...1
5....53.
6 6
S35#.5
11..5.
5..33.
##135#
##513#
E.351#
1 8
S#3#51#E
1 8
S#3.51#E
```

~ Input continues next column ~

~ Input continued ~

```
1 8
S133511E
1 8
S133531E
8 8
S#.3.5.1
#.3.5.1.
.3.5.1.#
3.5.1.#.
.5.1.#.3
5.1.#.3.
.1.#.3.5
1.#.3.5E
6 6
S..1.5
.#..31
531.##
E531.#
531.##
.#..35
```

~ Input continues next column ~

~ Input continued ~

```
8 8
S.#3#5#1
.#3#5#1#
#3#5#1#.
3#5#1#.#
#5#1#.#3
5#1#.#3#
#1#.#3#5
1#.#3#5E
10 10
531..#S3#1
5..3#1..#.
1313.51.3#
531##..#31
55.#31##..
3..351##.#
E#.135##.#
5..513##.#
33##15#..#
1515..#.51
```

Test Output To Screen:

```
The Great Escape.
Guess I won't be home in time for dinner.
The Great Escape.
Guess I won't be home in time for dinner.
The Great Escape.
Guess I won't be home in time for dinner.
Guess I won't be home in time for dinner.
The Great Escape.
Guess I won't be home in time for dinner.
The Great Escape.
```