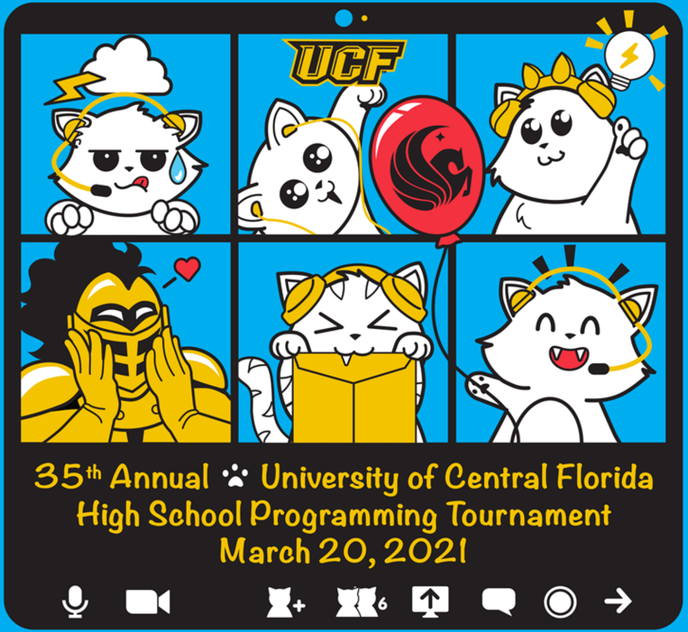
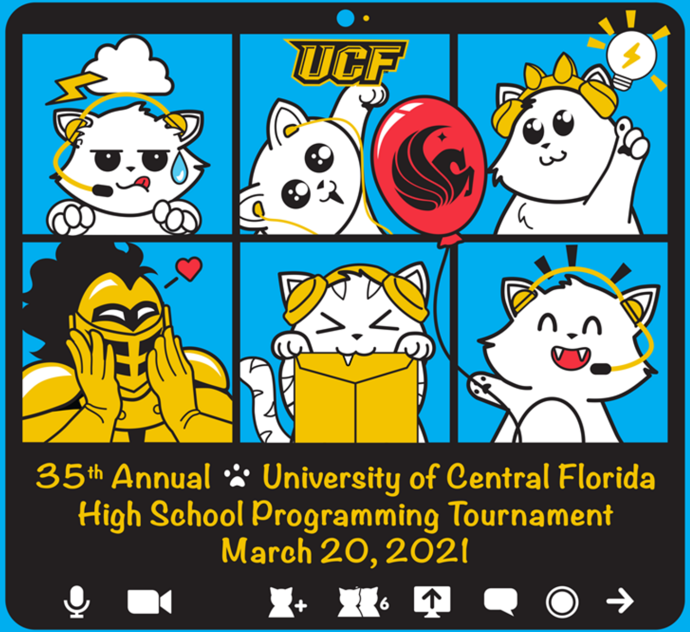

UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Problem Review

Some Overall Statistics

- **Submission Stats**
  - **Total Number of Submissions:** 1098
  - **Average Judging Time:** 1.03 minutes
    - **Min:** 0 minutes
    - **Max:** 21 minutes
    - **Standard Deviation:** 7.391 minutes

- **Per Language**
  - **C:** 0 submissions
  - **C++:** 119 submissions
  - **Java:** 757 submissions
  - **Python:** 212 submissions


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Build-a-House (house)

83/107 solved/submissions

Build-A-House (house)

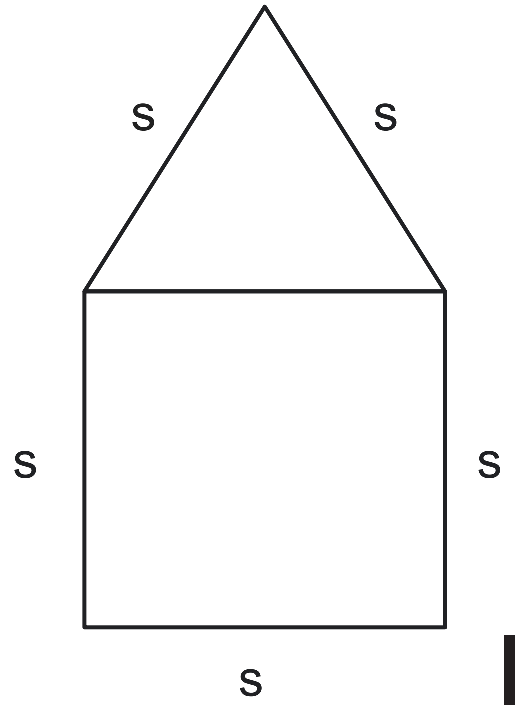
Given a side length for a house, calculate the length of the *outer* perimeter.

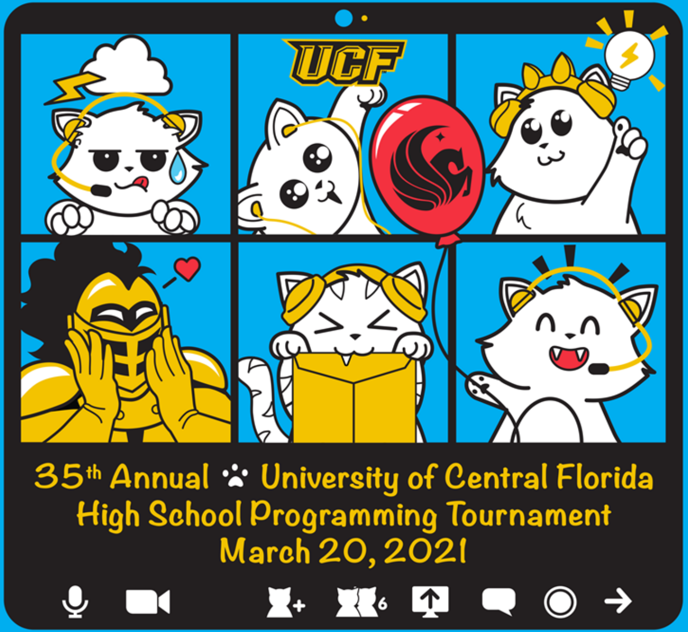
Solution:

From the picture on the right, we can see there are 5 segments, each of the same length.

Thus the solution is to multiply the input length by 5.

Make sure not to multiply by 6, since we want the *outer* perimeter, and don't want to include the middle segment!




UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

No Mor (nomor)
79/117 solved/submissions

No Mor (nomor)

Remove all letters “e” from the inputted string.

Solution:

Read in each input line by line (not word by word!), and remove the letter “e”.

Examples:

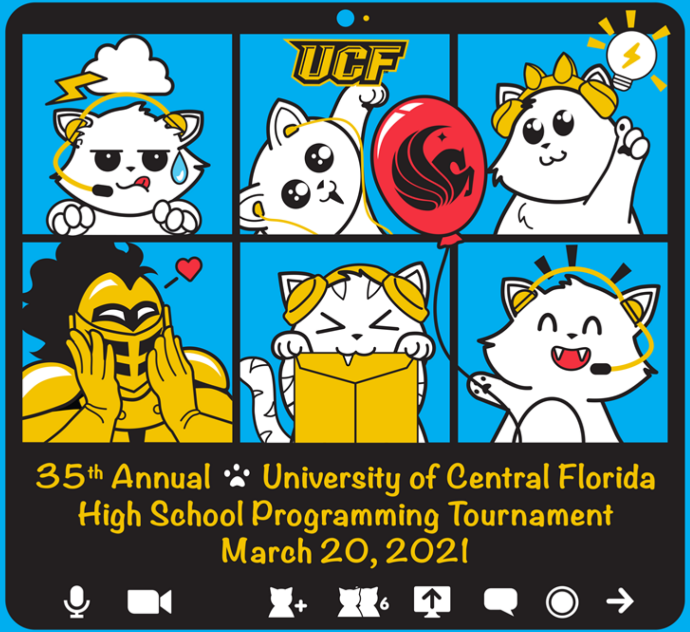
C++: `string myStringWithoutEs = regex_replace(myString, regex(“e”), “”);`

Java: `String myStringWithoutEs = myString.replaceAll(“e”, “”);`

Python: `myStringWithoutEs = myString.replace(“e”, “”)`

Print the string character by character unless the character is equal to ‘e’ (so skip ‘e’s).

Make sure you print each changed string on its own line!


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Interstellon Melon (melon)
80/92 solved/submissions

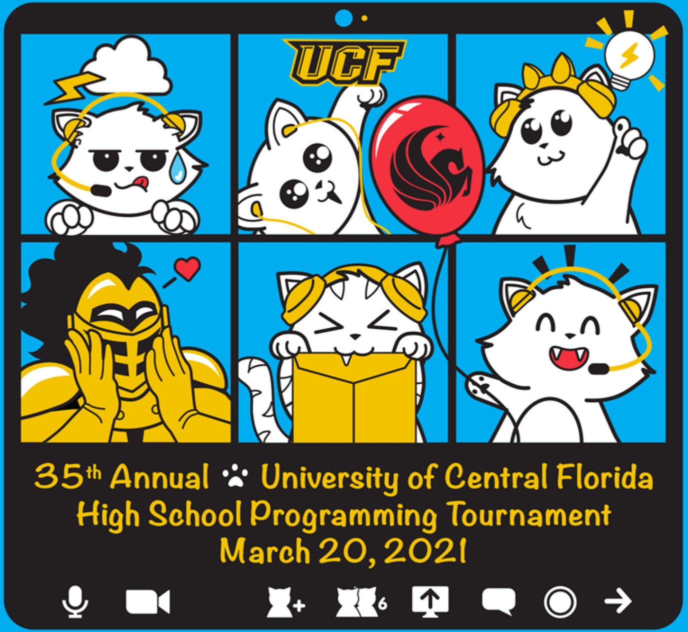
Interstellon Melon (melon)

Our goal is to travel from Earth located at the point (0, 0) to the melon's point (x, y) and back.

According to the problem, we cannot move diagonally, only parallel to either the x or y axis. This means we can just find the distance in x coordinates separately and y coordinate separately and add them together to find the distance.

- totalDistance = |x2 - x1| + |y2 - y1|.
- This distance is known as Manhattan Distance.

Since Earth is located at the origin: totalDistance = (x - 0) + (y - 0) = x + y. We are not done! Remember we have to find the round trip, and since it is the same distance to and back we can just multiply our answer by 2: totalRoundTripDistance = totalDistance * 2 and we can just print out this value.


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Worse Code (worse)

77/101 solved/submissions

Worse Code (worse)

Given a word written in ‘Worse Code’, translate it to plaintext.

Worse Code:

‘.’	= ‘A’
‘..’	= ‘B’
‘...’	= ‘C’
‘....’	= ‘D’

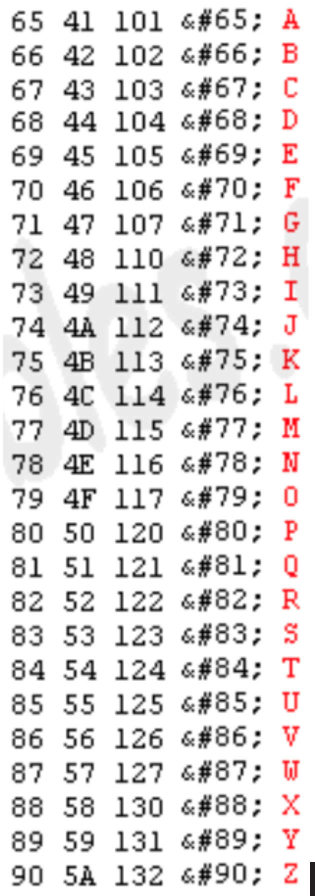
etc…

Solution, use ASCII!

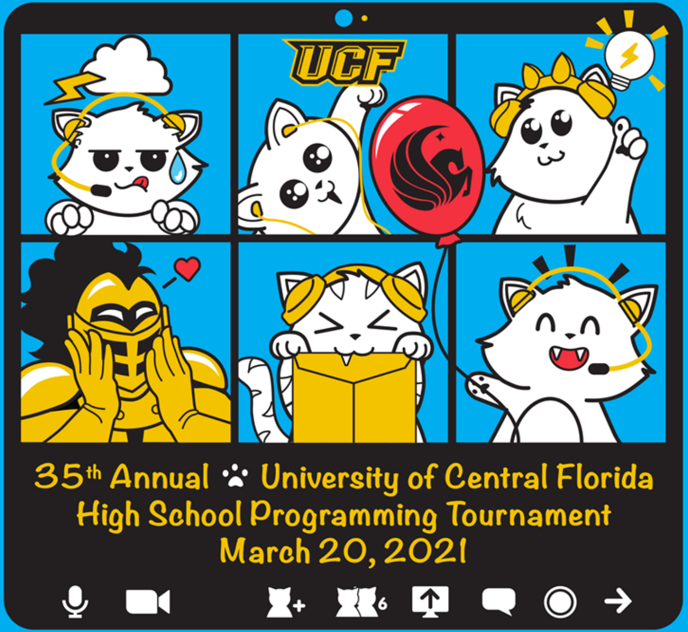
For each string of ‘.’, we can obtain a distance from ‘A’ using its length - 1.

ie:
len(‘...’)-1 = 2
‘A’ + 2 = ‘C’

Print each character in line with the following algorithm



65	41	101	&#65;	A
66	42	102	&#66;	B
67	43	103	&#67;	C
68	44	104	&#68;	D
69	45	105	&#69;	E
70	46	106	&#70;	F
71	47	107	&#71;	G
72	48	110	&#72;	H
73	49	111	&#73;	I
74	4A	112	&#74;	J
75	4B	113	&#75;	K
76	4C	114	&#76;	L
77	4D	115	&#77;	M
78	4E	116	&#78;	N
79	4F	117	&#79;	O
80	50	120	&#80;	P
81	51	121	&#81;	Q
82	52	122	&#82;	R
83	53	123	&#83;	S
84	54	124	&#84;	T
85	55	125	&#85;	U
86	56	126	&#86;	V
87	57	127	&#87;	W
88	58	130	&#88;	X
89	59	131	&#89;	Y
90	5A	132	&#90;	Z


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Improovian Expectations (expectations)

42/196 solved/submissions

Improovion Expectations (expectations)

Given an array of integers, determine by how much you must increase values in order to be left with a strictly increasing array.

- Since each value must be larger than the value before it, we have the following relationship:
  - a[i] > a[i-1]
- We can apply this rule in order from left to right, until we reach the end of the array, keeping track of our modifications.
- However, there's one important constraint that makes this problem tricky, which we'll cover in a few slides.

Improovion Expectations (expectations) cont’d

Let’s apply our algorithm on this example:

Array: [4, 2, 8, 8]

Final: [4, 5, 8, 9]

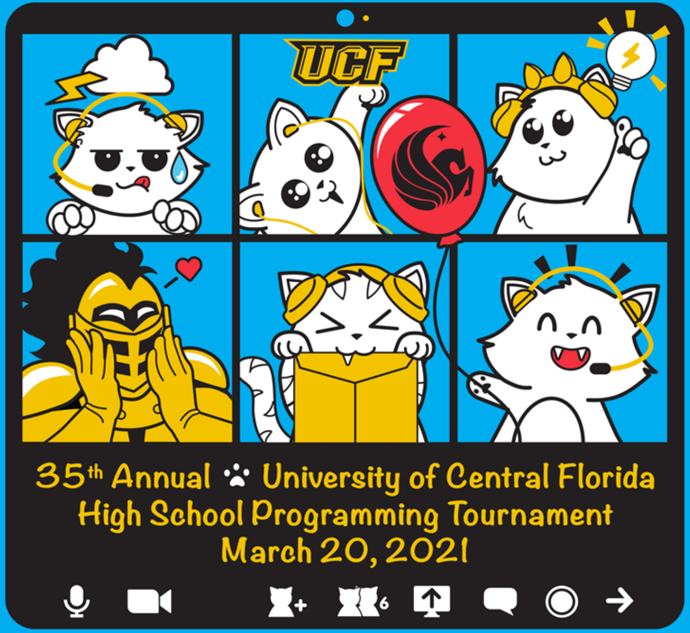
Index	Array	Changed
0	[4, 2, 8, 8] ▲	0
1	[4, 5, 8, 8] ▲	(0+3) = 3
2	[4, 5, 8, 8] ▲	(3+0) = 3
3	[4, 5, 8, 9] ▲	(3+1) = 4

Improovion Expectations (expectations) cont’d

- **IMPORTANT NOTE**: Integer Overflow
- Because computers don’t have infinite memory, computers often store information in set amounts.
    - For a mix of speed and versatility, many programming languages have chosen to store integers in 32 bits, letting each integer represent a value from **-2^{31} (-2,147,483,648) to $2^{31}-1$ (2,147,483,647)** (inclusive).
    - For example, here are the bounds of an “int” in a few languages
        - C++: **-2^{31} (-2,147,483,648) to $2^{31}-1$ (2,147,483,647)** (inclusive)
        - Java: **-2^{31} (-2,147,483,648) to $2^{31}-1$ (2,147,483,647)** (inclusive)
        - Python: No limit! Python is smart with how it stores

Improovion Expectations (expectations) cont'd

- In this problem, the value you have could grow to be very large, which overflows an integer
    - One example: 10^6 "1"s
        - So you'll increase the array to 1, 2, 3, 4, … 10^6. So the answer is (1-1) + (2-1) + (3-1) + … (10^6 -1) = 500,000,500,000, which is too large for a standard integer
- Solution:
    - Use a different variable type
        - C++: "long long": -2^{63} (-9,223,372,036,854,775,807) to 2^{63}-1 (9,223,372,036,854,775,808) (inclusive)
        - Java: "long": -2^{63} (-9,223,372,036,854,775,807) to 2^{63}-1 (9,223,372,036,854,775,808) (inclusive)
        - Python: You're all good!


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Minimum Excluded String (mexstr)
31/90 solved/submissions

Minimum Excluded String (mexstr)

Given a string *s*, find the minimum string *x* such that *x* is not a substring of *s*.

Claim: The length of the Minimum Excluded String will be at most length 2.

Proof: Let's find the lower bound for the length of the string *s* if we want our *x* to be 3. If we can construct the string *s* such that every two adjacent characters will form all unique string of length 2, then our *x* will have to be length 3.

There are 26 ^ 2 unique strings of length 2 that can be achieved with a minimum string *s* of length 26 ^ 2 - 1 = 675. Notice that in the problem, we can only get a string *s* of up to length 500. Since 675 > 500 we are guaranteed to have a Minimum Excluded String (string x) of length 1 or 2.

Minimum Excluded String (mexstr) (cont.)

Now that we know our Minimum Excluded String can only be length 1 or length 2, we can just check all of them and take the minimum string.

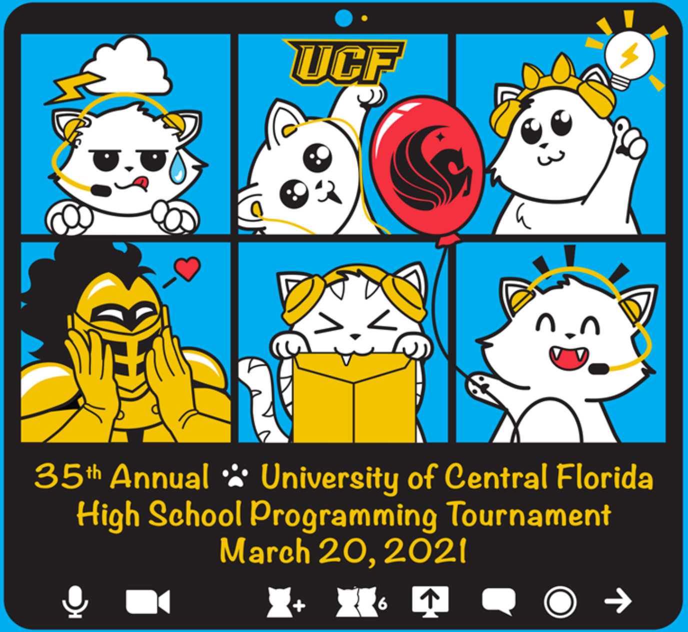
- You can implement this problem cleanly and quickly if you know the contains function for strings in your language like *s*.contains(*x*) in Java.

Check strings of length 1:

- Loop from letters in the order a,b,c,d,...z. If you find that one of these letters is not contained within *s*, print it out as your answer.

Check strings of length 2 when all length 1 strings are contained in *s*:

- Outer loop: Loop from letters in the order a,b,c,d,...z.
    - Inner Loop: Also loop from letters in the order a,b,c,d,...z.
    - Our current string *x* to check for will be *x* = outerLoopLetter + innerLoopLetter
    - If x is not contained within *s*, print it out as your answer


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Bouncing DVD (dvd)

28/49 solved/submissions

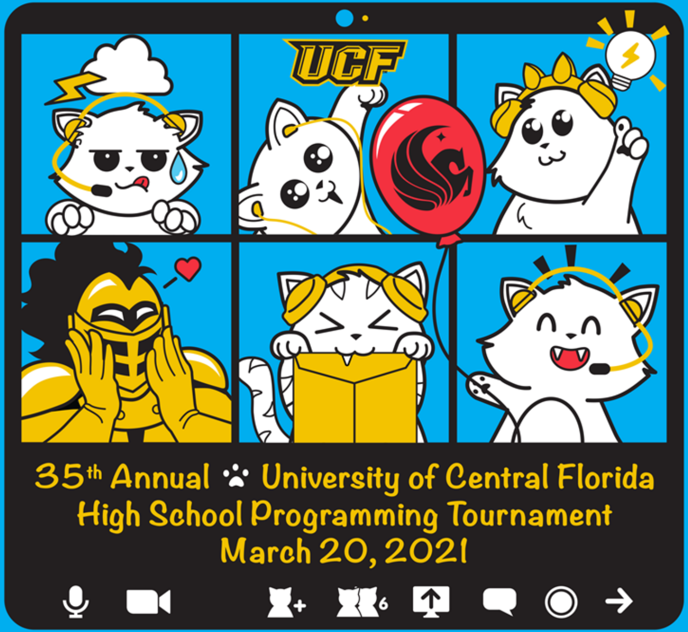
Bouncing DVD (dvd)

Simplify: We can pretend the DVD logo is 1x1 if we subtract the height of the window by the height of the logo and width of the window by the width of the logo.

Simulate it. Maintain the logo’s position and whether the logo moves in a positive X direction and positive Y direction on every move. Otherwise it moves in a negative direction.

When the logo hits a vertical wall it changes X direction, when it hits a horizontal wall it changes Y direction. When it hits both walls at the same time we are done.

Check pos_x = 0, pos_x = width, pos_y = 0, pos_y = height.


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Blob Tag (blob)

24/81 solved/submissions

Blob Tag (blob)

Given a list of collisions in Blob Tag, print the size of the largest group.

- Since each child can have at most 2 connections, we can keep 2 arrays of size n, where each value is initialized to -1.
- For each collision, make sure both children have a space to connect, and if they do, mark the connection in each of their respective arrays.
- After every collision has been processed, for each child, determine the size of the blob they're in.

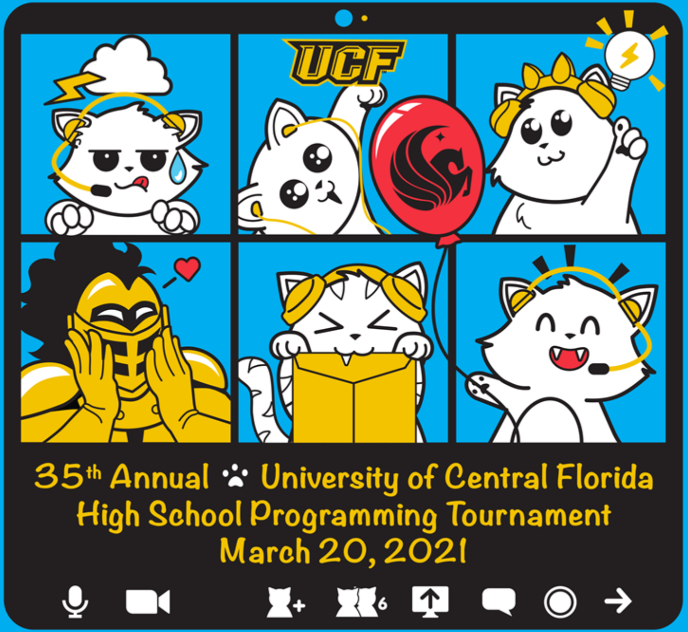
Blob Tag (blob)

For each child, to determine the size of the blob they're in:

- Keep a boolean array of which children you've processed. To process a child, add their neighbors to a queue and mark them as processed. You can consider this the "leader" of the group they're part of.
- Start a counter groupSize at 1. While the queue is not empty, pick one person off it, mark them as processed, add their unprocessed neighbors to the queue, and add 1 to groupSize.
- Once the queue becomes empty, everyone in the leader's group will have been processed and marked, and you will know the size of the leader's group.
- Only designate children to be the "leader" if they're not processed.

Blob Tag (blob)

- Once all the children have been processed, take the max of the group sizes of the leaders. You can store these in an array, or keep a running max.
- Notice that since you only added any one person to the queue one time, everyone was processed just one time, so the total time complexity is O(n) where n is the number of children.


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Triangles (triangles)

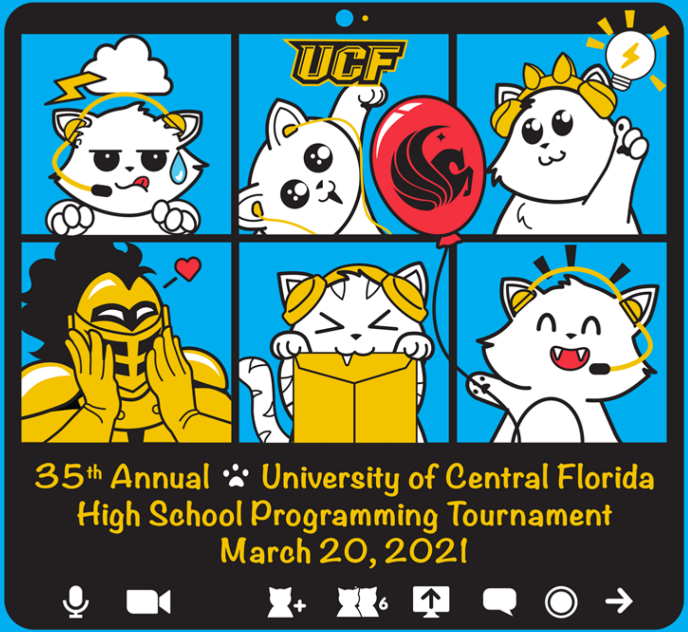
14/55 solved/submissions

Triangles (triangles)

- Using the pythagorean theorem, we can determine a triple of lengths (a, b, c) such that $a \leq b \leq c$ will form a right triangle if and only if $a^2 + b^2 = c^2$
- We can use the same logic to find acute and obtuse triangles:
  - Acute triangles: $a^2 + b^2 > c^2$ (the hypotenuse is too short to be right)
  - Obtuse triangles: $a^2 + b^2 < c^2$ (the hypotenuse is too long to be right)
- These above rules must also obey $a + b > c$ to form a non degenerate triangle (otherwise c would be too long)
- Now to efficiently count these triples...

Triangles (triangles) (cont.)

- The key way we will count is to brute force a and b, and then count the number of suitable values of c that exist to satisfy each of the 3 different triangle types
  - For each type, we will find a range of values that c will need to fall in:
  - Acute: $c > b$ and $c < \text{sqrt}(a^2 + b^2)$, range: $(b, \text{sqrt}(a^2 + b^2))$
  - Right: $c == \text{sqrt}(a^2 + b^2)$, range: $[\text{sqrt}(a^2 + b^2), \text{sqrt}(a^2 + b^2)]$
  - Obtuse: $c < a + b$ and $c > \text{sqrt}(a^2 + b^2)$, range: $(\text{sqrt}(a^2 + b^2), a + b)$
  - (*note: in implementation, both sides of each inequality can be squared to avoid precision issues)
- Each of these range sums can be found using the idea of Prefix Sums
- Total runtime: O(n^2)


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Stealing Spider-Man (spiderman)

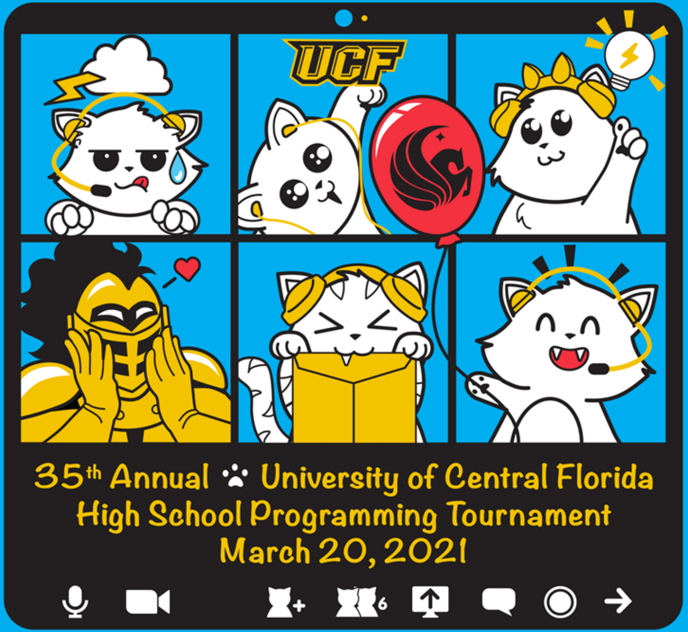
9/31 solved/submissions

Stealing Spider-Man (spiderman)

Need to check if removing one edge disconnects nodes 1 and n.

Iterate over which edge to remove (at most 1,000 of them).

For each iteration, use your favorite graph traversal algorithm (DFS/BFS are good options). Start it at node 1, see if node n is visited by the end. In each iteration make sure to pretend the edge you want to remove doesn’t exist (do not traverse it).

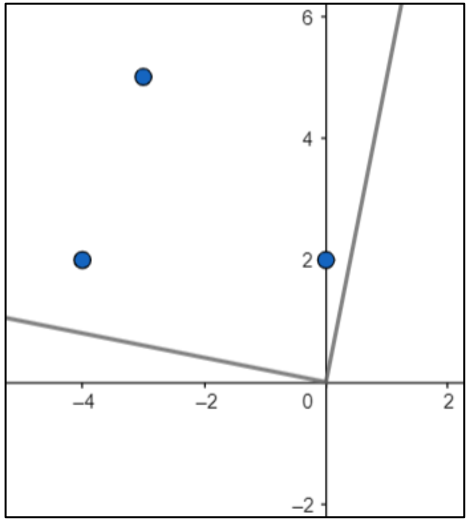

UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

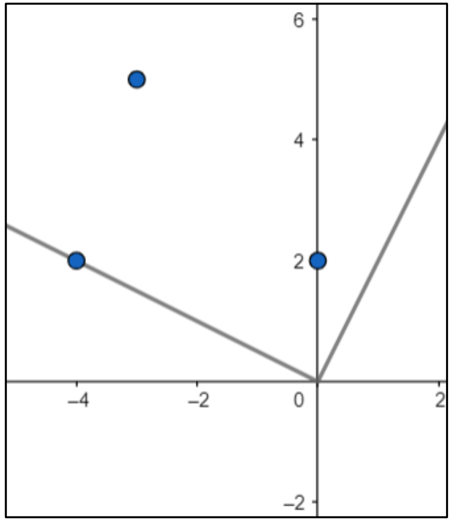
Night Watch (night)

6/29 solved/submissions

Night Watch (night)

- The key observation that can be made is that for any field of view in which no point is exactly on the boundaries, an equivalent field of view can be created by rotating the field of vision clockwise or counterclockwise:

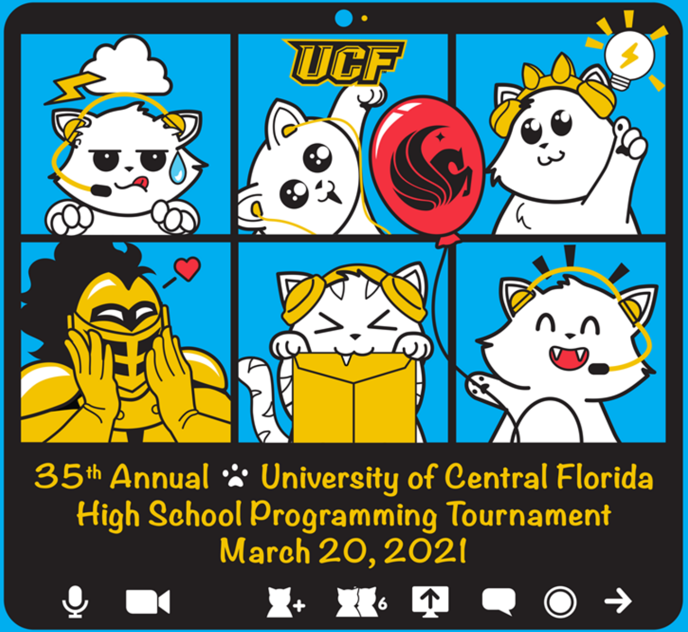




→

Night Watch (night) (cont.)

- This allows us to analyze exactly n different fields of view
- Each field of view will be uniquely defined by the point with the smallest unit circle angle
- We will brute force each point for our start and make it the smallest angle θ
  - This means that every point inside this newly defined field of view will have an angle in the range [θ, θ + 90]
  - We can look through every other point and see if it falls in this range in order to count the number of points in this field of view
- Out of all n fields of view we tried, we will take the largest count
- Total runtime: $O(n^2)$


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Octopus Garden (octopus)

1/5 solved/submissions

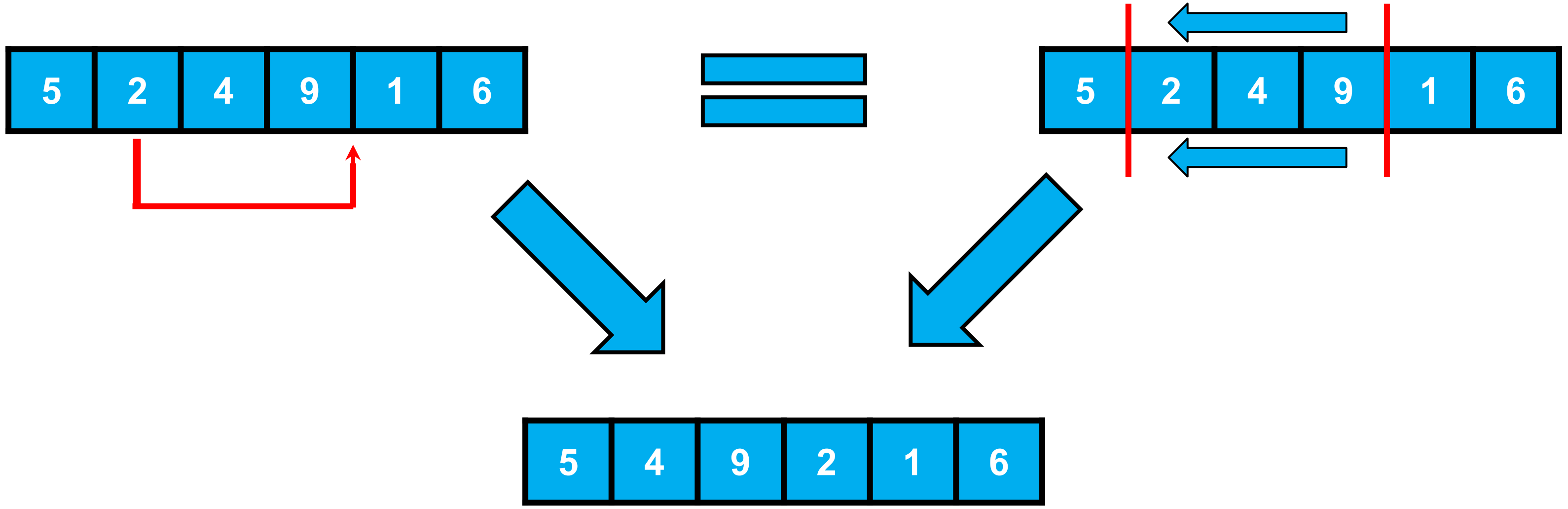
Octopus Garden (octopus)

Given an array of distinct values, determine the minimum number of cyclic subarray rotations by 1 needed to sort the array.

- Since each subarray’s start, end, and shift direction is up to you, we can reduce the operation to moving one element anywhere in the array.

Octopus Garden (octopus) cont'd

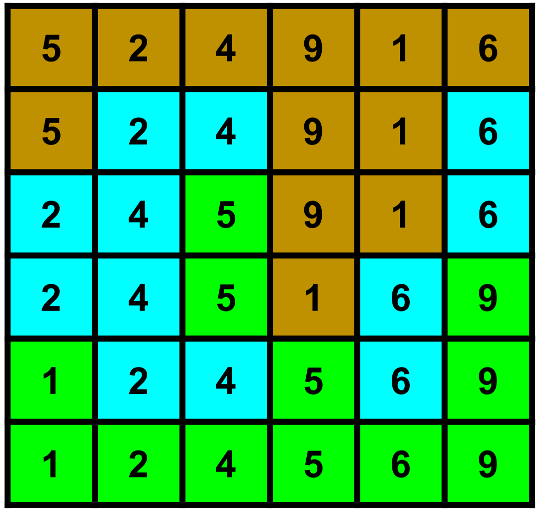
- We can reduce the operation to moving one element anywhere in the array:



Octopus Garden (octopus) cont’d

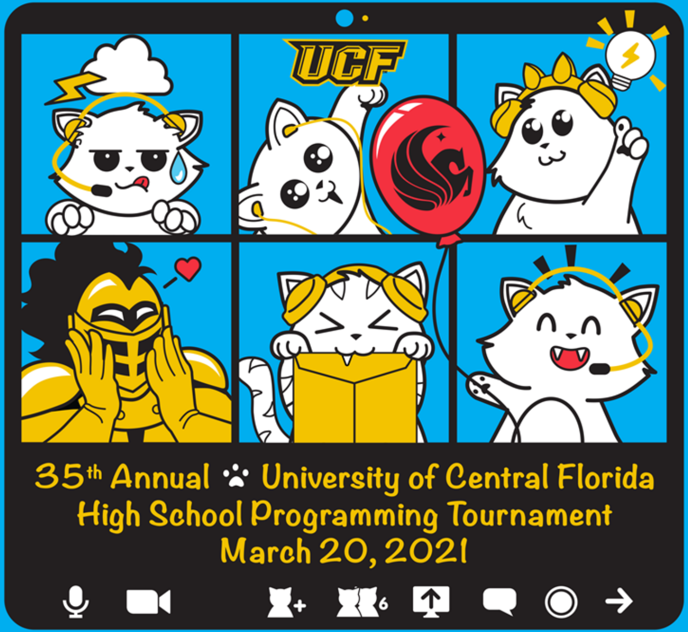
- Since each subarray’s start, end, and shift direction is up to you, we can reduce the operation to moving one element anywhere in the array.
- We never want to transpose any element more than once, because if we moved it from index i -> j -> k, we could have done i -> k (taking into account moves elsewhere in between). This yields upper bound on answer, n.
- Upper bound can be n-1, since we can pick some element to not move.
- If there exists some i, j, such that i < j and arr[i] < arr[j], upper bound can be n-2, since we can not move the elements at i, j and move the rest.
- It follows, then, that the best we can do is pick the **longest increasing subsequence** and not move those. Thus, the answer is n-LIS
- Compute the LIS with one of several techniques found online.

Octopus Garden (octopus) cont’d



Let’s see this solution on some input:	5	2	4	9	1	6
Find some LIS:	5	2	4	9	1	6
Move the 5:	2	4	5	9	1	6
Move the 9:	2	4	5	1	6	9
Move the 1:	1	2	4	5	6	9
Done!	1	2	4	5	6	9

Note: We do not have to print which moves we need to make, so we only care about the length of some LIS


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Knapsack? (knapsack)
1/72 solved/submissions

Knapsack? (knapsack)

We finished a raid, and we're picking up loot! Every item has a weight and a value. We have a maximum capacity that we can carry - what's the maximum value we can collect given that we don't exceed our capacity?

One common answer is greedy.

1) Let's try to take the items with the greatest value, and if they're the same, then take the ones with the smallest weight.

   This doesn't work. Consider a capacity of 10 and this array of items (value, weight):

   (6, 6), (4, 5), (3, 5)

   If we take the 6, we can't take anything else. But if we don't take it, then we can take 4 AND 3 for a total value of 7, instead. This breaks this greedy.

1) How about taking the items with the best value-to-weight ratio? Consider the same array. We would still take the 6 first, leading to a less than optimal solution.

Knapsack? (knapsack)

Instead, we would use a dynamic programming concept called a “knapsack” DP (haha, get it?).

In a regular knapsack DP, we would want to answer the question: from my current index and remaining capacity, what is the maximum value I can reach?

And then at each index we can either “take” the item (and, therefore, carry the weight and gain the value) or “leave” the item and move on (AKA a take-it-or-leave-it DP).

But we can’t use a knapsack DP! Why not?

Knapsack? (knapsack)

It would require us to create a memo table of size 10^9 (the state would be [index][capacity] = [1,000][1,000,000]).

Let’s start making observations! First, what if our capacity is greater than the total sum of all of our items’ weights? TAKE THEM ALL!

But what if the total sum of weights is greater than capacity? Instead of saying, “What’s the max value without exceeding capacity?”, ask the question, “What’s the minimum value I need to get rid of to meet my capacity requirements?”

Then the bounds become much better.

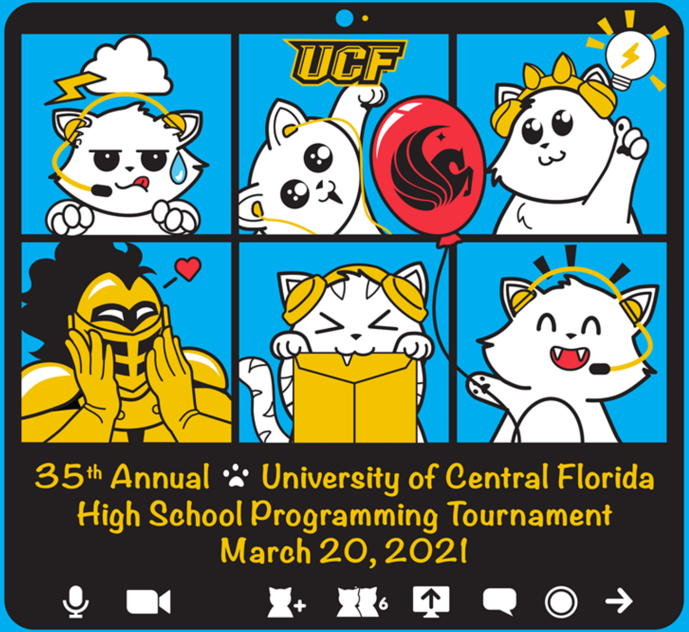
Knapsack? (knapsack)

Now we have a memo state of size 5x10^6. Why?

Well, we still need to keep track of what index we're at (size: 1,000). But instead of our remaining capacity, we have to keep track of our remaining amount to drop (1,050,000 [max total weight possible] - 995,000 [min capacity] = 55,000 [max amount to drop]).

Then, we do a drop-it-or-keep-it DP (copyrighted). At each index, we either drop the current item (and, therefore, we subtract from the amount of weight we need to drop and add the value to the total amount that we lose) or we keep it (suffering no consequence).

We memoize the minimum answer from dropping and keeping, subtract it from our total value, and that's our answer!


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Glenn and Glenn’s Pizza (pizza)

1/13 solved/submissions

Glenn and Glenn’s Pizza (pizza)

Glenn did not write this problem.

Given a list of n people and what toppings they hate, find the minimum number of people Glenn has to share the pizza with by adding up to k toppings.

Although there can be 50,000 toppings to choose from, there are a few observations that simplify the problem.

Glenn and Glenn’s Pizza (pizza)

Observation 1:

- If you can choose a topping for each person (k >= n), then you can make everyone hate the pizza by choosing one topping from each person’s list of hated toppings.
- From the input we know that every person has at least one topping they hate so this is always possible if k >= n.
- This means that k>=n is an easier case (always output 0). So we only have to worry about k<n cases.

Person	Hated Ingredients
Person 1	A, B, C
Person 2	C, D
Person 3	A, B, C
Person 4	D

Glenn and Glenn’s Pizza (pizza)

Observation 2:

Some ingredients are “identical” as they cover the exact same people.

For example:

- With the table of people on the right
- Ingredient A = Ingredient B
  - Since they both are hated by people 1 and 3

Person	Hated Ingredients
Person 1	**A**, **B**, C
Person 2	C, D
Person 3	**A**, **B**, C
Person 4	D

Glenn and Glenn’s Pizza (pizza)

Observation 2 continued:

Since some ingredients are “identical”, we can now think in terms of “ingredient categories”. Each ingredient category has a set of people that hate this ingredient category.

For example:

Person	Hated Ingredients
Person 1	**A**, **B**, C
Person 2	C, D
Person 3	**A**, **B**, C
Person 4	D

$\Rightarrow$

Ingredients	People who hate them
A, B	**1, 3**
C	1, 2, 3
D	2, 4

Glenn and Glenn’s Pizza (pizza)

Observation 2 continued:

We can show that there are at most 2^n of these “categories” of ingredients.

For each person, they can either hate or not hate an ingredient category, which is 2 choices. 2*2*2…*2 for all n people means there’s 2^n unique categories.

2^n

Ingredients	People who hate them
A, B	**1, 3**
C	1, 2, 3
D	2, 4

Glenn and Glenn’s Pizza (pizza)

Observation 2 continued:

Now that we know there’s at most 2^n categories, and n <= 10, that means we have at most 2^n = 1,024 different ingredient categories.

So now we’ve gone from 50,000 ingredients to 1,024 ingredient categories.

Glenn and Glenn’s Pizza (pizza)

Observation 3:

It seems promising that you’d naturally want to choose the ingredient categories that cause the most people to hate your pizza.

For example, if there’s an ingredient category that makes 4 people hate your pizza, and an ingredient that makes 3 people hate your pizza, you’d seemingly want to always pick the ingredient that makes 4 people hate your pizza.

However, this greedy approach does not work, as shown in the test case on the next slide.

Glenn and Glenn’s Pizza (pizza)

Observation 3 continued:

Break Test Case:

n = 6, k = 2. Ingredients are on the right table.

You can make *everyone* hate your pizza by choosing A and B.

However, if you greedily choose based on how many people hate an ingredient, then you’d choose C (which has 4 people hate your pizza). Then with your remaining topping choice you’d choose A or B, but that will leave with one remaining person who likes your pizza!

Thus, the greedy approach does not work.

Person	Hated Ingredients
Person 1	A
Person 2	A, C
Person 3	A, C
Person 4	B, C
Person 5	B, C
Person 6	B

Glenn and Glenn’s Pizza (pizza)

To recap

- Observation 1: If k>=n, then you can always choose toppings to make everyone hate your pizza. In other words, k>=n is an easier case, k<n is where we have to do work.
- Observation 2: You only have 2^n (2^10 = 1024) different topping categories (not 50,000).
- Observation 3: Greedy doesn’t work.

So with all this in mind, what does work?

Dynamic Programming!

Glenn and Glenn’s Pizza (pizza)

Dynamic Programming Solution Idea:

State:

We need to keep track of two things:

- How many toppings we’ve taken (k).
- What people currently hate our pizza so far (2^n).
  - We’ll store this as a sequence of n bits, where a 0 means this person doesn’t hate the pizza, and a 1 means the person hates the pizza.
  - As there’s n bits that are either a 0 or a 1, each bit has a decision of size 2, so 2*2*...2 = 2^n, so there’s 2^n possible combinations of people that hate our pizza.

So there’s a total of k*(2^n) different states. O(9 * 1024)

Glenn and Glenn’s Pizza (pizza)

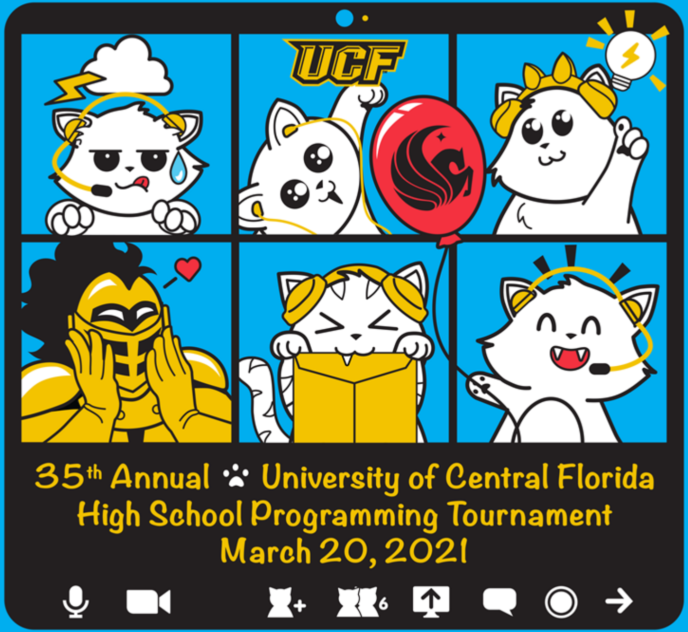
Dynamic Programming Solution Idea continued:

Transition:

For each topping type, we’ll either take this topping, or skip it.

If you do take a topping, you can see the effect of taking this topping by using the binary OR operation between the current state of people who currently hate your pizza and the people who hate the topping you’re taking.

People who hate the pizza so far	00110
People who hate this topping you’re considering	10101
People who hate the pizza if you take the topping	10111


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Molecular Mole (mole)

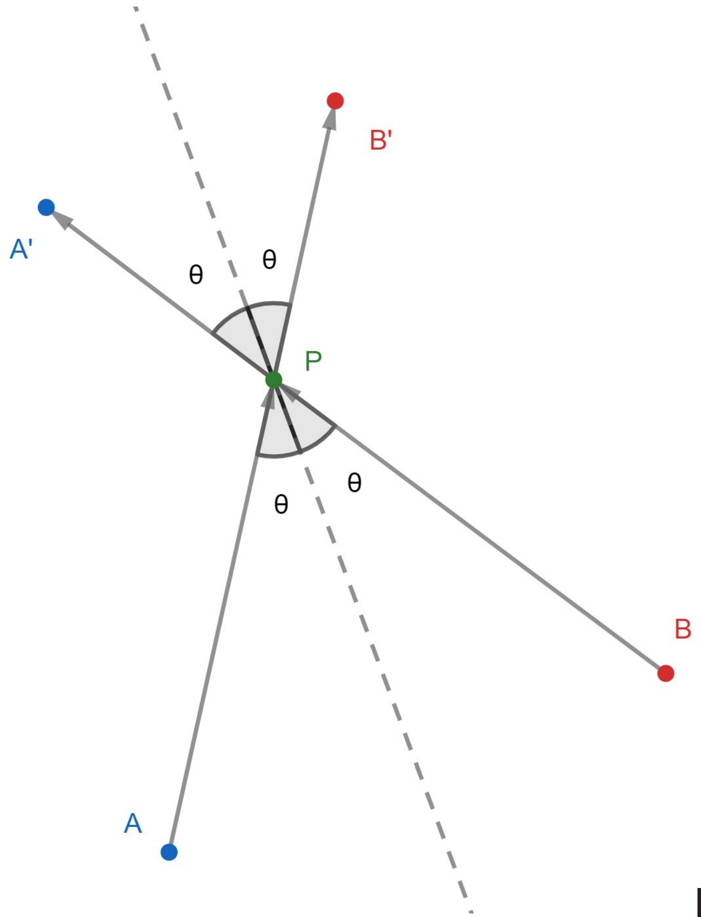
0/1 solved/submissions

Molecular Mole (mole)

Main observation: notice that after a collision, all we need to do is swap the rays which represent the molecules.

The ray for A’ is the ray for B, and the ray for B’ is the ray for A.

We can compute all collision points between all pairs of rays and sort them, then simulate.

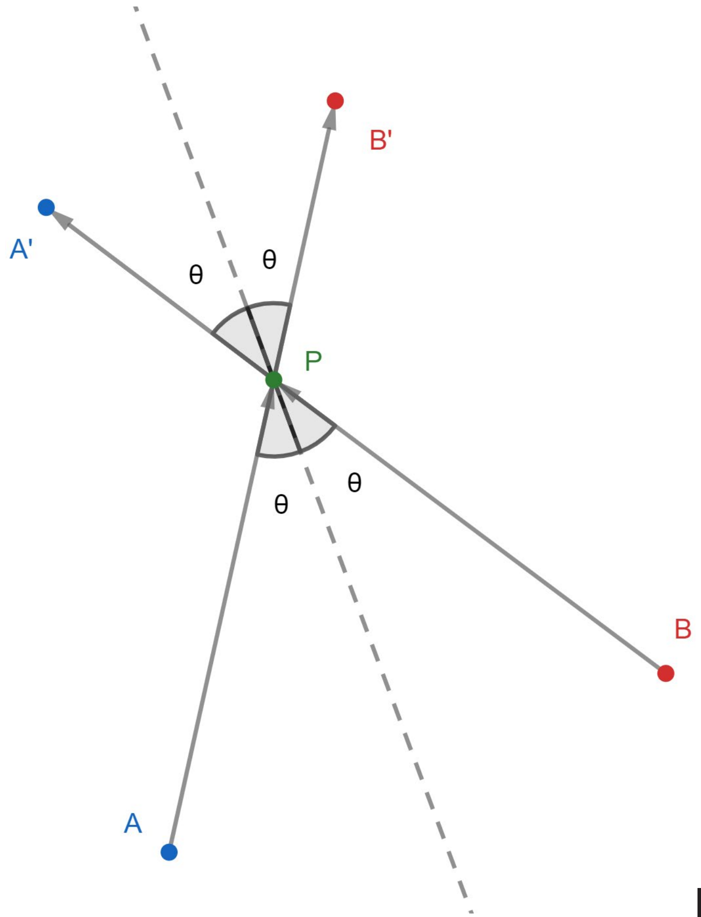


Molecular Mole (mole) (cont.)

Computing collision points:

For non-parallel rays: There is one intersection. Two moles collide at that intersection if they are equally distant to it.

Parallel rays: If the two rays are collinear and facing each other, the collision point is the midpoint between the two ray endpoints.

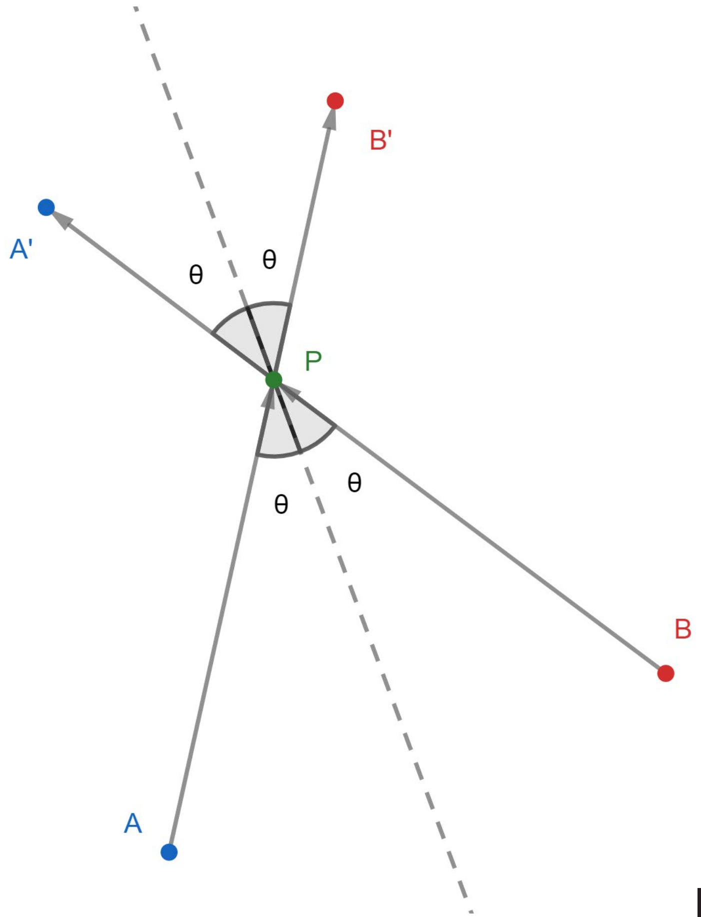


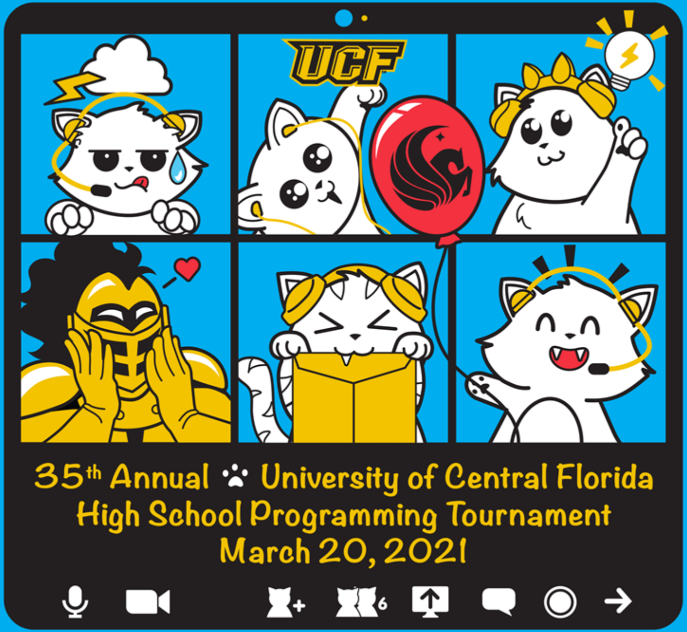
Molecular Mole (mole) (cont.)

Once we computed the collision points and sorted them, we initialize index[i] to be the mole being represented by each ray i.

For each collision point, we maintain which two rays a and b collide at this point.

We increment the answer for the two moles represented by the rays, and swap index[a] <-> index[b].




UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Alphabetic Road Trip (roadtrip)
0/3 solved/submissions

Alphabetic Road Trip (roadtrip)

Given a map of cities and the roads between them, what is the shortest length of an alphabetic road trip?

A few key details from the flavor text:

1) An alphabetic road trip is a path that visits a city beginning with each letter of the alphabet (up to ‘J’) in alphabetical order.
2) We can choose which cities we consider “visited”, i.e., we can pass through a city without counting it as a city on our trip.
3) We can start at ANY city that begins with an ‘A’.
4) We’re done as soon as we reach a city that begins with a ‘J’.

Alphabetic Road Trip (roadtrip)

Let’s solve some subproblems. First, what if we only had to find the shortest distance from point A to point B with edge weights?

We would use an algorithm called Dijkstra's, which uses a priority queue (or min heap) to find the shortest distance from one node to every other node in a graph. In short, it does this by keeping track of the current minimum path’s current node and adding all of that node’s edges to the PQ with the new path weight!

Now that we know how to find the shortest path between two nodes with edge weights, let’s tackle the next subproblem. We need to know what letter of the alphabet we just visited so we can go to the next letter.

Alphabetic Road Trip (roadtrip)

Let’s consider brute forcing all possible pairs of ‘A’s to ‘B’s, then ‘B’s to ‘C’s… etc. and use Dijkstras to find the minimum path length between each pair.

While this seems promising, our decisions have too many repercussions. For example, the minimum path from Alabama to Bermuda might be of length 6, but the min path from Bermuda to Canada is length 100. Meanwhile, Alabama to Bahama is length 10 and from Bahama to Canada is length 4.

The answer here is clear, and it is shows how this kind of solution could get complicated fast (let alone considering the run-time, which is n^2 or 10^{10}, plus a log factor).

Oh no!

Alphabetic Road Trip (roadtrip)

Instead of trying to force our order, let’s add a dimension to our Dijkstras which keeps track of what letter we last visited, as well as our current node index.

Then, when we look at our current minimum path, we can always go to the next node with our current letter (i.e., going through a city without visiting it), and if the city happens to begin with the next letter of the alphabet, we can go to the next node and update our current letter (i.e., visiting this city).

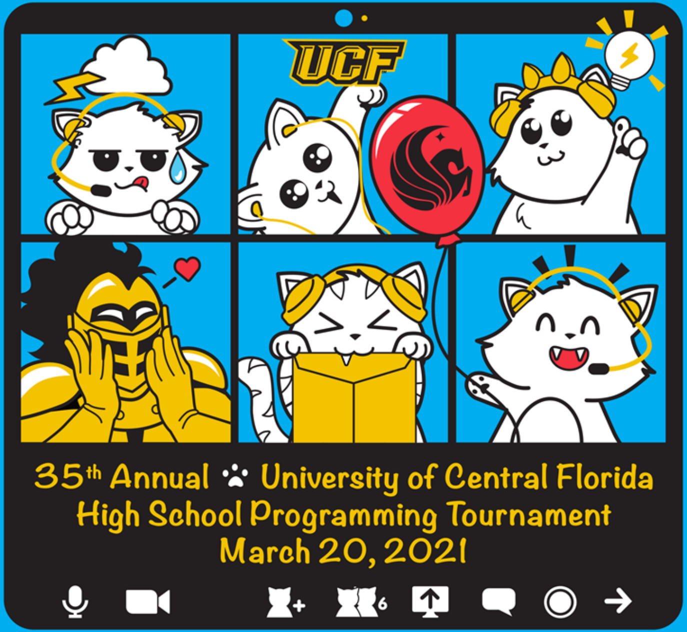
This is called Split-Node Dijkstras! The run-time is the size of the state ([alphabet_size][index] = [10][100,000] = 10^6 times a log factor to sort).

Alphabetic Road Trip (roadtrip)

Last important note!!

Remember the fact that we can start at any city beginning with ‘A’? We need to add all cities that begin with ‘A’ to our PQ with a distance of 0 before we start to make sure we consider all of our options. This is called a “Multisource” search.

Likewise, at the end, we need to consider all possible cities starting with ‘J’ that we could have ended at to ensure we find our minimum answer!


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Sharon’s Sausages (sausage)
0/23 solved/submissions

Sharon’s Sausages (sausage)

Sharon did not write this problem.

We want to count subsequences of the form ABBA.

Idea: For each AB we find, how many BAs can we find to the right to match with?

We can brute force with a quadruple for-loop, but it’s too slow ($n = 10^5$, $n^4 = 10^{20}$ operations)

The bound on values $x \le 100$, how can we use this to our advantage?

Sharon’s Sausages (sausage) (cont.)

Let’s keep next[i]: the next index with the same value as index i (n-sized array).

We can do this by storing the previous index i when a value appears, and setting next[prev[i]] = i.

Let’s also keep count[i][j]: how many pairs (a[i], j) appear in the suffix of the array starting at i, where a is the input array and j is a value from 1 to 100 (100*n-sized array).

Maintaining a frequency array for the suffix, for each position in the array (back-to-front), set count[i][j] = freq[j]. This is equal to the number of pairs (x, j) where x is the element at i. We can then add count[i][j] += count[next[i]][j] to include the rest of the pairs (x, j) where x is an element equal to a[i] in the range (i+1, n).

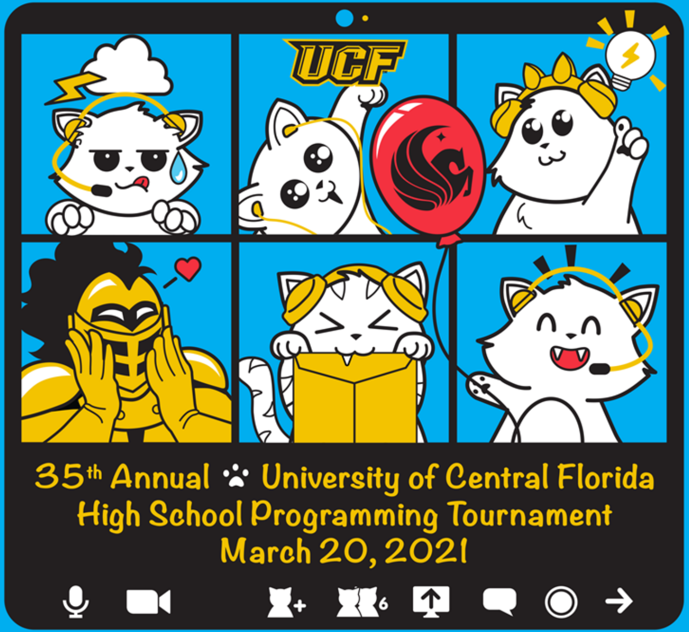
Sharon’s Sausages (sausage) (cont.)

Now we go front-to-back. For each position of i, for each value (from 1-100), the frequency of the value j is the amount of times the pair (j, a[i]) appears (same ideas as before).

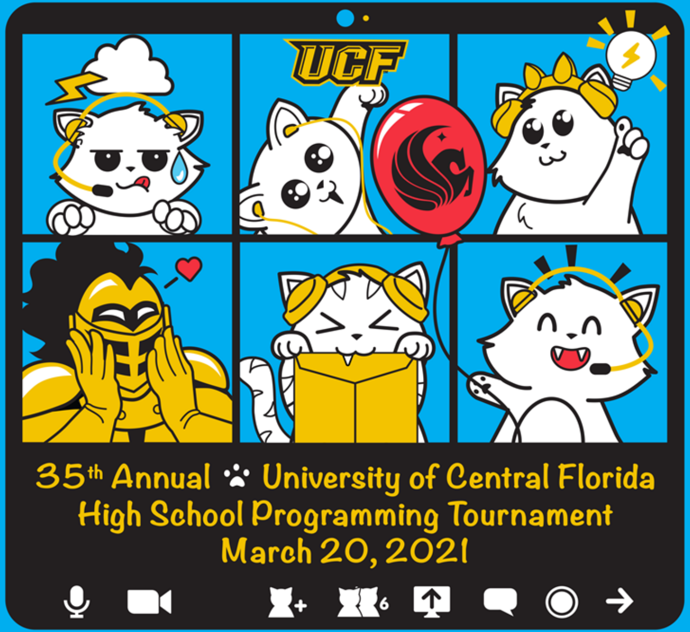
We multiply freq[j] by count[next[i]][j] to get the number of quadruplets (j, a[i], a[i], j) where the second element in the pair is fixed to be the element at index i.

The sum of the results across all indices i is our final answer.

Runtime: ~(100*n = 10^7 operations)


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021

Questions?


UCF
35th Annual University of Central Florida
High School Programming Tournament
March 20, 2021
6

Thank You!