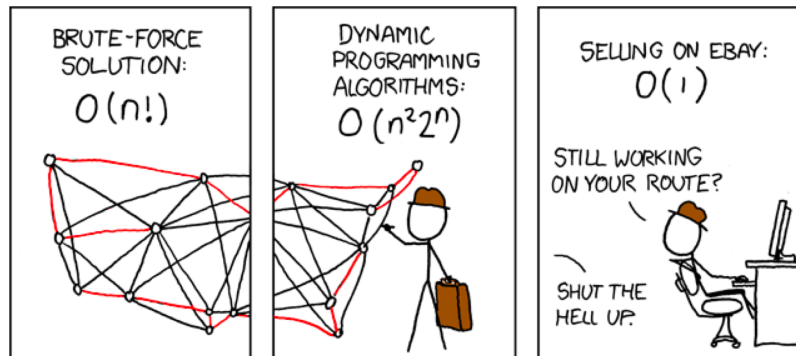


Algorithms: Dynamic Programming



Dennis J. Zhang
Washington University in St. Louis

Overview – Dynamic Programming

- **Dynamic Programming** is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping sub-instances.
- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS

Overview – Dynamic Programming

- **Dynamic Programming** is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping sub-instances.
- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS
- **Main idea:**
 - set up a recurrence relating a solution to a larger instance to solutions of some smaller instances
 - solve smaller instances once
 - record solutions in a table
 - extract solution to the initial instance from that table

An Example -- Fibonacci numbers

- Think about the Fibonacci number problem in Session 3:
- Let us change the problem to
 - “Giving an $n > 1$, return the n th Fibonacci number”

Generate the first N numbers of the Fibonacci sequence where

$$a_0 = 1$$

$$a_1 = 1$$

$$a_n = a_{n-1} + a_{n-2}$$

An Example -- Fibonacci numbers

- Think about the Fibonacci number problem in Session 3:
- Let us change the problem to
 - “Giving an $n > 1$, return the n th Fibonacci number”

Generate the first N numbers of the Fibonacci sequence where

$$a_0 = 1$$

$$a_1 = 1$$

$$a_n = a_{n-1} + a_{n-2}$$

```
: #Write your code here
N= 10

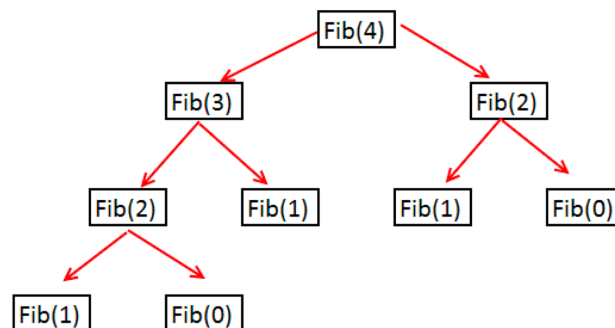
fibonacci=[1,1]

for i in range(N-2):
    nextTerm = fibonacci[-1] + fibonacci[-2]
    fibonacci+= [nextTerm]

fibonacci
```

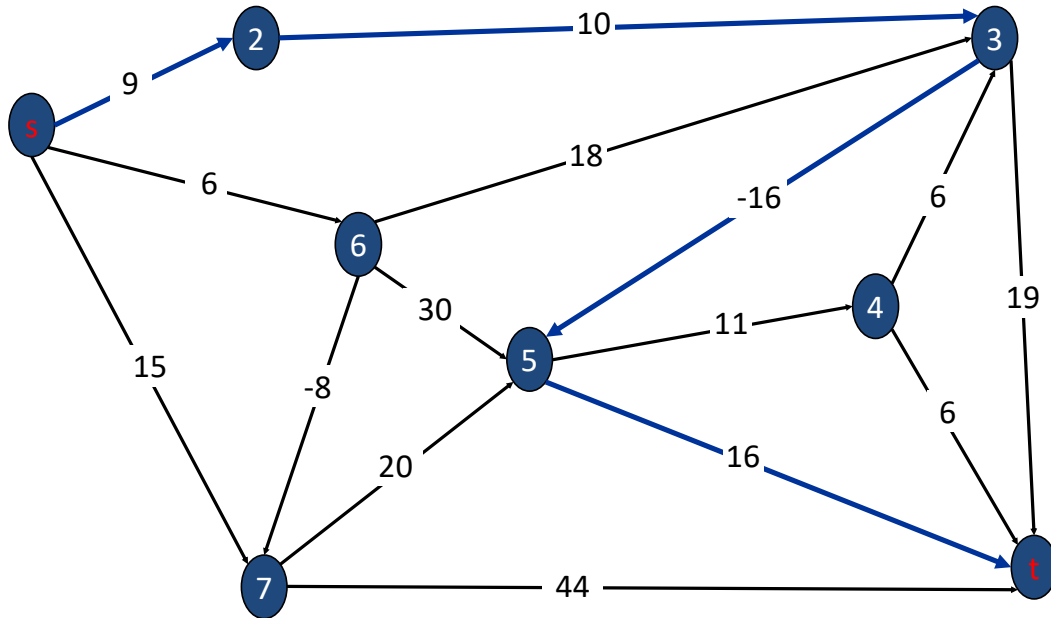
An Example -- Fibonacci numbers

- Why is this dynamic programming?
- 1. Divide the problem into sub-problems:
 - We divide Fib problem of N into sub-problems of N-1, N-2, N-3, ..., 1
- 2. Represent the larger instance by smaller instances:
 - $\text{Fib}(N) = \text{Fib}(N-1) + \text{Fib}(N-2)$
- 3. Solve the smaller cases first:
 - $\text{Fib}(0) = 1, \text{Fib}(1) = 1$
- 4. Record solution in a list / list of lists, solve large problems using smaller instances.



Another Example -- Shortest Path

- **Shortest path problem.** Given a directed graph $G = (V, E)$, with edge weights $c(v,w)$, find shortest path from node s to node t .



Another Example -- Shortest Path

- Def. $OPT(i, v)$ = length of shortest v - t path P using at most i edges.
- Case 1: **P uses at most $i-1$ edges.**
 - $OPT(i, v) = OPT(i-1, v)$
- Case 2: **P uses exactly i edges.**
 - if (v, w) is first edge, then OPT uses (v, w) , and then selects best w - t path using at most $i-1$ edges

$$OPT(i, v) = \begin{cases} 0 & \text{if } v = t, \infty \text{ otherwise} \\ \min \left\{ OPT(i-1, v), \min_{(v,w) \in E} \{ OPT(i-1, w) + c_{vw} \} \right\} & \text{otherwise} \end{cases} \quad \begin{matrix} \text{if } i = 0 \\ \text{otherwise} \end{matrix}$$

Another Example -- Shortest Path

```
Shortest-Path(G, t) {  
  foreach node v ∈ V  
    M[0, v] ← ∞  
  M[0, t] ← 0  
  
  for i = 1 to n-1  
    foreach node v ∈ V  
      M[i, v] ← M[i-1, v]  
    foreach edge (v, w) ∈ E  
      M[i, w] ← min { M[i, w], M[i-1, v] + cvw }  
}
```

Another Example -- Knapsack

- **Given a fixed Knapsack size, our midterm problem can be solved optimally with dynamic programming.**
- Here are some hints on how to think about the Knapsack problem as DP:
 - Define $V(i, j)$ as the optimal value of knapsack problem for the first i items with capacity j .
 - i is an integer between 0 and 50 in our case.
 - j can be treated as an integer between 100 and 150, in our case.

Another Example -- Knapsack

- **Given a fixed Knapsack size, our midterm problem can be solved optimally with dynamic programming.**
- Here are some hints on how to think about the Knapsack problem as DP:
 - Define $V(i, j)$ as the optimal value of knapsack problem for the first i items with capacity j .
 - i is an integer between 0 and 50 in our case.
 - j can be treated as an integer between 100 and 150, in our case.
 - $V(0, j) = 0$ for all j , $V(i, 0) = 0$
 - Suppose the first item has weight 10 and value 5,
 - Then $v(1, j) = 0$ if $j < 10$ and 5 if $j \geq 10$

Another Example -- Knapsack

- **Given a fixed Knapsack size, our midterm problem can be solved optimally with dynamic programming.**
- Here are some hints on how to think about the Knapsack problem as DP:
 - Define $V(i, j)$ as the optimal value of knapsack problem for the first i items with capacity j .
 - i is an integer between 0 and 50 in our case.
 - j can be treated as an integer between 100 and 150, in our case.
 - $V(0, j) = 0$ for all j , $V(i, 0) = 0$
 - Suppose the first item has weight 10 and value 5,
 - Then $V(1, j) = 0$ if $j < 10$ and 5 if $j \geq 10$
 - Question 1: how do you represent $v(i, j)$ as smaller instances (i.e., instances with smaller i and j)
 - Question 2: What information you should store along the way so that you can solve $V(50, 100)$ (assuming that knapsack size is fixed at 100)?