Dynamic Programming

Identify DP

- Must have:
  - Optimal substructure
  - Overlaping subproblems
- Top down - Recursion + Memoization
- Bot Up - Tabulation

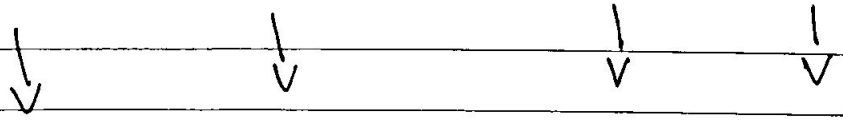
Def - Optimal substructure means that optimal solutions for each subproblem results in an optimal solution for the problem.

- Can be either:
  - Combinatorial (e.g. How many?)
  - Optimization (e.g. Maximize or Minimize)
- DP typically allows for at least polynomial solutions:
  - Such as $O(n^2)$ or $O(n^7)$
  - Not exponential $O(2^n)$ or $O(7^n)$

Steps to solve DP

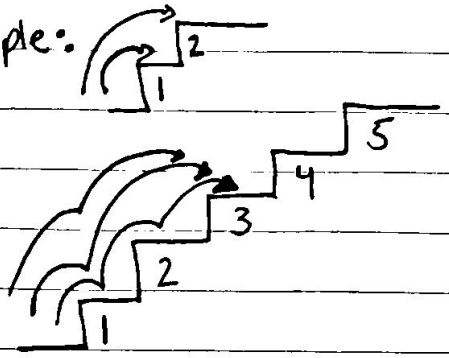
The Framework:

1. Define objective function. (e.g. f(i) = the number of ways to reach i)
2. Identify base cases.
3. Write down transition function. (e.g. $f(n) = f(n-1) + f(n-2)$)
4. Identify order of execution.
5. Identify location of the answer.



Climbing Stairs How many ways can we reach step i? We can either take 1 or 2 steps at a time.

Example:



f(3): (1,1,1)
(1,2)
(2,1)

Framework:

Solution

1. f(n) is the number of distinct possible ways to reach stair n.
2. f(0) = 0, f(1) = 1
3. f(n) = f(n-1) + f(n-2)
4. Bottom up approach
5. f(n)

Optimal Path What is the lowest cost to reach step i? We can either take 1 or 2 steps at a time.

index →	0	1	2	3	4	5
cost →	0	3	2	4	1	6

f(0) = 0
f(1) = 0+3 = 3
f(2) = 0+2 = 2
f(3) = 2+4 = 6
f(4) = 2+1 = 3
f(5) = 3+6 = 9

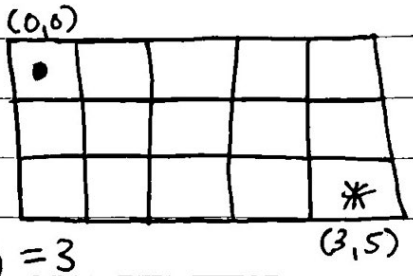
Framework:

Solution

1. f(n) is the lowest cost to reach step n.
2. f(0) = 0, f(1) = 3
3. f(n) = price(n) + min(f(n-1), f(n-2))
4. Bottom up

Unique Paths Given an m x n grid, how many unique ways can we reach a given cell i? You can move either down or right.

$m \times n = 3 \times 5$



$f(1,j) = 1 \quad f(3,2) = 3$
$f(i,1) = 1 \quad f(3,3) = 6$
$f(1,1) = 1$
$f(2,2) = 2$

Framework:

solution

1. $f(i,j)$ is the total possible ways to reach cell (i,j)
2. $f(1,j) = 1, f(i,1) = 1, f(1,1) = 1, f(2,2) = 2$
3. $f(i,j) = f(i-1,j) + f(i,j-1)$

Optimal Path Given an m x n grid, What is the optimal path to a given cell? You can move either down or right.

3	1	6	9	4
8	2	1	4	5
1	1	4	3	2

1. $f(i,j)$ is the cost of the optimal path to cell (i,j).
2. $f(1,1) = 3, f(1,2) = 4, f(2,1) = 8+3 = 11$
3. $f(i,j) = cost(i,j) + \min(f(i-1,j), f(i,j-1))$

(1,1)

3(1,1)	4(1,1)	10(1,2)	19(1,3)	23(1,4)
11(1,1)	6(1,2)	7(2,2)	11(2,3)	16(2,4)
12(2,1)	7(2,2)	11(2,3)	14(2,4)	16(3,4)

(3,5)

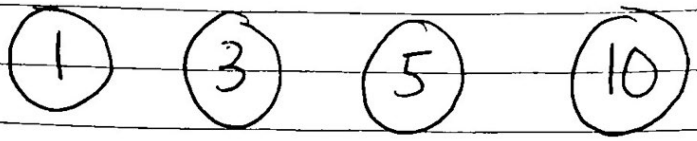
(3,5), (3,4), (2,4), (2,3), (2,2), (1,2), (1,1)

2 ← 3 ← 4 ← 1 ← 2 ← 1 ←

Coins

Given coins of different values and a total amount of money find the number of combinations that can make up that amount.

- Combinatoric



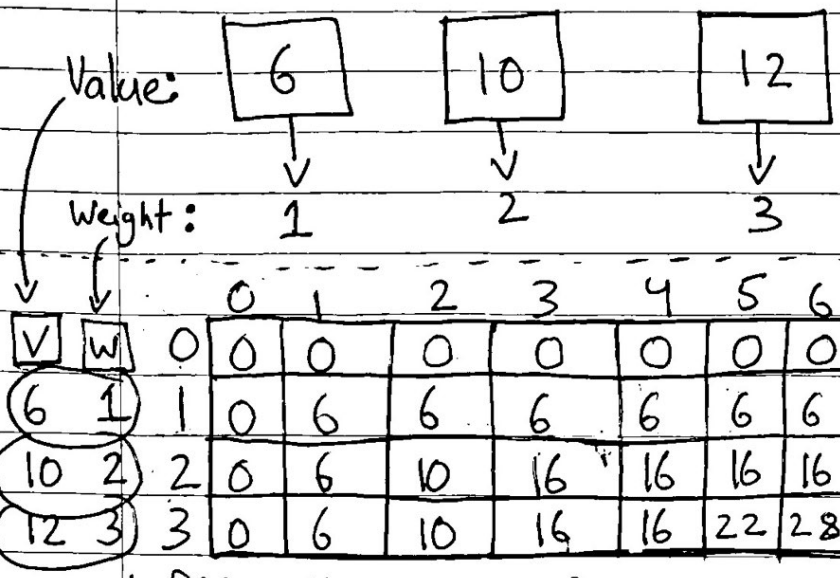
f(0) = 1 f(4) = 2 + 1.
f(1) = 1 f(5) = 3 + 1 + 1 = 5
f(2) = 1 f(6) =
f(3) = 2

Framework:

1. f(i) is the total number of combinations possible for amount i.
2. f(0) = 1 f(1) = 1
3. f(n) = f(n-1) + f(n-3) + f(n-5) + f(n-10)

Solution

0/1 Knapsack Given a set of items, each with: weight and value, and a max weight the knapsack can carry. Determine the max value that can be put without exceeding weight.



V	W	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0
6	1	0	6	6	6	6	6	6	6	6
10	2	0	6	10	16	16	16	16	16	16
12	3	0	6	10	16	16	22	28	28	28

1. ~~f(i) is the max value for a knapsack that can hold weight i.~~
2. (...)

Solution

3. $f(3,1) = \max(f(2,1), f(2,1-3)+12)$

$$f(i,x) = \max(f(i-1,x), f(i-1, x-x(i))+val(i))$$

$$f(2,3) = \max(f(1,3)=6, (f(1,3-2)+10)=16)$$
$$= 16$$

- Seems like we are dealing with 3 values to care about not just the knapsack weight. Item = (val, weight) knapsack = capacity.

(val, weight) capacity

items = [(60,3) A , (20,1) B , (40,2) C , (70,4) D , (15,1) E]

max carrying capacity = 5

Knapsack capacity

items	0	1	2	3	4	5
(v, w) 0	0	0	0	0	0	0
(60,3)	0	0	0	60	60	60
(20,1)	0	20	20	60	80	80
(40,2)	0	20	40	60	80	100
(70,4)	0	20	40	60	80	100
(15,1)	0	20	40	60	80	100

* $f(i,x) = \max(f(i-1,x), f(i-1, x-x(i)) + val(i))$
- $f(1,1) = \max(f(0,1), f(0,1-3)+60) = 0$
- $f(1,2) = \max(f(0,2), f(0,2-3)+60) = 0$
- $f(1,3) = \max(f(0,2), f(0,3-3)+60) = 60$
- $f(1,4) = 60$
- $f(1,5) = 60$
- $f(2,1) = \max(f(1,1), f(1,1-1)+20) = 20$
- $f(2,2) = 20$

A
- $f(2,3) = \max(f(1,3), f(1,3-1)+20) = 60$
- $f(3,4) = \max(f(2,4), f(2,4-2)+40) = 80$

C
- $f(3,5) = \max(f(2,5), f(2,5-2)+40) = 100$

Answer: include items A & C for value 100

Longest Common Subsequence

Given two sequences, find the length of the longest subsequence present in both. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous (as opposed to a substring) in both sequences.

str1 = "A B C D G H"
str2 = "A E D F H R"

V		A	B	C	D	G	H
	0	0	0	0	0	0	0
A	0	1	1	1	1	1	1
E	0	1	1	1	1	1	1
D	0	1	1	1	2	2	2
F	0	1	1	1	2	2	2
H	0	1	1	1	2	2	3
R	0	1	1	1	2	2	3

solution

```
if str1[i] == str2[j]:
    V(i,j) = 1 + V(i-1, j-1)
elif str1[i] != str2[j]:
    V(i,j) = max(V(i-1,j), V(i,j-1))
```

		S	T	O	N	E
	0	0	0	0	0	0
L	0	0	0	0	0	0
O	0	0	0	1	1	0
N	0	0	0	1	2	2
G	0	0	0	1	2	2
E	0	0	0	1	2	3
R	0	0	0	1	2	3

"O, N, E"

Min Path Given mxn grid find cheapest way from top left to bottom right only moving down or left.

1	3	1
1	5	1
4	2	1

1	4	5
2	7	6
6	8	7

Path = (1, 3, 1, 1, 1) = 7

[(1,1), (1,2), (1,3), (2,3), (3,3)]

$$f(i,j) = \min(f(i-1,j), f(i,j-1)) + Val(i,j)$$

Edit Distance Given two strings str1, str2, you are allowed to preform the following three operations on str1:

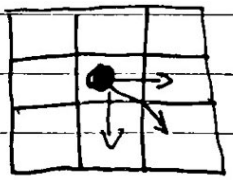
1. insert a char 2. remove a char 3. Replace a char

Find the minimum of operation to convert str1 -> str2.

		H	O	R	S	E
	R	3	3	2	3	3
	O	3	2	2	2	2
	S	4	3	2	1	1
		5	4	3	2	1	0

Solution:

```
if w1[i] = w2[j]:
    (i+1, j+1)
else:
    insert: (i, j+1)
    delete: (i+1, j)
    replace: (i+1, j+1)
```



Fibonachi Given n give the sequence of Fibonachi numbers until index n.

[0, 1, 1, 2, 3, 5, 8, 13, ...]

1. f(i) is the fibonachi number for index i.
2. f(0) = 0, f(1) = 1
3. Recurrence Relation: f(i) = f(i-1) + f(i-2)

Bottom Up w/ memo

```
memo_table = {}
for i in range(n):
    if i == 0:
        memo_table[i] = 0
        continue
    elif i == 1:
        memo_table[i] = 1
        continue
    curr_fib = memo_table[i] = memo_table[i-1] + memo_table[i-2]
```

Topdown

```
def fib(n)
    if n == 0: return 0
    if n == 1: return 1
    curr_fib = fib(n-1) + fib(n-2)
    return curr_fib
```

Top down w/ Memoization

```
def m_f(n, mem):
    if n in mem: return mem[n]
    if n == 0:
        mem[0] = 0
        return 0
    if n == 1:
        mem[1] = 1
        return 1
    curr = m_f(n-1, mem) + m_f(n-2, mem)
    return curr
```