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 (black).
- 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.

This grid may be folded in half in either way.

- Another way to visualize folding is to simply delete the part of the rectangle that goes on top.
Suppose we label the result of zero or more folds by:

- $w$: Width of the resulting grid.
- $h$: Height of the resulting grid.
- $r$: Row number of the top row in the original grid.
- $c$: Column number of the left column in the original grid.
- Label it $[w, h, r, c]$
The problem (continued)

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

Example 2

Answer = 9
Prototype and Constraints

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>int</td>
<td>Number of rows</td>
</tr>
<tr>
<td>$M$</td>
<td>int</td>
<td>Number of columns</td>
</tr>
<tr>
<td>$Grid$</td>
<td>vector &lt;string&gt;</td>
<td>The grid (in compressed format)</td>
</tr>
</tbody>
</table>

- **Return Value:** int
- **Constraints:** $N$ and $M$ are between 0 and 250.
  - (which is roughly $2^8$)
Thought Number 1 – Enumeration?

- How many potential \([w, h, r, c]\) are there?
  - \(M\) widths of 1, \(M-1\) widths of 2, \(M-2\) widths of 3....
  - \(N\) heights of 1, \(N-1\) heights of 2, \(N-2\) heights of 3...

\[
\sum_{i=1}^{M} i \times \sum_{i=1}^{N} i \approx (2^8*2^8) \times (2^8*2^8) = 2^{32}
\]

That's too slow!
• The horizontal and vertical folds are independent!

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

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

Count the potential horizontal folds, then the potential vertical folds, and multiply the results.
Thought Number 2

- Horizontal folds – how many \([w, c]\) combinations are there?
  - \(M\) widths of 1, \(M-1\) widths of 2, \(M-2\) widths of 3...

\[
\sum_{i=1}^{M} i \approx (2^8 \times 2^8) = 2^{16}
\]

- Vertical folds – same thing – now we're talking.
Details

- How do we count the \([w, c]\) combinations?
- How do we count the \([h, r]\) combinations?

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

- Return vector has \(N+1\) zero or one entries
  - \(rv[i] = 1\) iff row \(i\) can be a starting row.

\[rv = \{ 1, 0, 1, 0, 1, 0, 1, 0 \}\]
Details

You can use `starting_places()` to identify the potential ending rows too (reverse the Grid and then reverse the return value):

\[ rv = \{ 0, 0, 1, 0, 1, 0, 1, 0, 1 \} \]
vector <int> starting_places(vector <string> &Grid);

- Now, count all combinations of starting and ending rows that have positive height:

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

In this example, there are ten combinations.
- Let's do the same for starting and ending columns:

\[
\begin{align*}
S &: \{ 1, 0, 1, 0, 0, 0, 0 \} \\
E &: \{ 0, 0, 0, 0, 0, 1, 1, 1 \}
\end{align*}
\]
Details

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

- You can use the same `starting_places()` procedure for rows, columns, starting and ending places (just transpose and/or reverse the grid).

- Finally multiply the number of [w, c] combinations by the number of [h, r] combinations.
vector <int> starting_places(vector <string> &Grid);

- Given an index $j$, how can we determine that $j$ is a starting row index?
  
  - There must be a rectangle of height $w$ such that:
  - Row $j-w$ is a starting row index.
  - The rectangle of height $w$ above $j$ is the mirror image of the rectangle of height $w$ below $j.$
vector <int> starting_places(vector <string> &Grid);

- 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 as well.

It was fast enough for Topcoder, though.
One way to make it \( O(n^2) \):

- Iterate \( j \) from 0 to \( n \).
- Calculate the maximum \( w \) for each \( j \) and store it.

Now repeat the previous algorithm.
vector <int> starting_places(vector <string> &Grid);

- Another way to make it $O(n^2)$:
- Iterate $j$ from 0 to $n$.
- Iterate $w$ from 1 to $j$.
- If the rectangles of width $w$ above and below $j$ match, and $j-w$ is a starting row, then $j$ is a starting row.
Running Time Summary

- You call `starting_places()` four times.
  - $O(n^2)$.

- Setting up the Grid for `starting_places()`:
  - $O(n^2)$.

- Calculating the combinations:
  - $O(n^2)$.

- That's $O(n^2)$ overall.

- So, with $n = 256$, this should be fast enough, and indeed it is.
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?

• A pretty tough one:
  - 476 (of 534) Topcoders opened the problem.
  - 130 (27%) submitted a solution.
  - 83 (64%) of the submissions were correct.
  - That's 17.2% of those who opened the problem.
  - Best time was 12:23
  - Average correct time was 39:07.