Algorithms

(Dynamic Programming)

Pramod Ganapathi
Department of Computer Science
State University of New York at Stony Brook

April 27, 2021

Contents

What is dynamic programming (DP)?

- DP = algorithm design technique
- DP finds optimal solutions to a problem by combining optimal solutions to its overlapping subproblems by saving solutions to subproblems in a table and never recomputing them
- DP = efficient table filling where the value at each cell (solution to a problem) depends on the value(s) at one or more other cells (solutions to overlapping subproblems)

DP problem-solving template

7-step process:

Step 1. Problem

Step 2. Subproblem

Step 3. Recurrence ▷ brain of DP

Step 4. Dependency

Step 5. Algorithm

Step 6. Table

Step 7. Complexity

Fibonacci number

Step 1. Problem

- [Link] Compute the nth Fibonacci number

Step 2. Subproblem

$F[i] = i$th Fibonacci number

Compute $F[n]$

Step 3. Recurrence

$$F[i] = \left\{ \begin{array}{ll} 0 & \text{if } i = 0, \\ 1 & \text{if } i = 1, \\ F[i-1] + F[i-2] & \text{if } i \geq 2. \end{array} \right\}$$

Step 4. Dependency

Step 5. Algorithm

FIBONACCI(n)

Input: Whole number n
Output: Fibonacci number F_n
1. $F[0] \leftarrow 0$; $F[1] \leftarrow 1$
2. **for** $i \leftarrow 2$ **to** n **do**
3. $\quad F[i] \leftarrow F[i-1] + F[i-2]$
4. **return** $F[n]$

Step 6. Table

i	0	1	2	3	4	5	6	7	8	9	10
$F[i]$	1	1	2	3	5	8	13	21	34	55	89

Step 7. Complexity

$$\text{Time} \in \Theta(n), \text{Space} \in \Theta(n)$$

Number expressed as a sum

Step 1. Problem

- [Link] Count the number of ways to express a number as a sum of 1, 3, and 4.
- Example: 4 can be expressed as a sum of 1, 3, and 4 in 4 ways:

$$4 = \left\{\begin{matrix} 1+1+1+1 \\ 1+3 \\ 3+1 \\ 4 \end{matrix}\right\}$$

Step 2. Subproblem

$C[i] = \#$Ways of expressing i as a sum of $1, 3, 4$

Compute $C[n]$

Step 3. Recurrence

$$C[i] = \left\{ \begin{array}{ll} 1 & \text{if } i = 1, \\ 1 & \text{if } i = 2, \\ 2 & \text{if } i = 3, \\ 4 & \text{if } i = 4, \\ C[i-1] + C[i-3] + C[i-4] & \text{if } i \geq 5. \end{array} \right\}$$

Step 4. Dependency

Step 5. Algorithm

NumberExpressedAsSum(n)

Input: Natural number n
Output: $\#$ Ways of expressing n as a sum of 1,3,4

1. $C[1] \leftarrow 1; C[2] \leftarrow 1; C[3] \leftarrow 2$; $C[4] \leftarrow 4$
2. **for** $i \leftarrow 5$ **to** n **do**
3. $\quad C[i] \leftarrow C[i-1] + C[i-3] + C[i-4]$
4. **return** $C[n]$

Step 6. Table

i	0	1	2	3	4	5	6	7	8	9
$C[i]$	1	1	2	4	6	9	15	25	40	64

Step 7. Complexity

$$\text{Time} \in \Theta\left(n\right), \text{Space} \in \Theta\left(n\right)$$

Tiling problem

Step 1. Problem

- [Link] Count the number of ways to tile a board of size $2 \times n$ using 1×2 or 2×1 tiles.

Step 2. Subproblem

$C[i] = \#$Ways to tile a $2 \times i$ board using 1×2 or 2×1 tiles

Compute $C[n]$

Step 3. Recurrence

$$C[i] = \begin{cases} 1 & \text{if } i = 1, \\ 2 & \text{if } i = 2, \\ C[i-1] + C[i-2] & \text{if } i \geq 3. \end{cases}$$

Step 4. Dependency

Step 5. Algorithm

TILINGS(n)

Input: #Columns $n \geq 1$ (in the $2 \times n$ board)
Output: #Ways to tile the board using 1×2 or 2×1 tiles

1. $C[1] \leftarrow 1; C[2] \leftarrow 2$
2. **for** $i \leftarrow 3$ **to** n **do**
3. $\quad C[i] \leftarrow C[i-1] + C[i-2]$
4. **return** $C[n]$

Step 6. Table

i	1	2	3	4	5	6	7	8	9	10
$C[i]$	1	2	3	5	8	13	21	34	55	89

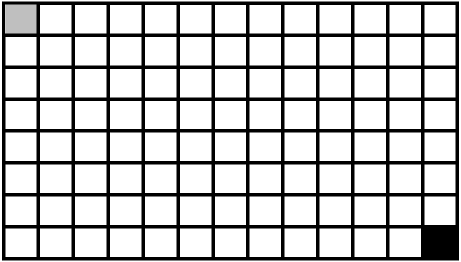
Step 7. Complexity

$$\text{Time} \in \Theta(n)\,, \text{Space} \in \Theta(n)$$

Count paths in a grid

Step 1. Problem

- [Link] Count all possible paths from top left corner [0, 0] to bottom right corner $[m, n]$ of a rectangular grid by only considering right and down moves.



Step 2. Subproblem

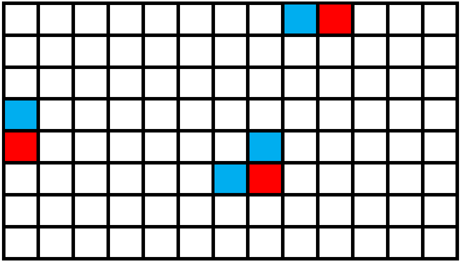
$C[i,j] = \#$Paths to reach $[i,j]$ from $[0,0]$ with $\rightarrow$, $\downarrow$

Compute $C[m,n]$

Step 3. Recurrence

$$C[i,j] = \left\{ \begin{array}{ll} 1 & \text{if } i = 0 \text{ and } j = 0, \\ \left\{ \begin{array}{l} C[i,j-1] \times \boxed{j \geq 1} + \\ C[i-1,j] \times \boxed{i \geq 1} \end{array} \right\} & \text{if } i \geq 1 \text{ or } j \geq 1. \end{array} \right\}$$

Step 4. Dependency



Step 5. Algorithm

$\textsc{PathsOnAGrid}(m, n)$

Input: Number of rows m and number of columns n
Output: #Paths from top left $(0, 0)$ to bottom right (m, n)
1. **for** $i \leftarrow 0$ **to** m **do** $C[i, 0] \leftarrow 1$
2. **for** $j \leftarrow 0$ **to** n **do** $C[0, j] \leftarrow 1$
3. **for** $i \leftarrow 1$ **to** m **do**
4.   **for** $j \leftarrow 1$ **to** n **do**
5. $C[i, j] \leftarrow C[i-1, j] + C[i, j-1]$
6. **return** $C[m, n]$

Step 6. Table

$C[i,j]$	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7	8
2	1	3	6	10	15	21	28	36
3	1	4	10	20	35	56	84	120
4	1	5	15	35	70	126	210	330
5	1	6	21	56	126	252	462	792
6	1	7	28	84	210	462	924	1716
7	1	8	36	120	330	792	1716	3432

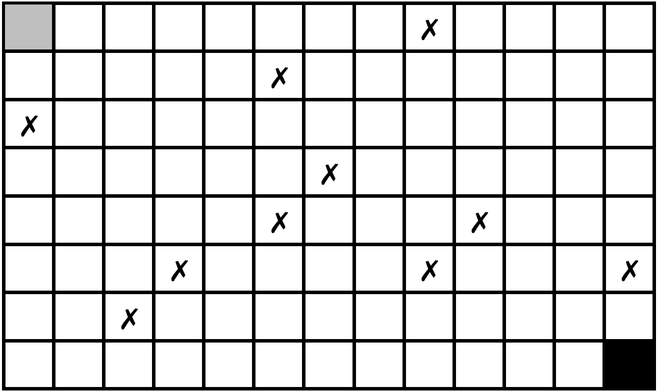
Step 7. Complexity

$$\text{Time} \in \Theta(mn), \text{Space} \in \Theta(mn)$$

Count paths in a grid with blocks

Step 1. Problem

- [Link] Count all possible paths from top left corner [0, 0] to bottom rightmost corner $[m, n]$ of a rectangular grid by only considering right and down moves.
- Cells with ✗ are blocked and cannot be passed.



Step 2. Subproblem

$C[i,j] = $ #Paths to reach $[i,j]$ from $[0,0]$ with $\rightarrow$, $\uparrow$

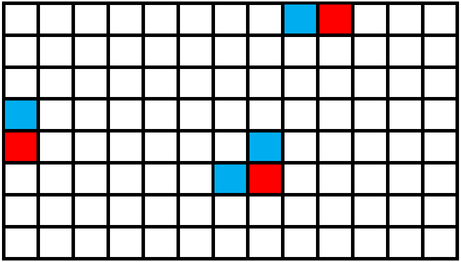
$$O[i,j] = \begin{cases} true & \text{if } [i,j] \text{ is open,} \\ false & \text{if } [i,j] \text{ is closed.} \end{cases}$$

Compute $C[m,n]$

Step 3. Recurrence

$$C[i,j] = \left\{ \begin{array}{ll} 1 \times \boxed{O[i,j]} & \text{if } i = 0, j = 0, \\ \left\{ \begin{array}{l} C[i,j-1] \times \boxed{j \geq 1} + \\ C[i-1,j] \times \boxed{i \geq 1} \end{array} \right\} \times \boxed{O[i,j]} & \text{if } i \geq 1 \text{ or } j \geq 1. \end{array} \right\}$$

Step 4. Dependency



Step 5. Algorithm

$\textsc{PathsOnAGridWithBlocks}(O[m,n])$

Input: Rectangular grid $O[m,n]$ with cells blocked denoted as $false$ and open cells as $true$.
Output: #Paths from top left $(0,0)$ to bottom right (m,n)

1. **for** $i \leftarrow 0$ **to** m **do** $C[i,0] \leftarrow (O[i,0] == true\ ?\ 1 : 0)$
2. **for** $j \leftarrow 0$ **to** n **do** $C[0,j] \leftarrow (O[0,j] == true\ ?\ 1 : 0)$
3. **for** $i \leftarrow 1$ **to** m **do**
4.   **for** $j \leftarrow 1$ **to** n **do**
5. $C[i,j] \leftarrow (C[i-1,j] + C[i,j-1]) \times (O[i,j] == true\ ?\ 1 : 0)$
6. **return** $C[m,n]$

Step 6. Table

$O[i,j]$	0	1	2	3	4
0	✓	✓	✗	✓	✓
1	✓	✓	✓	✗	✗
2	✓	✓	✓	✓	✓
3	✗	✓	✗	✓	✓
4	✓	✓	✓	✓	✓

$C[i,j]$	0	1	2	3	4
0	1	1	0	0	0
1	1	2	2	0	0
2	1	3	5	5	5
3	0	3	0	5	10
4	0	3	3	8	18

Step 7. Complexity

$$\mathsf{Time} \in \Theta\left(mn\right), \mathsf{Space} \in \Theta\left(mn\right)$$

Binomial coefficient

Step 1. Problem

- [Link] Compute n choose r i.e., $C[n,r]$, where,
$$(x+y)^n = \sum_{i=0}^{n} C[n,i] x^{n-i} y^i$$

Step 2. Subproblem

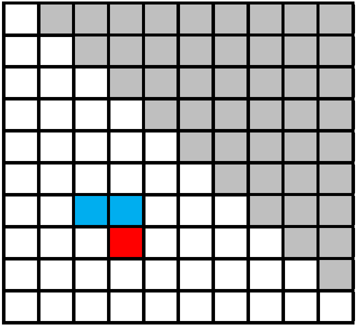
$C[i, j] = i$ choose j such that $0 \leq j \leq i$

Compute $C[n, r]$

Step 3. Recurrence

$$C[i,j] = \begin{cases} 1 & \text{if } j = 0 \text{ or } j = i, \\ C[i-1,j-1] + C[i-1,j] & \text{if } j \in [1, i-1]. \end{cases}$$

Step 4. Dependency



Step 5. Algorithm

BINOMIALCOEFFICIENT(n, r)

Input: Two whole numbers n and r such that $n \geq r$
Output: Number of r-sized subsets from an n-element set

```
for i ← 0 to n do
  for j ← 0 to min(i, r) do
    if j = 0 or j = i do
      C[i, j] ← 1
    else
      C[i, j] ← C[i − 1, j − 1] + C[i − 1, j]
return C[n, r]
```

Step 6. Table

$B[i,j]$	0	1	2	3	4	5	6	7	8	9
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

Step 7. Complexity

$$\text{Time} \in \Theta\left(nr\right), \text{Space} \in \Theta\left(nr\right)$$

Longest common substring

Step 1. Problem

- [Link] Compute length of the longest common substring between two strings $X[1..m]$ and $Y[1..n]$.

Step 2. Subproblem

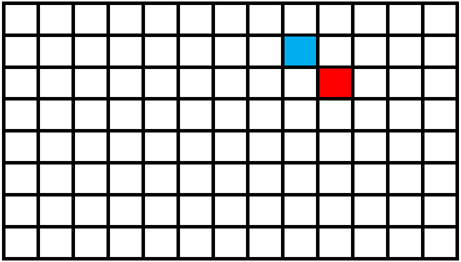
$L[i, j] =$ Length of the longest common substring between strings $X[1..i]$ and $Y[1..j]$.

Compute $L[m, n]$

Step 3. Recurrence

$$L[i,j] = \left\{ \begin{array}{ll} 0 & \text{if } i \times j = 0, \\ (L[i-1,j-1]+1) \times \boxed{X[i]=Y[j]} & \text{if } i,j \geq 1. \end{array} \right\}$$

Step 4. Dependency



Step 5. Algorithm

$\textsc{LongestCommonSubstring}(X[1..m], Y[1..n])$

Input: Strings $X[1..m]$, $Y[1..n]$
Output: Length of the longest common substring between strings X and Y

for $i \leftarrow 0$ **to** m **do** $L[i,0] \leftarrow 0$
for $j \leftarrow 1$ **to** n **do** $L[0,j] \leftarrow 0$
for $i \leftarrow 1$ **to** m **do**
  for $j \leftarrow 1$ **to** n **do**
    if $X[i] = Y[j]$ **then**
      $L[i,j] \leftarrow L[i-1, j-1] + 1$
    else if $X[i] \neq Y[j]$ **then**
      $L[i,j] \leftarrow 0$
return $L[m,n]$

Step 6. Table

$L[i,j]$	0 ($Y[0] = \varnothing$)	1 ($Y[1] = N$)	2 ($Y[2] = E$)	3 ($Y[3] = W$)	4 ($Y[4] = T$)	5 ($Y[5] = O$)	6 ($Y[6] = N$)
0 ($X[0] = \varnothing$)	0	0	0	0	0	0	0
1 ($X[1] = E$)	0	0	1	0	0	0	0
2 ($X[2] = I$)	0	0	0	0	0	0	0
3 ($X[3] = N$)	0	1	0	0	0	0	1
4 ($X[4] = S$)	0	0	0	0	0	0	0
5 ($X[5] = T$)	0	0	0	0	1	0	0
6 ($X[6] = E$)	0	0	1	0	0	0	0
7 ($X[7] = I$)	0	0	0	0	0	0	0
8 ($X[8] = N$)	0	1	0	0	0	0	1

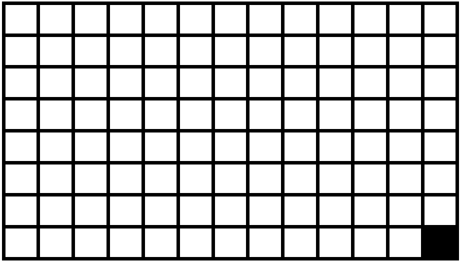
Step 7. Complexity

$$\text{Time} \in \Theta(mn), \text{Space} \in \Theta(mn)$$

Edit Distance

Step 1. Problem

- [Link] Compute the minimum number of edits required to convert string $X[1..m]$ into string $Y[1..n]$ using the three operations: insert, remove, and replace.



Step 2. Subproblem

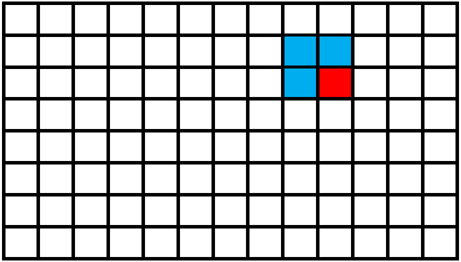
$D[i, j] =$ Least editing distance between strings $X[1..i]$ and $Y[1..j]$.

Compute $D[m, n]$

Step 3. Recurrence

$$D[i,j] = \begin{cases} 0 & \text{if } i \times j = 0, \\ \left\{ \begin{array}{l} D[i-1,j-1] \times \boxed{X[i] = Y[j]} + \\ \min \begin{Bmatrix} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+1 \end{Bmatrix} \times \boxed{X[i] \neq Y[j]} \end{array} \right\} & \text{if } i,j \geq 1. \end{cases}$$

Step 4. Dependency



Step 5. Algorithm

EDITDISTANCE($X[1..m], Y[1..n]$)

Input: Strings $X[1..m]$, $Y[1..n]$

Output: Least editing distance between X and Y

for $i \leftarrow 0$ **to** m **do** $D[i, 0] \leftarrow i$
for $j \leftarrow 1$ **to** n **do** $D[0, j] \leftarrow j$
for $i \leftarrow 1$ **to** m **do**
  for $j \leftarrow 1$ **to** n **do**
    if $X[i] = Y[j]$ **then**
      $D[i, j] \leftarrow D[i-1, j-1]$
    else if $X[i] \neq Y[j]$ **then**
      $D[i, j] \leftarrow \min(D[i-1, j], D[i, j-1], D[i-1, j-1]) + 1$
return $D[m, n]$

Step 6. Table

$D[i,j]$	0 ($Y[0] = \varnothing$)	1 ($Y[1] = N$)	2 ($Y[2] = E$)	3 ($Y[3] = W$)	4 ($Y[4] = T$)	5 ($Y[5] = O$)	6 ($Y[6] = N$)
0 ($X[0] = \varnothing$)	0	1	2	3	4	5	6
1 ($X[1] = E$)	1	1	1	2	3	4	5
2 ($X[2] = I$)	2	2	2	2	3	4	5
3 ($X[3] = N$)	3	2	3	3	3	4	4
4 ($X[4] = S$)	4	3	3	4	4	4	5
5 ($X[5] = T$)	5	4	4	4	4	5	5
6 ($X[6] = E$)	6	5	4	5	5	5	6
7 ($X[7] = I$)	7	6	5	5	6	6	6
8 ($X[8] = N$)	8	7	6	6	6	7	6

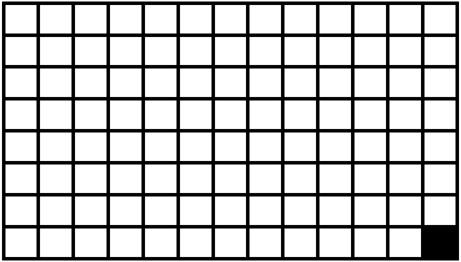
Step 7. Complexity

$$\text{Time} \in \Theta\left(mn\right), \text{Space} \in \Theta\left(mn\right)$$

Longest common subsequence

Step 1. Problem

- [Link] Compute the length of the longest common subsequence (LCS) between two strings X and Y of lengths m and n, respectively.



Step 2. Subproblem

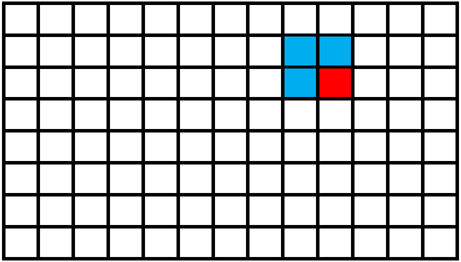
$L[i, j] =$ Length of the LCS between strings $X[1..i]$ and $Y[1..j]$.

Compute $L[m, n]$

Step 3. Recurrence

$$L[i,j] = \left\{ \begin{array}{ll} 0 & \text{if } i \times j = 0, \\ \left\{ \begin{array}{l} (L[i-1,j-1]+1) \times \boxed{X[i] = Y[j]} + \\ \max \left\{ \begin{array}{l} L[i,j-1] \\ L[i-1,j] \end{array} \right\} \times \boxed{X[i] \neq Y[j]} \end{array} \right\} & \text{if } i,j \geq 1. \end{array} \right\}$$

Step 4. Dependency



Step 5. Algorithm

LONGESTCOMMONSUBSEQUENCE($X[1..m], Y[1..n]$)

Input: Strings $X[1..m]$, $Y[1..n]$
Output: Length of the longest common subsequence between X and Y

for $i \leftarrow 0$ **to** m **do** $L[i, 0] \leftarrow 0$
for $j \leftarrow 1$ **to** n **do** $L[0, j] \leftarrow 0$
for $i \leftarrow 1$ **to** m **do**
  for $j \leftarrow 1$ **to** n **do**
    if $X[i] = Y[j]$ **then**
      $L[i, j] \leftarrow L[i-1, j-1] + 1$
    else if $X[i] \neq Y[j]$ **then**
      $L[i, j] \leftarrow \max(L[i-1, j], L[i, j-1])$
return $L[m, n]$

Step 6. Table

$S[i,j]$	0 $(Y[0] = \varnothing)$	1 $(Y[1] = N)$	2 $(Y[2] = E)$	3 $(Y[3] = W)$	4 $(Y[4] = T)$	5 $(Y[5] = O)$	6 $(Y[6] = N)$
0 $(X[0] = \varnothing)$	0	0	0	0	0	0	0
1 $(X[1] = E)$	0	0	1	1	1	1	1
2 $(X[2] = I)$	0	0	1	1	1	1	1
3 $(X[3] = N)$	0	1	1	1	1	1	2
4 $(X[4] = S)$	0	1	1	1	1	1	2
5 $(X[5] = T)$	0	1	1	1	2	2	2
6 $(X[6] = E)$	0	1	2	2	2	2	2
7 $(X[7] = I)$	0	1	2	2	2	2	2
8 $(X[8] = N)$	0	1	2	2	2	2	3

Step 7. Complexity

$$\text{Time} \in \Theta\left(mn\right), \text{Space} \in \Theta\left(mn\right)$$

Subset sum

Step 1. Problem

- [Link] Given a set of positive integers $A[1..n]$, and a value k, determine if there is a subset of A with sum equal to k.

Step 2. Subproblem

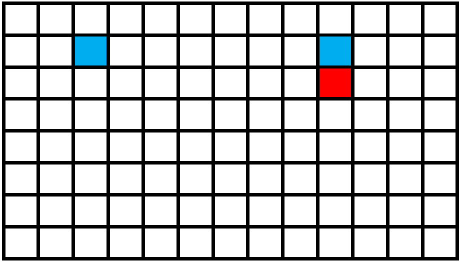
$S[i, j] =$ Boolean value representing the existence/non-existence of a subset of $A[0..i]$ with sum equal to j. ($A[0] = 0$)

Compute $S[n, k]$

Step 3. Recurrence

$$S[i,j] = \begin{cases} false & \text{if } i = 0, j \in [1,k], \\ true & \text{if } i \in [0,k], j = 0, \\ \left\{\begin{matrix} S[i-1,j] \text{ or} \\ S[i-1, j-A[i]] \times \boxed{j \geq A[i]} \end{matrix}\right\} & \text{if } i \in [1,n], j \in [1,k]. \end{cases}$$

Step 4. Dependency



Step 5. Algorithm

SubsetSum$(A[1..n], k)$

Input: Set of positive integers $A[1..n]$ and a value k
Output: Boolean value representing the existence/non-existence of a subset of elements in A with sum equal to k.

```
for i ← 0 to n do S[i, 0] ← true
for j ← 1 to k do S[0, j] ← false
for i ← 1 to n do
  for j ← 1 to k do
    if j ≥ A[i] then
      S[i, j] ← S[i − 1, j] or S[i − 1, j − A[i]]
    else if j < A[i] then
      S[i, j] ← S[i − 1, j]
```

Step 6. Table

$S[i,j]$	0	1	2	3	4	5	6	7	8
0 $(A[0] = 0)$	✓	✗	✗	✗	✗	✗	✗	✗	✗
1 $(A[1] = 2)$	✓	✗	✓	✗	✗	✗	✗	✗	✗
2 $(A[2] = 3)$	✓	✗	✓	✓	✗	✓	✗	✗	✗
3 $(A[3] = 5)$	✓	✗	✓	✓	✗	✓	✗	✓	✓
4 $(A[4] = 9)$	✓	✗	✓	✓	✗	✓	✗	✓	✓

Step 7. Complexity

$$\text{Time} \in \Theta\left(nk\right), \text{Space} \in \Theta\left(nk\right)$$

Coin change (mincoins)

Step 1. Problem

- [Link] We have coins of denominations $C[1], C[2], \ldots, C[m]$ such that $C[1] > C[2] > \cdots > C[m]$. Compute the minimum number of coins to make a change for n amount.
- Coin denominations $= \{9, 6, 5, 1\}$, $n = 11$, and mincoins $= 2$.

Step 2. Subproblem

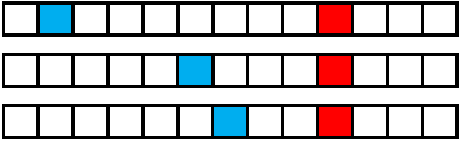
$M[i] =$ Minimum $\#$coins to make a change for value i
using coin denominations $C[1] > C[2] > \cdots > C[m]$

Compute $M[n]$

Step 3. Recurrence

$$M[i] = \begin{cases} 0 & \text{if } i = 0, \\ \min \left\{ \begin{array}{c} (1 + M[i - C[1]]) \times \boxed{C[1] \leq i} \\ (1 + M[i - C[2]]) \times \boxed{C[2] \leq i} \\ \cdots \\ (1 + M[i - C[m]]) \times \boxed{C[m] \leq i} \end{array} \right\} & \text{if } i \geq 1. \end{cases}$$

Step 4. Dependency



Step 5. Algorithm

$\textsc{MinCoins}(C[1..m], n)$

Input: Coin denominations $C[1] > C[2] > \cdots > C[m]$ and amount n
Output: Minimum #coins to make change for n

1. $M[0] \leftarrow 0$
2. **for** $i \leftarrow 1$ **to** n **do**
3. $\quad minimum \leftarrow \infty$
4. $\quad$ **for** $j \leftarrow 1$ **to** m **do**
5. $\quad\quad$ **if** $C[j] \leq i$ **then**
6. $\quad\quad\quad minimum \leftarrow \min(minimum, 1 + M[i - C[j]])$
7. $\quad M[i] \leftarrow minimum$

Step 6. Table

$$n = 11$$

i	1	2	3	4
$C[i]$	9	6	5	1

i	0	1	2	3	4	5	6	7	8	9	10	11
$M[i]$	0	1	2	3	4	1	1	2	3	1	2	2

Step 7. Complexity

$$\text{Time} \in \Theta(mn), \text{Space} \in \Theta(n)$$

Egg dropping

Step 1. Problem

- [Link] There is an n-floored building and we are given k identical eggs. A threshold floor of a building is defined as the highest floor in the building from and below which when the egg is dropped, the egg does not break, and above which when the egg is dropped, the egg breaks. Find the minimum number of drops required to find the threshold floor.

Step 2. Subproblem

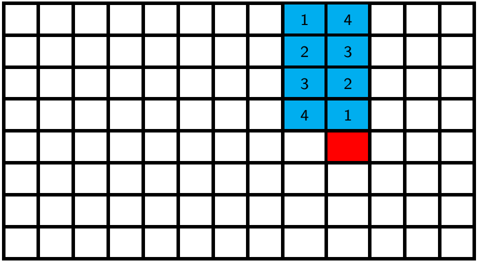
$D[i, j] = \#$Minimum drops to find the threshold floor
in a building of j floors using i eggs

Compute $D[n, k]$

Step 3. Recurrence

$$D[i,j] = \left\{ \begin{array}{ll} i & \text{if } j = 1, \\ i & \text{if } i = 0 \text{ or } 1, \\ 1 + \min_{x \in [1,i]} \left\{ \max \left\{ \begin{array}{c} D[x-1, j-1] \\ D[i-x, j] \end{array} \right\} \right\} & \text{if } i, j \geq 2. \end{array} \right\}$$

Step 4. Dependency



Step 5. Algorithm

```
EggProblemMinimizeDrops(n, k)
Input: Number of floors n and number of eggs k
Output: Minimum #drops D[n, k]
for i ← 1 to n do
  D[i, 1] ← i
for j ← 1 to k do
  D[0, j] ← 0; D[1, j] ← 1
for i ← 2 to n do
  for j ← 2 to k do
    minimum ← i
    for x ← 1 to i do
      maximum ← max(D[x − 1, j − 1], D[i − x, j])
      if maximum < minimum then minimum ← maximum
    D[n, k] ← minimum + 1
return D[n, k]
```

Step 6. Table

$D[i,j]$	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2
3	3	2	2	2	2	2	2	2	2	2
4	4	3	3	3	3	3	3	3	3	3
5	5	3	3	3	3	3	3	3	3	3
6	6	3	3	3	3	3	3	3	3	3
7	7	4	3	3	3	3	3	3	3	3
8	8	4	4	4	4	4	4	4	4	4
9	9	4	4	4	4	4	4	4	4	4
10	10	4	4	4	4	4	4	4	4	4
11	11	5	4	4	4	4	4	4	4	4

Step 7. Complexity

$$\text{Time} \in \Theta\left(n^2 k\right), \text{Space} \in \Theta\left(nk\right)$$