Plano West Wolf Invitational 2020 Editorial

Code can be found on the Github link posted on the Hackerrank Contest

Lisa Broke Up

We just need to print the ASCII Art. Make sure you use dashes and underscores at the appropriate times.

Valentine Gift

When we run out of one item, we can't make a gift. Therefore, we just need to find the min of **number of necklaces**, **number of bouquets of flowers / 2**, and **number of chocolates / 4**. All rounded down.

Oracle

Find number of **"not"** in the string. If the number of **"not"** is even, print LOVE, else print SAD.

Hop-scotch

We can create a visited array that keeps track of all the stones we have visited. Every time we visit a stone, we mark it as visited. If we visit a position that is already visited and we haven't reached the starting stone yet, then we print **NO** because we are going to visit the same set of stones again. Also, make sure to Modulo the jump distance by the total number of stones because the stones are laid out in a circular fashion.

Jam

We take in input and create two arrays that store the positions where both horizontal and vertical cuts are made, then we sort them in increasing order. After that, we run a nested for-loop of all the positions where we cut the cake. If both row and column indexes are odd/even, then we add that area to total area.

MooForces

We need to use a custom sorter, then print out the first **M** people in the sorted array. Because constraints are tight, using a BufferedReader in Java might be a good idea.

Align

One solution is to use a sliding window technique. Look through code for more explanation.

Busy Day

There are two time intervals that can be used to finish the errands. From time 0 to the time the date starts and from time the date ends to 24. Then, we sort the errands in decreasing order, and we subtract the errand from the smaller time interval. If there is no more space in both intervals, then you print **NO**.

Escape Valentine

We use BFS to find the shortest path.

However, when we encounter a teleport, we add its exit to the queue and mark it as visited because we have to teleport if we step on a teleport.

Valentine Street

This is a greedy problem.

The key is to realize that we have to group the farthest K homes together in one run and take the max distance - 1 in that group * 2 (for returning back to house 1). Then we process the next K groups until there is no more homes.

Candy Tower

This is a classic DP problem called the Longest Decreasing Subsequence, except in the problem, we consider values that are tied also as decreasing.

If you use recursion, it will time out.

Tower Elevators

This is a sweep line problem, and we need to check whether multiple intervals overlap together for more than 2 times.

There are 2 * N points of interest, which is the start and end time. We can mark starting time as even numbers (2 * S) and ending times as odd numbers (2 * S + 1).

Then, we iterate through the array. At any point if there's more than two even numbers in a row, then print **NO**.

Stations

This is a game theory problem playing off of pattern recognition.

First, we need to see which stations are winning and losing positions. Station 0 is always a losing position for a player because once the chocolate reaches station 0, they can't make a move. Station 1 is always a winning position because you can move the chocolate to station 0 (losing position), which forces the other player to lose. Station 2 is also a winning position because you can move the chocolate to station 0. Since Danny goes first, he would win if N was 1 or 2.

Now, let's ignore X and just focus on one or two steps. At station 3, if it's Danny's turn, he can only move to station 1 or 2, and Dimmy can move to station 0 no matter what. This forces Danny to lose. For station 4 or 5, they are winning positions because you can force the other player to station 3, which is a losing position. Then station 6 would become a losing position.

From this, we see a pattern (If station N % 3 == 0, then it's a losing position). We can also see that if a player can land on a losing station, then they will win because it forces the opponent to a losing position.

Now, let's take X into account. If X is 4, 5, 7, 8, ...

It is the same as before when X is ignored. Therefore, X can be ignored if X % 3 != 0.

However, when X is a divisible of 3, the pattern is different.

When X % 3 == 0, the stations go in a cycle every X+1 times, so we can re-number the stations starting from 0 when it reaches station X+2. After this, everything is just like the list when X is ignored, except position K is also winning rather than losing.

N-Gon

This is simply application of geometry and trigonometry. Since the polygon is normal, it can be drawn with constant side lengths and angles, allowing for a step-wise implementation.

The side length L can by found by the distance formula $\sqrt{(x1-x2)^2+(y1-y2)^2}$.

The initial direction T can be found by $\arctan((y2-y1)/(x2-x1))$, adjusted to include the third or fourth quadrant if necessary. The Java method Math.atan2 finds this adjusted angle.

The change in direction can be found be $2\pi/N$, where N is the number of sides.

To draw the polygon, simply draw N lines of length L, starting your direction at T, changing by A radians each time.

Branch Me Up, Please!

This is a DP problem. To solve it, you must find the number of asynchronous trees (trees that have a mirror reflection that is not itself) and symmetric trees. This can easily be done by keeping track of the total number of trees and the number of symmetrical trees. If you're using Java, you'll need to use the BigInteger class as results become very large prior to the modulo operation.

To start out the top-down DP array, the last branching of the tree has S possibilities, as given in the input.

For previous branching processes, each branch-out creates a smaller tree of S possibilities, each of which subsequently branching out with allTrees[i + 1] possibilities. As a result, the number of possible trees at point i is defined by $\sum(allTrees[i+1])^B$, where B is the number of branches a branching process could create; iterate through every B and add.

To find the number of symmetrical trees, a similar process is followed. This time, a tree is symmetrical if the left half of it is identical to the right half. Consequently, follow the same process, substituting $B/2$ for B. For odd-numbered B values, round down and add symTrees[i + 1].

The number of unique trees can be found by (allTrees[0] - symTrees[0]) / 2 + symTrees[0].

Roses

This problem can be solved greedily.

We go from the left side of the string to the right side of the string checking one pot at a time. When both pots are the same, we continue. When Kevin has a rose planted and Max doesn't, we just need to plant a rose in Max's pot, and add 1 to the total number of days.

When Max has a rose and Kevin doesn't, then we need to start at that pot and continue moving to the right until either we have hit the last pot or the pot we are on doesn't have a rose in it. After that, we add 1 to total days and plant a rose in the next pot.

Then, we repeat for every pot from left to right.

Daniel's Sandwich

This is a DP problem that can be best understood through viewing the implementation.

Use a 2D DP array dp[i][j] to represent the minimum number of bytes needed to eat the subset of the sandwich between points i and j, inclusive.

To fill out the value dp[i][j], find the minimum of dp[i - 1][k] + dp[k - 1][j] for $j \leq k \leq i-1$. This formula examines every possible way to eat the specified subset of the sandwich, assuming Sandra eats the interiors of vals[i-1] and vals[k] before eating vals[i-1] and vals[k] together, if possible (with the same process for vals[k - 1] and vals[j]). After every subset has been filled out, output the minimum number of bytes for the entire sandwich.

Crossing Paths

The key is to realize we only need distances to check path intersection. The main claim is the following, which is a cute exercise: Let f(x,y) denote the length of the path x $--> $ y. Then a $-->b$ and $c-->d$ intersect if and only if f(a,b) + f(c,d) $\geq$ f(a,c) + f(b,d).

As N is rather small, we can precompute all distances with N breadth-first searches. Then each query is answered in constant time, so we have a total complexity of O(N^2 + Q), which is well within limits.

Fill

Please look at the code.

Date Night

The formula is the sum of the 10 numbers divided by the product of the 10 numbers.