8 Dynamic Programming

Thursday, May 1, 2025 1:34 PM

by Richard Bellman

"Dynamic Programming"

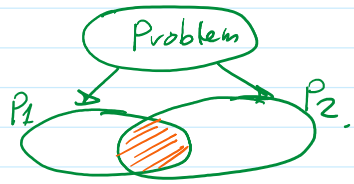
"nothing called "Dynamic" is ever bad"

~~Schedulling.~~

{ optimizing logistic schedules.

- An optimization technique for recursive solution implementation.

- What are we optimizing?
  - Many problems have recurrences
  - Many recurrences have overlapping subproblems

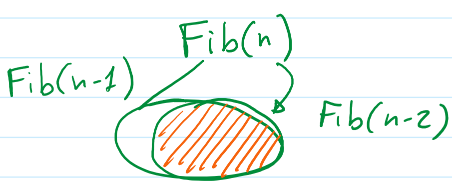


Dynamic Programming: (the cartoon version)

- solve small subproblems only once and store their solutions.

E.g Q: Fibonacci

- $Fib(n) = Fib(n-1) + Fib(n-2)$



Apply Dynamic Programming:

	0	1	2	3	4	5	6	......	☆
fib	0	1	1	2	3	5	8	13	○

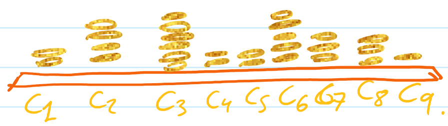
FOR $i=2$ TO n DO
$fib[i] = fib[i-1] + fib[i-2]$ $\in \Theta(n)$

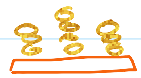
Note: not all Dynamic Programming Problems end up using a table Sometimes the table collapses.

- Dynamic Programming comonly used in optimization problems
  - "Principle of Optimality."

"An optimal solution to an optimization problem can be composed from the optimal solutions to it's subproblems"

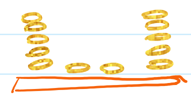
E.g Problem #1 "coins-in-the-table-problem"





. Pick lagest one.

- Pick up any collection of coin piles
- But, you cannot pick two adjacent piles

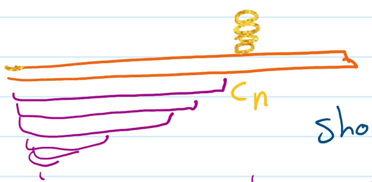


Goal: Maximize the coins you pick.

pick even/odd rows.

Brute Force: 2^n candidate solutions.

- Step 1: Formulate optimal solution as a recurrance. i.e. in terms of optimal solutions to smaller instances.



should you take pile C_n.

assume somebody else has solved the problem for piles $1 \ldots n-1$

$F(n)$: optimal solution for n piles

$$F(n) \begin{cases} C_n + F(n-2) & \text{pick pile } C_n \\ \text{or} & \\ F(n-1) & \text{reject pile } C_n \end{cases}$$

- $F(n) = \max\left(C_n + F(n-2),\ F(n-1)\right)$ } recurrence

Base case.

$F(0) = 0$
$F(1) = C_1$ } Base Case

- Step 2 Build a table of solutions from smaller to larger.

```
FUNCTION coin_row_select ( c[1..n] )
  VAR F[0...n]

  F[0] ← 0
  F[1] ← C[1]
  FOR i ← 2 TO n
      F[i] ← max ( C[i] + F[i-2], F[i-1] )

  RETURN F[n]
```

Trace:

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
C:	2	4	6	2	2	5	3	3	1

	0	1	2	3	4	5	6	7	8	9
F:	0	2	4	8	8	10	13	13	16	16
		✓	✓	✓	–	✓	✓	=	✓	–
		Δ		Δ			Δ		Δ	

Analysis:
Building table is $\in \Theta(n)$

Problem #2 Change Making Problem.

In cash: (american) 47¢ = quarter + dime + dime + penny + penny.

- given a quantity, and a supply of coins of different denominations.

d_1	d_2	d_3	d_4	d_5	d_6

complete the quantity using the least number of coins.

- Step 1
  $F(n)$: optimal solution for quantity n

$$F(n) \begin{cases} \cdot\, d_1 + F(n-d_1) \\ \cdot\, d_2 + F(n-d_2) \\ \cdot\, d_3 + F(n-d_3) \\ \vdots \end{cases}$$

one of these solution is the optimal

note $n - d_i$ cannot be negative.

$$F(n) = \left[\begin{array}{l} \cdot\, d_3 + F(n-d_3) \\ \vdots \\ \cdot\, d_k + F(n-d_k) \end{array}\right]$$

note $n-d_i$ cannot be negative.

- $F(n) = \min\limits_{j}\{F(n-d_j)\} + 1$ s.t. $n \geq d_j$

- $F(0) = 0$

- Step 2:

```
FUNCTION Make_Change ( D[1...m], n )      <- coins in increasing order.
  VAR F[0..n]
  F[0] <- 0
  FOR i <- 1 to m
     temp <- ∞
     j <- 1
     WHILE j ≤ m AND i ≥ D[j]
         temp <- min( F[i - D[j]], temp )
         j <- j+1
     F[i] <- temp+1
  RETURN F[N]
```

Trace:

$D = \{1, 5, 6, 9\}$ $n = 11$

9 + 1 + 1

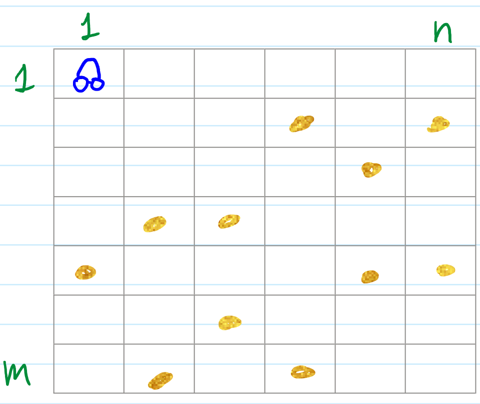
5 + 6

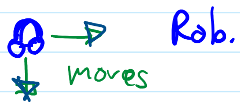
	0	1	2	3	4	5	6	7	8	9	10	11
F	0	1	2	3	4	1	1	2	3	1	2	2

Analysis:

worst case is $\Theta(n \cdot m)$

Problem #3 collecting coins in the grid world.





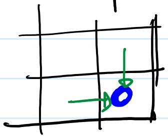
Rob's goal,
collect the maximum number of coins.

- Brute Force:
  consider all possible paths:
  - all paths consist of 5 moves to the right and 6 moves down.

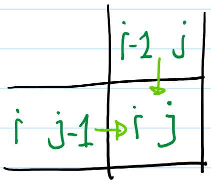
$\rightarrow \binom{11}{5}, \; C^{11}_5 = 462$

- Dynamic Programming.

Step 1:- represent solution in terms of smaller solutions.



$F(i,j)$: max coins collected ending in location i,j



$$F(i,j) \text{ or } \begin{cases} F(i-1,j) + C(i,j) \\ \text{max} \\ F(i,j-1) + C(i,j) \end{cases}$$

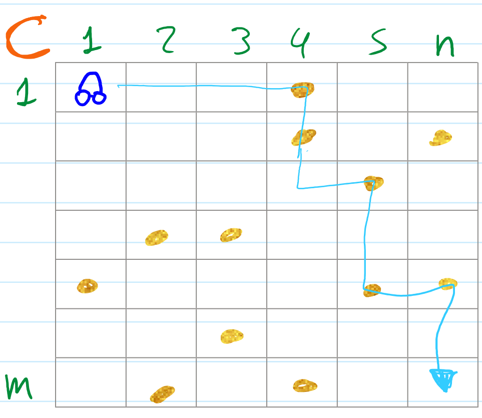
e.i.

$$F(i,j) = \max\big(F(i-1,j),\ F(i,j-1)\big) + C(i,j)$$
$$F(1,1) = 0$$

Step.2:- Build a table of solutions.

```
FUNCTION RobAdventure(C[1..n, 1..m])
VAR F[0..n, 0..m] initialized to zeroes
  F[1,1] ← 0
  FOR i ← 1 to n
    FOR j ← 1 to m
      F[i,j] = max(F[i-1,j], F(i,j-1))
               + C[i,j]
  RETURN F[n,m]
```

Trace:

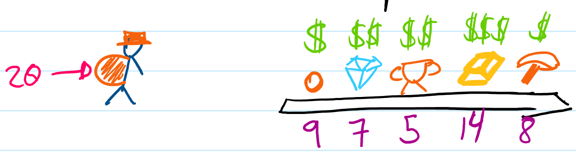


	0	1	2	3	4	5	n
0	0	0	0	0	0	0	0
1	0	0 →	0 →	0 →	1 →	1 →	1
2	0	0 →	0 →	0	2 →	2 →	3
	0	0 →	0 →	0	2 →	3 →	3
	0	0 →	1 →	2 →	2	3	3
	0	1 →	1	2 →	2	4 →	5
	0	1 →	1	3 →	3	4	5
m	0	1 →	2	3 →	4 →	4	5

Analysis:

Filling table is $\Theta(n \cdot m)$

Problem #4: KnapSack.



Given a collection of items $e_1 \quad e_2 \quad e_3 \quad e_4 \ \dots \ e_n$

each with a weight $w_1 \quad w_2 \quad w_3 \quad w_4 \ \dots \ w_n$

and a value $v_1 \quad v_2 \quad v_3 \quad v_4 \ \dots \ v_n$

and a "capacity" W

- find a subset of the items s.t the sum of the weights is less than W that maximizes value.

- Brute Force: check 2^n subsets
- Dynamic Programming

Step 1:- Formulate Solution in terms of solutions to Smaller problems.

$F(i,j)$: optimal solution for first i items, and a bag of capacity j

$F(i,j)$:
- carry i'th item $F(i-1, j-\underset{\text{weight of } i}{w_i}) + \underset{\text{value of } i}{v_i}$
- don't carry i'th item. $F(i-1, j)$

$$F(i,j) = \text{Max}\left(F(i-1, j-w_i) + v_i \ , \ F(i-1, j)\right)$$

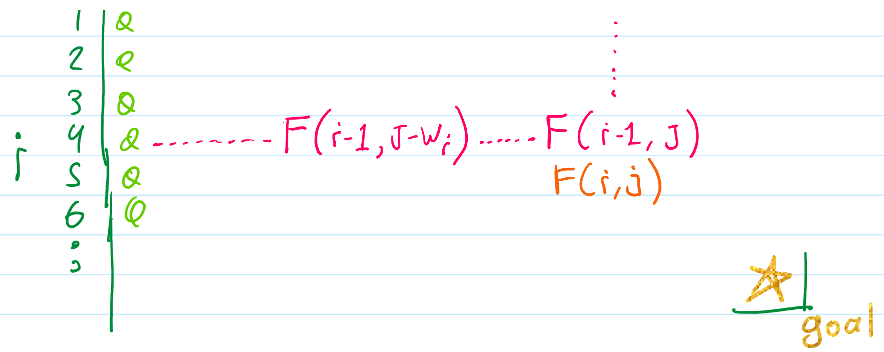
$$F(x, 0) = 0$$

$$F(0, y) = 0$$

Step-2 build a table of solutions:

Note:

	0	1	2	3	4	5	6	....		W
0	0	0	0	0	0	0	0	0	0	0
1	0									
2	0									
3	0									



FUNCTION DPKnapsack (w[1..n], v[1..n], W)
VAR F[0..n, 0..W] initialized to zeros.

```
FOR i ← 1 to n
    FOR j ← 1 TO W
        IF j - W[i] < 0 THEN
            F[i,j] ← F[i-1, j]
        ELSE
            F[i,j] ← Max { F[i-1, j]
                           F[i-1, j-W[i]] + V[i]
RETURN F[n,W]
```

Analysis

Fillin the table is $\Theta(n \cdot W)$ lot better than 2^n