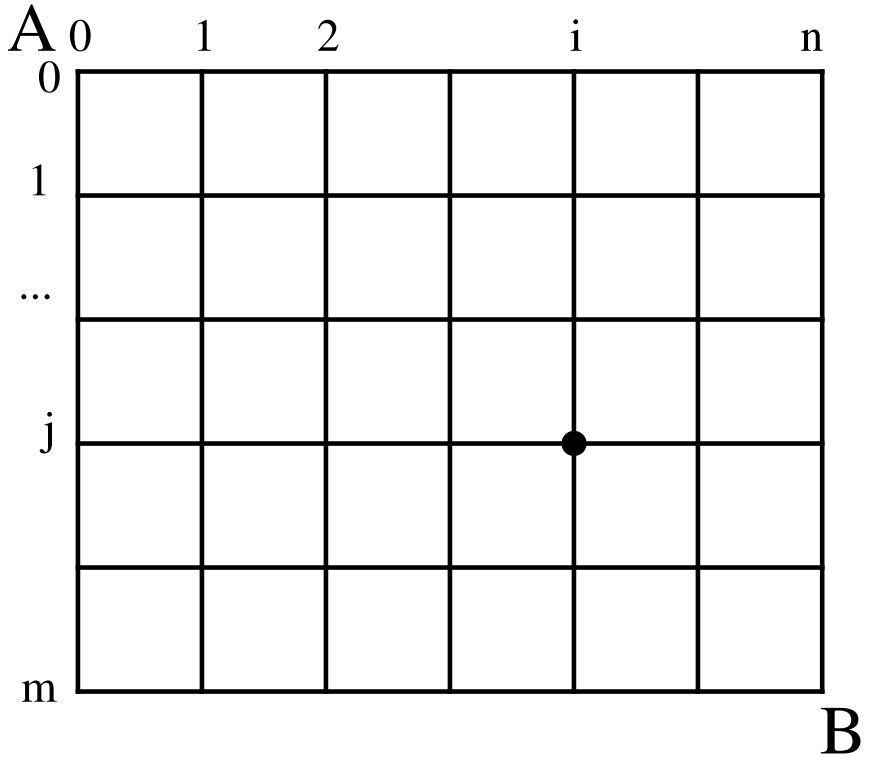
Online: `http://www.math.mcgill.ca/~bryant`

How many direct paths from A to B?

We consider only paths that go left to right and top to bottom.



Define $f(i,j)$ to be number of paths from A to (i,j). A path from A to (i,j) either takes the edge to the left of (i,j) or takes the edge above (i,j), but not both.

Recursive solution

$$
\begin{aligned}
f(0,0) &= 1 \\
f(i,0) &= f(i-1,0) \text{ for all } i > 0 \\
f(0,j) &= f(0,j-1) \text{ for all } j > 0 \\
f(i,j) &= f(i-1,j) + f(i,j-1) \text{ for all } i,j > 0
\end{aligned}
$$

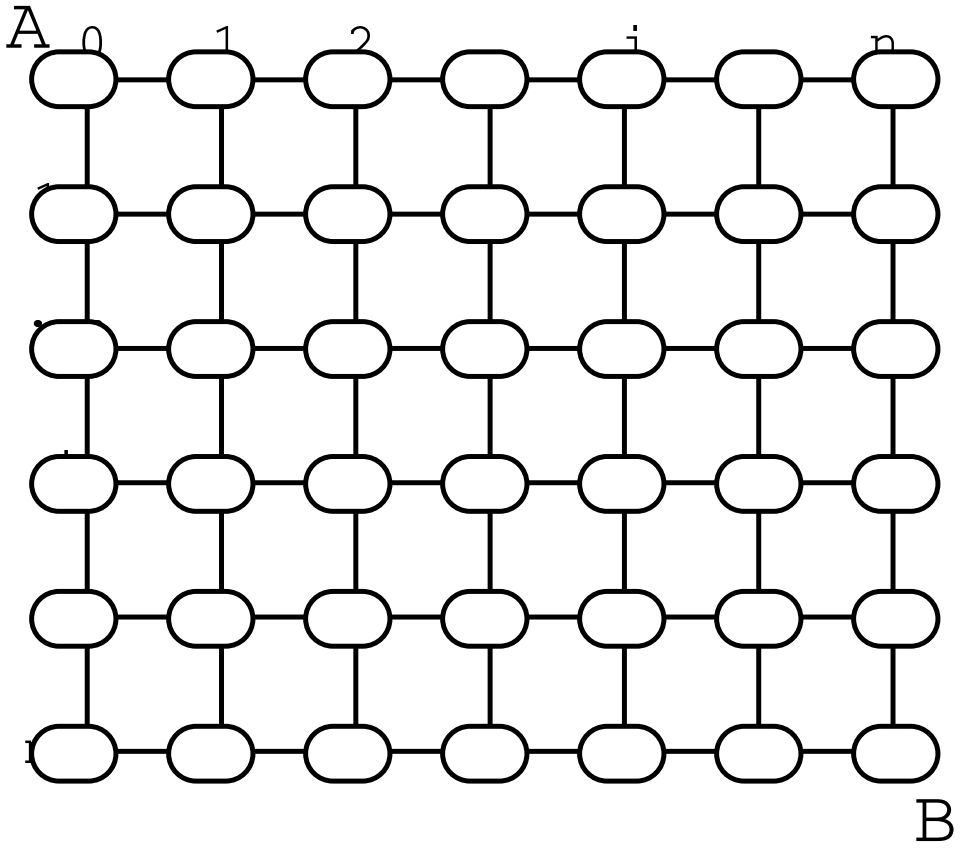
We can turn this directly into a recursive algorithm.

$NumPaths(i,j)$
1. **if** $i = 0$ and $j = 0$
2. **then** return 1
3. **else if** $i > 0$ and $j = 0$
4. return NumPaths($i-1$,0)
5. **else if** $i = 0$ and $j > 0$
6. return NumPaths(0,$j-1$)
7. **else**
8. return NumPaths($i-1$,j) + NumPaths(i,$j-1$)

This works, but it takes exponential time!

The problem with recursion

How many times does $NumPaths(i, j)$ get called?



The number of times called follows Pascal's Triangle. So, for example, $NumPaths(1, 1)$ gets called $\binom{n+m}{n}$ times! If $n = m = 100$ then this is around 10^{59}.

Fix: dynamic programming

The trick is to *remember* when we have already computed something, store the values, and look-up the values when we need it again.

Create an $m \times n$ table F to store values.

First *initialise* table to some dummy value (indicating that the value has not yet been calculated)

Modify $NumPaths(i, j)$ to check whether we have already evaluated $f(i, j)$ by looking at entry $F[i, j]$. If we have, we return the value already computed. If not, we compute it.

End result: total time taken reduces to $O(nm)$.

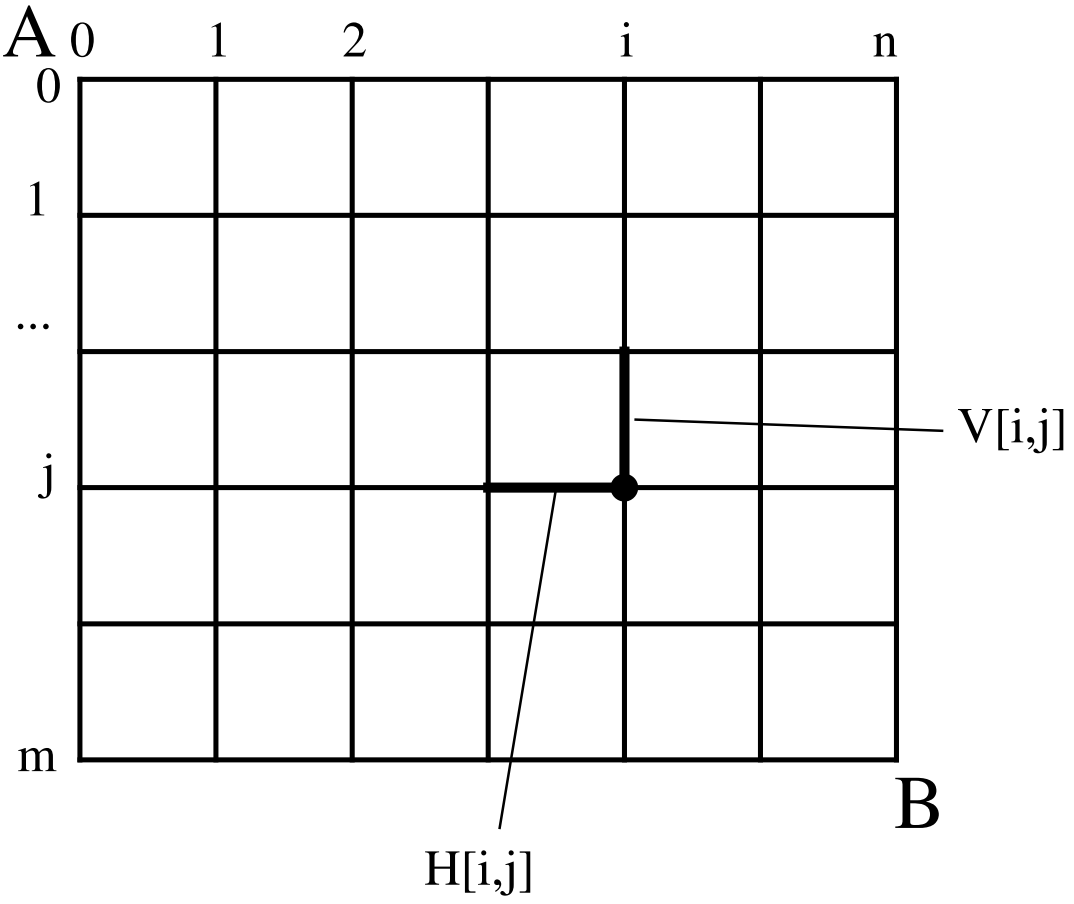
1. Construct an $m \times n$ table F [global variable]
2. Initialise each entry of F to -1.
3. return $FastNumPaths(i, j)$

$FastNumPaths(i, j)$
1. **if** $F[i, j] \geq 0$ **then**
2. **return** $F[i, j]$
3. **else**
4. **if** $i = 0$ and $j = 0$ **then**
5. $F[i, j] \leftarrow 1$
6. **else if** $i > 0$ and $j = 0$ **then**
7. $F[i, j] \leftarrow FastNumPaths(i - 1, 0)$
8. **else if** $i = 0$ and $j > 0$ **then**
9. $F[i, j] \leftarrow FastNumPaths(0, j - 1)$
10. **else**
11. $F[i, j] \leftarrow FastNumPaths(i - 1, j) + FastNumPaths(i, j - 1)$
12. **return** $F[i, j]$

For each i, j, $FastNumPaths(i, j)$ called at most three times, though goes through lines 3-12 at most once.

Collecting flags

Suppose that along each edge of the grid there are a different number of flags. Goal is to find a path from A to B that goes left to right and top to bottom, and collects as many flags as possible (the *score* equals the number of flags).



Suppose that the horizontal edge entering (i, j) has $H[i, j]$ flags and the vertical edge $V[i, j]$ flags.

Finding a recursion

Suppose that a path P has the highest score of any path from A to (i,j), and that P enters (i,j) along the horizontal edge. Let P' denote the part of P going from A to $(i-1,j)$. Then

- The score of P equals the score of P' plus $H[i,j]$.
- P' will have the highest score of any path going from A to $(i-1,j)$.

Carrying on the same idea we derive a recursion. Let $g(i,j)$ be the highest score of a path from A to (i,j).

$$\begin{aligned} g(0,0) &= 0 \\ g(i,0) &= g(i-1,0) + H[i,0] \text{ for all } i > 0 \\ g(0,j) &= g(0,j-1) + V[0,j] \text{ for all } j > 0 \end{aligned}$$

When $i,j > 0$, $g(i,j)$ is the **maximum** of

$$g(i-1,j) + H[i,j]$$

and

$$g(i,j-1) + V[i,j].$$

1. Construct an $m \times n$ table G [global variable[
2. Initialise each entry of G to -1.
3. return $GoodPaths(i, j)$

$GoodPaths(i, j)$
1. **if** $G[i, j] \geq 0$ **then**
2. **return** $G[i, j]$
3. **else**
4. **if** $i = 0$ and $j = 0$ **then**
5. $G[i, j] \leftarrow 0$
6. **else if** $i > 0$ and $j = 0$ **then**
7. $G[i, j] \leftarrow GoodPaths(i - 1, 0) + H[i, 0]$
8. **else if** $i = 0$ and $j > 0$ **then**
9. $G[i, j] \leftarrow GoodPaths(0, j - 1) + V[i, 0]$
10. **else**
11. $G[i, j] \leftarrow \max\{GoodPaths(i - 1, j) + H[i, j],$
$GoodPaths(i, j - 1) + V[i, j]\}$
12. **return** $G[i, j]$