Grid Paths

Madhavan Mukund

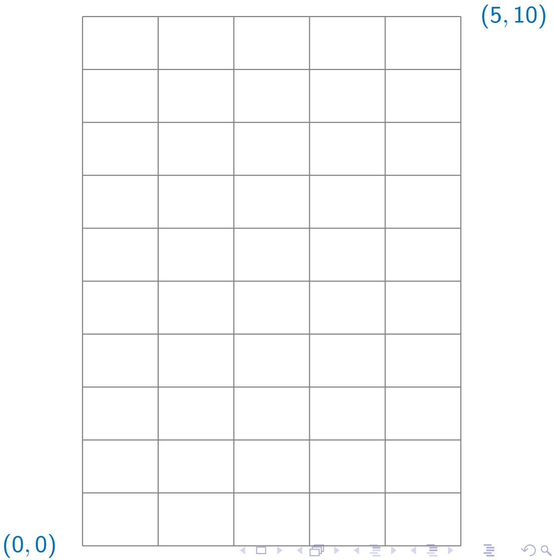
https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 9

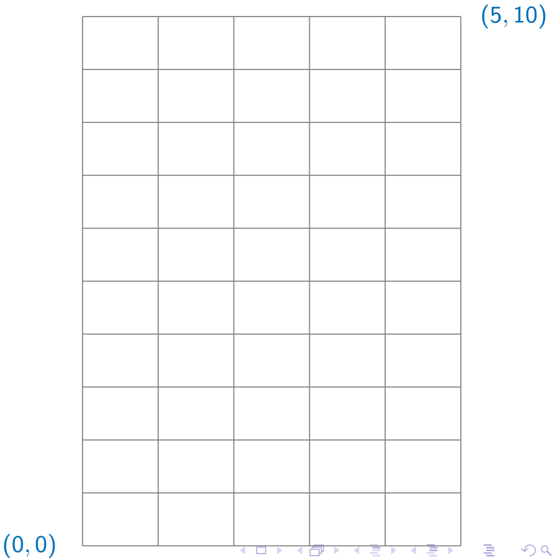
Grid paths

- Rectangular grid of one-way roads



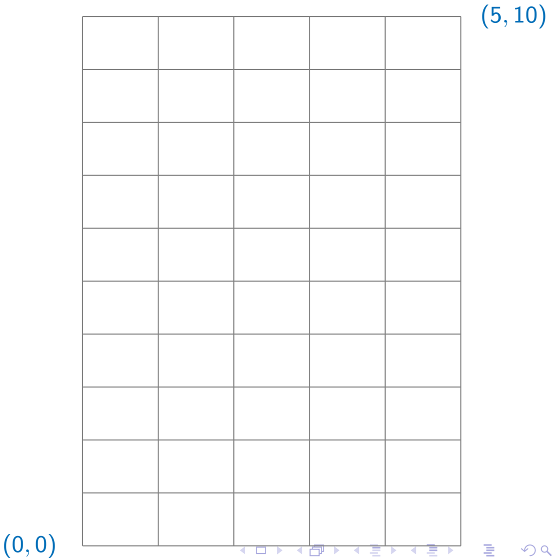
Grid paths

- Rectangular grid of one-way roads
- Can only go up and right



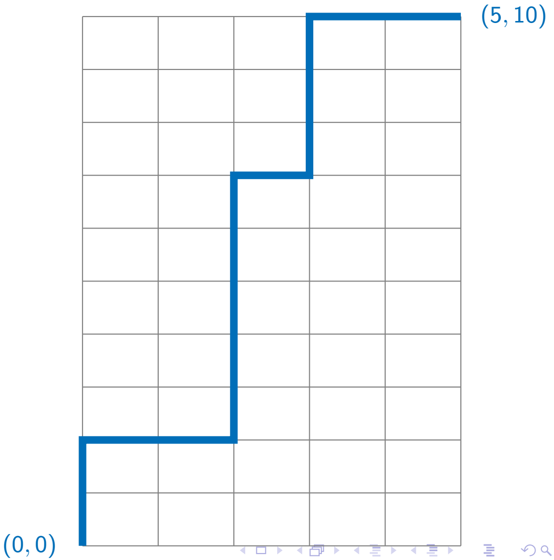
Grid paths

- Rectangular grid of one-way roads
- Can only go up and right
- How many paths from $(0,0)$ to (m,n)?



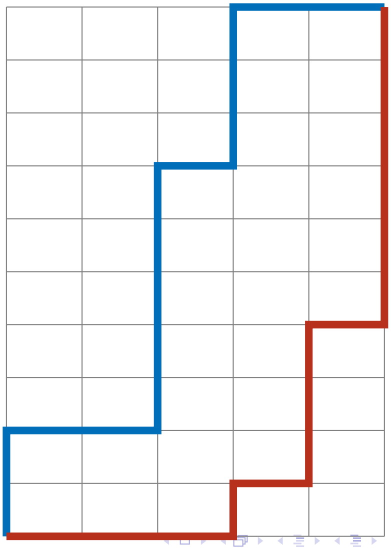
Grid paths

- Rectangular grid of one-way roads
- Can only go up and right
- How many paths from $(0, 0)$ to (m, n)?



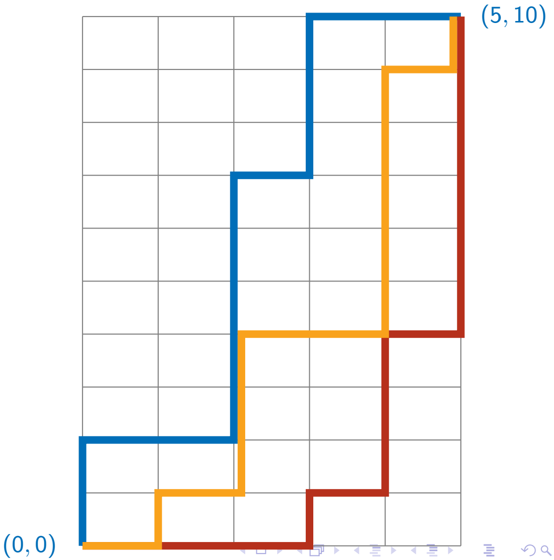
Grid paths

- Rectangular grid of one-way roads
- Can only go up and right
- How many paths from $(0,0)$ to (m,n)?



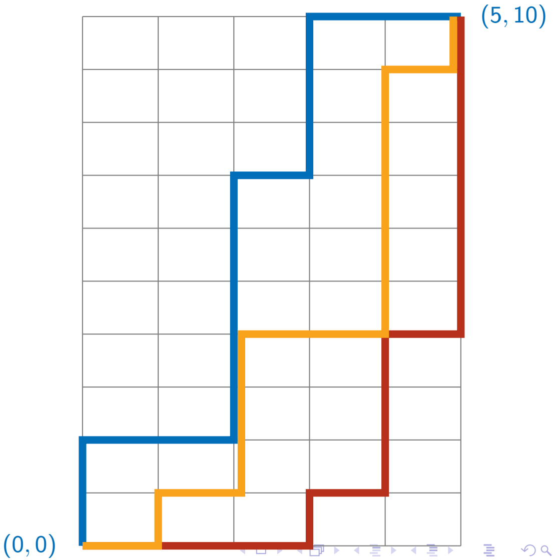
Grid paths

- Rectangular grid of one-way roads
- Can only go up and right
- How many paths from $(0,0)$ to (m,n)?



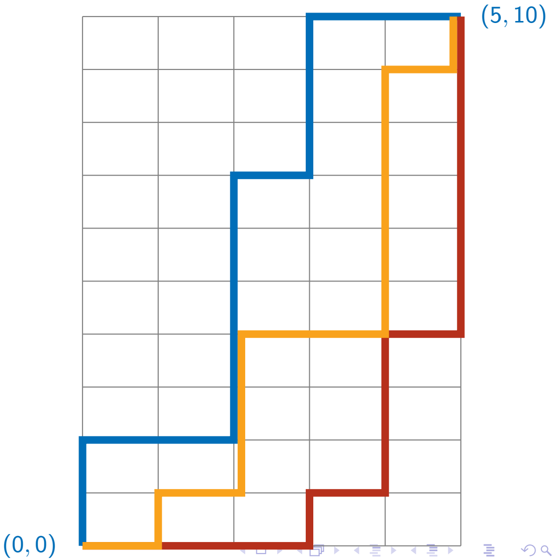
Combinatorial solution

- Every path from $(0,0)$ to $(5,10)$ has 15 segments
  - In general $m+n$ segments from $(0,0)$ to (m,n)



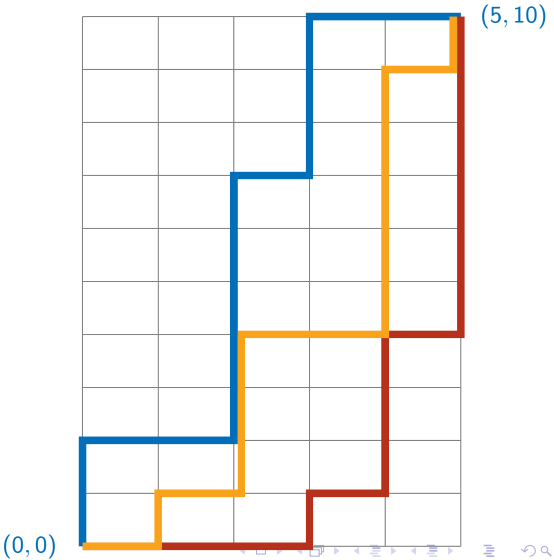
Combinatorial solution

- Every path from $(0,0)$ to $(5,10)$ has 15 segments
  - In general $m+n$ segments from $(0,0)$ to (m,n)
- Out of 15, exactly 5 are right moves, 10 are up moves



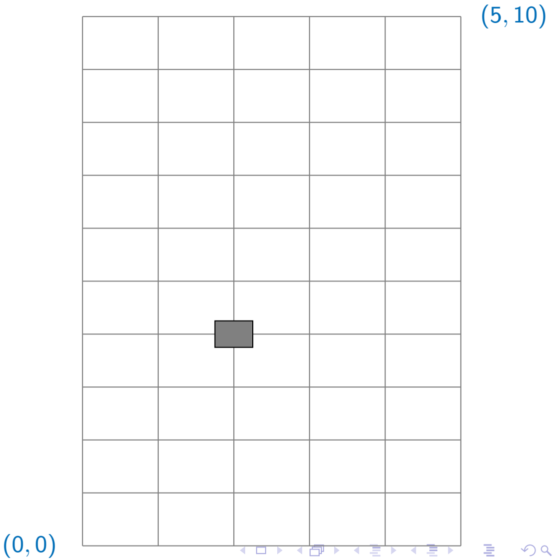
Combinatorial solution

- Every path from $(0,0)$ to $(5,10)$ has 15 segments
  - In general $m+n$ segments from $(0,0)$ to (m,n)
- Out of 15, exactly 5 are right moves, 10 are up moves
- Fix the positions of the 5 right moves among the 15 positions overall
  - $\binom{15}{5} = \frac{15!}{10! \cdot 5!} = 3003$
  - Same as $\binom{15}{10}$ — fix the 10 up moves



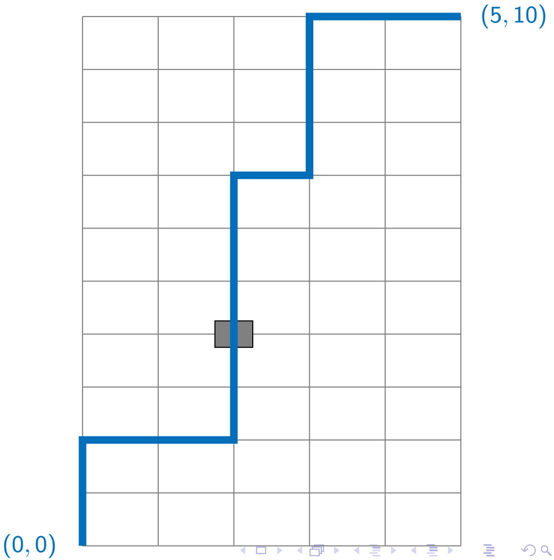
Holes

- What if an intersection is blocked?
  - For instance, $(2, 4)$



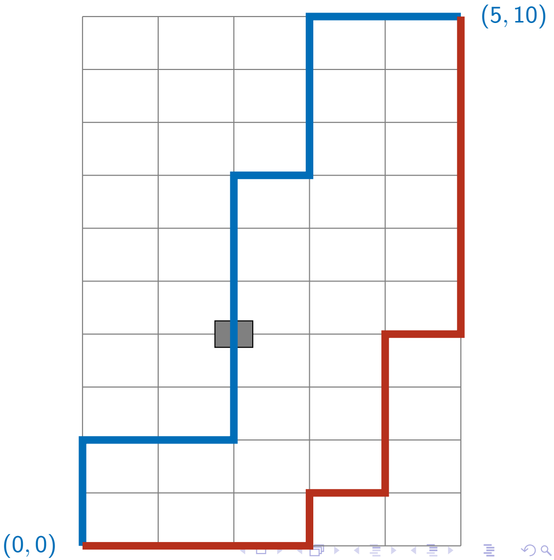
Holes

- What if an intersection is blocked?
  - For instance, $(2,4)$
- Need to discard paths passing through $(2,4)$
  - Two of our earlier examples are invalid paths



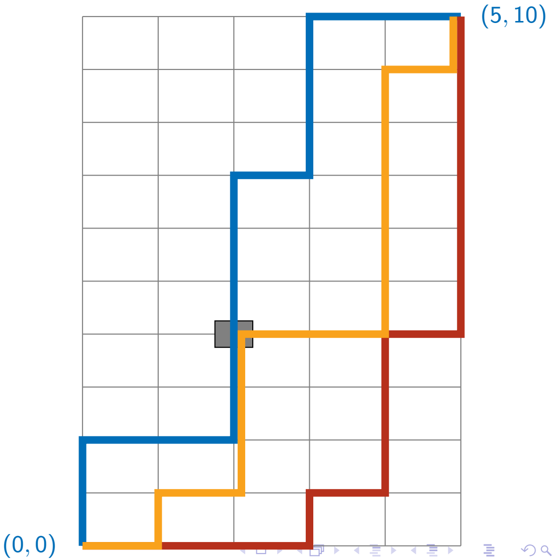
Holes

- What if an intersection is blocked?
  - For instance, $(2, 4)$
- Need to discard paths passing through $(2, 4)$
  - Two of our earlier examples are invalid paths



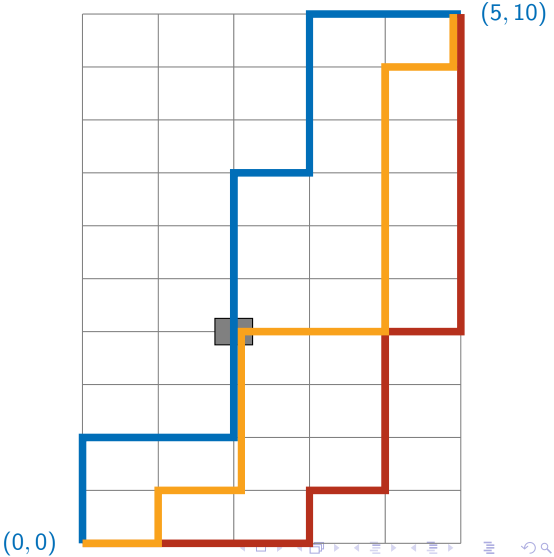
Holes

- What if an intersection is blocked?
  - For instance, $(2,4)$
- Need to discard paths passing through $(2,4)$
  - Two of our earlier examples are invalid paths



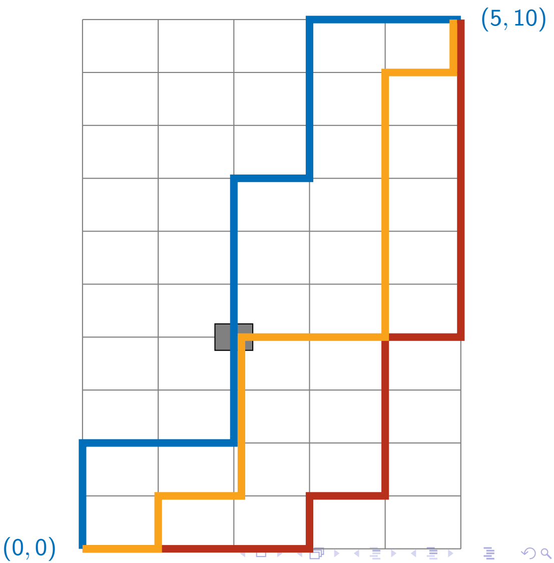
Combinatorial solution for holes

- Discard paths passing through $(2,4)$



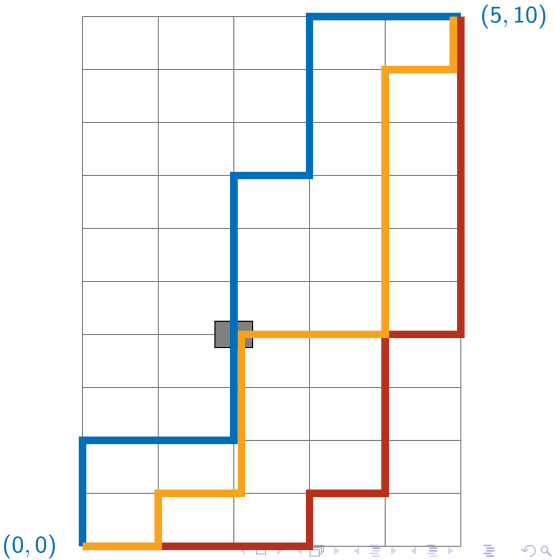
Combinatorial solution for holes

- Discard paths passing through $(2,4)$
- Every path via $(2,4)$ combines a path from $(0,0)$ to $(2,4)$ with a path from $(2,4)$ to $(5,10)$
  - Count these separately
  - $\binom{2+4}{2} = 15$ paths $(0,0)$ to $(2,4)$
  - $\binom{3+6}{3} = 84$ paths $(2,4)$ to $(5,10)$



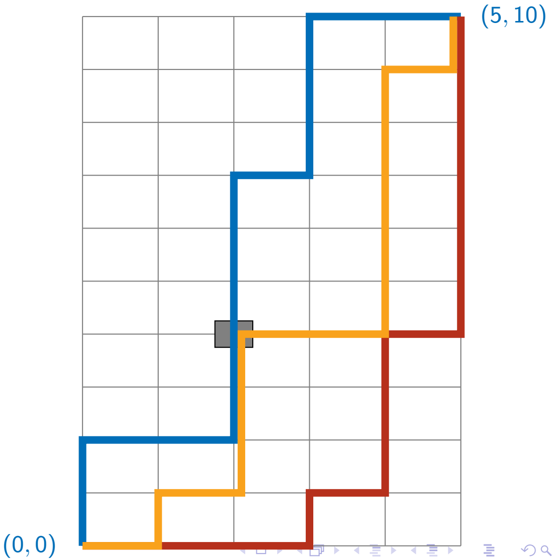
Combinatorial solution for holes

- Discard paths passing through $(2,4)$
- Every path via $(2,4)$ combines a path from $(0,0)$ to $(2,4)$ with a path from $(2,4)$ to $(5,10)$
  - Count these separately
  - $\binom{2+4}{2} = 15$ paths $(0,0)$ to $(2,4)$
  - $\binom{3+6}{3} = 84$ paths $(2,4)$ to $(5,10)$
- $15 \times 84 = 1260$ paths via $(2,4)$



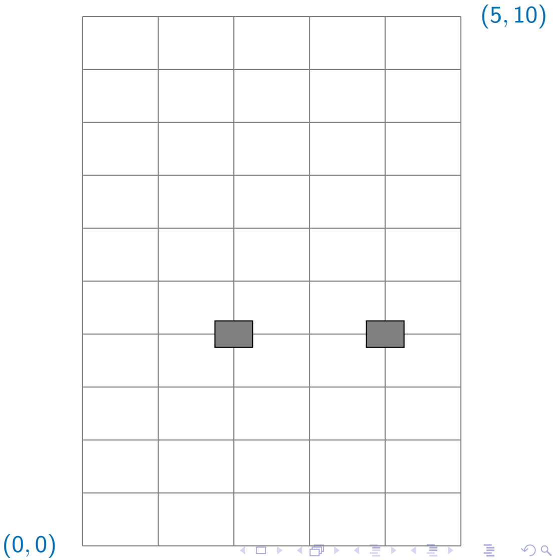
Combinatorial solution for holes

- Discard paths passing through $(2, 4)$
- Every path via $(2, 4)$ combines a path from $(0, 0)$ to $(2, 4)$ with a path from $(2, 4)$ to $(5, 10)$
  - Count these separately
  - $\binom{2+4}{2} = 15$ paths $(0, 0)$ to $(2, 4)$
  - $\binom{3+6}{3} = 84$ paths $(2, 4)$ to $(5, 10)$
- $15 \times 84 = 1260$ paths via $(2, 4)$
- $3003 - 1260 = 1743$ valid paths avoiding $(2, 4)$



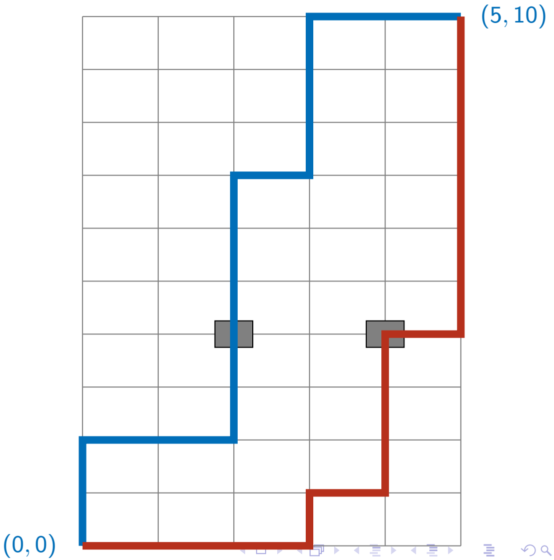
More holes

- What if two intersections are blocked?



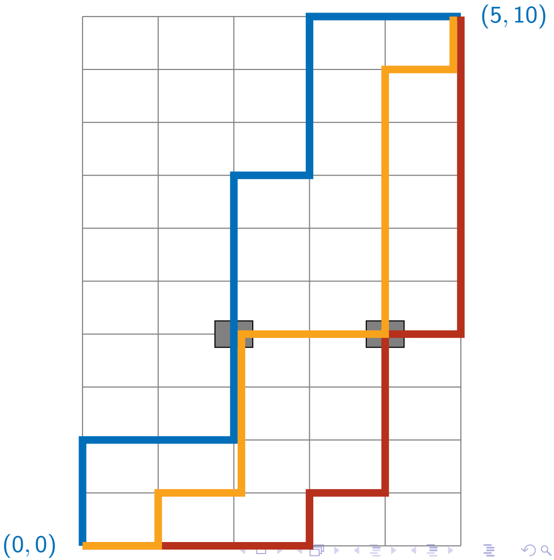
More holes

- What if two intersections are blocked?
- Discard paths via $(2,4)$, $(4,4)$



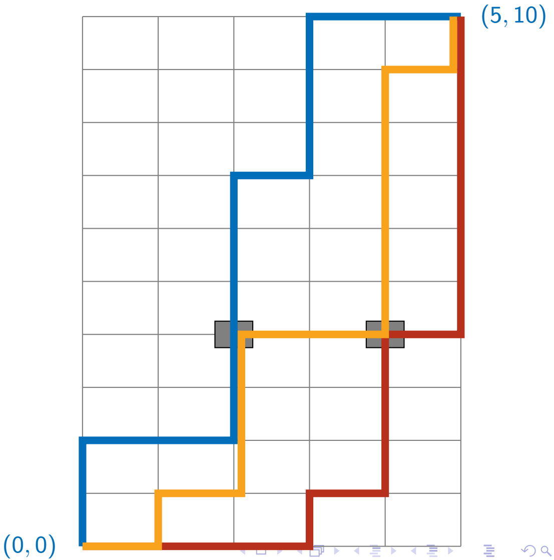
More holes

- What if two intersections are blocked?
- Discard paths via $(2, 4)$, $(4, 4)$
  - Some paths are counted twice



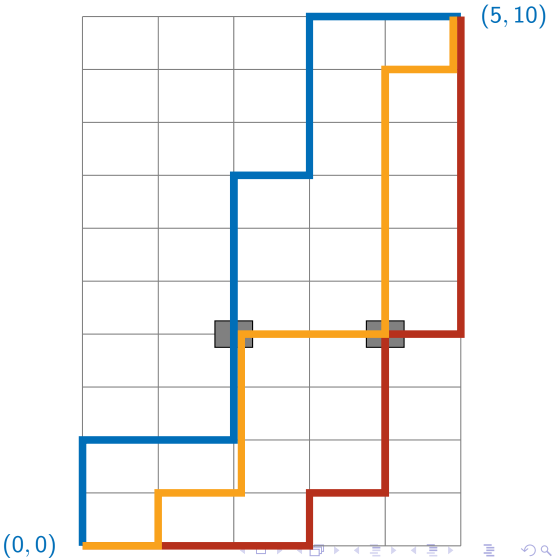
More holes

- What if two intersections are blocked?
- Discard paths via $(2,4)$, $(4,4)$
  - Some paths are counted twice
- Add back the paths that pass through both holes



More holes

- What if two intersections are blocked?
- Discard paths via $(2,4)$, $(4,4)$
  - Some paths are counted twice
- Add back the paths that pass through both holes
- Inclusion-exclusion — counting is messy



Inductive formulation

- How can a path reach (i, j)

Inductive formulation

- How can a path reach (i, j)
  - Move up from $(i, j-1)$

Inductive formulation

- How can a path reach (i, j)
  - Move up from $(i, j-1)$
  - Move right from $(i-1, j)$

$$\begin{tikzcd}
\boldsymbol{(i{-}1, j)} \arrow[r] & \boldsymbol{(i, j)} \\
 & \boldsymbol{(i, j{-}1)} \arrow[u]
\end{tikzcd}$$

Inductive formulation

- How can a path reach (i, j)
  - Move up from $(i, j-1)$
  - Move right from $(i-1, j)$
- Each path to these neighbours extends to a unique path to (i, j)

$$
\begin{tikzcd}
(\boldsymbol{i-1,j}) \arrow[r] & (\boldsymbol{i,j}) \\
 & (\boldsymbol{i,j-1}) \arrow[u]
\end{tikzcd}
$$

Inductive formulation

- How can a path reach (i,j)
  - Move up from $(i,j-1)$
  - Move right from $(i-1,j)$
- Each path to these neighbours extends to a unique path to (i,j)
- Recurrence for $P(i,j)$, number of paths from $(0,0)$ to (i,j)
  - $P(i,j) = P(i-1,j) + P(i,j-1)$

$$\begin{tikzcd}
\boldsymbol{(i{-}1,j)} \arrow[r] & \boldsymbol{(i,j)} \\
 & \boldsymbol{(i,j{-}1)} \arrow[u]
\end{tikzcd}$$

Inductive formulation

- How can a path reach (i, j)
  - Move up from $(i, j-1)$
  - Move right from $(i-1, j)$
- Each path to these neighbours extends to a unique path to (i, j)
- Recurrence for $P(i, j)$, number of paths from $(0, 0)$ to (i, j)
  - $P(i, j) = P(i-1, j) + P(i, j-1)$
  - $P(0, 0) = 1$ — base case

$$\begin{tikzcd} (\boldsymbol{i}-\mathbf{1},\boldsymbol{j}) \arrow[r] & (\boldsymbol{i},\boldsymbol{j}) \\ & (\boldsymbol{i},\boldsymbol{j}-\mathbf{1}) \arrow[u] \end{tikzcd}$$

Inductive formulation

- How can a path reach (i, j)
  - Move up from $(i, j-1)$
  - Move right from $(i-1, j)$
- Each path to these neighbours extends to a unique path to (i, j)
- Recurrence for $P(i, j)$, number of paths from $(0, 0)$ to (i, j)
  - $P(i, j) = P(i-1, j) + P(i, j-1)$
  - $P(0, 0) = 1$ — base case
  - $P(i, 0) = P(i-1, 0)$ — bottom row

$$\begin{tikzcd}
(\boldsymbol{i-1, j}) \arrow[r] & (\boldsymbol{i, j}) \\
 & (\boldsymbol{i, j-1}) \arrow[u]
\end{tikzcd}$$

Inductive formulation

- How can a path reach (i,j)
  - Move up from $(i,j-1)$
  - Move right from $(i-1,j)$
- Each path to these neighbours extends to a unique path to (i,j)
- Recurrence for $P(i,j)$, number of paths from $(0,0)$ to (i,j)
  - $P(i,j) = P(i-1,j) + P(i,j-1)$
  - $P(0,0) = 1$ — base case
  - $P(i,0) = P(i-1,0)$ — bottom row
  - $P(0,j) = P(0,j-1)$ — left column

$$\begin{tikzcd} (\boldsymbol{i-1,j}) \arrow[r] & (\boldsymbol{i,j}) \\ & (\boldsymbol{i,j-1}) \arrow[u] \end{tikzcd}$$

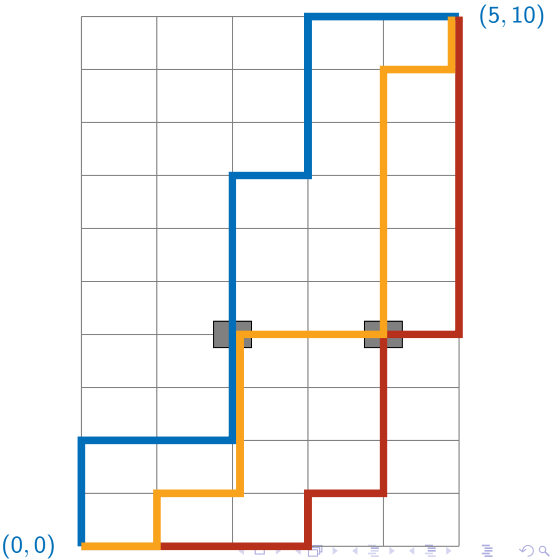
Inductive formulation

- How can a path reach (i, j)
  - Move up from $(i, j-1)$
  - Move right from $(i-1, j)$
- Each path to these neighbours extends to a unique path to (i, j)
- Recurrence for $P(i, j)$, number of paths from $(0, 0)$ to (i, j)
  - $P(i, j) = P(i-1, j) + P(i, j-1)$
  - $P(0, 0) = 1$ — base case
  - $P(i, 0) = P(i-1, 0)$ — bottom row
  - $P(0, j) = P(0, j-1)$ — left column
- $P(i, j) = 0$ if there is a hole at (i, j)

$$\begin{tikzcd} (\boldsymbol{i-1,j}) \arrow[r] & (\boldsymbol{i,j}) \\ & (\boldsymbol{i,j-1}) \arrow[u] \end{tikzcd}$$

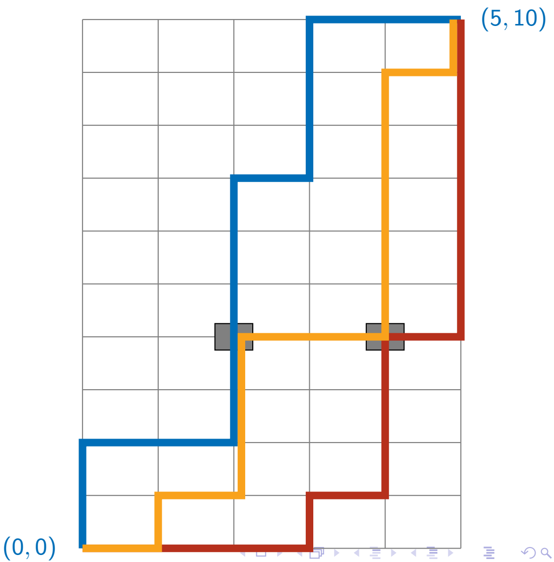
Computing $P(i,j)$

- Naive recursion recomputes same subproblem repeatedly



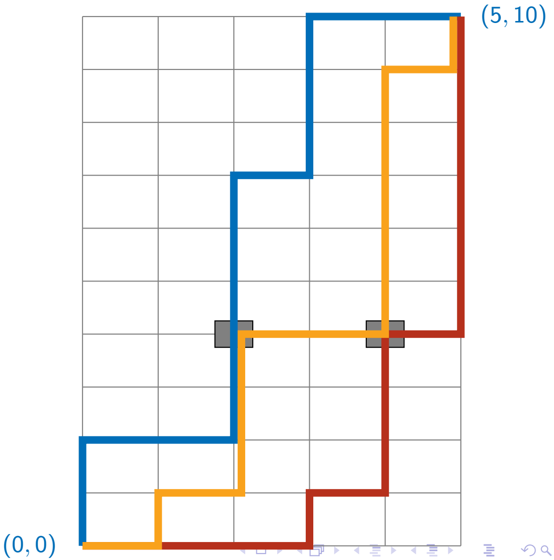
Computing $P(i,j)$

- Naive recursion recomputes same subproblem repeatedly
  - $P(5,10)$ requires $P(4,10)$, $P(5,9)$



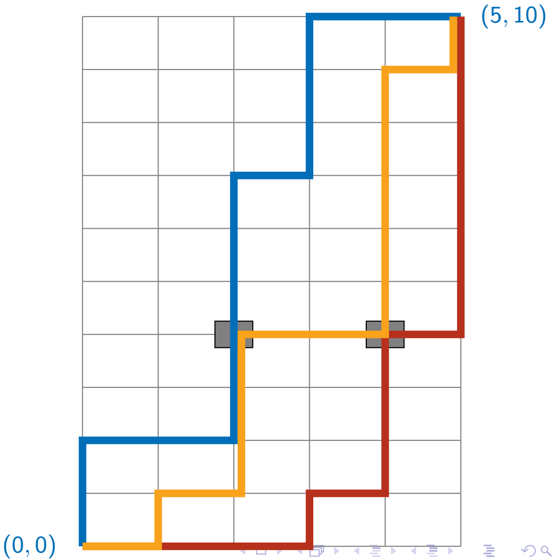
Computing $P(i,j)$

- Naive recursion recomputes same subproblem repeatedly
  - $P(5,10)$ requires $P(4,10)$, $P(5,9)$
  - Both $P(4,10)$, $P(5,9)$ require $P(4,9)$



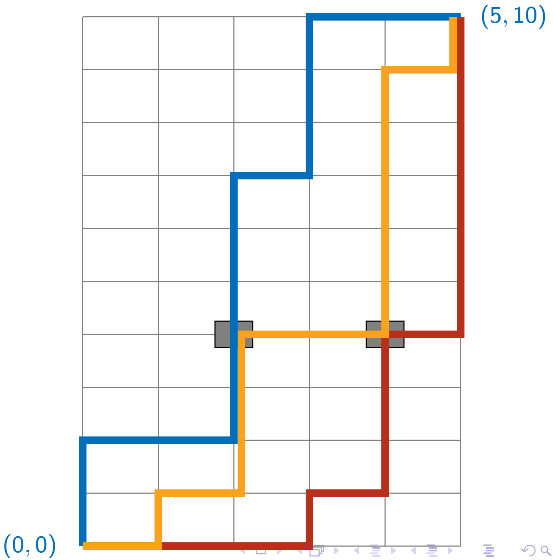
Computing $P(i, j)$

- Naive recursion recomputes same subproblem repeatedly
  - $P(5, 10)$ requires $P(4, 10)$, $P(5, 9)$
  - Both $P(4, 10)$, $P(5, 9)$ require $P(4, 9)$
- Use memoization . . .



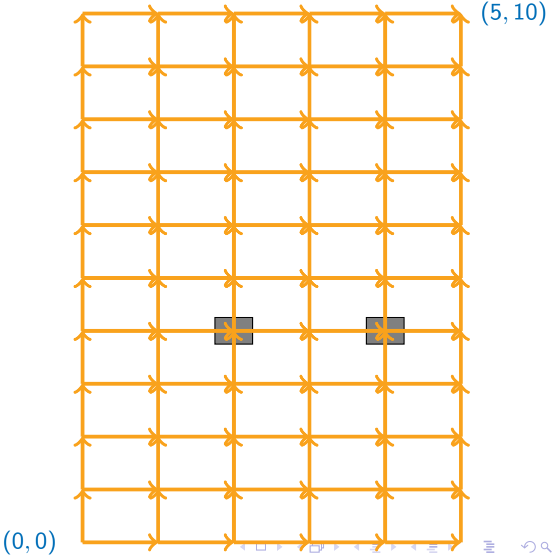
Computing $P(i, j)$

- Naive recursion recomputes same subproblem repeatedly
  - $P(5, 10)$ requires $P(4, 10)$, $P(5, 9)$
  - Both $P(4, 10)$, $P(5, 9)$ require $P(4, 9)$
- Use memoization ...
- ...or find a suitable order to compute the subproblems



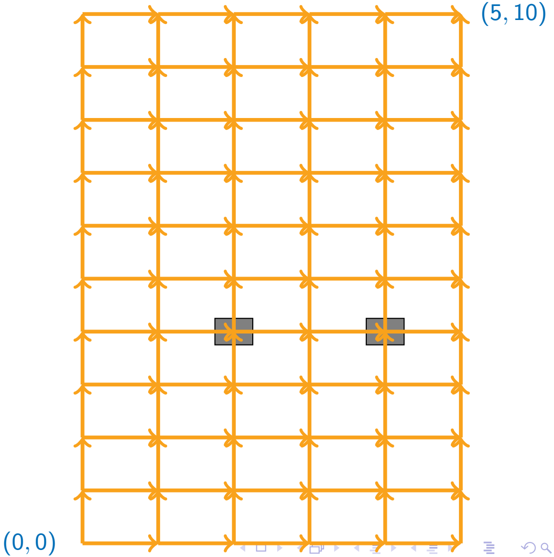
Dynamic programming

- Identify DAG structure



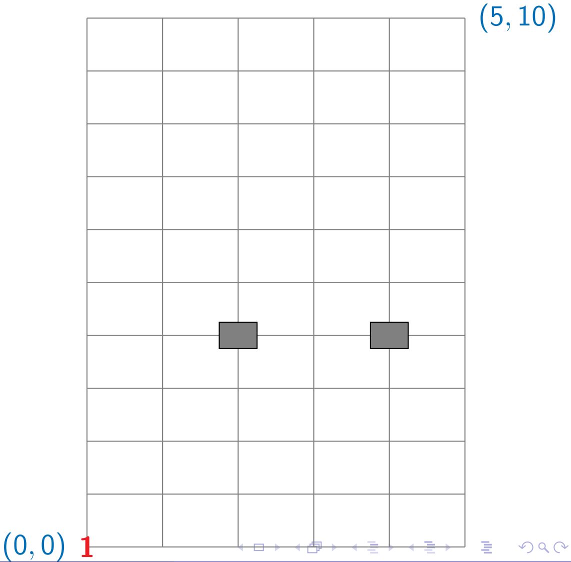
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies



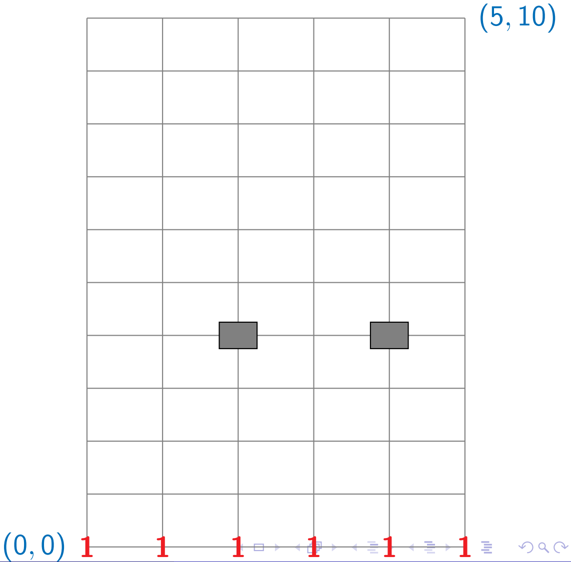
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$



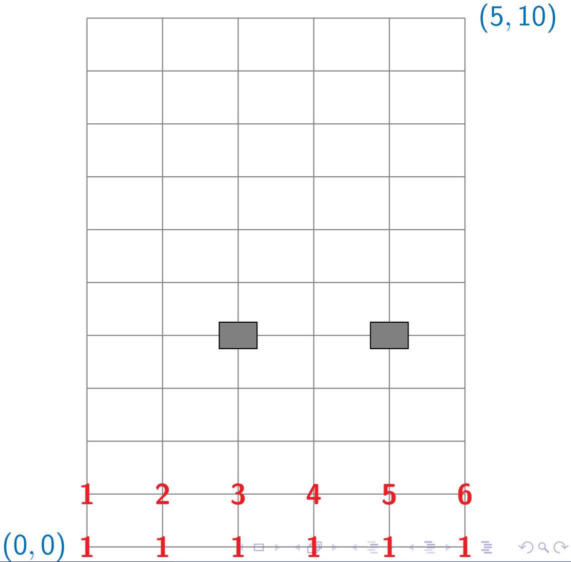
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row



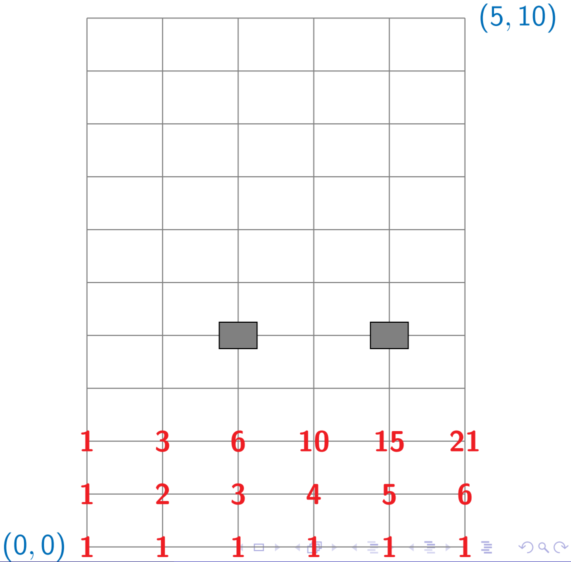
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row



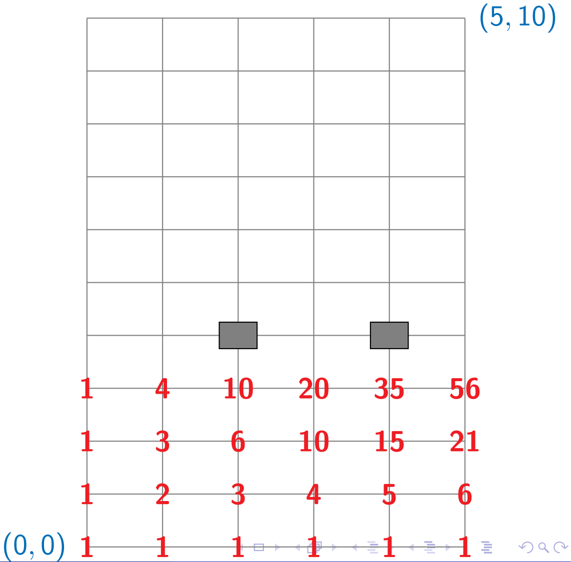
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row



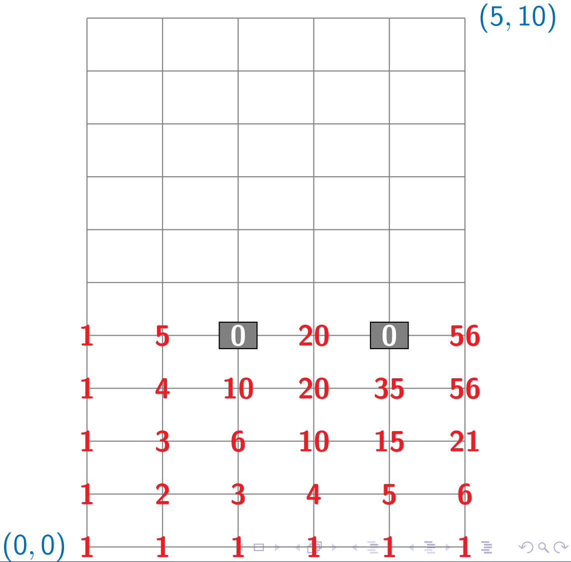
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row



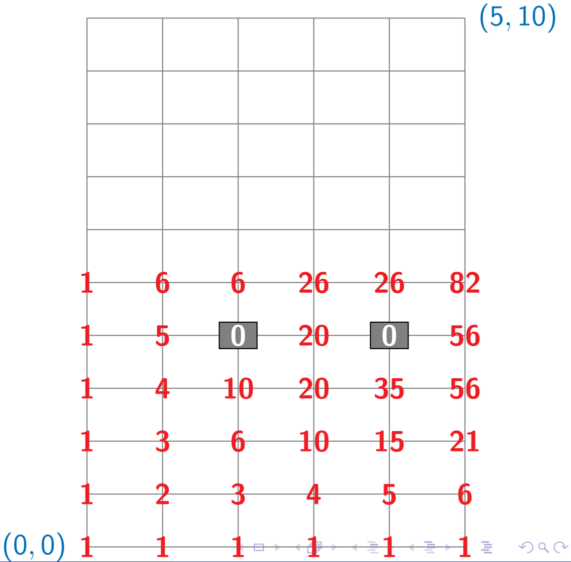
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row



Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row



Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row

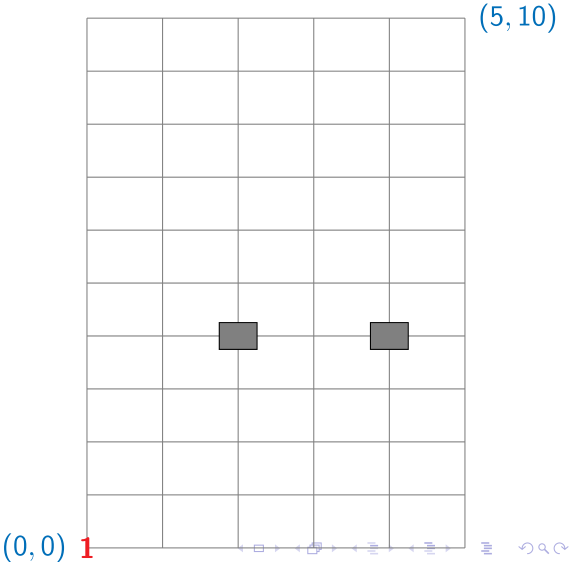
$(5,10)$

1	11	51	181	526	1358
1	10	40	130	345	832
1	9	30	90	215	487
1	8	21	60	125	272
1	7	13	39	65	147
1	6	6	26	26	82
1	5	$\boxed{0}$	20	$\boxed{0}$	56
1	4	10	20	35	56
1	3	6	10	15	21
1	2	3	4	5	6
1	1	1	1	1	1

$(0,0)$

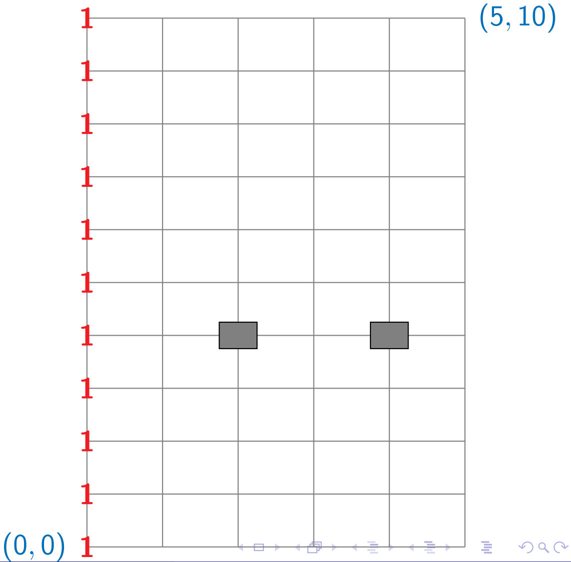
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row
- Fill column by column



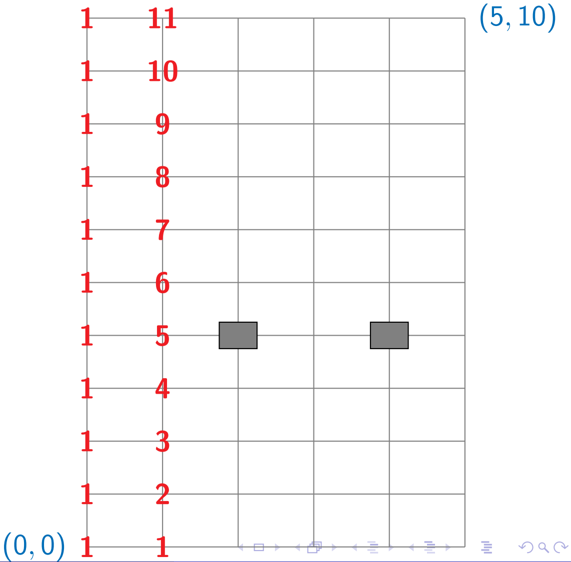
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row
- Fill column by column



Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row
- Fill column by column



Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row
- Fill column by column

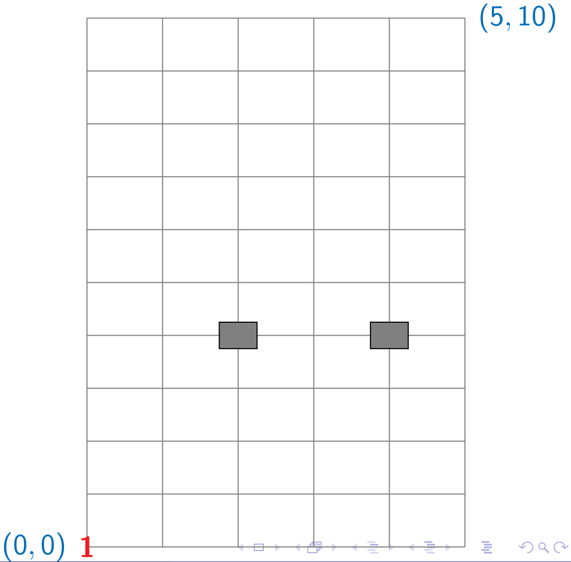
$(5,10)$

1	11	51	181	526	1358
1	10	40	130	345	832
1	9	30	90	215	487
1	8	21	60	125	272
1	7	13	39	65	147
1	6	6	26	26	82
1	5	0	20	0	56
1	4	10	20	35	56
1	3	6	10	15	21
1	2	3	4	5	6
1	1	1	1	1	1

$(0,0)$

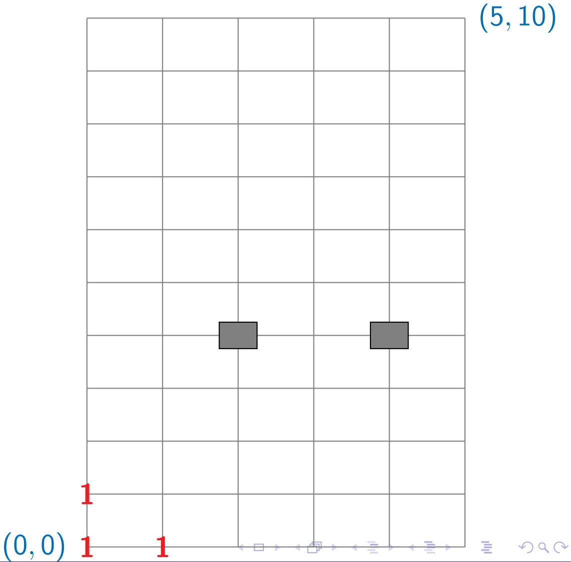
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row
- Fill column by column
- Fill diagonal by diagonal



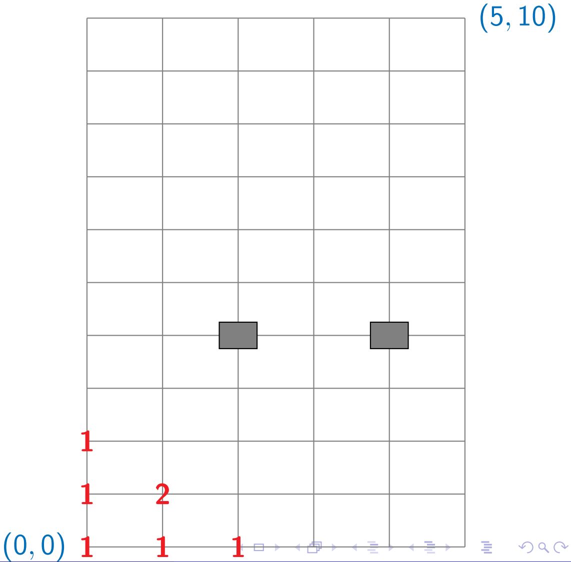
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row
- Fill column by column
- Fill diagonal by diagonal



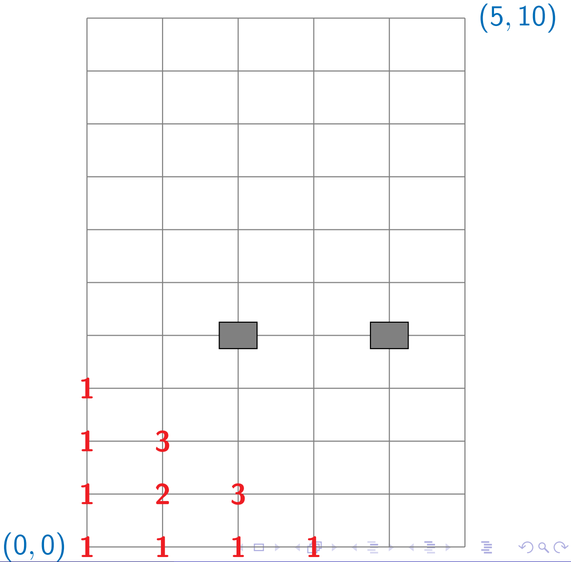
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row
- Fill column by column
- Fill diagonal by diagonal



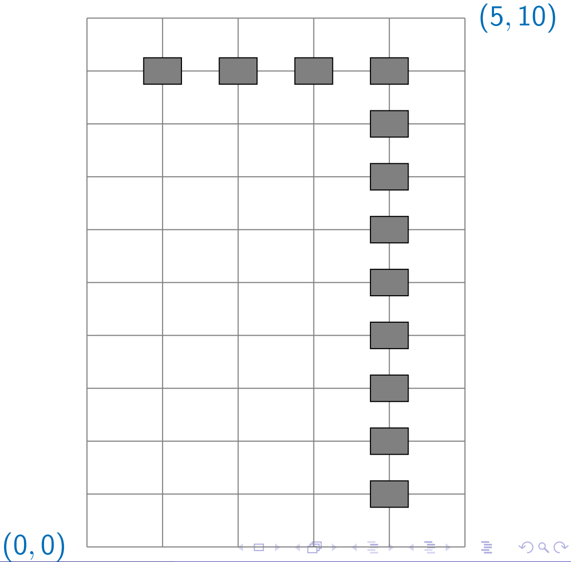
Dynamic programming

- Identify DAG structure
- $P(0,0)$ has no dependencies
- Start at $(0,0)$
- Fill row by row
- Fill column by column
- Fill diagonal by diagonal



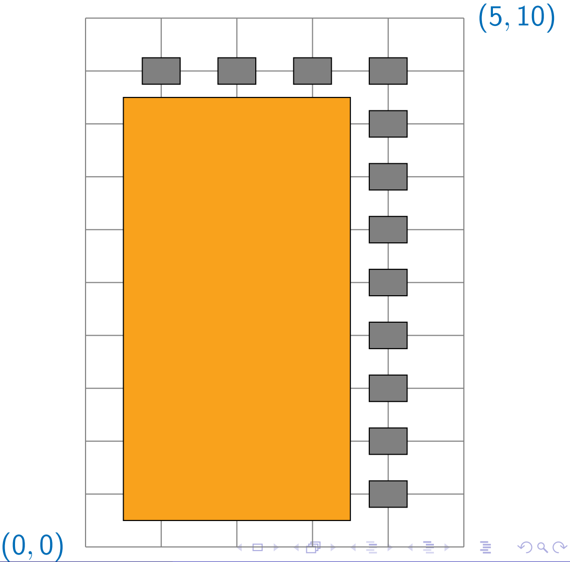
Memoization vs dynamic programming

- Barrier of holes just inside the border



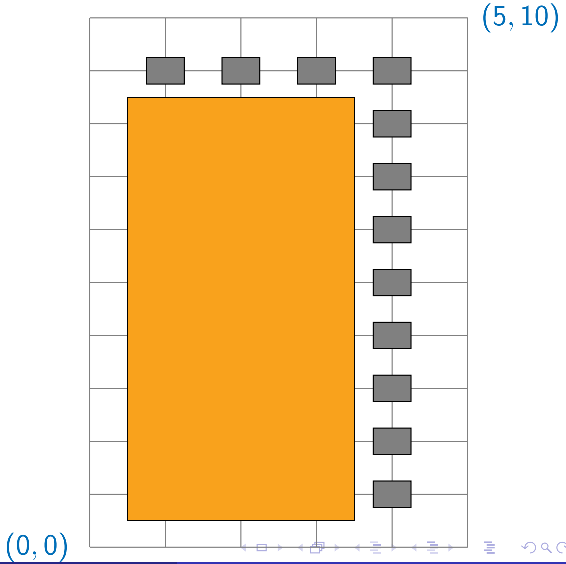
Memoization vs dynamic programming

- Barrier of holes just inside the border
- Memoization never explores the shaded region



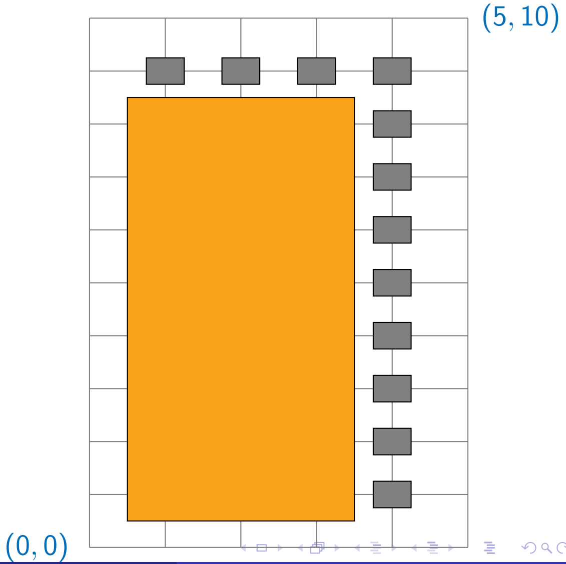
Memoization vs dynamic programming

- Barrier of holes just inside the border
- Memoization never explores the shaded region
- Memo table has $O(m+n)$ entries



Memoization vs dynamic programming

- Barrier of holes just inside the border
- Memoization never explores the shaded region
- Memo table has $O(m+n)$ entries
- Dynamic programming blindly fills all mn cells of the table



Memoization vs dynamic programming

- Barrier of holes just inside the border
- Memoization never explores the shaded region
- Memo table has $O(m+n)$ entries
- Dynamic programming blindly fills all mn cells of the table
- Tradeoff between recursion and iteration
  - "Wasteful" dynamic programming still better in general