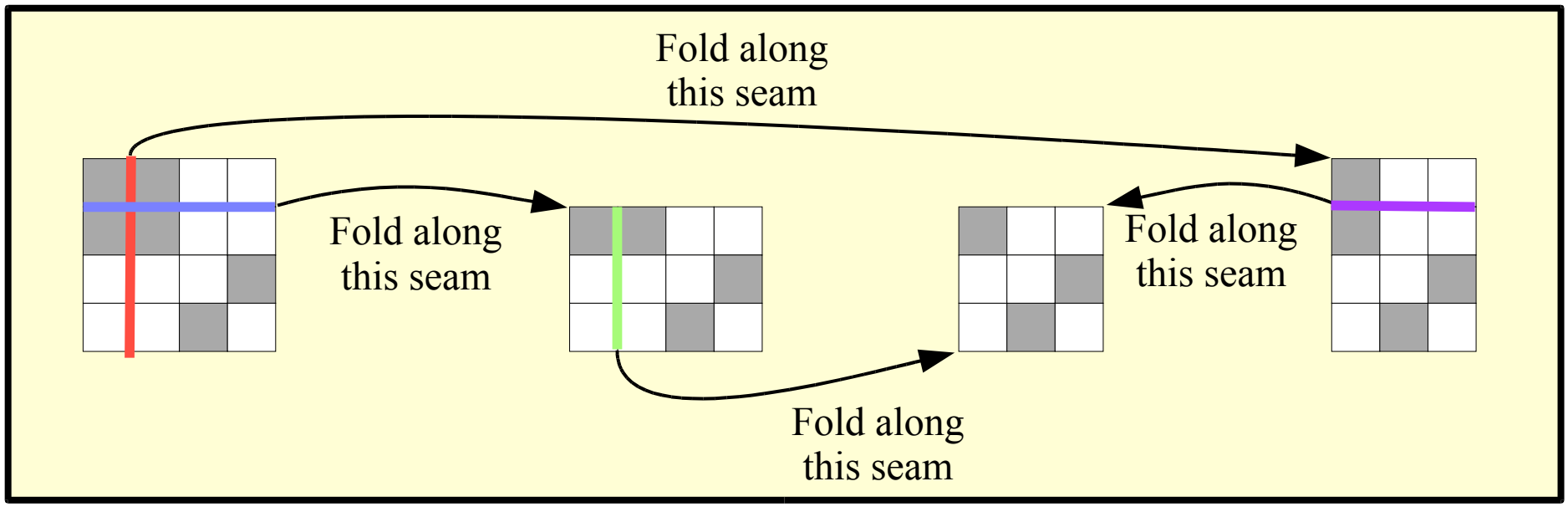
Topcoder SRM 639, D1, 500-Pointer "BoardFolding"

James S. Plank
EECS Department
University of Tennessee

CS494/594 Class
January 27, 2026

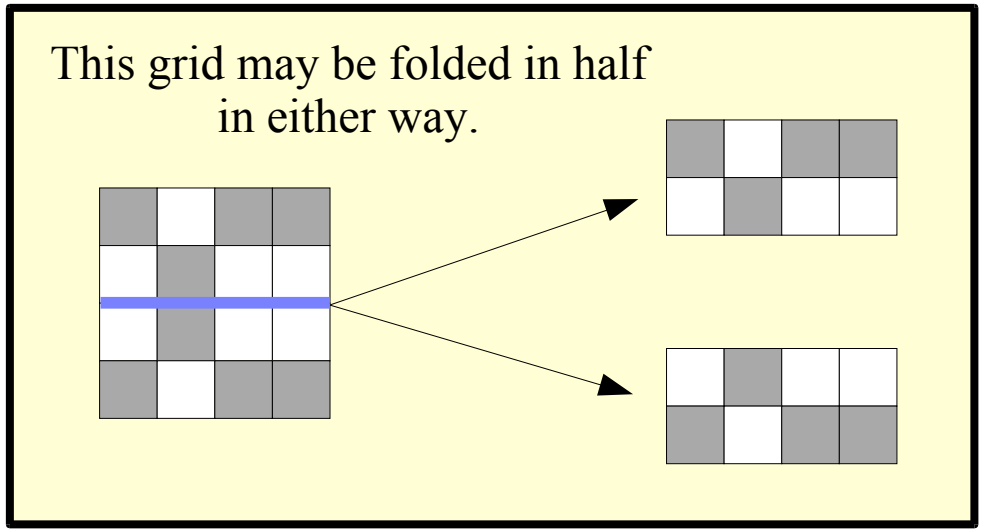
The problem

- You are given a rectangular grid with N rows and M columns.
- Each entry of the grid is either 0 (white) or 1 (gray).
- You may "fold" the grid along the seam between rows or columns as long as the smaller portion goes on top of the larger portion, and the two portions match exactly.



The problem (continued)

- If the two portions are of equal size, then either portion may go on top of the other.

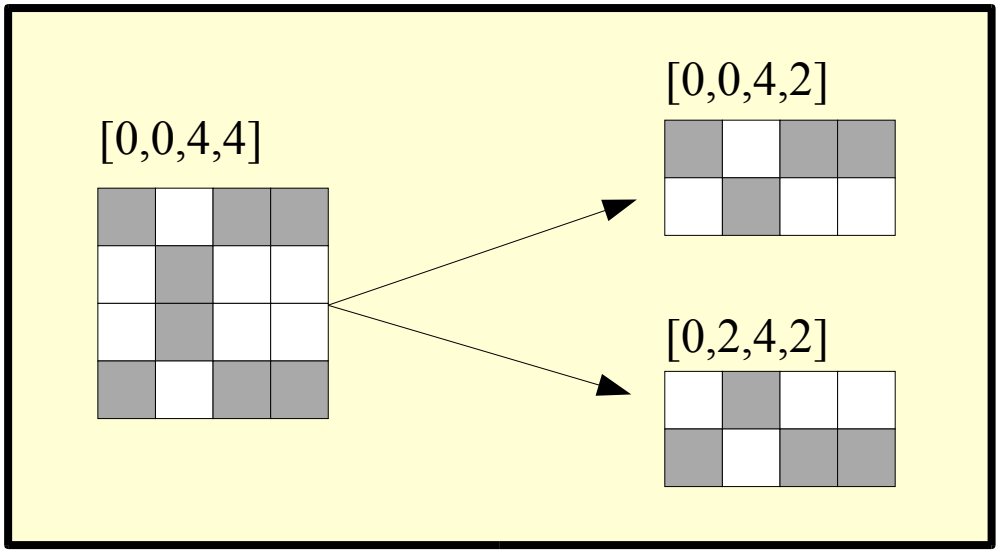


- Another way to visualize folding is to simply delete the part of the rectangle that goes on top.

The problem (continued)

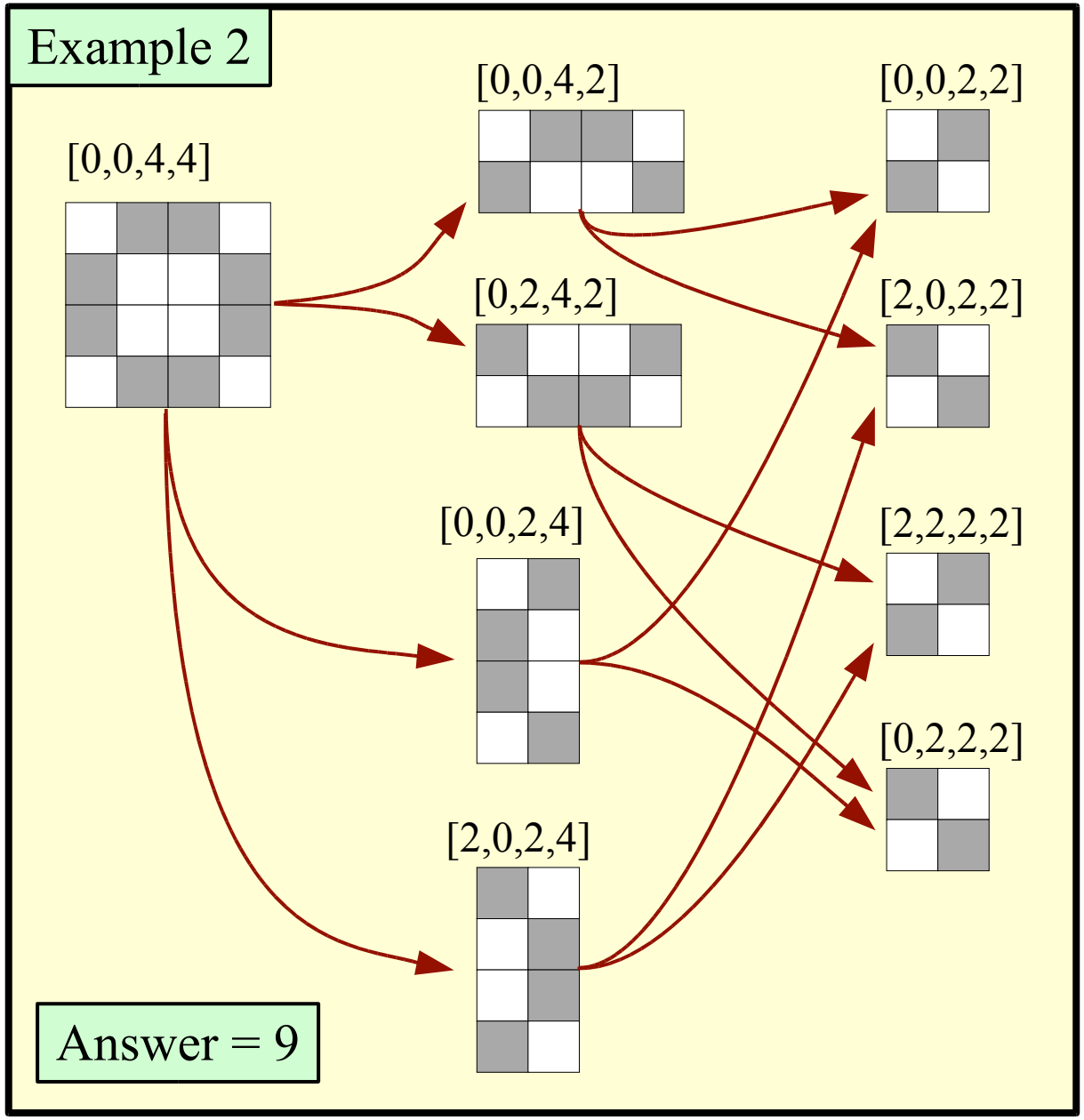
Suppose we label the result of zero or more folds by:

- *x*: Starting x position of upper-left cell.
- *y*: Starting y position of upper-left cell (0 at top).
- *w*: Width of the resulting grid.
- *h*: Height of the resulting grid.
- Label it *[x,y,w,h]*



The problem (continued)

Given a starting grid, how many unique labelings can result from zero or more folds?



Prototype and Constraints

- **Class name**: `BoardFolding`
- **Method**: `howMany()`
- **Parameters**:

N	`int`	Number of rows
M	`int`	Number of columns
Grid	`vector <string>`	The grid (in compressed format)

- **Return Value**: `int`
- **Constraints**: *N* and *M* are between 0 and 250.
  - (which is roughly 2^8)

Thought #1: Enumeration

How many potential *[x,y,w,h]* are there?

x & w:

- *C* columns with width 1
- *(C-1)* columns with width 2
- *(C-2)* columns with width 3

$$\sum_{i=1}^{M} i \approx (2^8 * 2^8) = 2^{16}$$

y & h:

- *R* rows with height 1
- *(R-1)* rows with height 2
- *(R-2)* rows with height 3

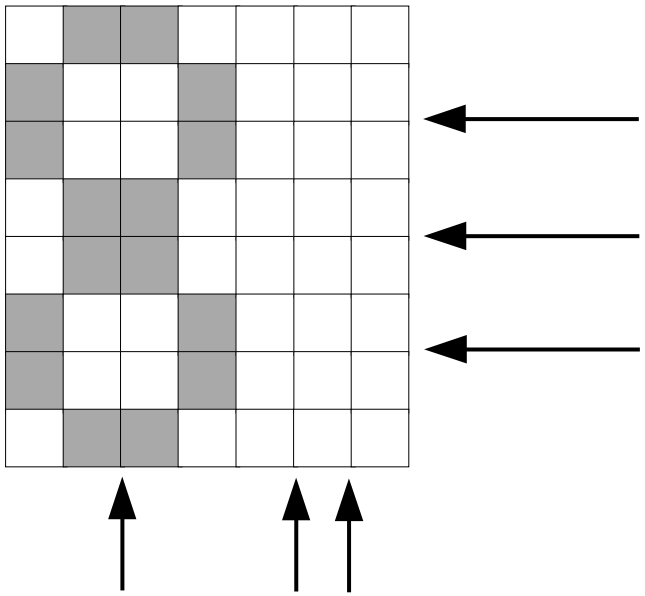
$$\sum_{i=1}^{N} i \approx (2^8 * 2^8) = 2^{16}$$

In total: $(x \,\&\, w) * (y \,\&\, h) = 2^{32}$

That's too slow!

Thought #2

The horizontal and vertical folds are independent!



You can make vertical folds here, regardless of when you make the horizontal folds.

You can make horizontal folds here, regardless of when you make the vertical folds.

- Count the horizontal folds
- Count the vertical folds
- Multiply the results

Back to this slide

How many potential *[x,y,w,h]* are there?

x & w:	*y & h:*
• *C* columns with width 1 • *(C-1)* columns with width 2 • *(C-2)* columns with width 3	• *R* rows with height 1 • *(R-1)* rows with height 2 • *(R-2)* rows with height 3
$\sum_{i=1}^{M} i \approx (2^8 * 2^8) = 2^{16}$	$\sum_{i=1}^{N} i \approx (2^8 * 2^8) = 2^{16}$

In total: $(x \,\&\, w) + (y \,\&\, h) = 2^{17}$

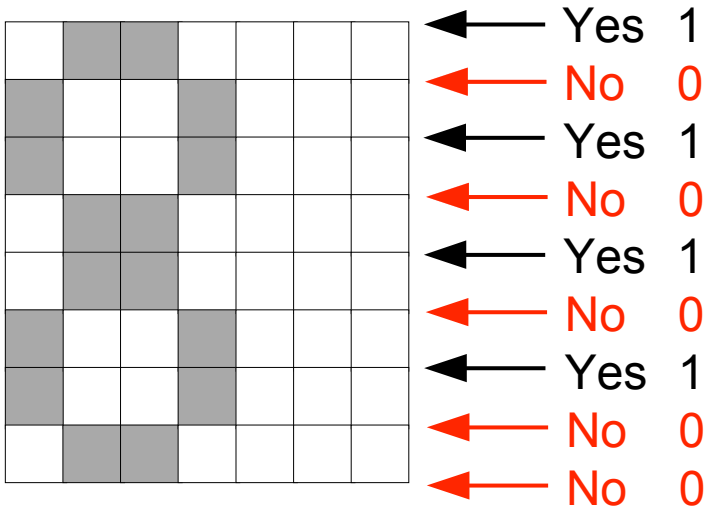
That's fast enough!

Details: A fundamental procedure

- Let's count the [y,h] combinations.

```
vector <int> starting_places(vector <string> &Grid);
```

- Return vector has *(R+1)* zero or one entries.
- `rv[i] = 1` iff row *i* can be a starting row
  - i.e. *i* can equal *y* in some labeling.



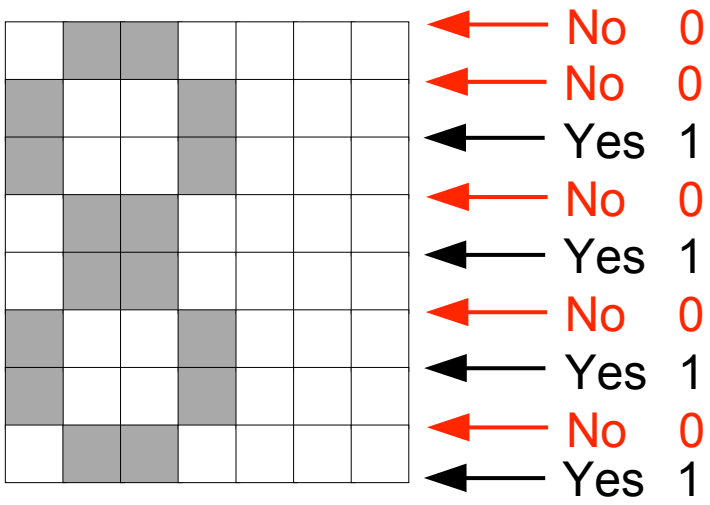

```
rv = { 1, 0, 1, 0, 1, 0, 1, 0, 0 }
```

Details: A fundamental procedure

- Let's count the [y,h] combinations.

```
vector <int> starting_places(vector <string> &Grid);
```

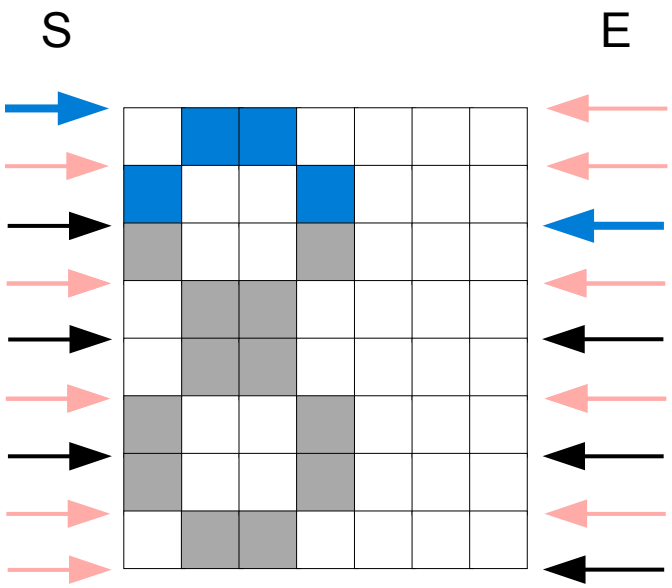
- Reverse the grid, call again and reverse the return value to get the ending places:
- `rv[i] = 1` iff $i = y+h$ in some labeling




```
rv = { 0, 0, 1, 0, 1, 0, 1, 0, 1 }
```

Details: A fundamental procedure

- Now, count all combinations of valid starting & ending rows

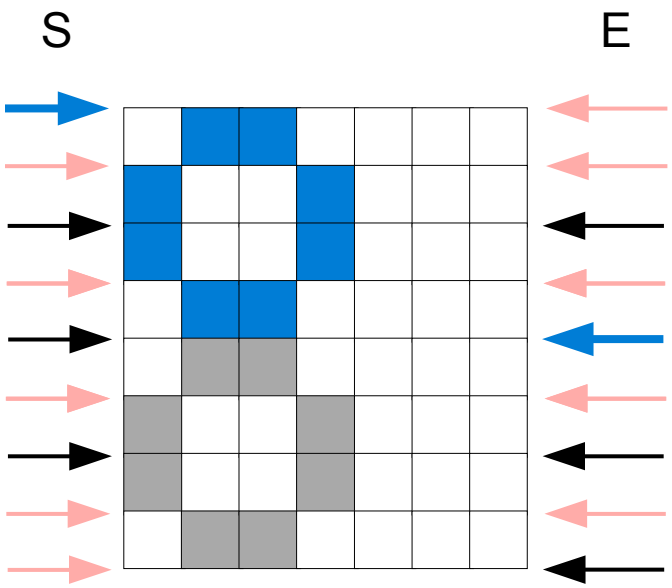


S: { 1, 0, 1, 0, 1, 0, 1, 0, 0 }

E: { 0, 0, 1, 0, 1, 0, 1, 0, 1}

Details: A fundamental procedure

- Now, count all combinations of valid starting & ending rows

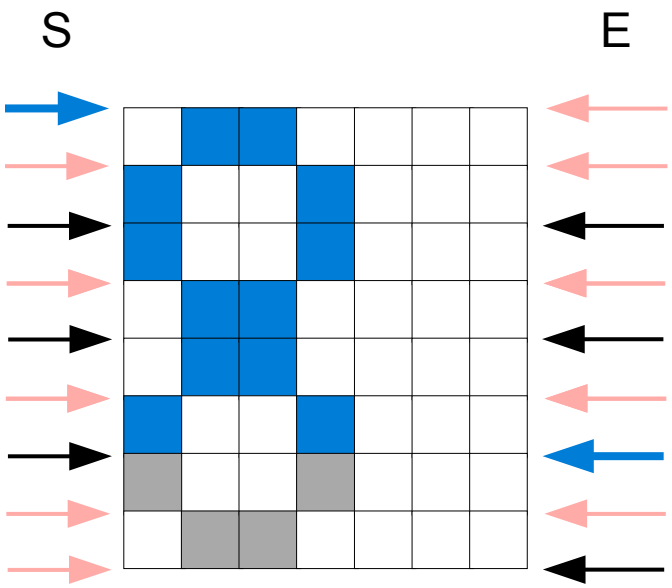


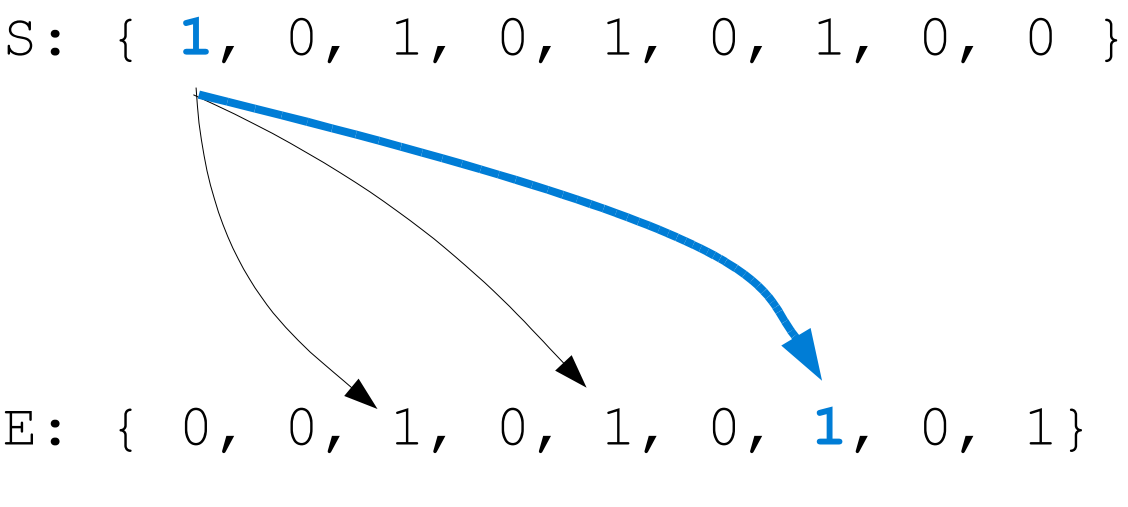
S: { **1**, 0, 1, 0, 1, 0, 1, 0, 0 }

E: { 0, 0, 1, 0, **1**, 0, 1, 0, 1}

Details: A fundamental procedure

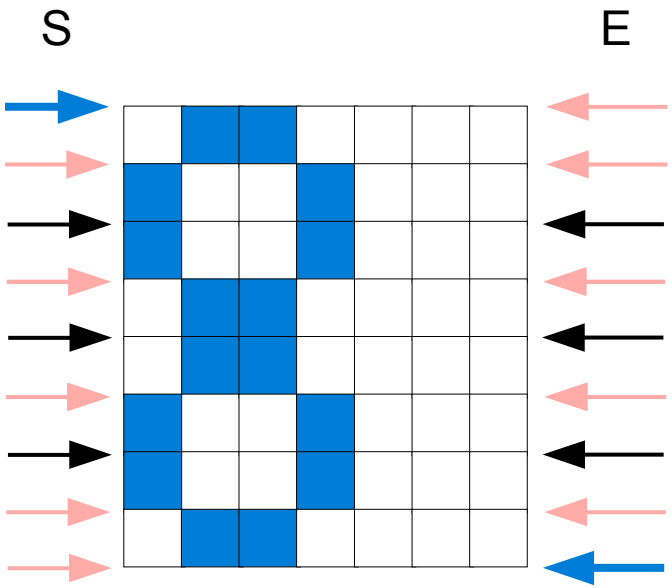
- Now, count all combinations of valid starting & ending rows

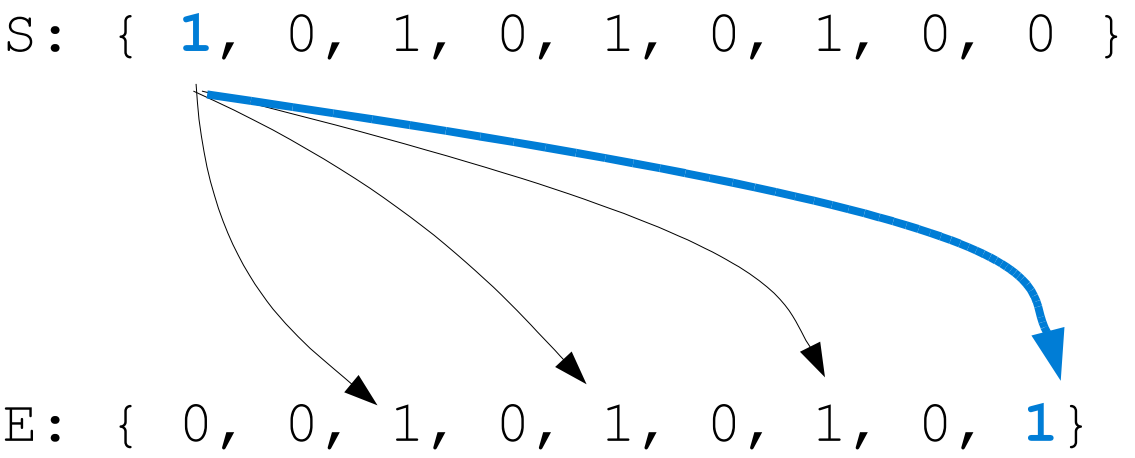




Details: A fundamental procedure

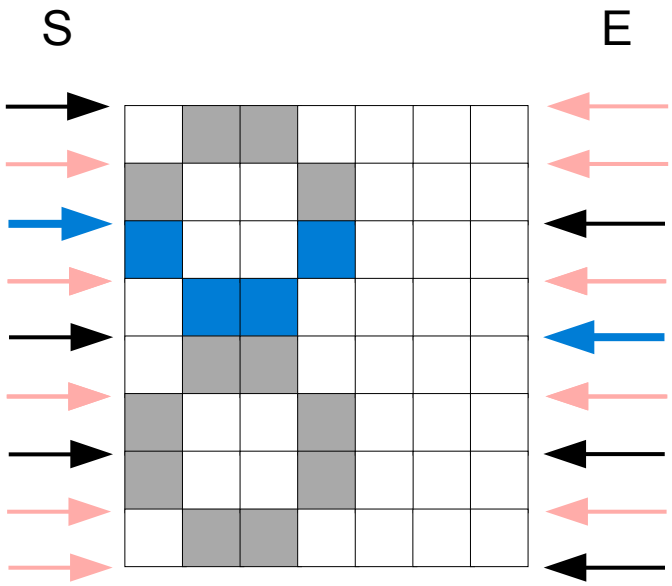
- Now, count all combinations of valid starting & ending rows

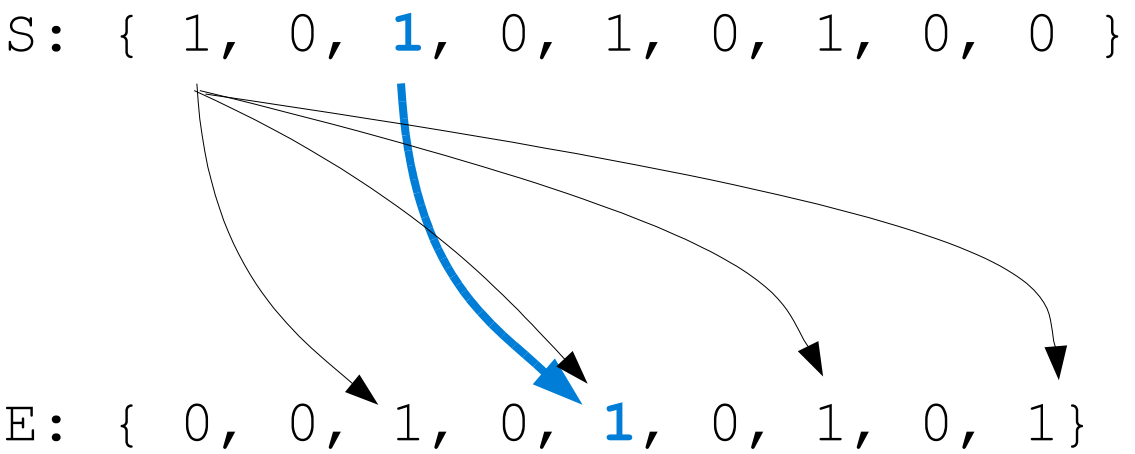




Details: A fundamental procedure

- Now, count all combinations of valid starting & ending rows

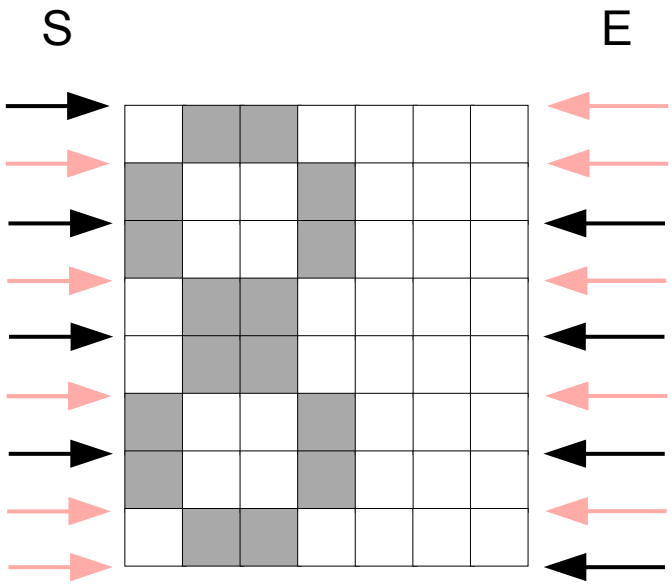


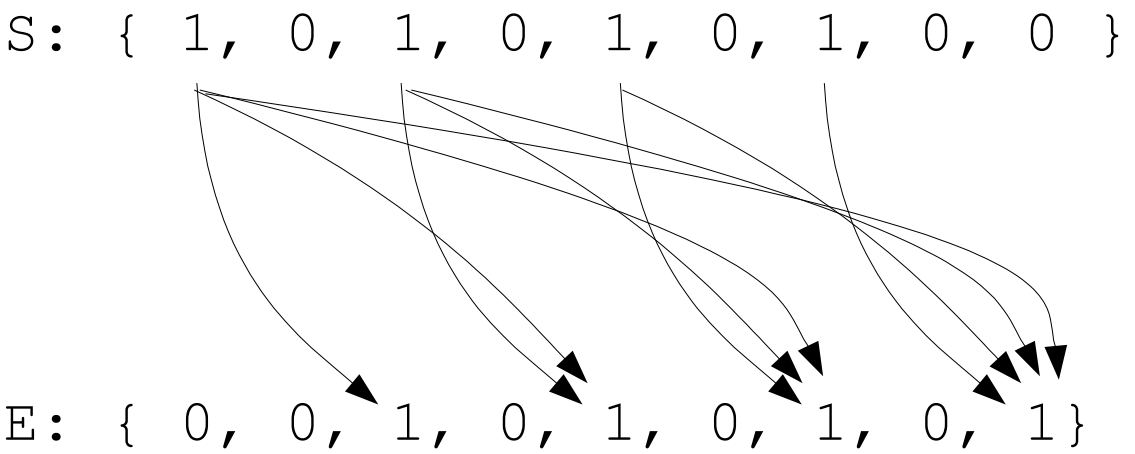


And So On.

Details: A fundamental procedure

- Now, count all combinations of valid starting & ending rows

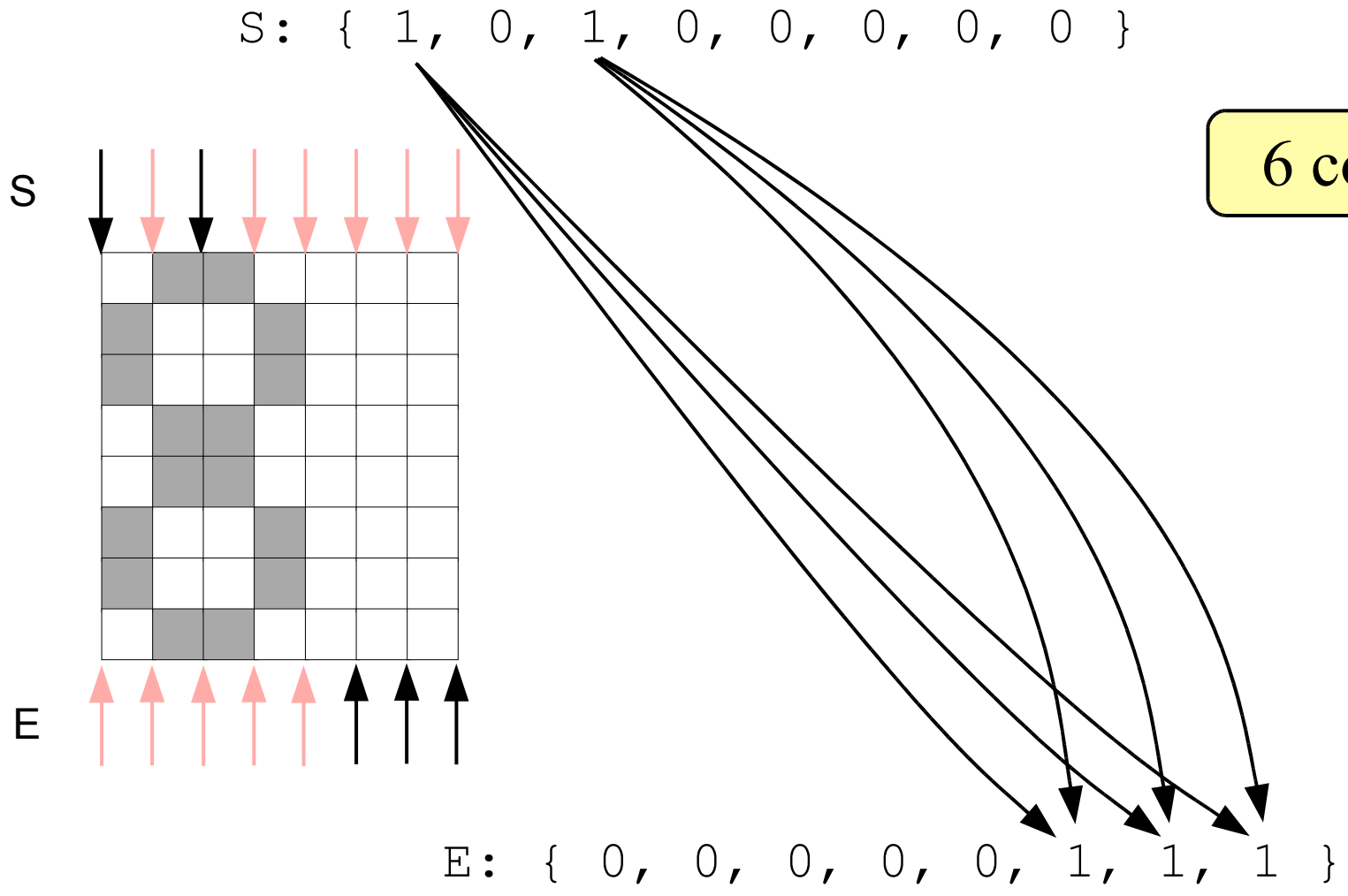




In this example, there are ten combinations.

Details: A fundamental procedure

- Do the same for starting and ending columns
- (You can transpose / reverse the grid for this)



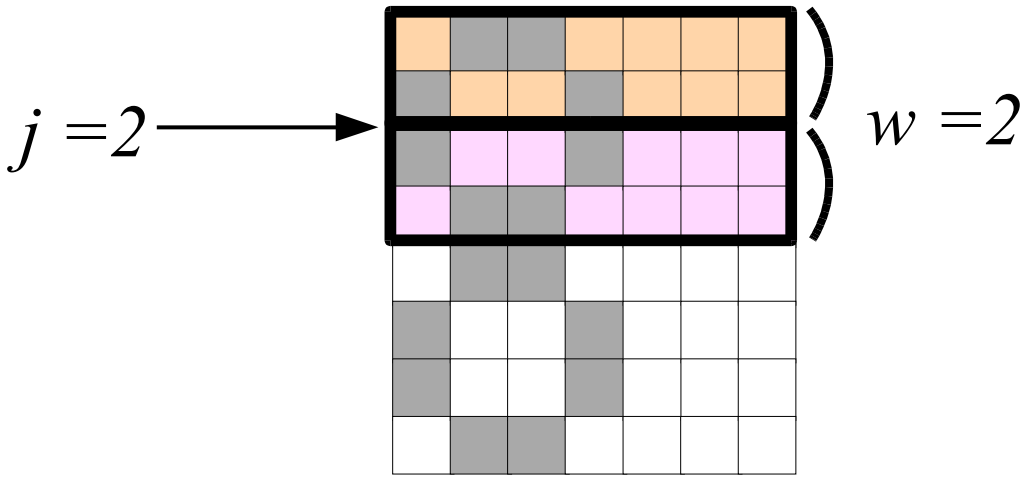
6 combinations

Multiply the answers and you're done!

(6*10 = 60)

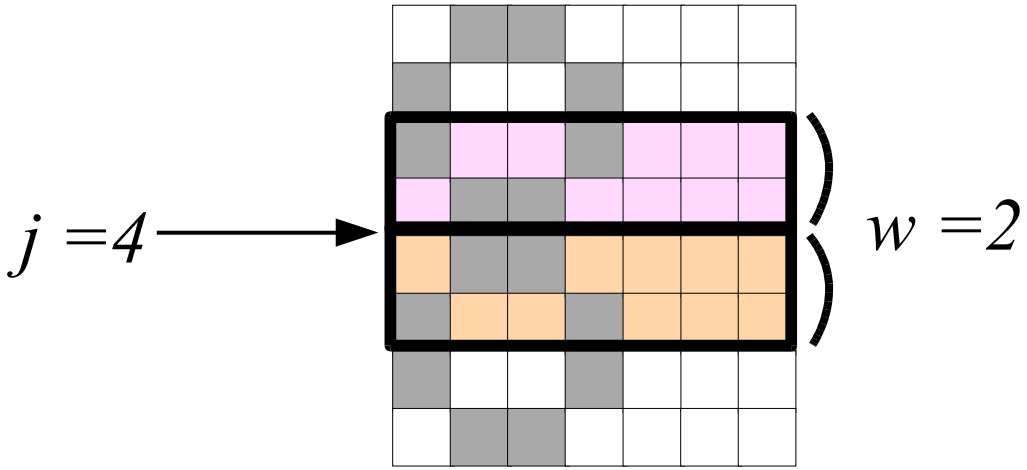
Implementing starting_places

- Given an index *j*, how do we determine that `rv[j] = 1`?
  - Start with `rv[0] = 1`.
  - There must be a rectangle of height *w* such that:
    - Row *(j-w)* is a starting row - `rv[j-w] = 1`
    - The rectangle of height *w* starting at *(j-w)* is the mirror image of the rectangle starting at *(j)*.



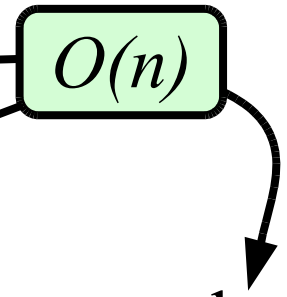
Implementing starting_places

- Given an index *j*, how do we determine that `rv[j] = 1`?
  - Start with `rv[0] = 1`.
  - There must be a rectangle of height *w* such that:
    - Row *(j-w)* is a starting row - `rv[j-w] = 1`
    - The rectangle of height *w* starting at *(j-w)* is the mirror image of the rectangle starting at *(j)*.

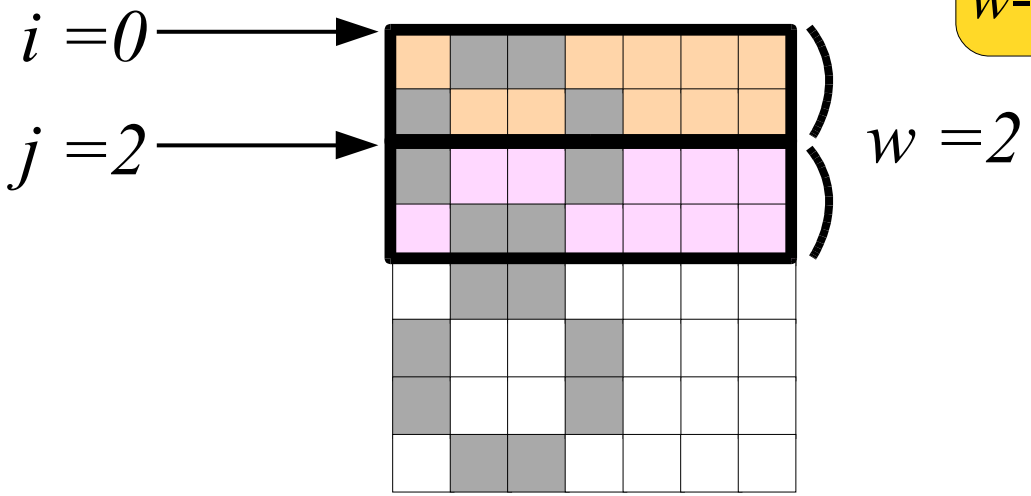


Implementing starting_places

- My initial realization of this was $O(n^3)$
  - Iterate i from 0 to n.
  - If i is a starting row, then for each $j > i$.
    - See if the rectangle from i to j matches the rectangle from j to $j+(j-i)$
    - If so, then j is a starting row.



Assumes that comparing w-row rectangles is O(w).



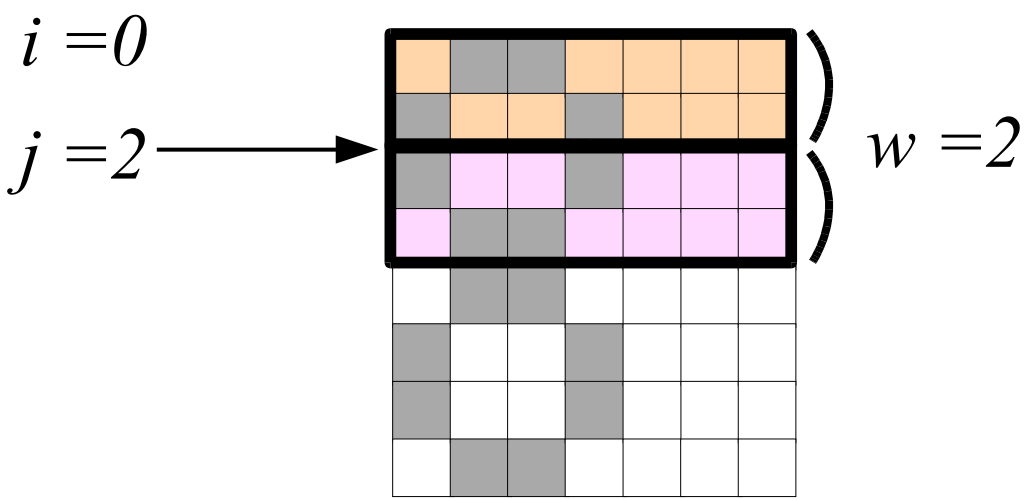
It was fast enough for Topcoder.

Implementing starting_places

- We can make this $O(n^2)$
  - Iterate j from 0 to n.
    - Calculate the maximum w for each j and store it.
  - Now repeat the previous algorithm.

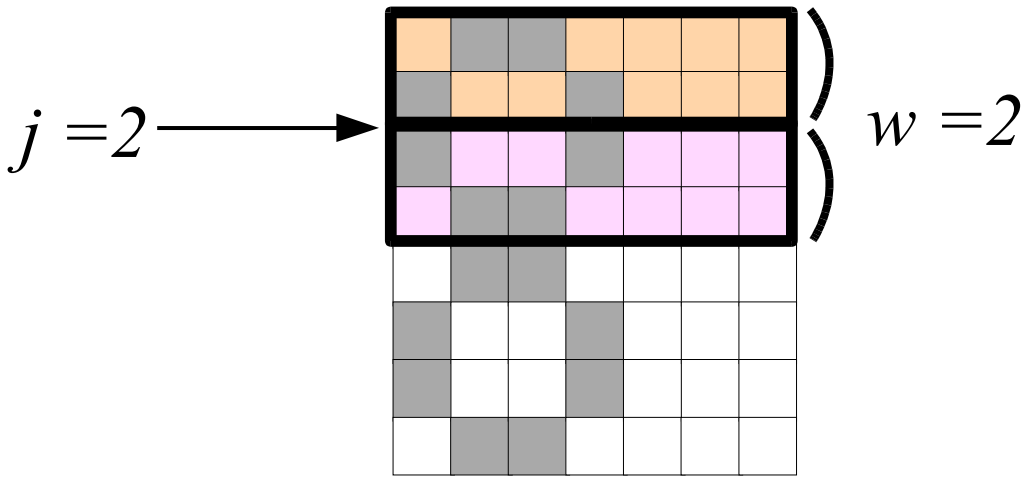
$O(n)$

$O(n^2)$



Implementing starting_places

- Another way to make this $O(n^2)$
  - Iterate j from 0 to n.
    - Iterate w from 1 to j
      - If the rectangles of height w above and below j match, and if $(j-w)$ is a starting row, then so is j.

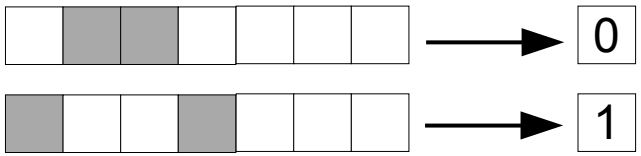


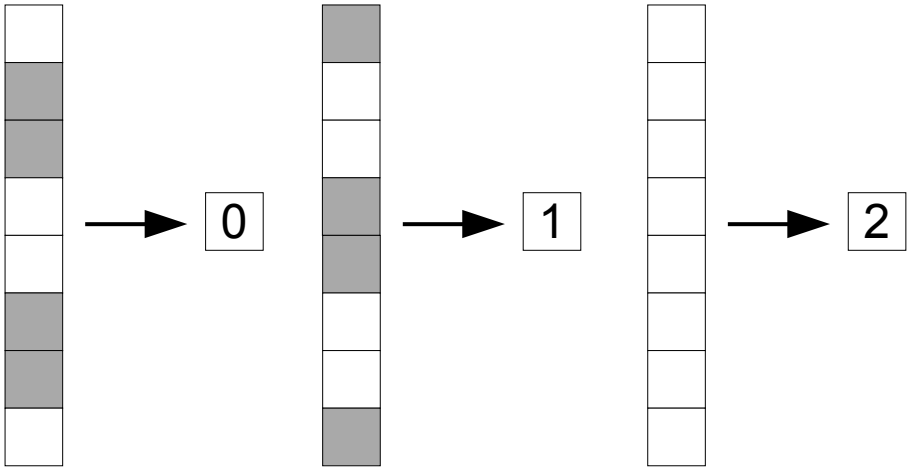
Running Time Summarized

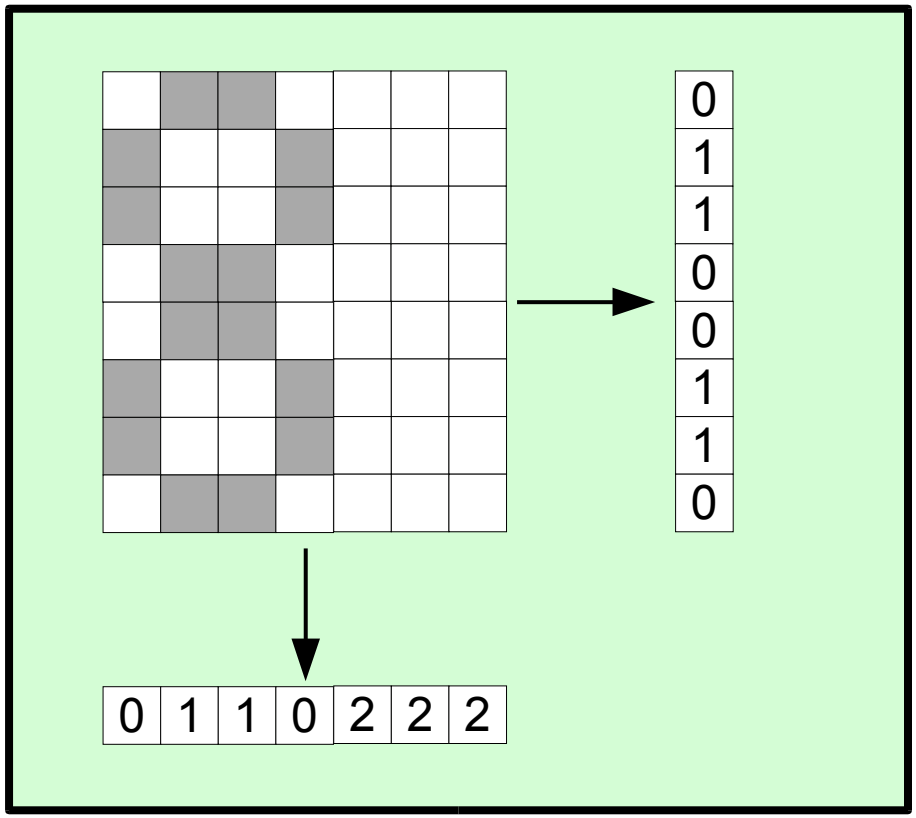
- Setting up the Grid for `starting_places()`: $O(n^2)$.
- Calling `starting_places()` four times: $O(n^2)$.
- Calculating the combinations: $O(n^2)$.
- That's $O(n^2)$ overall.

Can you make it faster?

- Yes – you can remove the string comparison by turning each string into an integer:

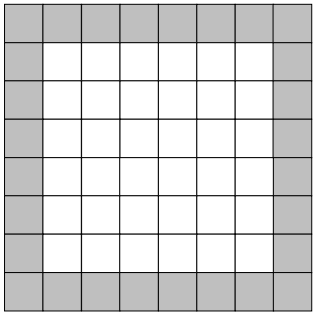


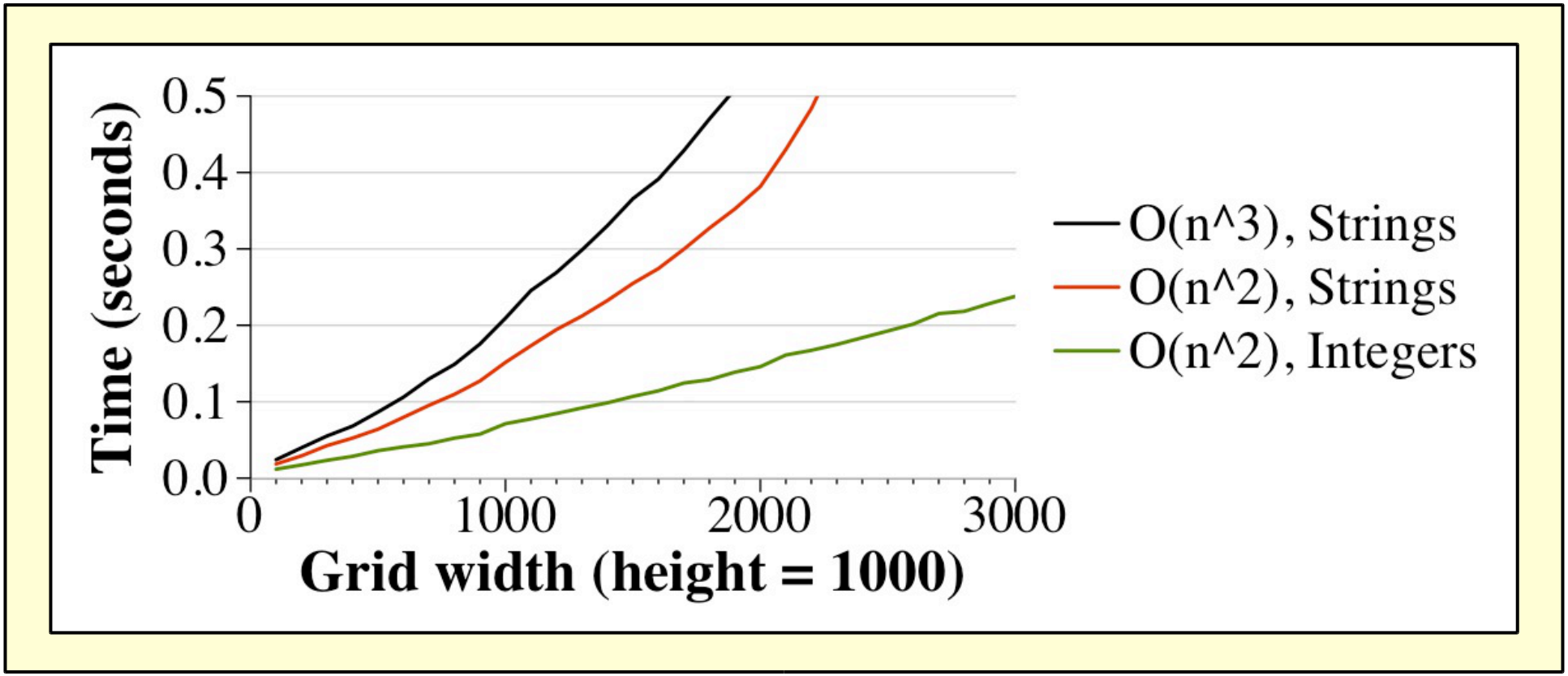




Experiment

- MacBook Pro 2.4 GHz
- Difficult Grid.





How did the Topcoders Do?

- By and large, those who submitted did well:
  - 476 (of 534) Topcoders opened the problem.
  - 130 (27%) submitted a solution.
  - 83 (64%) of the submissions were correct.
  - Best time was 12:23
  - Average correct time was 39:07.

Topcoder SRM 639, D1, 500-Pointer "BoardFolding"

James S. Plank
EECS Department
University of Tennessee

CS494/594 Class
January 27, 2026