36th Annual High School Programming Tournament

March 7, 2022

Problem Review

36th Annual High School Programming Tournament

March 7, 2022

A Very Hard Problem (hard)

Category: Math Difficulty: 1

A Very Hard Problem (hard)

Problem:

Given two problems described by a length and a complexity, output the difficulty of the harder problem.

Solution:

As described in the problem, the difficulty is the product of the length and complexity, so take the product for each problem and output the max of the two products.

36th Annual High School Programming Tournament

March 7, 2022

Catching those Zs! (sleep)

Category: Text Difficulty: 1

Catching those Zs! (sleep)

Problem:

Given a string of characters, determine if that string has 3 or more ‘z’s.

Solution:

Scan in the whole next line. Iterate through each character and increment a count if you are on a lower or uppercase ‘z’.

Caution: After scanning in an integer, we do not move to the next line so you must scan in a full line before starting to iterate through the test cases. This is not a concern in python.

36th Annual High School Programming Tournament

March 7, 2022

Ovoo? What's That? (ovoo)

Category: Simulation Difficulty: 1

Ovoo? What's that? (ovoo)

Problem:

Determine the favor of a clan on a worship day.

Solution:

- Favor is dependent on the number of ovoos so keep track of the total ovoos.
- Each work day adds one ovoo to the total. At each worship day, the total favor increments by the number of ovoos.
- In addition to incrementing, output the favor.

36th Annual High School Programming Tournament

March 7, 2022

The Queen's Reign (queen)

Category: Math Difficulty: 2

The Queen's Reign (queen)

Problem:

Given a queen in the center of a chess board, where k describes the number of squares between the queen and the edge of the board, determine how many squares the queen cannot reach.

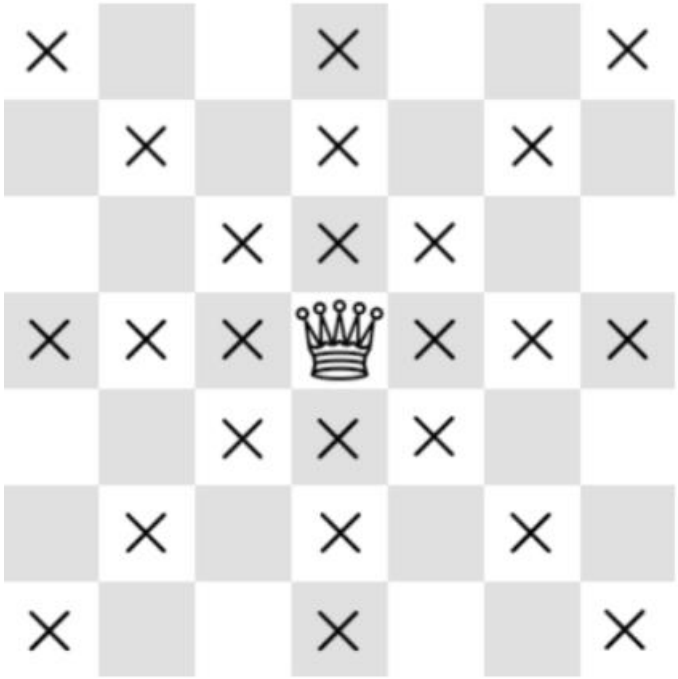
Solution:

- Start by finding the total number of squares in the board. The chessboard will be $2k+1$ long so the total number of squares is $(2k+1)^2$.
- Now we find how many total square the queen can reach. The queen can move in 8 directions, and each direction she can move k squares, so she can reach $8k$ squares.
- She also cannot reach the square she is currently on.
- So, the total number of squares she can reach is $(2k+1)^2 - 8k - 1$;

UCF

The Queen's Reign (queen)

- The figure to the right corresponds to $k = 3$.
- The total number of squares is $(2*3+1)^2 = 49$ squares.
- The queen can move k squares in each of the 8 directions so she can reach 24 total squares.
- In total, excluding the square she is in, she can reach 49 – 24 – 1 = 24 squares



36th Annual High School Programming Tournament

March 7, 2022

Campus Parade (parade)

Category: Simulation Difficulty: 2

Campus Parade (parade)

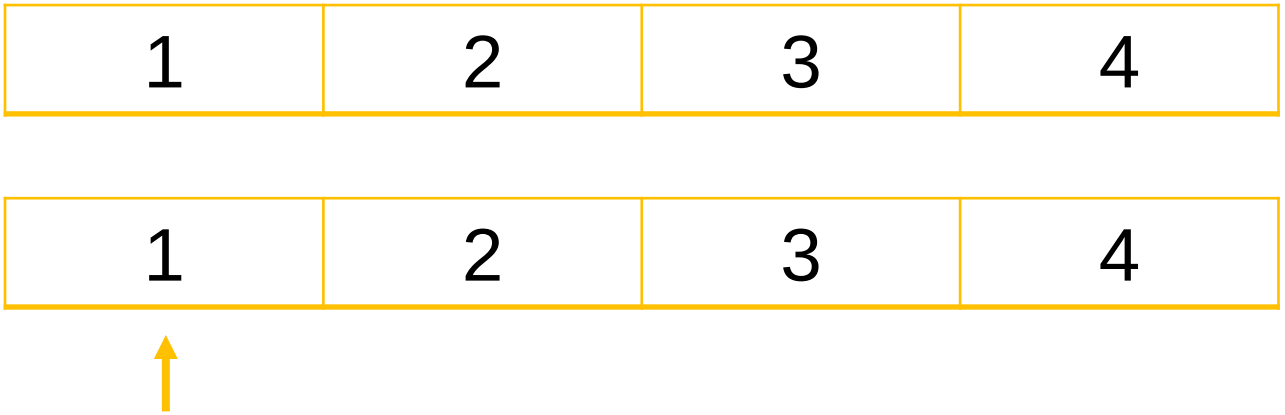
Problem:

Given the numbers 1 through *n*, determine the number at the beginning given *m* cyclic shifts to the left.

Solution:

- One way to approach this problem is to simulate moving the person from the front to the back *m* times.
- This can be done by creating a new array and shifting the positions.
- This is inefficient due to having to make a new array.
- Alternative approach: Have a pointer that points to the current front of the line.

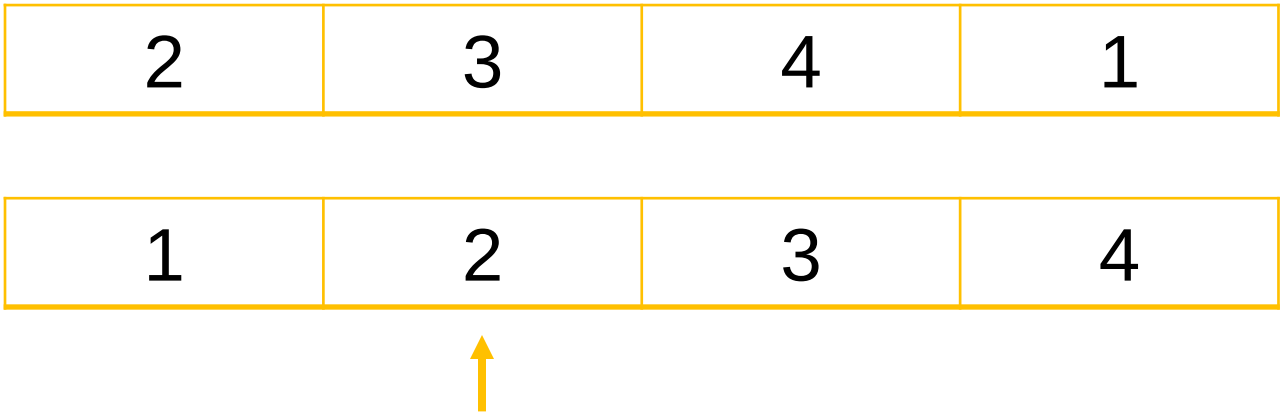
Campus Parade (parade)



Let's take a look at the case of $n = 4$ and $m = 6$.

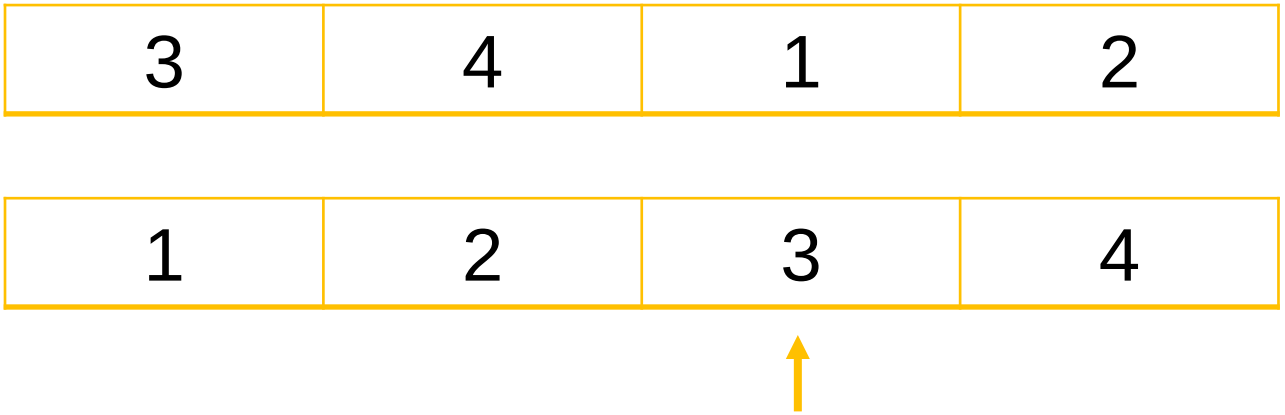
This is the order before any doors.

Campus Parade (parade)



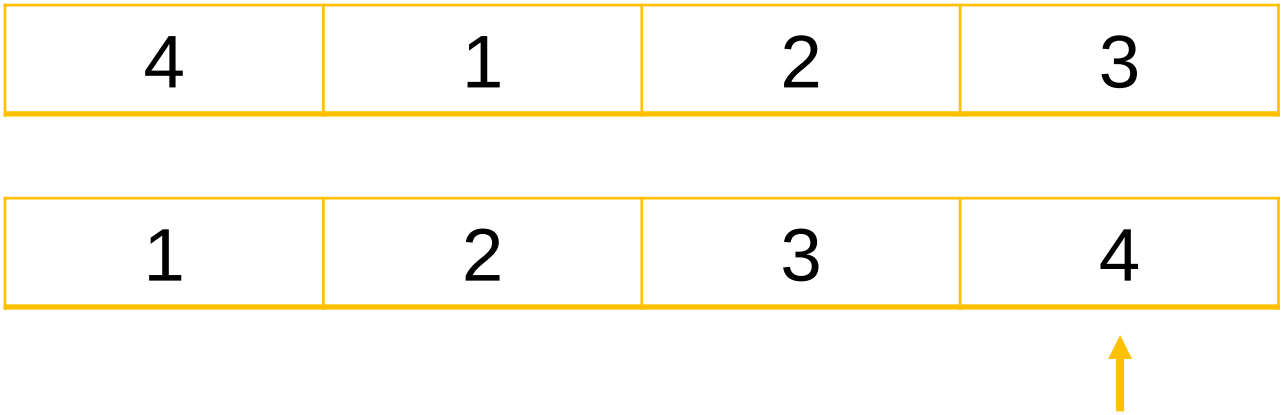
After the first door: 1 moves to the end of the line which can be shown with a shift of a pointer. 2 Is now in the front of the line.

Campus Parade (parade)



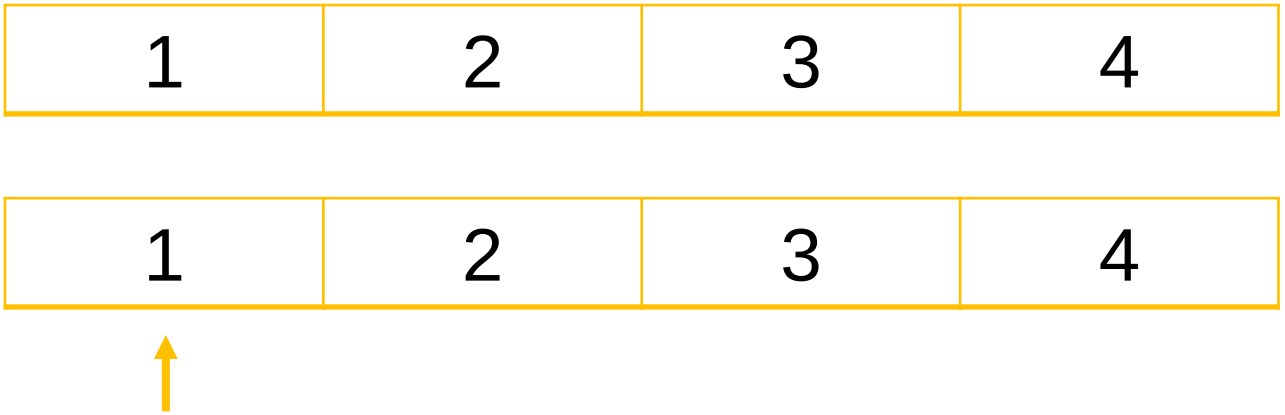
After the second door: 2 moves to the end of the line, 3 is now in front.

Campus Parade (parade)



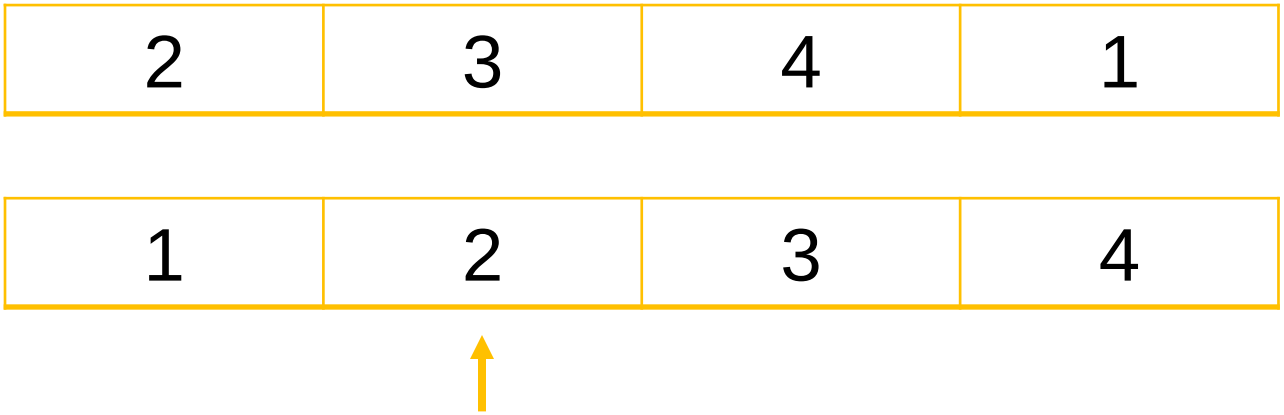
After the third door: 3 moves to the end of the line, 4 is now in front.

Campus Parade (parade)



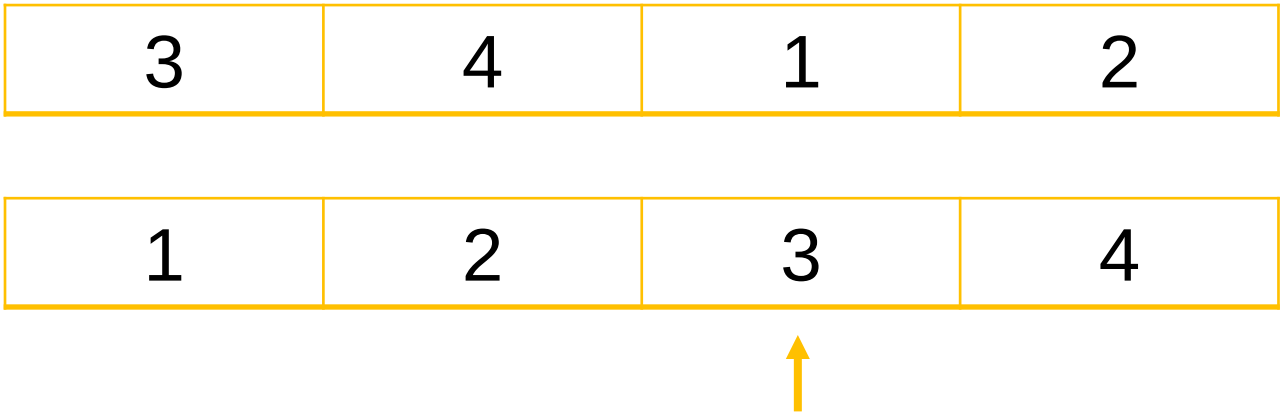
After the fourth door: 4 moves to the end of the line, 1 is now in front. (Notice, at the n^{th} door, we have fully cycled through the array)

Campus Parade (parade)



After the fifth door: 1 moves to the end of the line, 2 is now in the front of the line.

Campus Parade (parade)



After the sixth door: 2 moves to the end of the line, 3 is now in front, so 3 is out final answer.

Notice: The final position index of our pointer points to the element with the value of *index* + 1

Campus Parade (parade)

- Is there a even more efficient way we can approach this?
- After *n* doors, we are back to our original order with 1 at the beginning.
- After *k* ▪ *n* iterations (where *k* is an arbitrary integer), we will be back to our original order.
- So, all we care about is our last *m* - *k* ▪ *n* iterations.
- *m* - *k* ▪ *n* is the definition of the mod so we care about the last *m%n* iterations.
- This mod value is equivalent to the final index of our pointer, so our final answer is going to be *m%n* + 1.

36th Annual High School Programming Tournament

March 7, 2022

Dog Drama (dog)

Category: Geometry Difficulty: 2

Dog Drama (dog)

Problem:

Determine how many flowers (points) are within ℓ units away from you.

Solution:

Iterate through every point. Using the distance formula ($\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$) get the distance between you and the current point. If that point is within ℓ units, increment a total count.

36th Annual High School Programming Tournament

March 7, 2022

Let's Groove (groove)

Category: Greedy Difficulty: 3

Let's Groove (groove)

Problem:

Given an array of numbers, determine the largest number, *k*, such that the sequence [1, 2, …, k, k, …, 2, 1] can be found as a subsequence of the array.

Solution:

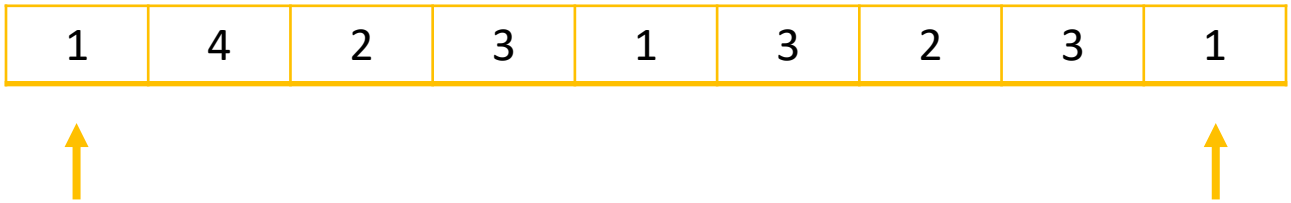
This can be solved with a two pointer sweep.

- Start your left pointer at the beginning and your right pointer at the end of the array.
- While your left pointer is less than your right pointer:
  - increment the next *k* value by 1 if both values are equal to the next *k*.
  - If the left pointer is not equal to the next *k*, increment its index. If the right pointer is not equal to the next *k*, decrement its index.

Let's Groove (groove)

Example:

Current next *k* value: 1

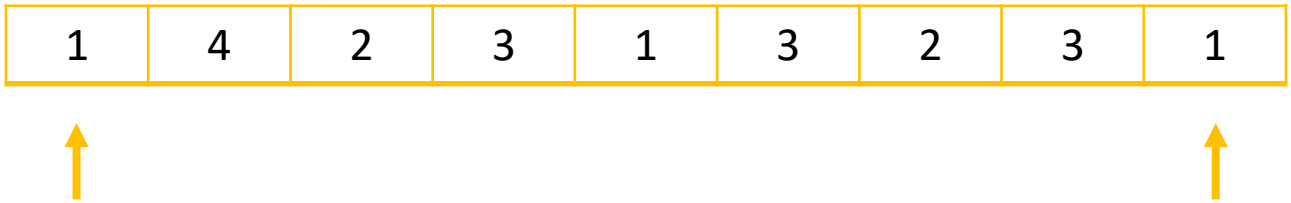


- This corresponds to the first sample case
- We start with 2 pointers: one at the beginning and one at the end

Let's Groove (groove)

Example:

Current next *k* value: 1

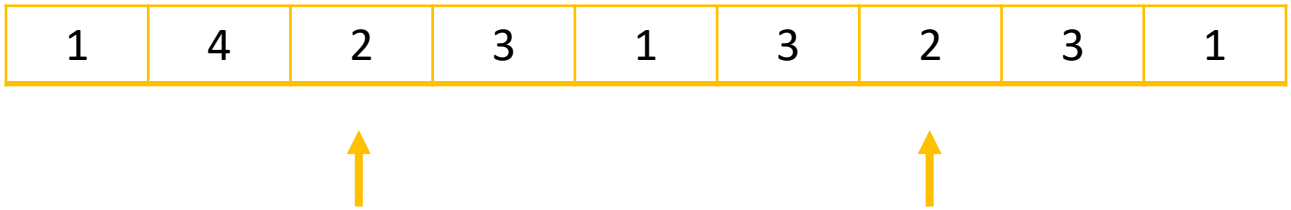


- Both pointers point to a 1.
- Increment the next *k* value.
- Increment the left pointer.
- Decrement the right pointer.

Let's Groove (groove)

Example:

Current next *k* value: 2

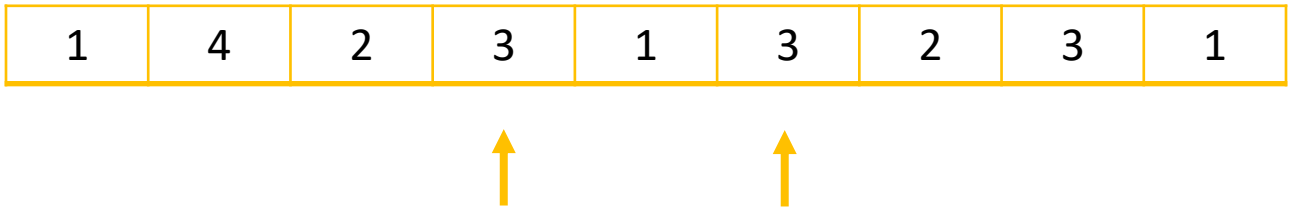


- Neither pointer points to our next *k* value
- Increment the left pointer
- Decrement the right pointer

Let's Groove (groove)

Example:

Current next *k* value: 3

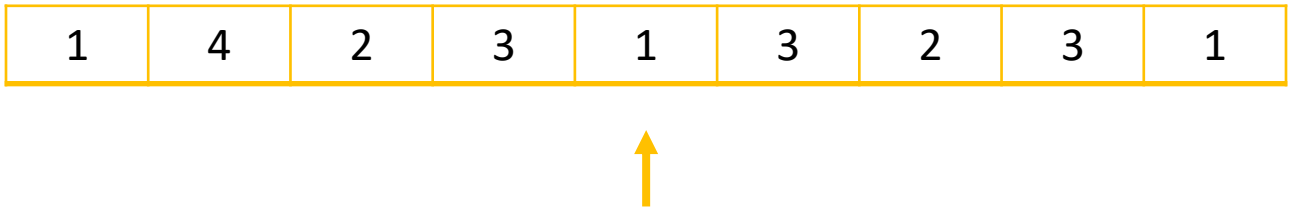


- Both pointers point to a 3.
- Increment the next *k* value.
- Increment the left pointer.
- Decrement the right pointer.

Let's Groove (groove)

Example:

Current next *k* value: 4



- Since our pointers now overlap, our sweep is complete.
- Our last successful *k* was 3, so that is our answer.

36th Annual High School Programming Tournament

March 7, 2022

What are the Odds? (odds)

Category: Text Difficulty: 3

What are the odds? (odds)

Problem:

Given two strings, find the probability that removing one character randomly from the first string results in the second string.

Solution:

- We need to find the number of characters that can be removed such that the answer results in the second string.
- First calculate the length of the longest common prefix and longest common suffix.
- The result can then be derived from the inclusion-exclusion principle.
- Add the two lengths together. This counts every letter in the second string so subtract out the length of that string.
- If this value is 0 or negative, it is impossible to get to the second string.

What are the odds? (odds)

- Otherwise, we must add 1 to the count to account for the first string being one longer for the second.
- Then create the fraction and simplify using the Euclidean algorithm for greatest common divisor

Example

n	o	o	o	o	o	o	n	o	o

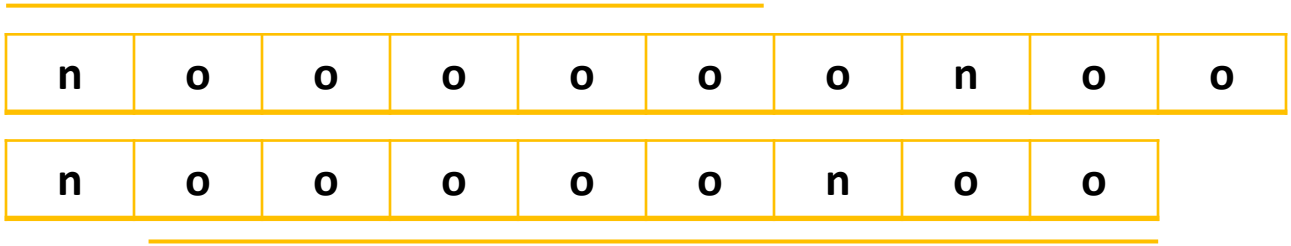
n	o	o	o	o	o	n	o	o

This corresponds to the 2nd test case in the samples

What are the odds? (odds)

- Otherwise, we must add 1 to the count to account for the first string being one longer for the second.
- Then create the fraction and simplify using the Euclidean algorithm for greatest common divisor

Example



Find the longest common prefix (LCP) and suffix (LCS).

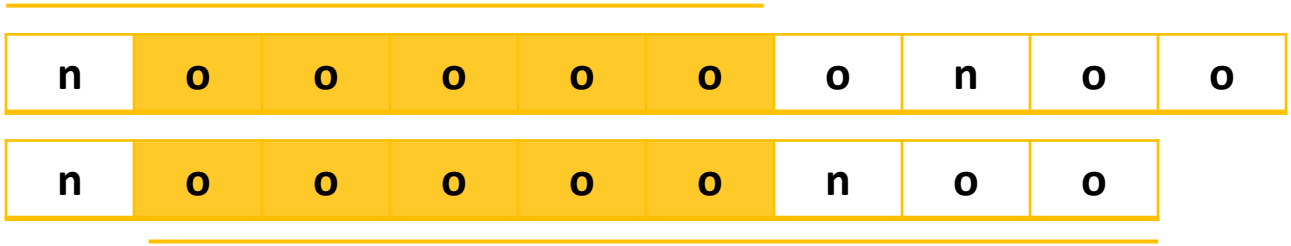
The top line shows the LCP

The bottom line show the LCS

What are the odds? (odds)

- Otherwise, we must add 1 to the count to account for the first string being one longer for the second.
- Then create the fraction and simplify using the Euclidean algorithm for greatest common divisor

Example



Now look at the overlap between the LCS and LCP.

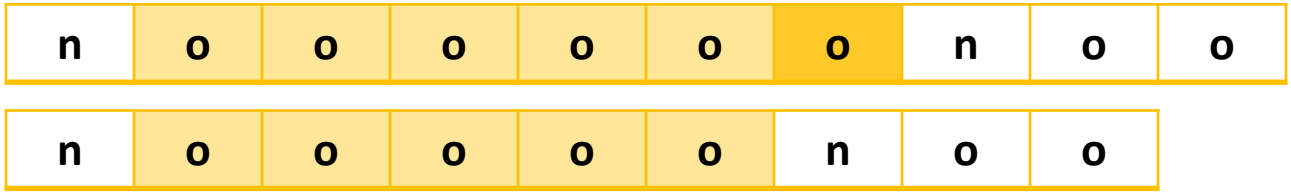
This overlap can be computed by adding the lengths of the LCS and LCP and subtracting out the length of our second string.

This gives us an overlap of length 5.

What are the odds? (odds)

- Otherwise, we must add 1 to the count to account for the first string being one longer for the second.
- Then create the fraction and simplify using the Euclidean algorithm for greatest common divisor

Example



As can be seen in the first word, we have one extra letter that can be removed (highlighted in dark yellow).

This occurs naturally due to the differing lengths of the word so we must add 1 to our length of overlap to get 6 letters.

UCF

What are the odds? (odds)

- Otherwise, we must add 1 to the count to account for the first string being one longer for the second.
- Then create the fraction and simplify using the Euclidean algorithm for greatest common divisor

Example

n	o	o	o	o	o	o	n	o	o

So now we have our final number of successes: 6

Our final answer will be the number of successes over the total number of possibilities: 6/10

Simplified, our final answer is 3/10.

36th Annual High School Programming Tournament

March 7, 2022

Problematic Picasso (picasso)

Category: Greedy Difficulty: 3.5

Problematic Picasso (picasso)

Problem:

Given 2 arrays, find the maximum sum such that you take one element at each index and you take *x* elements from the second array

Solution:

Greedy:

- Find the sum of all elements in the first array.
- Take the difference between the elements of both arrays at each index.
- Sort the differences and add the *x* largest differences to the sum.

36th Annual High School Programming Tournament

March 7, 2022

Astronomy Rules! (astronomy)

Category: Ad Hoc Difficulty: 3.5

Astronomy Rules! (astronomy)

Problem:

Given 2 arrays, find if you can get from the first array to the second array solely by either rotating or flipping the array.

Solution:

- In order to get from the first array to the second, the relative order of the numbers must remain the same.
- Because we can flip the array, the order of the first array must be the same or the reverse of the second array.
- Traverse the arrays starting at a position in the first array that shares the same value with a position of the second array.
- If the order is the same or reversed, then you are all good.

36th Annual High School Programming Tournament

March 7, 2022

Arcane Artifacts (artifacts)

Category: Geometry Difficulty: 4

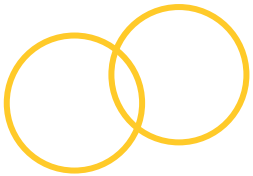
Arcane Artifacts (artifacts)

Problem:

Given a set of circles that merge if they intersect, determine how many circles remain after they all merge. Circles appear in the order they are given.

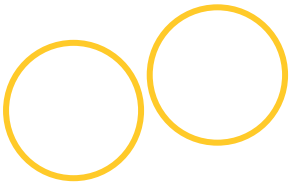
Solution:

- First how do we tell if two circles intersect?
  - Two circles intersect if the distance between their circles is less than the sum of their radii
  - If the distance is equal to the sum of their radii, then they are tangent (which is a valid for this problem



$d < r_1 + r_2$

$d = r_1 + r_2$



$d > r_1 + r_2$

Arcane Artifacts (artifacts)

- Perform a simulation while scanning in the circles
- For our current circle, loop through all of the circles that currently exist, keeping track of three sums: sum of *x*-values, sum of *y*-values, and sum of areas
  - If the current circle intersects the circle in the list, add the values to our sums.
  - Otherwise add the circle from the list into a “keep” array
- Divide the sum of *x*-values and *y*-values by the number of intersections that occurred (the simple average)
- Compute the new radius from the sum of areas.
- If a change has occurred, loop through the circles again checking for intersections.
- Continue looping until no changes occur.
- If no change occurs add the merged circle to the “keep” array, replace the current list of circles with the “keep” array, and scan in the next circle.

36th Annual High School Programming Tournament

March 7, 2022

Monkey Madness (monkey)

Category: Graph Difficulty: 4

Monkey Madness (monkey)

Problem:

Given a grid of numbers 0-9 with 0's corresponding to the monkeys' positions, determine the lowest number, *k*, such that all monkeys can reach the edge of the grid such that they can only pass through grid cells less than *k*.

Solution:

- Do a linear search on *k.*
- For a certain *k* value, perform a Breadth First Search (BFS) on the grid.
- Start by inserting into the BFS queue all grid cell values less than *k* that lie on the edge.
- While running the BFS, if our current cell value is zero, increment a count (keeping track of the number of seen monkeys).
- If the count is equal to our total number of monkeys, print the value of *k*, and end our search.

36th Annual High School Programming Tournament

March 7, 2022

Team Naming (naming)

Category: Text Difficulty: 5

Team Naming (naming)

Problem:

Given 3 strings, determine the shortest string that contains all 3 strings as a subsequence.

Solution:

Dynamic Programming:

- 3 States: Index in the first, second, and third string
- Transition
  - First gather a set of the letters for our current state.
  - For each of these characters, we will transition to the next state.
  - To get the indices of our new state:
    - For each string, if the character matches the one in the string for our current index, increment the index by one. Otherwise keep that index the same.
  - If one more than our next state results in a shorter string, update our current state.

Team Naming (naming)

- This unfortunately only gives us the length of our optimal string.
- We must build back our answer.
- This can be done by storing a table of characters indicating what is the optimal character to take at our current state.
- At a current state, we try each of our current characters.
  - If this character results in a shorter string, put that character into the table.
  - If this character results in an equally long string, put that character into the table if it is lower lexicographically.
- After the length is computed, traverse our optimal characters/states to build back the string.