

CS 9

Week 1 Problems
"Watch out for bugs!"

Andrew Benson
Ian Tullis



Game plan for Thursdays

- Introduce 3 problems (~5 minutes)
- ~35 minutes of working time
 - There is no implicit expectation that you should be able to fully solve all three, or even one, let alone write full code. Just get as far as you can.
- Go over solution to second problem together
 - First problem - video walkthrough posted on Canvas
 - Third problem - written solution at end of deck

Star system for difficulty

★: not "easy" per se but has a straightforward solution

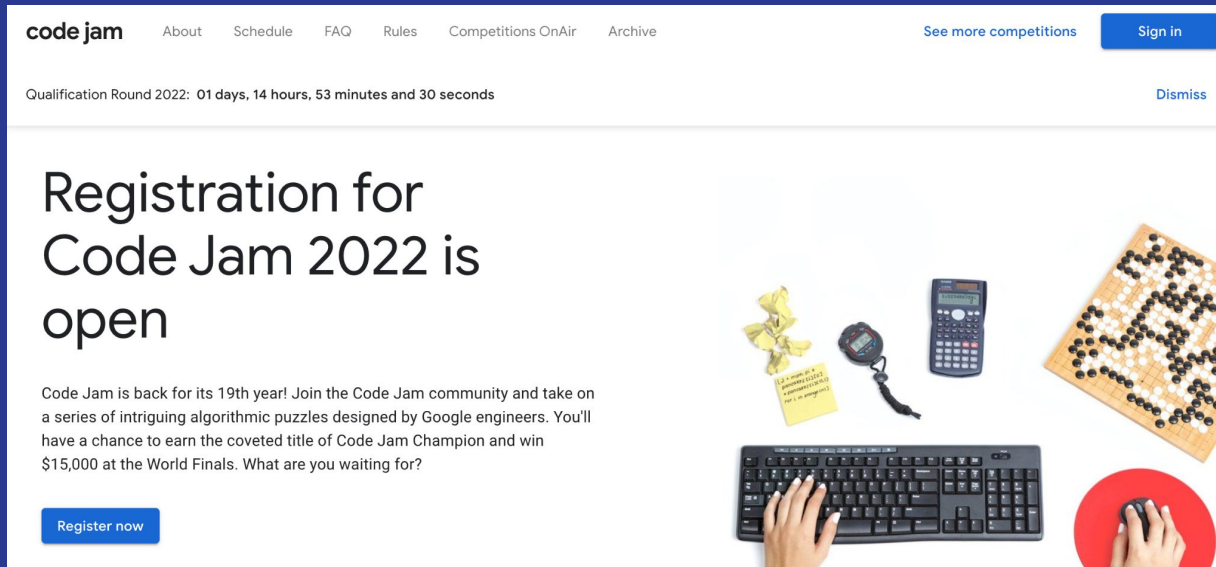
★★: potentially tricky, requires a significant insight

★★★: quite intricate/challenging, would be a difficult interview (but these do still show up)

kind of like Leetcode Easy / Medium / Hard...

A quick plug (since this was once Ian's job...)

- Google Code Jam starts this Friday! It's good practice and can get you an interview! (You have to advance in the Qual Round this weekend to participate in future rounds)



The screenshot shows the Google Code Jam 2022 registration page. At the top, there is a navigation bar with links for 'code jam', 'About', 'Schedule', 'FAQ', 'Rules', 'Competitions OnAir', and 'Archive'. On the right side of the navigation bar, there are links for 'See more competitions' and a blue 'Sign in' button. Below the navigation bar, a progress indicator shows 'Qualification Round 2022: 01 days, 14 hours, 53 minutes and 30 seconds' with a 'Dismiss' link on the right. The main content area features a large heading 'Registration for Code Jam 2022 is open'. Below the heading, there is a paragraph of text: 'Code Jam is back for its 19th year! Join the Code Jam community and take on a series of intriguing algorithmic puzzles designed by Google engineers. You'll have a chance to earn the coveted title of Code Jam Champion and win \$15,000 at the World Finals. What are you waiting for?'. At the bottom left of the main content area, there is a blue 'Register now' button. The background of the main content area is a collage of images related to coding and puzzles, including a keyboard, a mouse, a calculator, a Go board, a stopwatch, and a sticky note.

code jam About Schedule FAQ Rules Competitions OnAir Archive See more competitions Sign in

Qualification Round 2022: 01 days, 14 hours, 53 minutes and 30 seconds Dismiss

Registration for Code Jam 2022 is open

Code Jam is back for its 19th year! Join the Code Jam community and take on a series of intriguing algorithmic puzzles designed by Google engineers. You'll have a chance to earn the coveted title of Code Jam Champion and win \$15,000 at the World Finals. What are you waiting for?

Register now

General problem-solving advice

- Make sure you understand the problem. (Ask the interviewer questions if needed!) Consider working through a small test case.
- What would a brute force solution look like?
 - Don't code it up unless you don't get farther than that and need to get some code down
- What makes that solution inefficient? Can you see how to remove those inefficiencies?

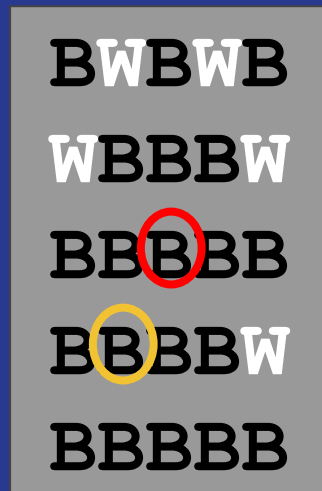
★ Problem 1-1: Legs



- I'm at the insect store looking for new pets. There are N (up to 1000000) individual insects for sale in the store, and I want to buy *exactly two different* ones. Moreover, I want my two insects to have a combined total of exactly L legs, where L is between 0 and $2 * 10^9$.
- I have an array: the i -th element is the number of legs the i -th insect has, from 0 to 10^9 .
- Determine whether I can get what I want. If so, give a pair of distinct indices of insects (counting starting from 0) that satisfy my needs.
- Example 1: $[0, 3, 14, 6, 4, 4]$, $L = 10$: one acceptable answer is 3, 5.
- Example 2: $[0, 3, 14, 6, 4, 4]$, $L = 12$: return IMPOSSIBLE

☆☆ Problem 1-2: Spider Count

- I have an N by N grid of cells, each of which is either black (B) or white (W). N can be as large as 1000.
- A k -spider is defined as a black cell from which there are at least $k-1$ more consecutive black cells immediately to the right, at least $k-1$ more consecutive black cells immediately to the upper right, and so on for all eight directions.
- In the example to the right, the central (red) cell is a 3-spider. The yellow cell is a 2-spider. (Spiders can overlap!)
- A cell is given the highest number possible, so e.g. a 3-spider is not also considered a 2-spider. Any black cell that is not a 2-spider (or higher) is a 1-spider.
- Your goal: count the number of spiders of each size in the grid. (Here we have 1 3-spider, 2 2-spiders, and 17 1-spiders.)



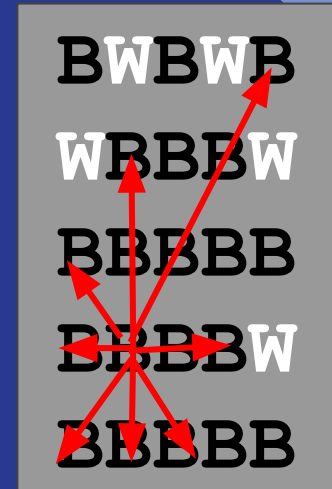
☆☆ Problem 1-3: Caterpillars

- There are C caterpillars dangling from the oak trees on campus. The i -th caterpillar is C_i inches off the ground. (C and the C_i s can be as large as 100000.)
- You have M caterpillar-collecting machines. The i -th machine can reach caterpillars that are up to M_i inches off the ground. Each caterpillar is reachable by at least one machine. (M and the M_i s can be as large as 100000.)
- You want to assign each caterpillar to a machine, but you do not want there to be one machine that doesn't do much work. Let N_1, \dots, N_M be the total numbers of caterpillars collected by the different machines. Your goal is to maximize $\min(N_1, \dots, N_M)$.
- Example 1 : $C = 5$; $C_i = [60, 47, 33, 70, 50]$; $M = 2$; $M_i = [70, 65]$. Answer: 2.
One optimal solution is to assign the first two caterpillars to machine 2 and the last three to machine 1.
- Example 2 : $C = 3$; $C_i = [50, 60, 55]$; $M = 2$; $M_i = [40, 60]$. Answer: 0.
The first machine can't reach any caterpillars, so we are out of luck.



Spider Count: 🦋 Brute Force 🦋

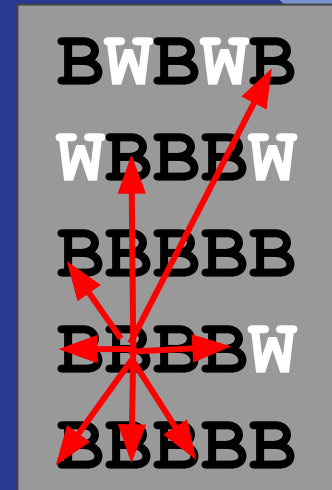
- For each cell in the grid:
 - For each of the eight directions:
 - Count the number of B cells in that direction (including the center) until we either see a W or reach the edge of the grid.
 - Take the minimum k of these counts. Increment the number of k -spiders by 1.



234
2 3 min=2
222

Spider Count: 🦋 Brute Force 🦋

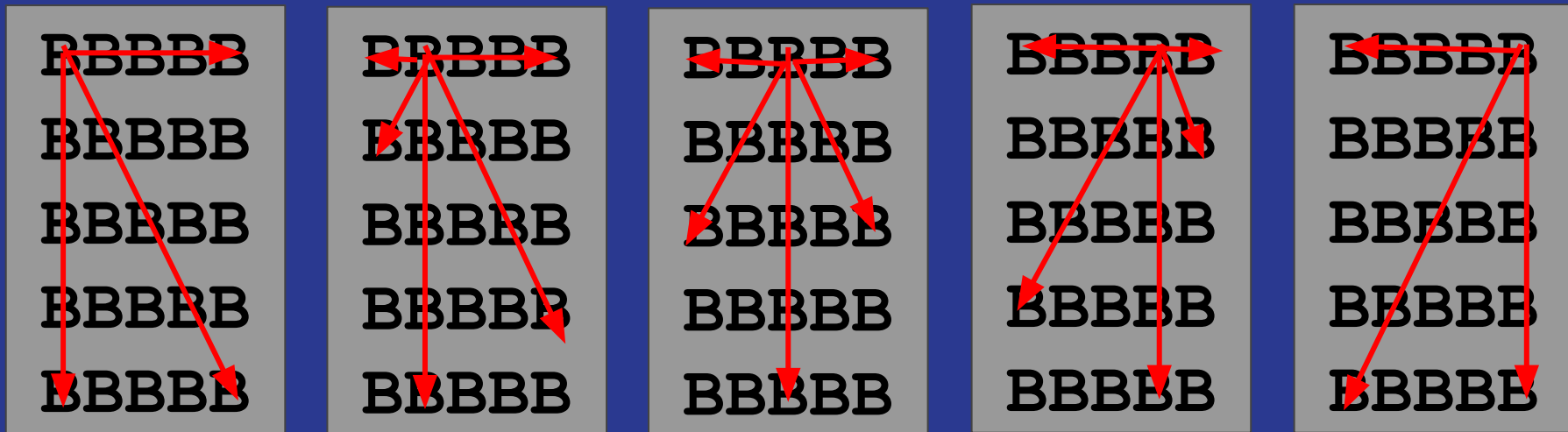
- For each cell in the grid: $O(N^2)$
 - For each of the eight directions: $O(1)$
 - Count the number of B cells in that direction (including the center) until we either see a W or reach the edge of the grid. $O(N)$
 - Take the minimum k of these counts. Increment the number of k -spiders by 1. $O(1)$
- Overall running time: $O(N^3)$



234
2 3 min=2
222

What's inefficient?

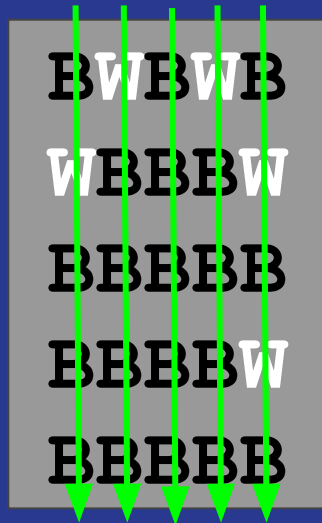
- Explores/learns the same stuff over and over!
e.g. the first row here...



Preprocessing idea

- For each cell in the grid, for each direction, learn the number of consecutive black cells in that direction.
 - maybe make 8 separate grids, one for each direction...
- If we can manage to do this only once for each direction, we avoid redundant work!
- Once we have these 8 grids, counting spiders is easy! For each grid cell, we just take the minimum of those 8 numbers.

Creating a grid for one direction



1	0	1	0	1
0	1	2	1	0
1	2	3	2	1
2	3	4	3	0
3	4	5	4	1

Notice that this actually finds the number of consecutive black cells going in the direction *opposite* to the one we just used. But that's fine as long as we check each direction and its opposite!

Diagonals are more annoying

BWBWB

WBWBW

BBBBB

BBBBW

BBBBB

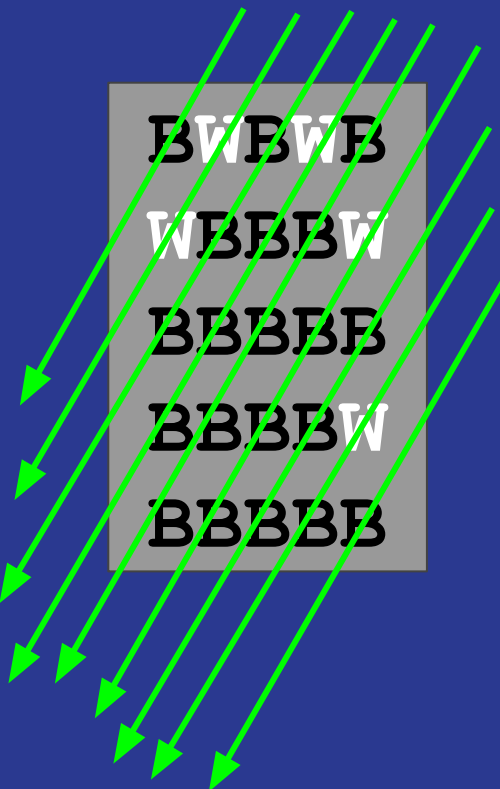
10101

02120

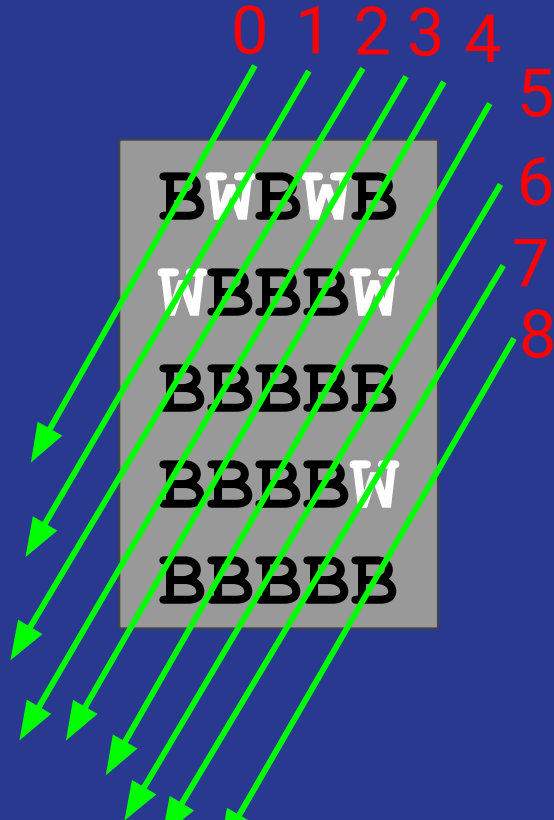
32311

34220

53311



A useful tip (not for this problem)



- Down-left (up-right) diagonals are defined by the *sums of* row/column indices.
- Down-right (up-left) diagonals are defined by the *difference in* row/column indices.

Spider Count: Improved Solution

- For each of the eight directions:
 - Make an empty grid that is the size of the original grid.
 - Traverse each line in that direction. Keep a cumulative count of consecutive black cells seen, and record these in the new grid. Reset the count to 0 when a white cell is encountered.
- For each cell in the grid:
 - Take the minimum k of the values in that position across all 8 grids. Increment the number of k -spiders by 1.

Spider Count: Improved Solution

- For each of the eight directions: $O(1)$
 - Make an empty grid that is the size of the original grid. $O(N^2)$
 - Traverse each line in that direction. Keep a cumulative count of consecutive black cells seen, and record these in the new grid. Reset the count to 0 when a white cell is encountered. $O(N^2)$
- For each cell in the grid: $O(N^2)$
 - Take the minimum k of the values in that position across all 8 grids. Increment the number of k -spiders by 1. $O(1)$
- Overall running time: $O(N^2)$

Slightly Nicer Improved Solution

- Make an empty grid that is the size of the original grid. Fill each cell with some huge placeholder value like $N+1$.
- For each of the eight directions:
 - Traverse each line in that direction. Keep a cumulative count of consecutive black cells seen, and "record" these in the new grid by taking the minimum of this value and the grid's current value. Reset the count to 0 when a white cell is encountered.
- For each cell in the new grid:
 - Read the value k in that cell. Increment the number of k -spiders by 1.

One implementation

```
def solve(grid):
    n = len(grid)
    new_grid = [[n+1 for _ in range(n)] for __ in range(n)]
    for row_dir, col_dir, start_points in (
        # orthogonal directions
        (1, 0, [(0, c) for c in range(n)]),
        (-1, 0, [(n-1, c) for c in range(n)]),
        (0, 1, [(r, 0) for r in range(n)]),
        (0, -1, [(r, n-1) for r in range(n)]),
        # diagonals
        (-1, -1, [(r, n-1) for r in range(n-1)] + [(n-1, c) for c in
range(n)]),
        (-1, 1, [(r, 0) for r in range(n)] + [(n-1, c) for c in
range(n-1)]),
        (1, -1, [(r, n-1) for r in range(1, n)] + [(0, c) for c in
range(n)]),
        (1, 1, [(r, 0) for r in range(n)] + [(0, c) for c in range(1,
n)])):
        check(grid, n, new_grid, row_dir, col_dir, start_points)
    counts = [0]*(n+1)
    for r in range(n):
        for c in range(n):
            counts[new_grid[r][c]] += 1
    return(counts)

def check(grid, n, new_grid, row_dir, col_dir, start_points):
    for start_row, start_col in start_points:
        curr_row, curr_col = start_row, start_col
        cumulative = 0
        while 0 <= curr_row < n and 0 <= curr_col < n:
            if grid[curr_row][curr_col] == 'W':
                cumulative = 0
            else:
                cumulative += 1
            new_grid[curr_row][curr_col] = min(
                new_grid[curr_row][curr_col], cumulative)
            curr_row += row_dir
            curr_col += col_dir
```

this would be hard to code fully in a real interview...

Can we do better?

- We obviously need to at least look at every cell*, and there are $O(N^2)$ cells, so no! At least not asymptotically...
- (However, there *might* be something with better constant factors. Remember that industry does care about those! Let us know if you find an alternative...)

**ok, maybe not if we use a quantum computer, but probably don't go there in an interview unless you have a good reason*

Other stuff to think about

- Can you generalize this to multiple dimensions (e.g. "sea urchins" in a 3D grid)? How would the running time depend on the dimension D ?
- Can this solution be parallelized across multiple machines? If so, how?
- How would you test your solution? (What cases would have the longest running time? Are there edge / corner cases that are easy to mess up?)

*Now Andrew will say the
magic password*

Caterpillars answer

Trying all ways of assigning individual caterpillars to individual machines would be exponentially complex, so we have to do some thinking.

Conceptually, we can have the shortest machine collect all the caterpillars it can, then have the next shortest machine collect all the *remaining* caterpillars it can, and so on. Then we keep finding the machine that currently has the minimum number of caterpillars, and improving it by passing one caterpillar up from the next shortest machine. (Here, "passing" means changing the assignment of who collected that caterpillar.) This is always safe because anything a shorter machine collected could have also been collected by a machine at least as tall. Once the current minimum can't be improved (which happens only when the shortest machine has the minimum), stop. This can be done directly via simulation, but it might take around as many steps as there are caterpillars!

Caterpillars answer, continued

A more efficient way is to see that the solution must be the *smallest* one of the following:

- the total number of caterpillars that the shortest machine can reach. (No other machine can pass anything to it!)
- the total number of caterpillars that the second shortest machine can reach, divided by 2. (That is, the second shortest machine can only get surplus from the shortest machine. The best strategy is to distribute the caterpillars as evenly as possible. If this is not possible, e.g. the shortest can only reach 5 but the second shortest can reach 15, then this situation is already covered by the first bullet point.)
- the number of caterpillars the third shortest machine can reach, divided by 3, and so on.

So we can sort the lists of caterpillars and machines, and keep cumulative counts of how many total caterpillars can be reached by the first k machines. Then we divide the first value by 1, the second by 2, the third by 3, etc., and take the smallest result, rounded down. This takes only $O(N \log N)$ time -- where N is the maximum of M and C -- because a sorting step dominates.