Advanced Dynamic Programming

Dynamic Programming - Reminder

- Given a grid of numbers, find the path from the top-left to the bottom right that minimizes the sum of numbers along this path. You can only move down or right.

5	4	2	8
8	1	6	9
4	5	2	7

$$dp(i,j) = \mathrm{A[i][j]} + \min(dp(i-1,j), dp(i,j-1))$$

DP table

5	9	11	19
13	10	16	25
17	15	17	24

Fill the cells column by column (or row by row), this way, when computing $dp(i,j)$ both $dp(i-1,j)$ and $dp(j-1,i)$ are already known.

Dynamic Programming - Reminder

- Given a grid of numbers, find the path from the top-left to the bottom right that minimizes the sum of numbers along this path. You can only move down or right. **No two consecutive down moves are allowed.**

5	4	2	8
8	1	6	9
4	5	2	7

$$dp(i,j,R) = \mathrm{A[i][j]} + \min\big(dp(i-1,j,D), dp(i-1,j,R)\big)$$
$$dp(i,j,D) = \mathrm{A[i][j]} + dp(i,j-1,R)$$

DP table

D

5	∞	∞	∞
13	10	17	26
∞	19	18	32

R

5	9	11	19
∞	14	16	25
∞	∞	21	25

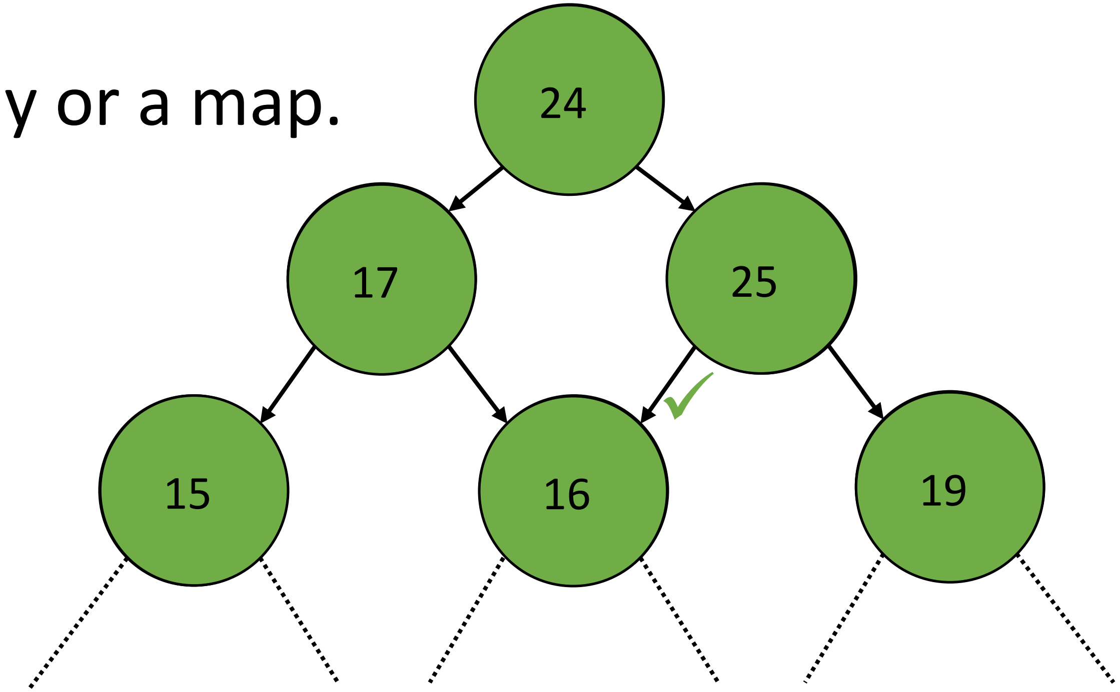
Dynamic Programming - Reminder

- In dynamic programming we usually use iterative implementation
- Although the formula is recursive by nature, the implementation (usually) **does not use recursion**
- The DP-table is built bottom-up, i.e. each value is derived from the previous values when it can be computed.

Memoization - Reminder

- Another approach is to compute the value of a state only when needed to the computation of another state.
- In this approach, we compute the values in top-down fashion, and **use recursion** (usually).
- We will save the values using array or a map.

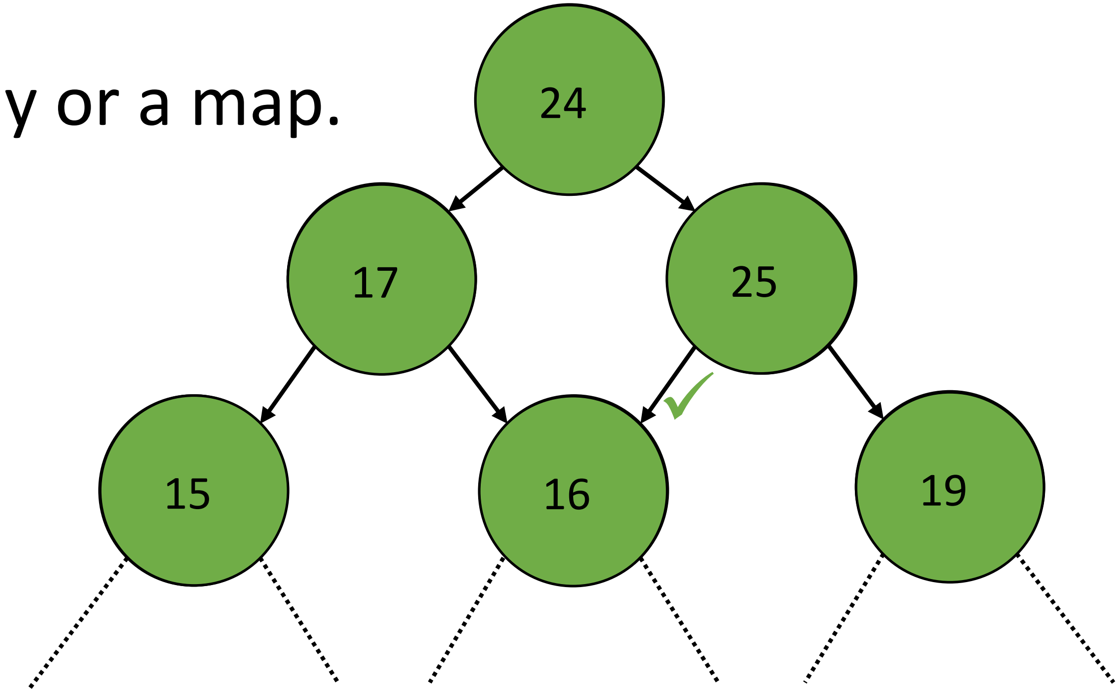
5	4	2	8
8	1	6	9
4	5	2	7



Memoization - Reminder

- Another approach is to compute the value of a state only when needed to the computation of another state.
- In this approach, we compute the values in top-down fashion, and **use recursion** (usually).
- We will save the values using array or a map.

5	4	2	8
8	1	6	9
4	5	2	7

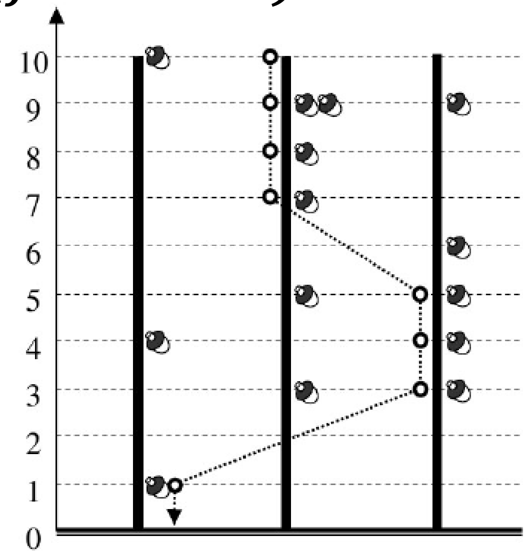


Complexity

- In both DP and memoization, the worst case complexity is:
$$\#[states] \cdot T(state)$$
- $\#[states]$ is the number of possible states
  - Number of cells in the DP-table
  - Number of nodes in the computation graph
- $T(state)$ is the time needed to compute the value of a cell given its predecessors values
  - Usually, number of outgoing edges in the computation graph

Reducing complexity

- Jayjay the squirrel wants to collect acorns.
- There are T trees ($T \leq 2000$)
- He starts in a tree at height H ($H \leq 2000$)
- At each step, he can either stay in the same tree and go down one meter, or go to another tree and plunge f meters ($f \leq 500$).
- How many acorns can Jayjay collect?
- Solution:
  $$dp[h][t] = \max(dp[h-1][t], \max\{dp[h-f][i], i \in (1,T)\})$$

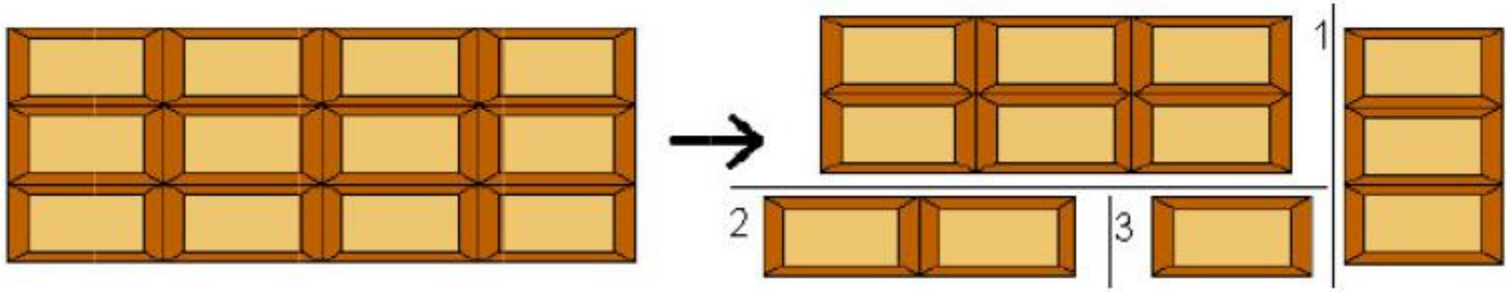


Reducing complexity

- Solution:
  $dp[h][t] = \max(dp[h-1][t],$
  $\max\{dp[h-f][i], i \in (1,T)\})$
- Complexity: $\#[states] = 2000 \times 2000 = 4M$
  $T(state) = \Theta(T) = 2000$
- $\#[states] \cdot T(state) \approx 8B$ ☹
- Notice that $\max\{dp[h-f][i], i \in (1,T)\}$ is the same for all trees, thus, compute it for each height (when possible) and use it.
- $T(state) = O(1)$
- $\#[states] \cdot T(state) \approx 4M$ ☺

Reducing complexity

- We are given a chocolate bar, of size $h \times w$, and the number cubes each person in a group of n people wants.
  - $h, w \leq 100, \mathrm{n} \leq 15$
- At each step, we want to cut a piece (either horizontally or vertically) and give it to a person in the group who wants this amount.
- For example, for a 3×4 bar, if 4 people want to get 6,3,2,1 respectively we can separate it as follows:



Reducing complexity

- Notice that after each cut, we remain with rectangular bar, and we ask if it can be divided for the remaining people in the group
- Solution: use DP over the states defined by
$$(h, w, bitmask)$$
where $bitmask$ represent the people who were given their piece.
- Complexity: $\mathrm{T}(state) = O(1), \#[states] = h \cdot w \cdot 2^n \approx 327M$ ☹
- We clearly have to reduce the number of states. How?
- Notice that at the beginning, the size of the bar, $h \cdot w$ is equal to the total requirements by the group.
- This is also true in each intermediate state, thus, given h and $bitmask$ we can deduce w, so we only need the state to be $(h, bitmask)$
- $\#[states] \approx 3.3M$ ☺

Using Fast Exponentiation

- Previously, we saw that by raising a matrix to a certain power we can find the value of $fib(n)$ in $O(\log n)$ time
- Lets derive how we get to this trick
- From now on $f(n) = fib(n)$
- Define the vector $v_n = \ [f(n), f(n-1)]$
- We look for a matrix $A = \begin{bmatrix} a\ b \\ c\ d \end{bmatrix}$, s.t $Av_n = v_{n+1}$
- It is easy to see that , $A^{n-1}V_1 = V_n$
- How do we find the matrix A?

Using Fast Exponentiation

- $A = \begin{bmatrix} a\ b \\ c\ d \end{bmatrix}, v_n = [f(n), f(n-1)], Av_n = v_{n+1}$
- We want:

$$\begin{bmatrix} a\ b \\ c\ d \end{bmatrix} \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix}$$

$$\Downarrow$$

$$af(n) + bf(n-1) = f(n+1)$$
$$cf(n) + df(n-1) = f(n)$$

$$\Downarrow$$

$$A = \begin{bmatrix} 1\ 1 \\ 1\ 0 \end{bmatrix}$$

Using Fast Exponentiation

- Let's do another one.
- Given $p = a + b, q = a \cdot b$, compute $a^n + b^n$ for a given (large) n.
- First, we find a recurrence, notice that:

$$(a + b) \cdot (a^{n-1} + b^{n-1}) = a^n + b^n + (a \cdot b)\,(a^{n-2} + b^{n-2})$$

- Denote $f(\mathrm{n}) = \ a^n + b^n$, and rearrange:

$$f(n) = pf(n-1) - qf(n-2)$$

Using Fast Exponentiation

- $f(n) = pf(n-1) - qf(n-2)$

$$\begin{bmatrix} a\ b \\ c\ d \end{bmatrix} \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix}$$

$$\Downarrow$$

$$af(n) + bf(n-1) = f(n+1)$$
$$cf(n) + df(n-1) = f(n)$$

$$\Downarrow$$

$$A = \begin{bmatrix} p & \text{-}q \\ 1 & 0 \end{bmatrix}$$

Misc.

- Grading
- Course competition
- Regional competition