# Computer Science Competition
# State 2022
# 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 | Andrei |
| Problem 2 | Charles |
| Problem 3 | Dmitry |
| Problem 4 | Fatima |
| Problem 5 | Frances |
| Problem 6 | Honghui |
| Problem 7 | Manuel |
| Problem 8 | Michelle |
| Problem 9 | Prateek |
| Problem 10 | Richard |
| Problem 11 | Sanjay |
| Problem 12 | Urvashi |

# 1. Andrei

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

Andrei really enjoys working with prime numbers. He appreciates their characteristics and their importance to all manners of mathematics. In fact, on his wall he has the following chart in which the first 25 prime numbers are separated by their tens digit.

Your job is to duplicate that chart.

**Input:**
None

**Output:**
A list of the first 25 prime numbers separated into "tens."

**Sample input:**
None

**Sample output:**
```
2 3 5 7
11 13 17 19
23 29
31 37
41 43 47
53 59
61 67
71 73 79
83 89
97
```

## 2. Charles

**Program Name: Charles.java** **Input File: charles.dat**

Charles is curious about some of the properties of factors. To begin his study, Charles needs a program that will allow him to find the sum of all of the factors of an integer and display those factors in the form of a sum of integers. From there, he will proceed with the investigation.

**Input:**
The first line consists of a number N, representing the number of lines of data to follow. N will be in the range of [1,50]. The next N lines of data consist of one whole number in the range of [1,100].

**Output:**
The output will list the factors of the input listed in ascending order separated by a plus sign. At the end of the list of factors, there will be one equals mark followed by the sum of the factors. There will be no spaces in the output.

If there is only one factor, there of course will be no plus signs.

**Sample input:**
```
5
30
28
25
1
17
```

**Sample output:**
```
1+2+3+5+6+10+15+30=72
1+2+4+7+14+28=56
1+5+25=31
1=1
1+17=18
```

# 3. Dmitry

**Program Name: Dmitry.java** **Input File: dmitry.dat**

Dmitry has had pancakes for breakfast every morning since he was three. Because of this, he is obsessed with stacking things. He noticed that the pancakes on his plate are rarely the same size. So the cooks always put the smaller pancakes on top of larger pancakes. Thus, the smallest is on top and the pancake sizes increase in size all of the way to the bottom of the stack.

Dmitry wants a program that will read in a list of integers. The program will sort those integers and display them with X's from the smallest value until the largest.

Example, if the numbers were, 1, 7, 2 and another 2, the numbers would be rearranged to create 1,2,2,7. This stack will be represented by 1X, 2Xs, 2Xs, and finally 7Xs.
X
XX
XX
XXXXXXX

**Input:**
The first line consists of a number N, representing the number of lines of data to follow. N will be in the range of [1,50]. The next N lines of data consist of at least one whole number, but not more than 20. And each number in the list will be in the range [1,25].

**Output:**
Each data set will be represented by a series of lines consisting of upper case Xs.

**Sample input:**
```
5
1 2 3 4 5 6 7 8 9 10
5 4 3 2 1
2 4 6 7 5 3
12
4 4 6 6 8 8 1 1
```

**Sample output:**
```
X
XX
XXX
XXXX
XXXXX
XXXXXX
XXXXXXX
XXXXXXXX
XXXXXXXXX
XXXXXXXXXX
X
XX
XXX
XXXX
XXXXX
XX
XXX
XXXX
XXXXX
XXXXXX
XXXXXXX
XXXXXXXXXXXX
X
X
XXXX
XXXX
XXXXXX
XXXXXX
XXXXXXXX
XXXXXXXX
```

## 4. Fatima

**Program Name: Fatima.java** **Input File: fatima.dat**

Fatima has been working with nested loops but has encountered a situation she cannot quite solve. She wants to produce heat index charts and has discovered that they are "jagged" as shown in the sample output.

Fatima found the following formula on the Web where ambient air temperature (t) is measured in °F while relative humidity (h) is a percentage and Heat Index is abbreviated HI:

$$\begin{aligned} HI = & -42.379 + 2.04901523t + 10.14333127h - 0.22475541th - 6.83783 \cdot 10^{-3}t^2 \\ & - 5.481717 \cdot 10^{-2}h^2 + 1.22874 \cdot 10^{-3}t^2h + 8.5282 \cdot 10^{-4}th^2 - 1.99 \cdot 10^{-6}t^2h^2 \end{aligned}$$

However, the above formula has some limitations and under certain conditions must be adjusted with an additional calculation. When the calculated heat index is over 140 °F it is considered invalid and should not be displayed. When humidity is greater than 85%, the following adjustment must be added to HI:

ADJUSTMENT = ( (h – 85.0) / 10 ) * ( (87.0 – t) / 5.0 )

There are some other special situations but they will be ignored for this program.

Can you create a heat index chart program for Fatima?

**Input:** First line of data file contains a positive integer N, the number of test cases that follow with $1 \leq N \leq 10$. Each test case contains just one line with two floating point numbers separated by a space: step size, A, for the air temp labels across the top of the chart and step size, H, for the humidity labels down the left side of the chart with $1.0 \leq A, H \leq 10.0$.

**Output:** For each test case, display a heat index chart formatted as shown in the samples. The air temp labels across the top always start with 80.0 and do not exceed 125.0. The line of equal signs below the air temp label line always contains 95 equal signs. The humidity labels on down the left always start with 20.0 and do not exceed 100.0. The final values may be less than maximums depending on step sizes. All values in the chart are displayed with 1 digit after decimal point. After the final line of the chart, display a line containing 30 equal signs "==============================".

**Sample input:**
```
3
5.0 5.0
4.5 7.5
3.0 10
```

***~ Sample output on next page ~***

***Fatima, continued***

**Sample output:**

```
Temp   80.0  85.0  90.0  95.0 100.0 105.0 110.0 115.0 120.0 125.0
Humid ===============================================================================================
 20.0  78.6  82.0  86.3  91.5  97.5 104.3 112.0 120.5 129.9
 25.0  78.9  82.4  87.0  92.7  99.6 107.6 116.8 127.1 138.5
 30.0  79.2  82.9  87.9  94.4 102.3 111.6 122.3 134.5
 35.0  79.5  83.5  89.2  96.5 105.5 116.2 128.6
 40.0  79.9  84.3  90.7  99.0 109.3 121.5 135.7
 45.0  80.3  85.3  92.5 101.9 113.5 127.4
 50.0  80.8  86.5  94.6 105.2 118.3 133.9
 55.0  81.3  87.8  97.0 108.9 123.6
 60.0  81.8  89.3  99.7 113.1 129.5
 65.0  82.4  90.9 102.7 117.6 135.9
 70.0  83.0  92.7 105.9 122.6
 75.0  83.6  94.7 109.5 128.0
 80.0  84.2  96.8 113.3 133.8
 85.0  84.9  99.1 117.5 140.0
 90.0  86.3 101.8 121.6
 95.0  87.8 104.6 126.0
100.0  89.3 107.6 130.7
==============================
Temp   80.0  84.5  89.0  93.5  98.0 102.5 107.0 111.5 116.0 120.5 125.0
Humid ===============================================================================================
 20.0  78.6  81.7  85.4  89.8  95.0 100.8 107.3 114.5 122.3 130.9
 27.5  79.0  82.2  86.4  91.6  97.8 105.1 113.4 122.7 133.1
 35.0  79.5  83.0  87.9  94.1 101.7 110.7 121.0 132.7
 42.5  80.1  84.2  90.0  97.5 106.7 117.6 130.1
 50.0  80.8  85.8  92.8 101.8 112.8 125.8
 57.5  81.5  87.7  96.1 106.9 119.9 135.3
 65.0  82.4  89.9 100.0 112.8 128.2
 72.5  83.3  92.5 104.6 119.6 137.5
 80.0  84.2  95.4 109.7 127.2
 87.5  85.6  98.8 115.3 135.4
 95.0  87.8 102.7 121.4
==============================
Temp   80.0  83.0  86.0  89.0  92.0  95.0  98.0 101.0 104.0 107.0 110.0 113.0 116.0 119.0 122.0 125.0
Humid ===============================================================================================
 20.0  78.6  80.6  82.8  85.4  88.3  91.5  95.0  98.8 102.9 107.3 112.0 117.0 122.3 128.0 133.9
 30.0  79.2  81.2  83.8  86.8  90.4  94.4  99.0 104.1 109.6 115.7 122.3 129.4 137.1
 40.0  79.9  82.3  85.4  89.3  93.8  99.0 104.9 111.5 118.9 126.9 135.7
 50.0  80.8  83.9  87.9  92.8  98.5 105.2 112.8 121.2 130.6
 60.0  81.8  85.9  91.1  97.4 104.7 113.1 122.6 133.1
 70.0  83.0  88.4  95.1 103.0 112.2 122.6 134.3
 80.0  84.2  91.3  99.8 109.7 121.0 133.8
 90.0  86.3  95.1 105.4 117.3 130.8
100.0  89.3  99.7 111.8 125.7
==============================
```

## 5. Frances

**Program Name: Frances.java** **Input File: frances.dat**

Frances's mom works for the local university's academic advising office. As part of her job, she must evaluate degree plans students submit. A degree plan is a student's plan of action for completing a degree and constitutes the order in which he or she plans to take given courses. At the university level, many upper level courses have prerequisites that must be completed. For example, for a computer science major to be able to take a senior level operating systems course, the student must complete a course on data structures and algorithms. But for the student to take data structures and algorithms, the student must take computer science II, but to take computer science II, the student must complete computer science I. Not all courses have prerequisites though. For example, most sophomore level courses can be taken by anyone, at any time. So it doesn't matter if the student takes the class at the very beginning of their college career, at the very end, or somewhere in between.

France's mom sometimes has trouble determining if a given degree plan is in fact valid or not. She has asked Frances to write a program that can verify if a degree plan is legal, i.e. all course prerequisites are met in order and all required courses are taken, or if the degree plan is illegal, i.e. a student plans to take a course without the required prerequisites completed or the student doesn't take all the necessary classes needed to graduate. Can you help Frances write a program that, if given a set of all courses that must be taken as well as the prerequisites present for any given course, determines if the degree plan is in fact legal or not?

**Input:** The input will consist of an integer *T*, the number of test cases. *T* will be in the range of [1,10]. For each test case, input will consist of four lines. Line 1 will contain the names of all the courses a student must take to have a legal degree plan. The number of courses will be greater than or equal to two, and will not exceed forty. Course names are not limited to one-word names. It will be guaranteed that no two courses have the same exact name. Names in the list will be separated by a comma ",". Line 2 will consist of all the course prerequisites. The prerequisites will be given in the form: "`Name1->Name2`" this means `Name1` must be taken before `Name2`. Prerequisites will be separated by a comma ",". There will be at least one prerequisite present. Line 3 is the order of classes in which the student plans to take the courses. This will be a comma separated list similar to Line 1. Line 4 is 20 dashes and serves to separate all test cases. Note, it is not guaranteed that a course has exactly one prerequisite. It is possible for a course to have more than one prerequisite or zero prerequisites. It is also possible for a course to be the prerequisite of more than one course.

**Output:** For each test case you are to output: `Degree plan #X is legal.` or `Degree plan #X is illegal.` Where X is the degree plan number. Note, for a degree plan to be legal, all courses must be taken, no course can be taken twice, and all prerequisites must be taken in order.

**Sample input: (*Lines indented from left are continuation of previous line*)**

```
9
CS I,CS II,Introduction to Programming and Problem Solving,Data Structures and
       Algorithms,Operating Systems
CS I->CS II,CS II->Data Structures and Algorithms,Data Structures and Algorithms->Operating
       Systems
CS I,Introduction to Programming and Problem Solving,CS II,Data Structures and
       Algorithms,Operating Systems
--------------------
CS I,CS II,Introduction to Programming and Problem Solving,Data Structures and
       Algorithms,Operating Systems
CS I->CS II,CS II->Data Structures and Algorithms,Data Structures and Algorithms->Operating
       Systems
CS II,Introduction to Programming and Problem Solving,CS I,Data Structures and
       Algorithms,Operating Systems
--------------------
CS I,CS II,Introduction to Programming and Problem Solving,Data Structures and
       Algorithms,Operating Systems
CS I->CS II,CS II->Data Structures and Algorithms,Data Structures and Algorithms->Operating
       Systems
CS I,CS II,Data Structures and Algorithms,Operating Systems
--------------------
CS I,CS II,Introduction to Programming and Problem Solving,Data Structures and
       Algorithms,Operating Systems
```

```
CS I->CS II,CS II->Data Structures and Algorithms,Data Structures and Algorithms->Operating
       Systems
CS I,CS II,Data Structures and Algorithms,Operating Systems,CS I
--------------------
Course 0,Course 1,Course 2,Course 3,Course 4,Course 5
Course 5->Course 2,Course 5->Course 0,Course 4->Course 0,Course 4->Course 1,Course 2->Course
       3,Course 3->Course 1
Course 4,Course 5,Course 2,Course 0,Course 3,Course 1
--------------------
Course 0,Course 1,Course 2,Course 3,Course 4,Course 5
Course 5->Course 2,Course 5->Course 0,Course 4->Course 0,Course 4->Course 1,Course 2->Course
       3,Course 3->Course 1
Course 5,Course 4,Course 2,Course 0,Course 3,Course 1
--------------------
Course 0,Course 1,Course 2,Course 3,Course 4,Course 5
Course 5->Course 2,Course 5->Course 0,Course 4->Course 0,Course 4->Course 1,Course 2->Course
       3,Course 3->Course 1
Course 4,Course 5,Course 2,Course 0,Course 1,Course 3
--------------------
Course 0,Course 1,Course 2,Course 3,Course 4,Course 5
Course 5->Course 2,Course 5->Course 0,Course 4->Course 0,Course 4->Course 1,Course 2->Course
       3,Course 3->Course 1
Course 4,Course 5,Course 2,Course 0,Course 1
--------------------
Course 0,Course 1,Course 2,Course 3,Course 4,Course 5
Course 0->Course 1,Course 1->Course 2,Course 2->Course 3,Course 3->Course 4,Course 4->Course
       5,Course 5->Course 0
Course 0,Course 1,Course 2,Course 3,Course 4,Course 5
--------------------
```

**Sample output:**

```
Degree plan #1 is legal.
Degree plan #2 is illegal.
Degree plan #3 is illegal.
Degree plan #4 is illegal.
Degree plan #5 is legal.
Degree plan #6 is legal.
Degree plan #7 is illegal.
Degree plan #8 is illegal.
Degree plan #9 is illegal.
```

**Explanation of output:**

Degree plan #1: All prerequisites are taken in the correct order and all needed courses are taken.

Degree plan #2: CSII has a prerequisite of CSI, however the student attempted to take CS II before CS I.

Degree plan #3: The student will not take Introduction to Programming and Problem Solving, a required course.

Degree plan #4: The student will not take Introduction to Programming and Problem Solving, a required course. The student attempts to take CS I twice.

Degree plan #5: All prerequisites are taken in the correct order and all needed courses are taken.

Degree plan #6: All prerequisites are taken in the correct order and all needed courses are taken.

Degree plan #7: Course 1 has a prerequisite of Course 3, however the student attempts to take Course 1 before Course 3.

Degree plan #8: The student will not take Course 0, a required course

Degree plan #9. This degree plan is impossible to complete no matter the order the student chooses. This is due to Course 5 having the prerequisite of Course 0.

# 6. Honghui

**Program Name: Honghui.java** **Input File: honghui.dat**

Honghui walked into an old mathematics classroom where he discovered a sequence of colored symbols on the chalkboard. The symbols were faded, but each was either an opening or closing parenthesis (`'('` or `')'` respectively).

Honghui wants to erase some symbols on the board to form a valid colored parenthesis sequence. Valid colored parenthesis sequences are described by the following grammar:

- An empty sequence is a valid colored parenthesis sequence.
- If `X` is a valid colored parenthesis sequence, then `(X)` is a valid colored parenthesis sequence if and only if the outer parentheses have the same color.
- If `X` and `Y` are valid colored parenthesis sequences, then their concatenation `XY` is a valid colored parenthesis sequence.

Each symbol takes a different amount of effort to erase. What is the minimum amount of effort Honghui has to put in such that the remaining sequence is a valid colored parentheses sequence?

**Input:** The first line of input is T (1 <= T <= 25), the number of test cases. Each test case starts with an integer N (1 <= N <= 400), the number of symbols on the chalkboard. The next line is a string of length N where every character is either '(' or ')', describing the symbols on the chalkboard. Then follows a line with N integers, the color of each symbol. The color is given as a positive integer in the range [1, N]. The next line of each test case has N integers, the effort required to erase each symbol in order. Each effort value is a positive integer at most 1,000,000.

**Output:** For each test case, output the minimum amount of effort required to make a valid parenthesis sequence by erasing some symbols. Format each answer with the case numbers as in the sample.

**Sample input:**
```
2
7
(()))())
1 2 4 2 3 3 1
3 1 4 1 5 9 2
3
)))
3 3 3
1 2 3
```

**Sample output:**
```
Case #1: 4
Case #2: 6
```

**Sample Explanation:** In the first test case, erase the only symbol with color 4 to get the sequence `(()())`, which is balanced and has all matching colors. In the second case, all parentheses must be erased.

## 7. Manuel

**Program Name: Manuel.java** **Input File: manuel.dat**

In Manuel's Algebra II class, he has just learned about solving two equations with two unknowns. For example, if Manuel is given:

$$2x - 1y = -4$$
$$1x + 3y = 5$$

Manuel knows the solution to be $x = -1$ and $y = 2$. Manuel was taught there is more than one way to arrive at the solution. Examples include elimination, substitution, and even some slick matrix operations to derive this solution. Manuel doesn't have too much trouble with the two equations and two unknown problems, but every now and then, Manuel's teacher gives him three equations and three unknowns. For example:

$$1x + 3y - 1z = 2$$
$$4x + 2y + 5z = 1$$
$$3x + 0y + 1z = 12$$

Manuel knows the solution to be $x = 5$, $y = 3$, and $z = -2$. Manuel has the idea to write a Java program that can solve either two equations with two unknowns problems, or three equations with three unknowns problems. Do you think you can assist Manuel with this problem?

**Input:** Input will consist of a single integer T, the number of test cases. T will be in range of [1,20]. Each test case will then begin with an integer V, either a 2 or a 3 on the first line. V signifies where the test case has either two equations unknowns, or three equations and three unknowns. The following V lines will consist of the equations. Equations with two unknowns will have the form: `Avar1+Bvar2=D` where `A,B,` and `D` are guaranteed to be integers from `[-2147483648,-2147483647]` and `var1,` and `var2` are guaranteed to be a single, lowercase letter from `'a' - 'z'`. Variables are not allowed to be the same and are not allowed to be duplicated within the same test case. Equations with three unknowns will have the form: `Avar1+Bvar2+Cvar3=D` where `A,B, C,` and `D` are guaranteed to be integers from `[-2147483648,-2147483647]` and `var1, var2,` and `var3` are guaranteed to be a single, lowercase letter from `'a' - 'z'`. Variables will appear in the same order for each of the equations in the test case. Variables are not allowed to be the same and are not allowed to be duplicated within the same test case.

**Output:** For each test case you are to output on a single line: `var1=NUM1,var2=NUM2,var3=NUM3` where `var1, var2,` and `var3` are the variables read in as input (see above), and `NUM1, NUM2,` and `NUM3` are floating point values rounded to three decimal places. An output of zero must be postive. `NUM1, NUM2,` and `NUM3` when input into the original equations, should satisfy the equations. The order of `var1, var2,` and `var3` should match the order as given in the equations. It can also be guaranteed that there is one and only one solution to the equations.

***~Sample Input and Output on next page~***

***~Manuel continued~***

**Sample input:**
```
7
3
1x+1y+1z=6
0x+2y+5z=-4
2x+5y-1z=27
2
3t+1a=-10
1t-2a=4
3
-4x-5y-1z=18
-2x-5y-2z=12
-2x+5y+2z=4
2
3a-1b=7
2a+3b=1
2
5i+4m=1
3i-6m=2
3
4x+4y+1z=24
2x-4y+1z=0
5x-4y-5z=12
2
-1x+1y=2
1x+0y=-3
```

**Sample output:**
```
x=5.000,y=3.000,z=-2.000
t=-2.286,a=-3.143
x=-4.000,y=0.000,z=-2.000
a=2.000,b=-1.000
i=0.333,m=-0.167
x=4.000,y=2.000,z=0.000
x=-3.000,y=-1.000
```

# 8. Michelle

**Program Name: Michelle.java** **Input File: michelle.dat**

Michelle has always been interested in patterns. Mathematics has patterns, Computer Science has patterns as do many fields of study. She never really thought too much about languages like English beyond the obvious poetry but now wonders how she might look for other types of patterns. Obviously, there is looking at how frequently the various letters are used but she was thinking about word sizes.

Michelle asked your team for assistance in creating a program that has the flexibility to tweak the analysis as she decides what might be most interesting and revealing. Her basic idea is to build histograms for various ranges of word sizes. However, since prose can get very lengthy and wordy, Michelle recognizes that she will need to normalize the results by using percentages instead of raw counts. She cannot print histograms that are thousands of characters wide.

Can your team help Michelle with this project?

**Input:** First line of data file contains a positive integer T, the number of test cases that follow with $1 \leq T \leq 10$. Each test case starts with a single line of integers separated by single spaces with at least one and no more than 50 integers that define the upper-size limit on each of the size ranges to be used for the histogram. Each of the integers are in [1,50] and are listed in strictly ascending order. All words with lengths greater than the last integer would be placed in a final interval with a maximum size of 50. Unless the data includes 1 as the upper bound of the first interval, all words with lengths smaller than the first integer are placed in the first interval. Following the line of integers will be one or more lines of non-zero length text eventually followed by a line containing a single **‘#’** signaling the end of text for that test case. The count of total words will be (0,10000).

**Output:** For each test case, display a horizontal histogram like those shown in the sample output. Each line of output starts with the lower and upper sizes of that specific interval with leading 0s when < 10, separated by a single colon. The line of output then continues with **“ -> ”** and the percentage of words placed in that bracket rounded to a whole number, left aligned in a column 5 positions wide. A single space and the histogram then follow on the same line where each ‘x’ represents 1% of the total number of words. Following each test case output a line containing 15 equal signs “===============”.

***~ Sample input and output on next page ~***

***Michelle, continued***

**Sample input:**
```
3
1 4 7 12
We the People of the United States in Order to form a more perfect Union
establish Justice insure domestic Tranquility provide for the common defence
promote the general Welfare and secure the Blessings of Liberty to ourselves
and our Posterity do ordain and establish this Constitution for the
United States of America
#
5 9 12
We the People of the United States in Order to form a more perfect Union
establish Justice insure domestic Tranquility provide for the common defence
promote the general Welfare and secure the Blessings of Liberty to ourselves
and our Posterity do ordain and establish this Constitution for the
United States of America
We the People of the United States in Order to form a more perfect Union
establish Justice insure domestic Tranquility provide for the common defence
promote the general Welfare and secure the Blessings of Liberty to ourselves
and our Posterity do ordain and establish this Constitution for the
United States of America
#
3 5 7 9 11
We the People of the United States in Order to form a more perfect Union
establish Justice insure domestic Tranquility provide for the common defence
promote the general Welfare and secure the Blessings of Liberty to ourselves
and our Posterity do ordain and establish this Constitution for the
United States of America
We the People of the United States in Order to form a more perfect Union
establish Justice insure domestic Tranquility provide for the common defence
promote the general Welfare and secure the Blessings of Liberty to ourselves
and our Posterity do ordain and establish this Constitution for the
United States of America
We the People of the United States in Order to form a more perfect Union
establish Justice insure domestic Tranquility provide for the common defence
promote the general Welfare and secure the Blessings of Liberty to ourselves
and our Posterity do ordain and establish this Constitution for the
United States of America
#
```

**Sample output:**
```
Test case #1
01:01 -> 2     xx
02:04 -> 44    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
05:07 -> 38    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
08:12 -> 15    xxxxxxxxxxxxxxx
13:50 -> 0
===============
Test case #2
01:05 -> 50    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
06:09 -> 46    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
10:12 -> 4     xxxx
13:50 -> 0
===============
Test case #3
01:03 -> 40    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
04:05 -> 10    xxxxxxxxxx
06:07 -> 35    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
08:09 -> 12    xxxxxxxxxxxx
10:11 -> 2     xx
12:50 -> 2     xx
===============
```

## 9. Prateek

**Program Name: Prateek.java** **Input File: prateek.dat**

The Uncannily Intelligent Lexicographers (UIL) is testing new methods of compressing dictionaries. A dictionary is a collection of words. The words in the dictionary can be stored in any order, and different ways to store a word in the dictionary can take up differing amounts of space. The UIL's dictionary structure has two methods of representing a word:

1. Record a whole word. By UIL mandate, this takes A bytes of memory, regardless of the length of the word.
2. Record a word by difference. This method begins with a B byte reference to a previous word T and D bytes for each difference between T and the current word S. The total space is B + D * dist(S, T) bytes of memory where dist(S, T) is the Levenshtein distance between S and T.

The Levenshtein distance between two words is the number of single-character additions, deletions, and changes to go from one word to the other. For example, the Levenshtein distance between "cat" and "dog" is 3 (change each character). The Levenshtein distance between "apple" and "orange" is 5 (delete the l, change the first 'p' to an 'n', change the second 'p' to a 'g', and add 'o' and 'r' at the beginning).

Given a dictionary of words, what is the minimum number of bytes needed to encode all the words?

**Input:** The first line of input is T (1 <= T <= 30), the number of test cases. Each test case begins with 4 integers N A B D where N (1 <= N <= 100) is the number of words in the dictionary, A is the number of bytes storing a whole word takes, B is the number of bytes to record a word by difference, and D is the cost per difference. 1 <= A, B, D <= 1,000. Then follow N lines, each containing a word in the dictionary. Each word consists of at most 20 lowercase letters (a-z).

**Output:** For each test case, output the minimum cost to send all words, formatted with the case number as in the samples.

**Sample input:**
```
2
3 10 4 1
apple
orange
strawberry
5 4 1 1
ab
ac
xy
yz
az
```

**Sample output:**
```
Case #1: 29
Case #2: 13
```

**Sample Explanation:** For the first sample, the best strategy is to send "apple" and "strawberry" normally, and then send "orange" as a difference of "apple". This saves one byte over sending all 3 words as whole words.

## 10. Richard

**Program Name: Richard.java** **Input File: richard.dat**

Richard has some really bad news! His personal server was just hacked! Richard isn't sure exactly how the server was hacked, but he thinks his old password of "password123" might be the issue. In an attempt to better safeguard his server, Richard has the idea to use a password generator to create a much stronger, and harder to crack password. The problem is, Richard no longer trusts anyone or any application created by anyone other than himself or someone he knows personally!

With this said, Richard wants to write a program that generates a password and he has the idea to take a given string of characters made up of letters (uppercase and lowercase), digits, and special characters and determine the ith permutation of that string. For example, suppose Richard chooses the string "`AbC!`". If Richard were to generate all possible permutations of this string and sort them by their ASCII values he would have: `"!ACb", "!AbC", "!CAb", "!CbA", "!bAC", "!bCA", "A!Cb", "A!bC", "AC!b", "ACb!", "Ab!C". "AbC!", "C!Ab", "C!bA", "CA!b", "CAb!", "Cb!A", "CbA!", " b!AC", "b!CA", "bA!C", "bAC!", "bC!A", "bCA!".` So, if Richard chose his password to be the 16th permutation of the string "`AbC!`", his password would be: `"CAb!"`

Can you assist Richard in writing a program to generate new passwords for his server?

**Input:** The input will consist of an integer *T*, the number of test cases. *T* will be in the range of [1,20]. For each test case, input will consist of two lines. Line 1 will be a number `i`, in range [1, 2147483647]. This is the ith permutation number Richard has selected. Line 2 will consist of a string of characters (upper and lower case), digits, and printable special characters. The length of the string will be guaranteed to be at least one character long and no more than 50 characters long. The string is also guaranteed to have no two characters with the same ASCII value, and the ith permutation is guaranteed to exist.

**Output:** For each test case, you are to output `"Password #X: GENERATED_PASSWORD"` where `X` is the test case number and `GENERATED_PASSWORD` is the ith permutation of the given string of characters.

**Sample input:**
```
6
16
AbC!
2
abcde
598
AaBbcC!
104567
t%&avcdef3
1
zAaBb
123456
8rTv3@AM!
```

**Sample output:**
```
Password #1: CAb!
Password #2: abced
Password #3: !bcaBCA
Password #4: %aet3cvf&d
Password #5: ABabz
Password #6: @!MArvT83
```

# 11. Sanjay

**Program Name: Sanjay.java** **Input File: sanjay.dat**

During his fishing trip, Sanjay was hit by a rogue wave! In the commotion he was able to keep hold of his fishing pole and ocean map, but unfortunately lost his paddle. Due to this, he is at the mercy of the ocean currents, but has come up with a plan. Sanjay would like to use his map to determine the closest his boat will drift to an island. From there he can hook onto the island with his fishing pole and reel himself in to subsist on bananas and coconuts until he is eventually rescued.

Given the direction that the boat is drifting, and a description of all the islands (given as polygons on the y = 0 ocean plane), write a program to find the closest island to the path of Sanjay's boat.

**Input:** The first line of input will contain a single integer `T`, the number of test cases to follow (`1` <= `T` <= `10`). The first line of each test case will contain an integer `N` denoting the number of islands on the map , followed by 4 floating point numbers `x`, `z`, `dx`, and `dz`, denoting the x and z position, and the x and z velocity of Sanjay's boat on the ocean plane. Sanjay's boat is very small compared to the islands and can be viewed as a point on the plane. The following `N` lines of each test case will each contain a single island description. Each island description will begin with an integer `V`, the number of vertices in the island's polygon, followed by `2*V` space separated floating point numbers representing the position of the island in the order $x_0, z_0, x_1, z_1, \ldots, x_V, z_V$. These points will form a polygon and be given in clockwise order.

**Constraints:**
There will be at most `10` test cases
There will be at most `50` islands per test case
no island will consist of more than `50` vertices
No X or Z coordinate of a boat or island point will have an absolute value greater than 10,000.
`-100` <= `dx`, `dz` <= `100`
Sanjay will not start on an island

**Output:** For each test case on its own line, output the minimum distance Sanjay's boat will be from any island, rounded to 3 places after the decimal point. This distance is calculated by the length of the shortest line whose endpoints are incident to both Sanjay's boat and some island during the course of the boat's drift.

**Sample input:**
```
2
3 -3.0 -1.0 1.0 1.0
5 -8.0 -7.0 -7.0 -7.0 -7.0 -9.0 -8.0 -9.0 -9.0 -8.0
5 -1.0 -3.0 3.0 0.0 4.0 -2.0 3.0 -5.0 1.0 -5.0
5 7.0 4.0 6.0 6.0 8.0 7.0 9.0 7.0 10.0 4.0
2 -3 -1 1 1
4 -4.0 8.0 -2.0 8.0 -2.0 6.0 -4.0 6.0
4 6.0 13.0 7.0 13.0 9.0 11.0 6.0 12.0
```

**Sample output:**
```
1.414
0.000
```

# 12. Urvashi

**Program Name: Urvashi.java** **Input File: urvashi.dat**

Urvashi is playing a video game set in a faraway land with many towns and M bidirectional roads connecting them. Urvashi wants to travel from the starting town S to the main city T. To cross a road, she must pay a toll.

In order to potentially make the journey cheaper, Urvashi has a powerful toll reversal spell. When used, the tolls on all roads are reversed. For example, if the toll was originally 314, after applying the spell, the new toll is 413. When reversing a toll, leading 0s are discarded. For example, casting the spell changes a toll of 2040 to a toll of 402.

The reversal spell is too powerful to be cast recklessly, and may be cast at most K times. Given this, what is the cheapest cost to travel from town S to town T?

**Input:** The first line of input is C (1 <= C <= 25), the number of test cases. The first line of each test case is M, K, S, and T where M (1 <= M <= 2,000) is the number of roads, K (0 <= K <= 2,000) is the maximum number of times the spell may be cast, S is the name of the starting town and T is the name of the destination town. Then follow M lines, each of the form U V W where U and V are the names of the towns connected by the road and W (1 <= W <= 10^9) is the tax paid to cross that road before the reversal spell is applied. No road will connect a town to itself, but there may be multiple roads between the same pair of towns.

All names use only lowercase Latin letters (a-z) and have length at most 15.

**Output:** For each test case, output the minimal total cost to get from S to T using at most K reversal spells. If it is impossible to travel from S to T using the roads, output "IMPOSSIBLE". Format the output with the case numbers as in the samples.

**Sample input:**
```
3
3 1 shire mordor
shire middleearth 1
middleearth mordor 32
shire mordor 303
3 1 a d
a b 12
b c 32
c d 34
3 2 a d
a b 12
b c 32
c d 34
```

**Sample output:**
```
Case #1: 24
Case #2: 78
Case #3: 69
```

**Sample Explanation:** In the first test case, Urvashi can travel from "shire" to "middleearth" with cost 1, apply the reversal spell, then travel from "middleearth" to "mordor" with cost 23 for a total cost of 24. In the second test case, it's best if Urvashi never uses the reversal spell.