ACM ICPC 2015 Recap

(some of the Division 2 problems)

Intro

All problems can be found at the links below:

Division 1:
http://serjudging.vanb.org/wp-content/uploads/SER-2015-Problems-D1.pdf

Division 2:
http://serjudging.vanb.org/wp-content/uploads/SER-2015-Problems-D2.pdf

Test inputs and outputs can be found here:
http://serjudging.vanb.org/

More info about the ICPC:

Blur

"You have a **black and white image**. You decide to represent this image with one number per pixel: **black is 0, and white is 1**. Someone asks you to **blur the image**, resulting in
various shades of gray."

Blur: New value of a pixel is the average of the 9 pixel box around it. Wrap around the edges. Repeat this process a fixed number of times and **determine the number of unique colors in the final image**.

First three numbers are *w* (image width), *h* (image height), and *b* (number of times to blur). The next *h* lines each contain *w* space-separated integers with a 0 or 1, denoting the color of that pixel.

Output: A single number (the **number of unique colors after blurring**)

Sample Input

```
5 4 1
0 0 1 1 0
0 0 1 1 0
0 0 1 1 0
0 0 1 1 0
```

Sample Output

```
3
```

Blur

Limits:

- 3 <= w, h <= 100
- 0 <= b <= 9

Thoughts:

- Could use **doubles** to represent fractions, but we might **lose precision**. However, in terms of **unique colors**, we can take the **sum instead of averaging**, and use **long ints** instead.
- Brute force approach:
  - Make a temp array, put the blurred value of the image into it, and then replace the old array with the temp array (can't do in-place blurring)
  - Time Complexity: O(w*h*b), but the operations are very fast (addition and multiplication). Given our limits,100*100*9 = 100,000 operations is doable.
  - Space Complexity: O(w*h)
- Upper bound on sum: 9^9 = 387,420,489 will fit inside an integer

Blur

Input	Blur Once
5 4 1	0 3 6 6 3
0 0 1 1 0	0 3 6 6 3
0 0 1 1 0	0 3 6 6 3
0 0 1 1 0	0 3 6 6 3
0 0 1 1 0	

Count unique colors = 3

A Classy Problem

"In his memoir *So, Anyway*, comedian John Cleese writes of the class difference between his father (who was 'middle-middle-middle-lower-middle class' and his mother (who was 'upper-upper-lower-middle class')."

Goal: **Sort a group of people** by their given classes.

Parameters: **Start from left and work backwards**. For example, "middle lower" is below "lower middle". Additionally, "middle middle lower" is identical to "middle lower" (the beginning is assumed to be middles)

Maximum number of identifiers: ~50

First line: Number of people, *n* (no specified bound on *n*, so **we'll assume *n* is pretty small**)
Following *n* lines are formatted like:
mona: upper upper class

A Classy Problem

Sample Input:	Sample Output:
5	queenelizabeth
mom: upper upper lower middle class	mom
dad: middle middle lower middle class	dad
queenelizabeth: upper upper class	unclebob
chair: lower lower class	chair
unclebob: middle lower middle class	

A Classy Problem

Thoughts:

- Don't need to think about complexity so much. In fact, we could use Java's **built-in sorting methods** to do the sorting part for us.
- Could use Java's **Comparable interface**? Then we put everything in an array, sort the array, and print everyone's names in the order they appear in the array.
- Our approach: Make an object **Person** that holds a string *name* and an array *classes* of length 50ish.
    - For each element in the array, -1 means lower, 0 means middle, and 1 means upper.
    - *compareTo(Person p)* method needs to **correctly compare two people**
- Alternative approaches: Could think about this as a ternary tree, then traverse the tree from low to high.
    - Space complexity could get out of hand.

Grid

You are given an ***n* x *m* grid** where each element contains a number **0 - 9**.

The number of an element of the grid is the **number of spaces you can jump** in any of the four directions up, down, left, or right. You can't jump outside the grid, so **sometimes no path will exist**.

Goal: Find the **shortest path** from the **top left corner to the bottom right corner**.

Output: Single integer for the **length of the shortest path** (-1 if no path exists)

Grid

Input
2 2
11
1**1**

Output
2

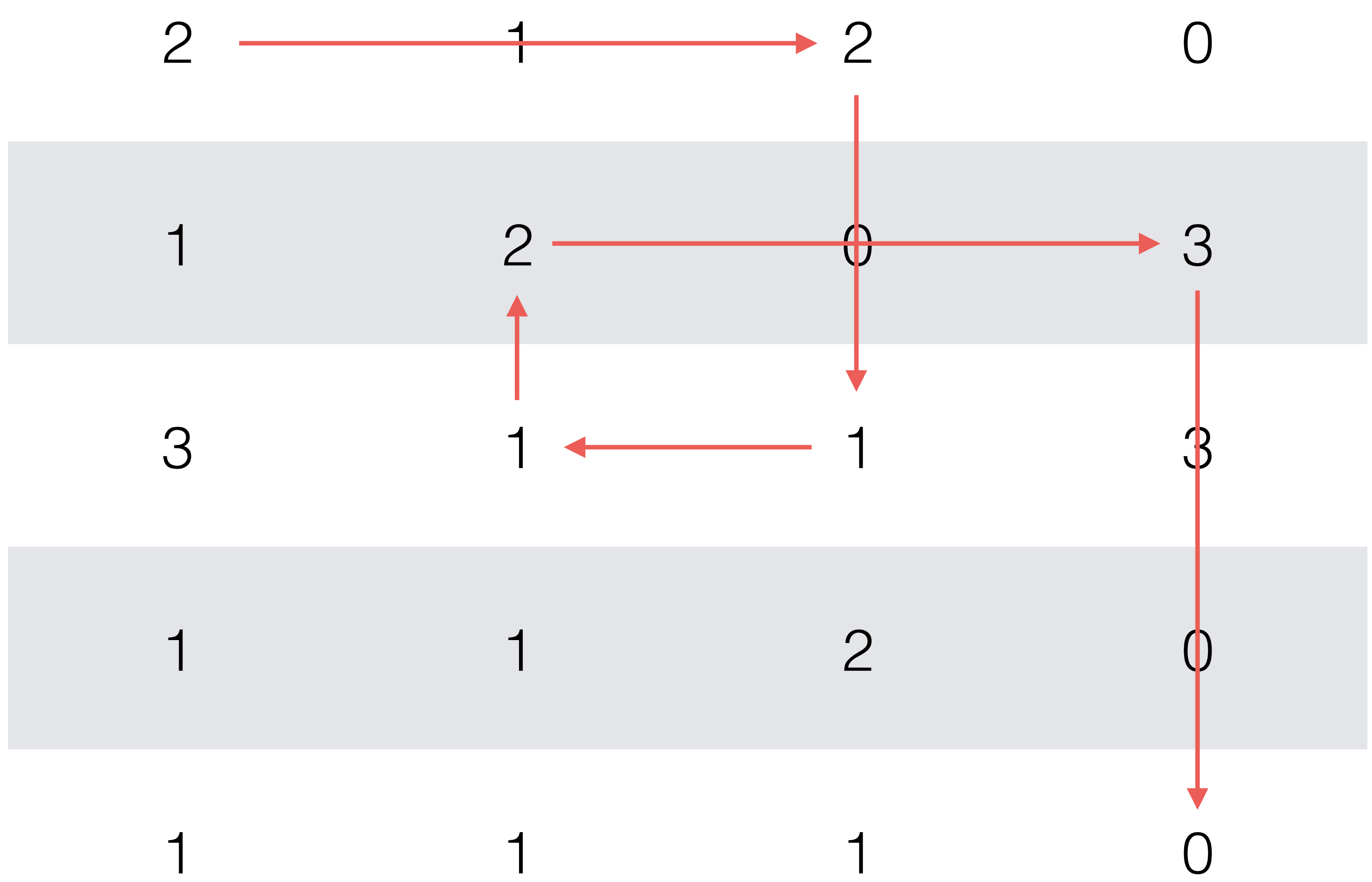
Input
2 2
22
22

Output
-1

Input
5 4
21**2**0
1**2**0**3**
3**11**3
1120
111**0**

Output
6

Grid



Input

5 4

21**2**0

1**2**0**3**

3**11**3

1120

111**0**

Output

6

Grid

Limits:
- 1 <= n, m <= 500

Thoughts:
- What type of problem does this resemble?
- How do we solve that type of problem?
- What do we need to look out for?

The Magical 3

Given some number ***n***, find the smallest base ***b*** in which the last digit of that number is a 3.

Examples:

- Input: 11. 11 in base 4 is 23, so output 3
- Input: 42. 42 in base 13 is 33, so output 3

The Magical 3

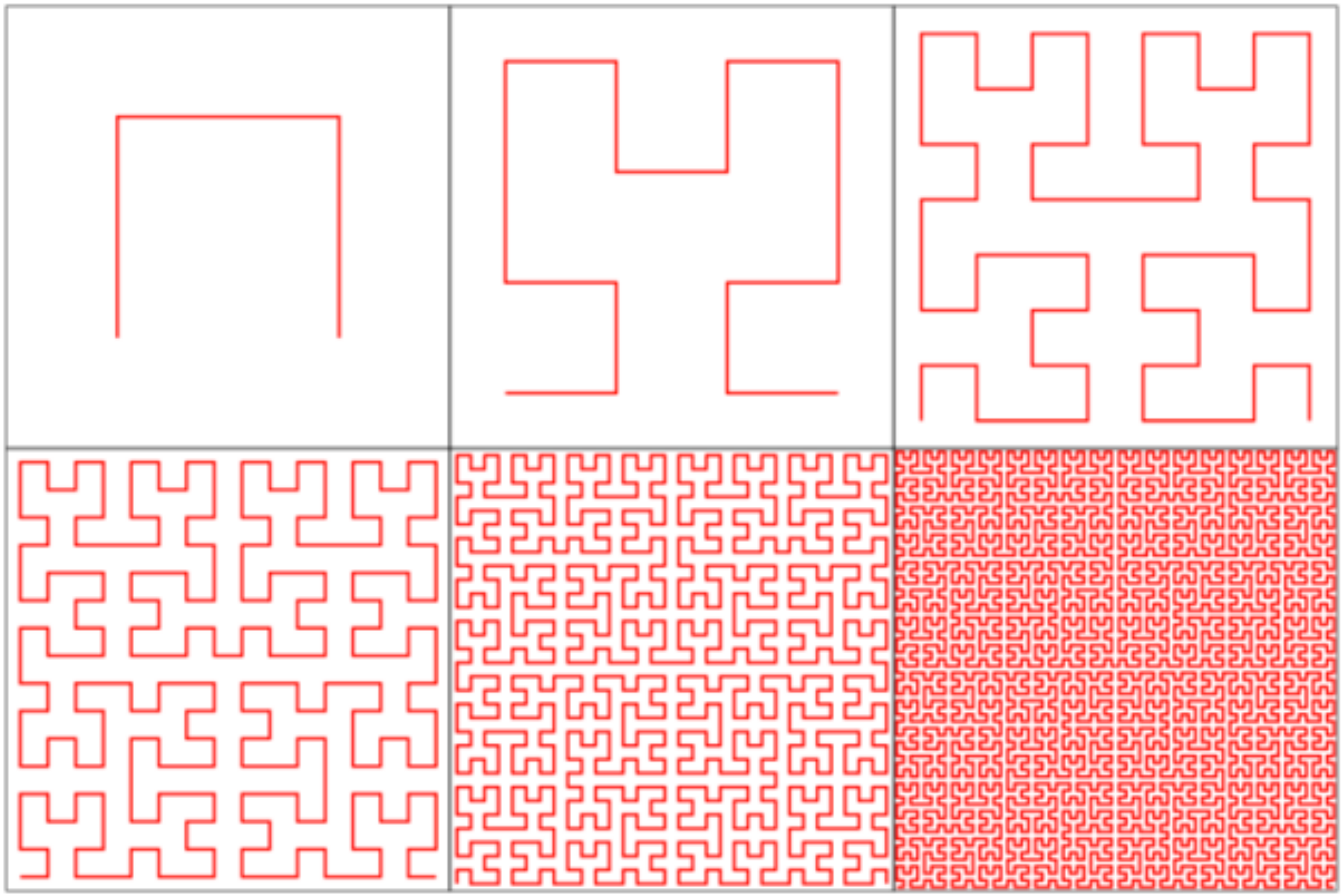
Limits:

- 7 <= *n* <= 2^31 (really big number!)

Thoughts:

- What is the equivalent problem?
- Can this be done efficiently?
- Square root of 2^31 is around 2^16 = 65,536

Hilbert Sort



The Hilbert Curve

Sort points based on when they appear on the Hilbert curve

Hilbert Curve

Limits:

- 1 <= *number of points* <= 100,000
- 1 <= *point value* <= 10^9

Thoughts:

- Can you think about this problem recursively?
  - Bucket / radix sort?

Fun Fact:

- Hilbert sorting can be used to approximate a solution to the traveling salesman problem.

The End