

FIT2004

Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Companez, Aamir Cheema, Arun Konagurthu and Lloyd Allison



Faculty of Information Technology, Monash University

COMMONWEALTH OF AUSTRALIA
Copyright Regulations 1969

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act. Do not remove this notice



Ready?

Agenda

- Dynamic Programming

Agenda

- Dynamic Programming
 - Brute force (aka the starting point)

Agenda

- Dynamic Programming
 - Brute force (aka the starting point)
 - Overlapping sub-problems
 - Backtracking

Agenda

- Dynamic Programming
 - Brute force (aka the starting point)
 - Overlapping sub-problems
 - Backtracking
 - Fibonacci
 - Coin change
 - Knapsack
 - Unbounded
 - 0/1
 - Edit distance

Agenda

- Dynamic Programming
 - Brute force (aka the starting point)
 - Overlapping sub-problems
 - Backtracking for **solution reconstruction**
 - Fibonacci
 - Coin change
 - Knapsack
 - Unbounded
 - 0/1
 - Edit distance



Let us begin...

Dynamic Programming

... a smarter way to brute force

Dynamic Programming

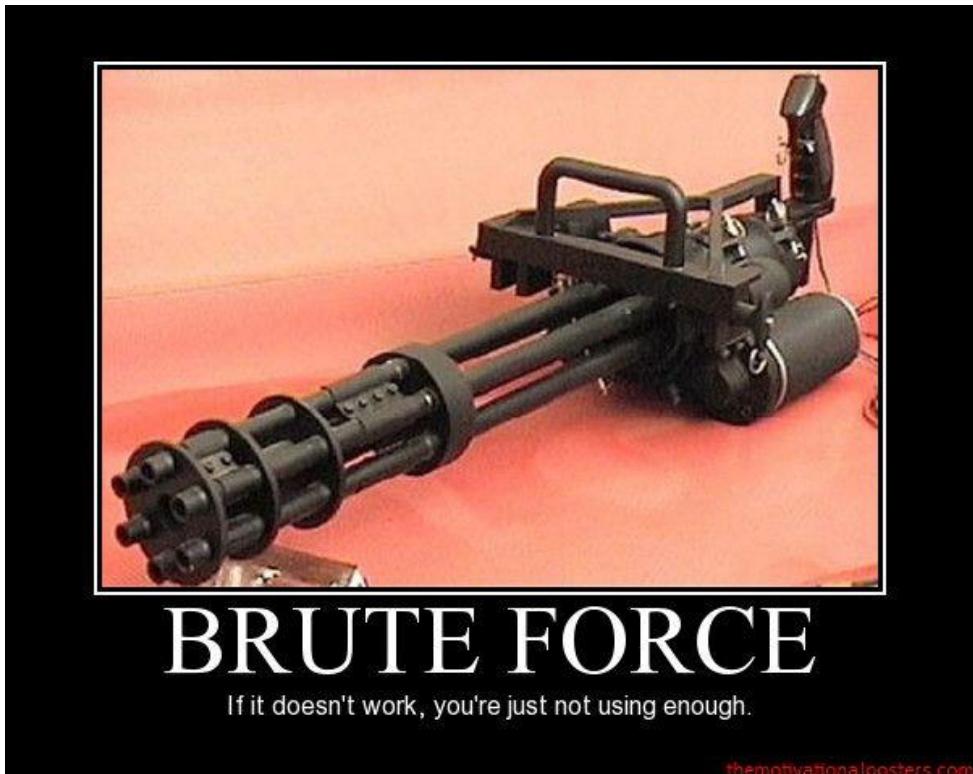
... a smarter way to brute force

- Can we brute force everything?

Dynamic Programming

... a smarter way to brute force

- Can we brute force everything?



Dynamic Programming

... a smarter way to brute force

- Can we brute force everything?
 - So if we can brute force anything, why not just use it?

Dynamic Programming

... a smarter way to brute force

- Can we brute force everything?
 - So if we can brute force anything, why not just use it?
 - Because brute forcing needs **too much effort**...

Dynamic Programming

... a smarter way to brute force

- Can we brute force everything?
 - So if we can brute force anything, why not just use it?
 - Because brute forcing needs **too much effort**...
- Consider the Fibonacci problem
 - Find the n-th Fibonacci number

Questions?

Fibonacci

n-th number in the series

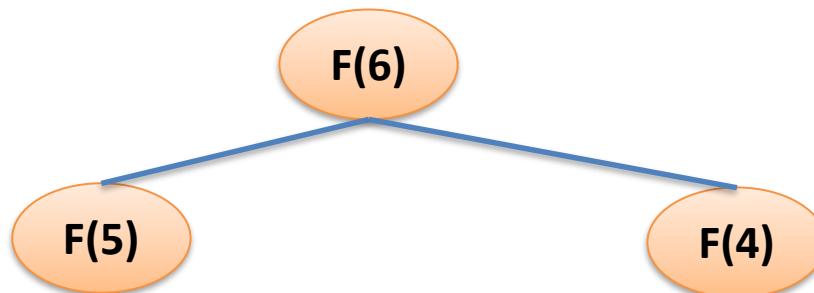
- Now if we want the 6th Fibonacci number...
 - How would we get it?

$F(6)$

Fibonacci

n-th number in the series

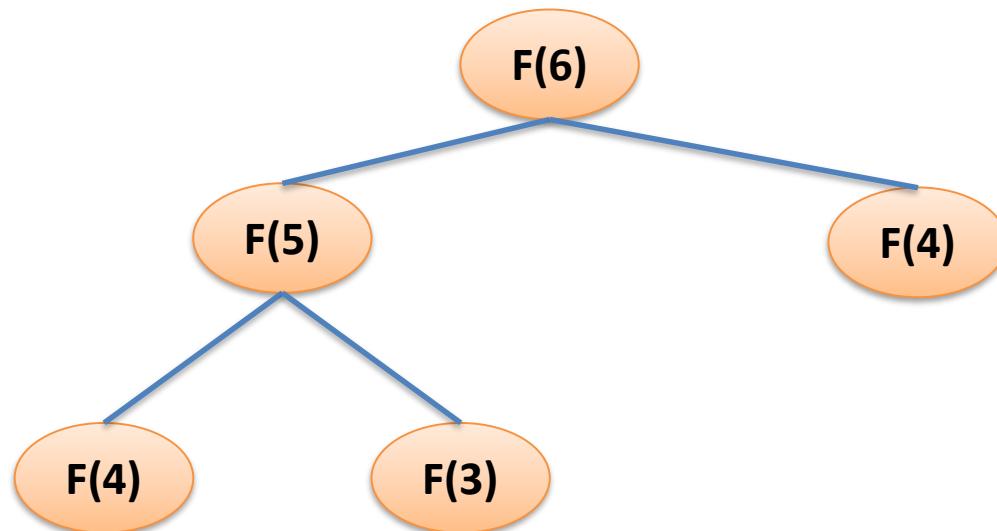
- Now if we want the 6th Fibonacci number...
 - How would we get it?



Fibonacci

n-th number in the series

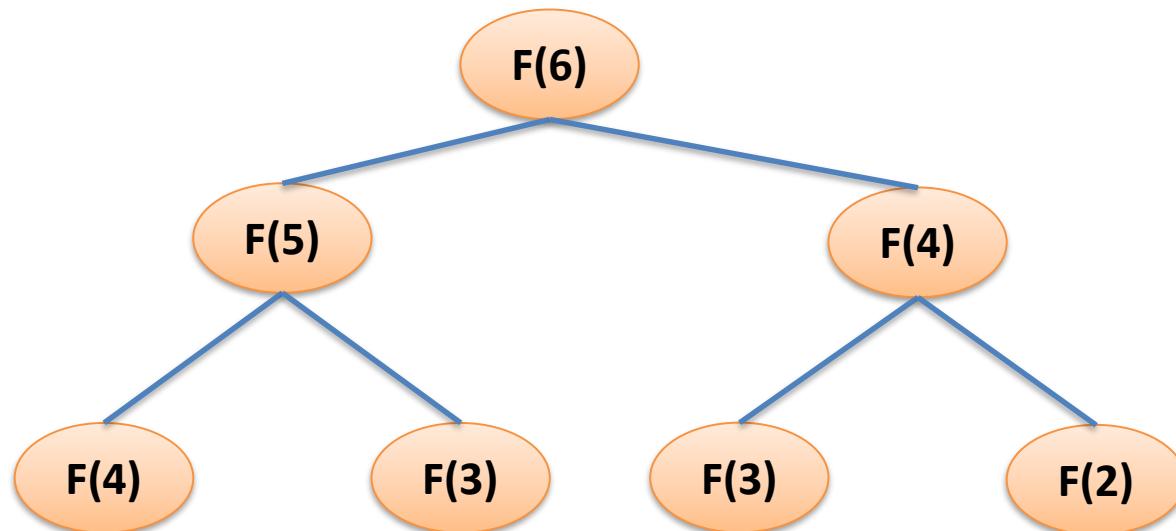
- Now if we want the 6th Fibonacci number...
 - How would we get it?



Fibonacci

n-th number in the series

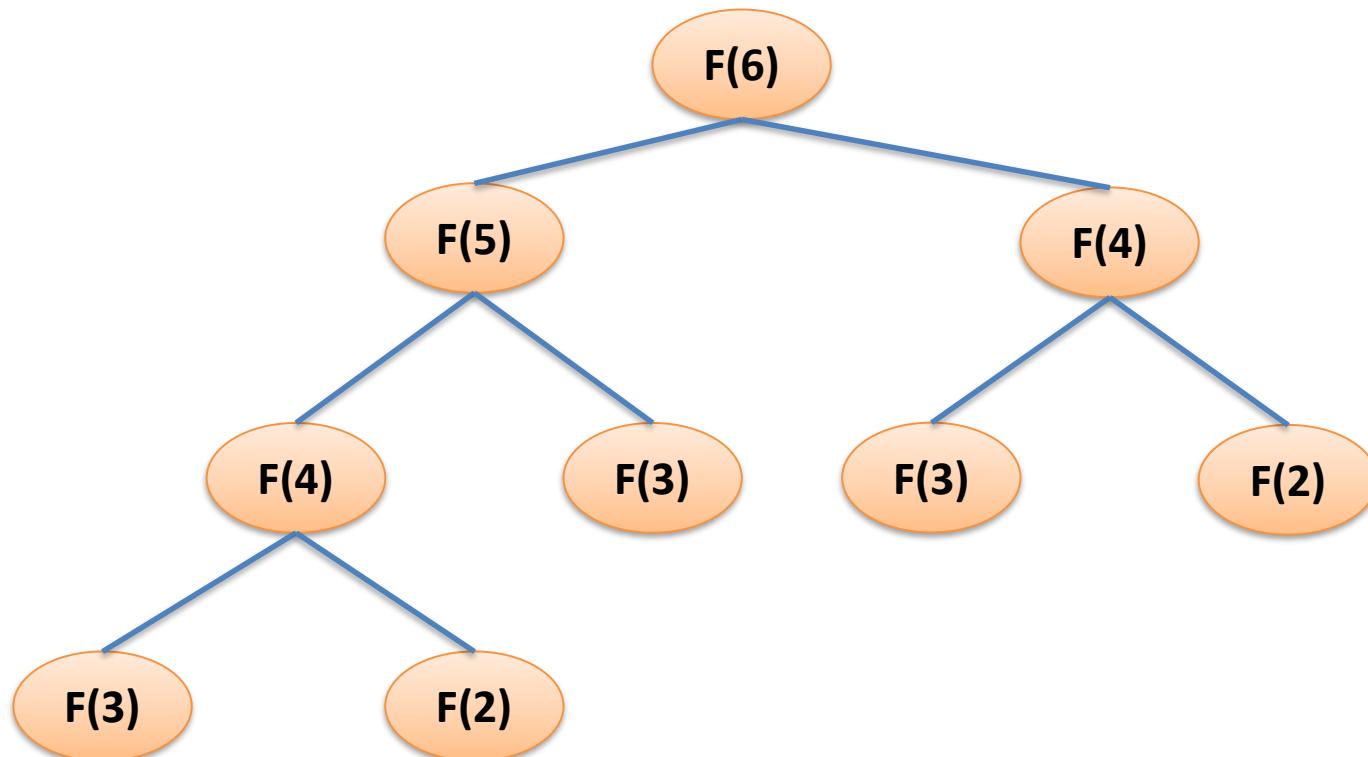
- Now if we want the 6th Fibonacci number...
 - How would we get it?



Fibonacci

n-th number in the series

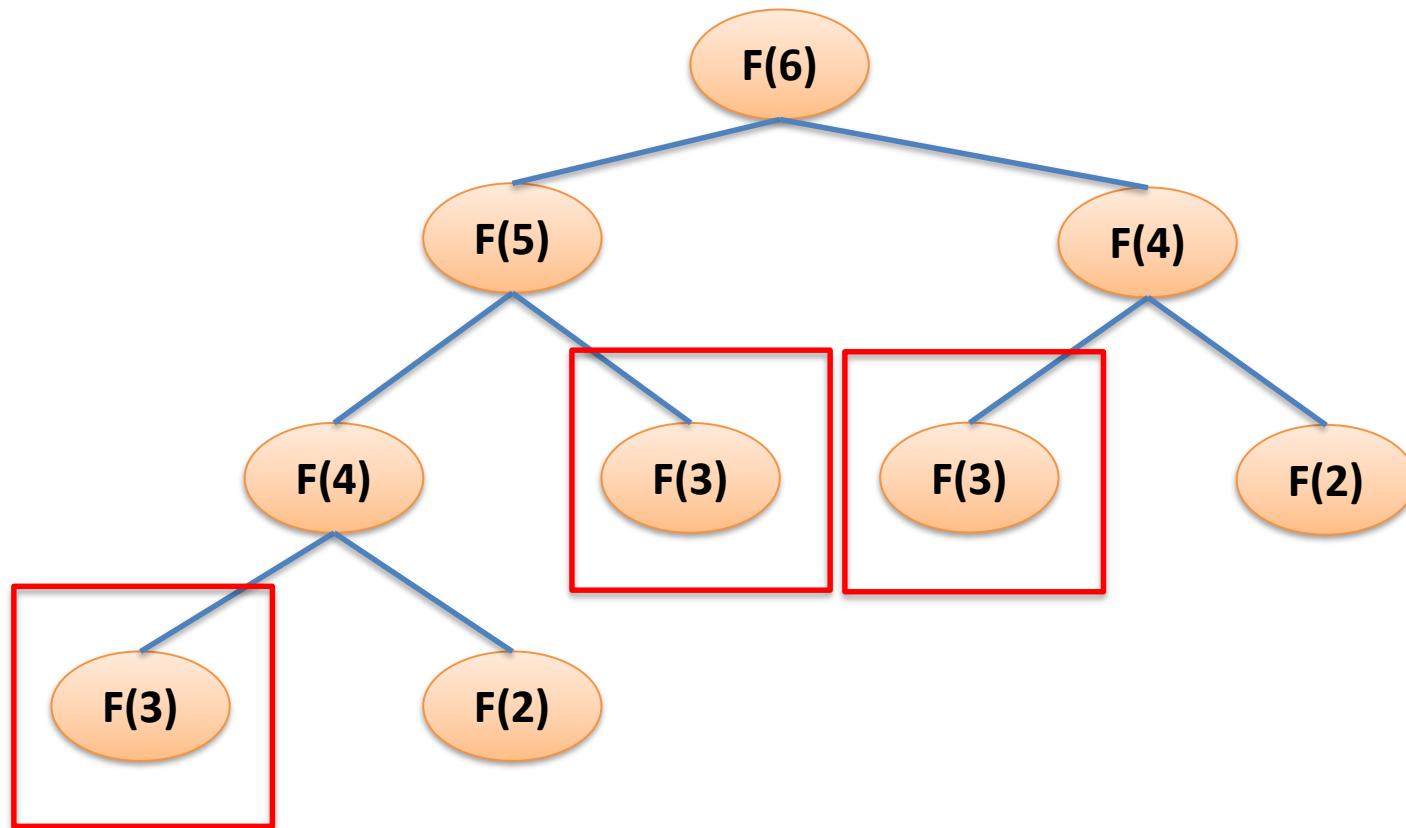
- Now if we want the 6th Fibonacci number...
 - How would we get it?



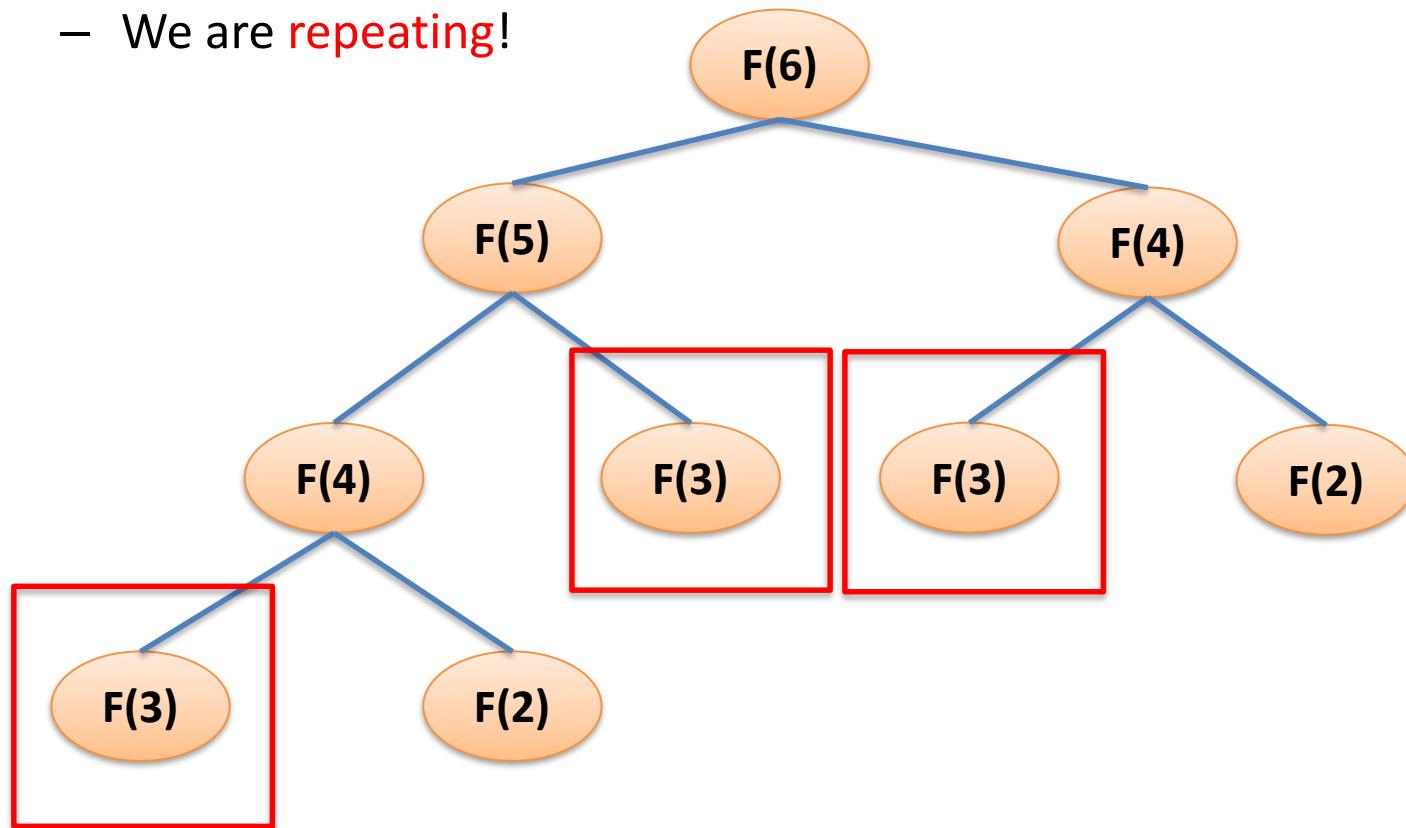
Fibonacci

n-th number in the series

- Now if we want the 6th Fibonacci number...
 - How would we get it?



- Now if we want the 6th Fibonacci number...
 - How would we get it?
 - We are **repeating!**



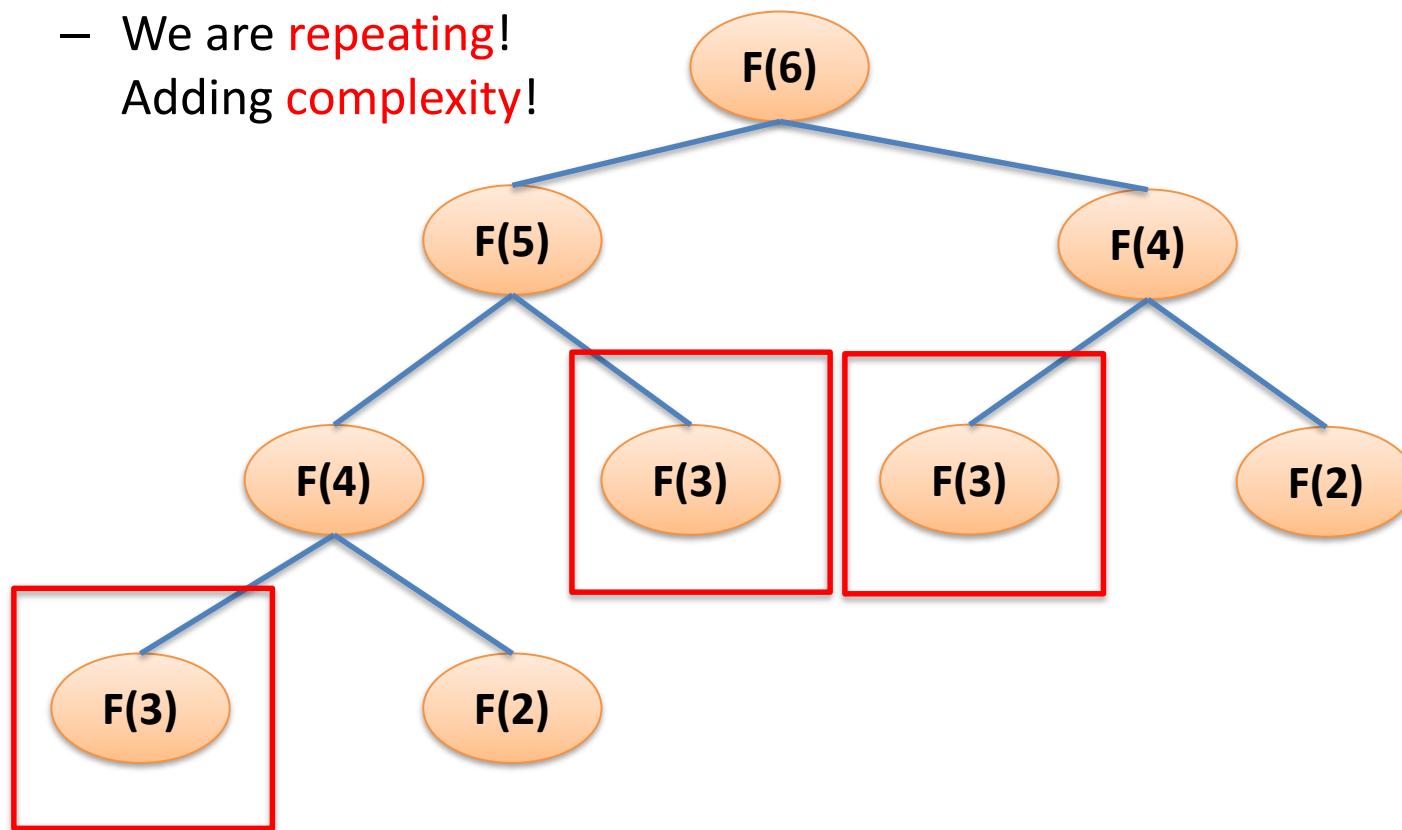
Fibonacci

n-th number in the series

- Now if we want the 6th Fibonacci number...

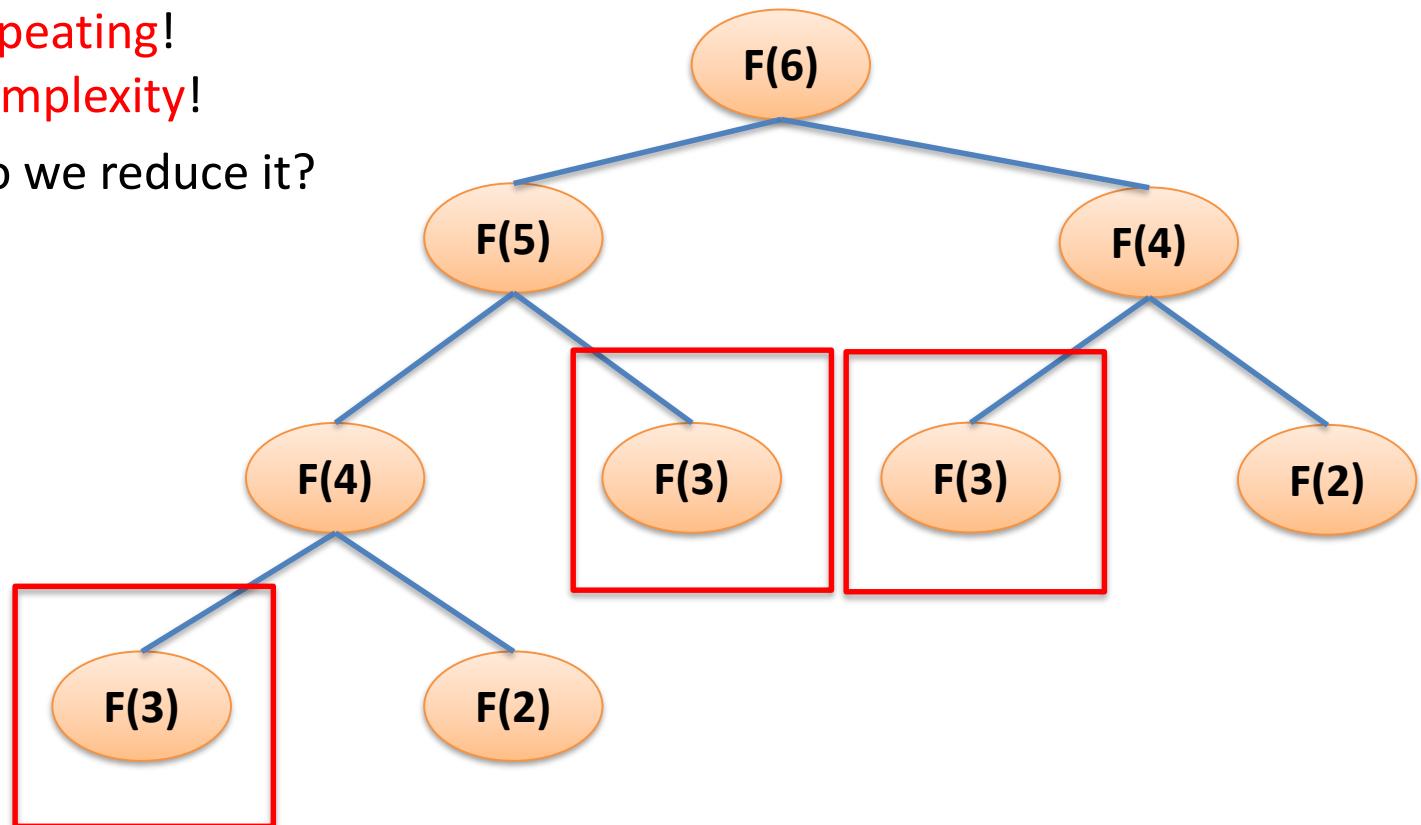
- How would we get it?

- We are **repeating!**
Adding **complexity!**



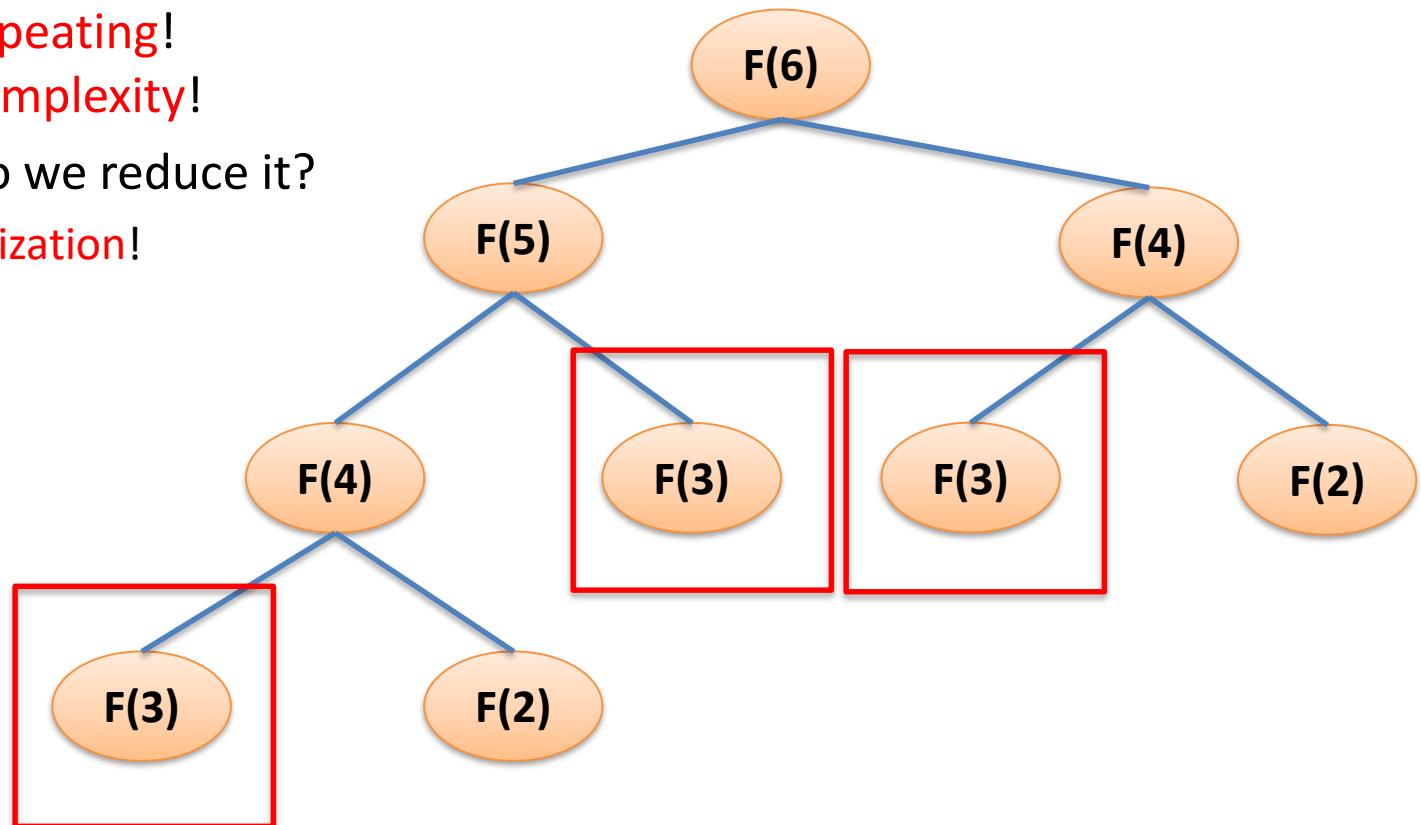
- Now if we want the 6th Fibonacci number...

- How would we get it?
- We are **repeating!**
Adding **complexity!**
- So how do we reduce it?



- Now if we want the 6th Fibonacci number...

- How would we get it?
- We are **repeating!**
Adding **complexity!**
- So how do we reduce it?
 - **Memoization!**



Questions?

- Now if we want the 6th Fibonacci number...
 - How would we get it?
 - We are **repeating!**
Adding **complexity!**
 - So how do we reduce it?
 - **Memoization!**
 - Quick real world examples...

- Now if we want the 6th Fibonacci number...
 - How would we get it?
 - We are **repeating!**
Adding **complexity!**
 - So how do we reduce it?
 - **Memoization!**
 - Quick real world examples...
 - What is 12x12?

- Now if we want the 6th Fibonacci number...
 - How would we get it?
 - We are **repeating!**
Adding **complexity!**
 - So how do we reduce it?
 - **Memoization!**
 - Quick real world examples...
 - What is 12x12?
 - What is 12x13?

- Now if we want the 6th Fibonacci number...
 - How would we get it?
 - We are **repeating!**
Adding **complexity!**
 - So how do we reduce it?
 - **Memoization!**
 - Quick real world examples...
 - What is 12x12?
 - What is 12x13?
 - What is 12x14?

- Now if we want the 6th Fibonacci number...
 - How would we get it?
 - We are **repeating!**
Adding **complexity!**
 - So how do we reduce it?
 - **Memoization!**
 - Quick real world examples...
 - What is 12x12? **Remember this**
 - What is 12x13? Used for this...
 - What is 12x14? Used for this...

- Now if we want the 6th Fibonacci number...
 - How would we get it?
 - We are **repeating!**
Adding **complexity!**
 - So how do we reduce it?
 - **Memoization!**
 - Quick real world examples...
 - What is 12x12? **Remember this**
 - What is 12x13? Used for this...
 - What is 12x14? Used for this...
 - Let's apply to Fibonacci

Questions?

Fibonacci

n-th number in the series

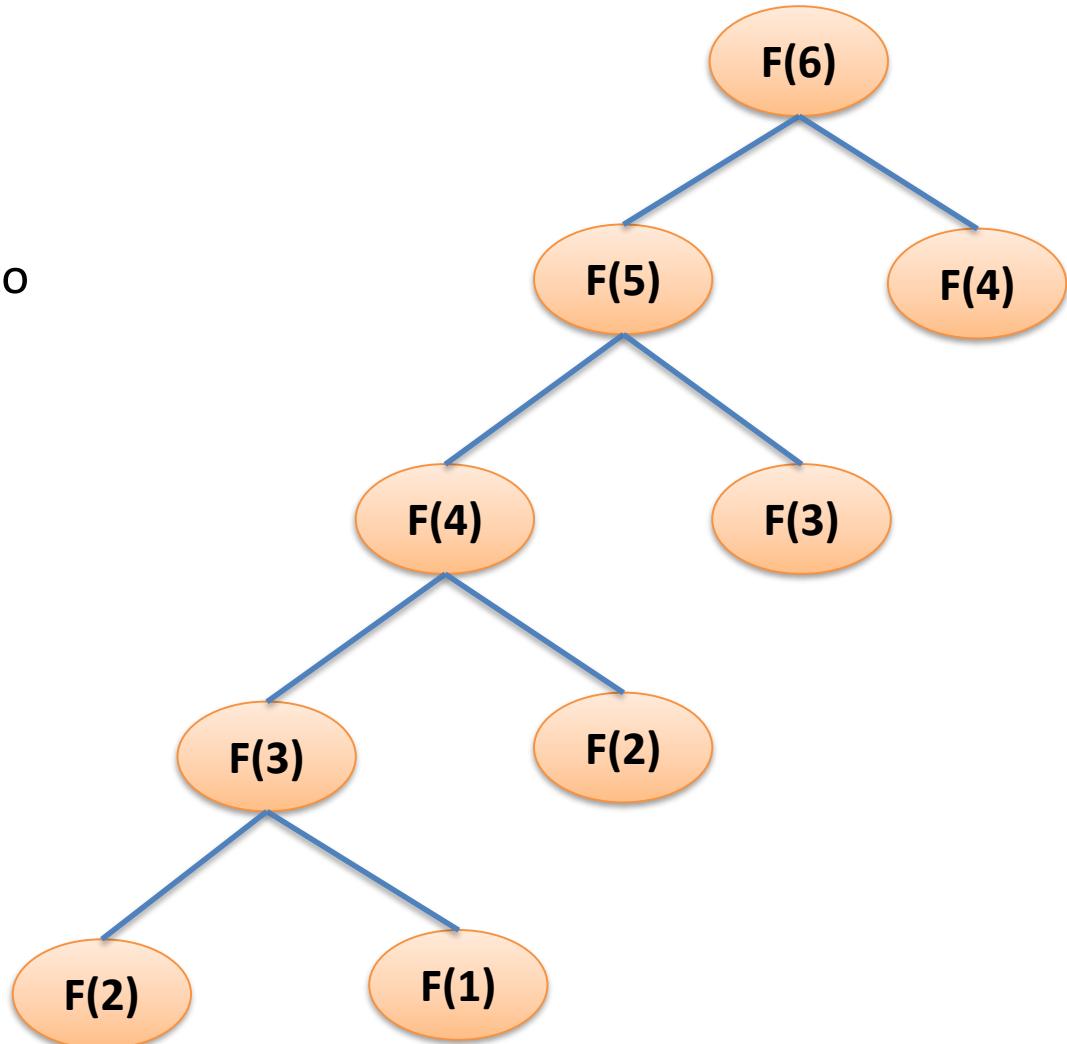
- Top-down approach
 - Start from top

F(6)

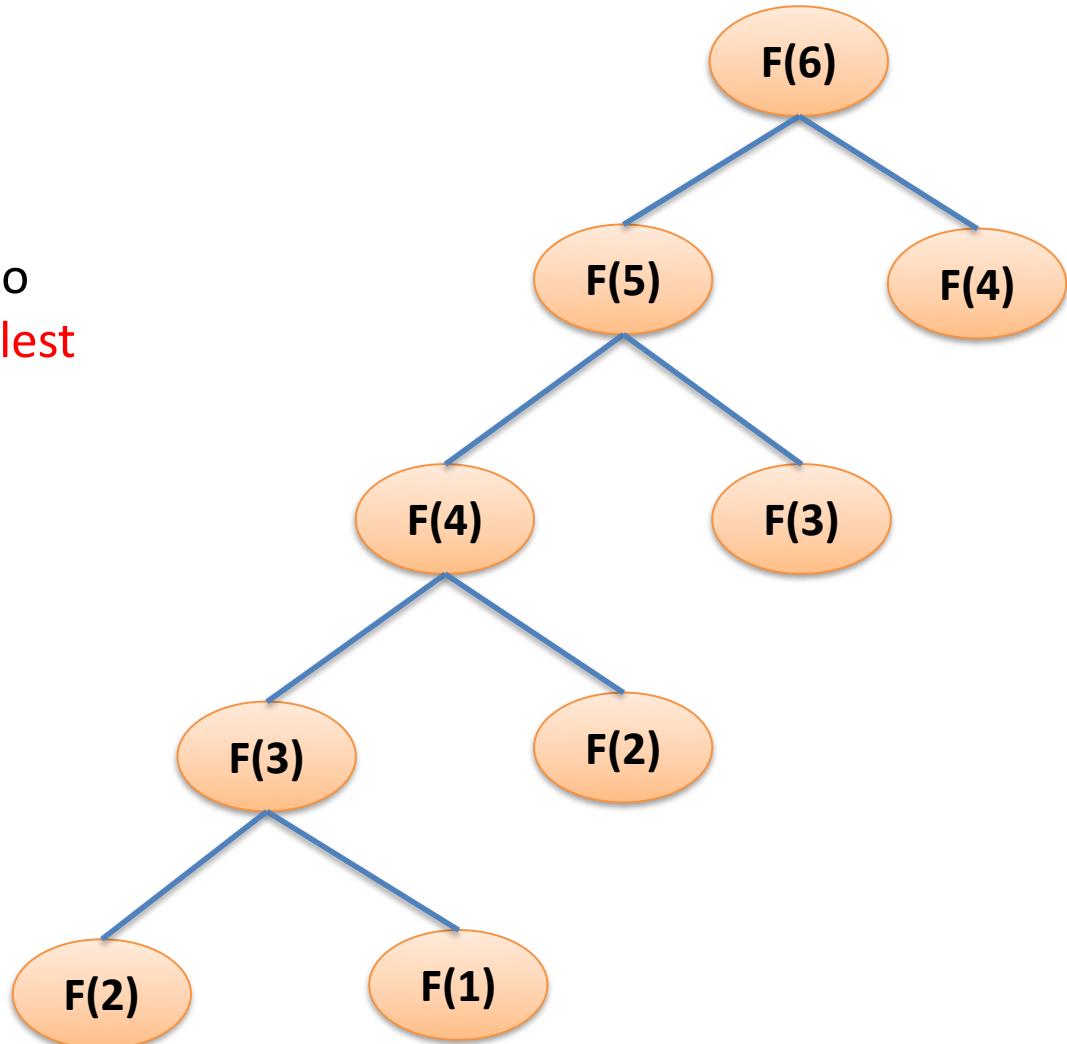
Fibonacci

n-th number in the series

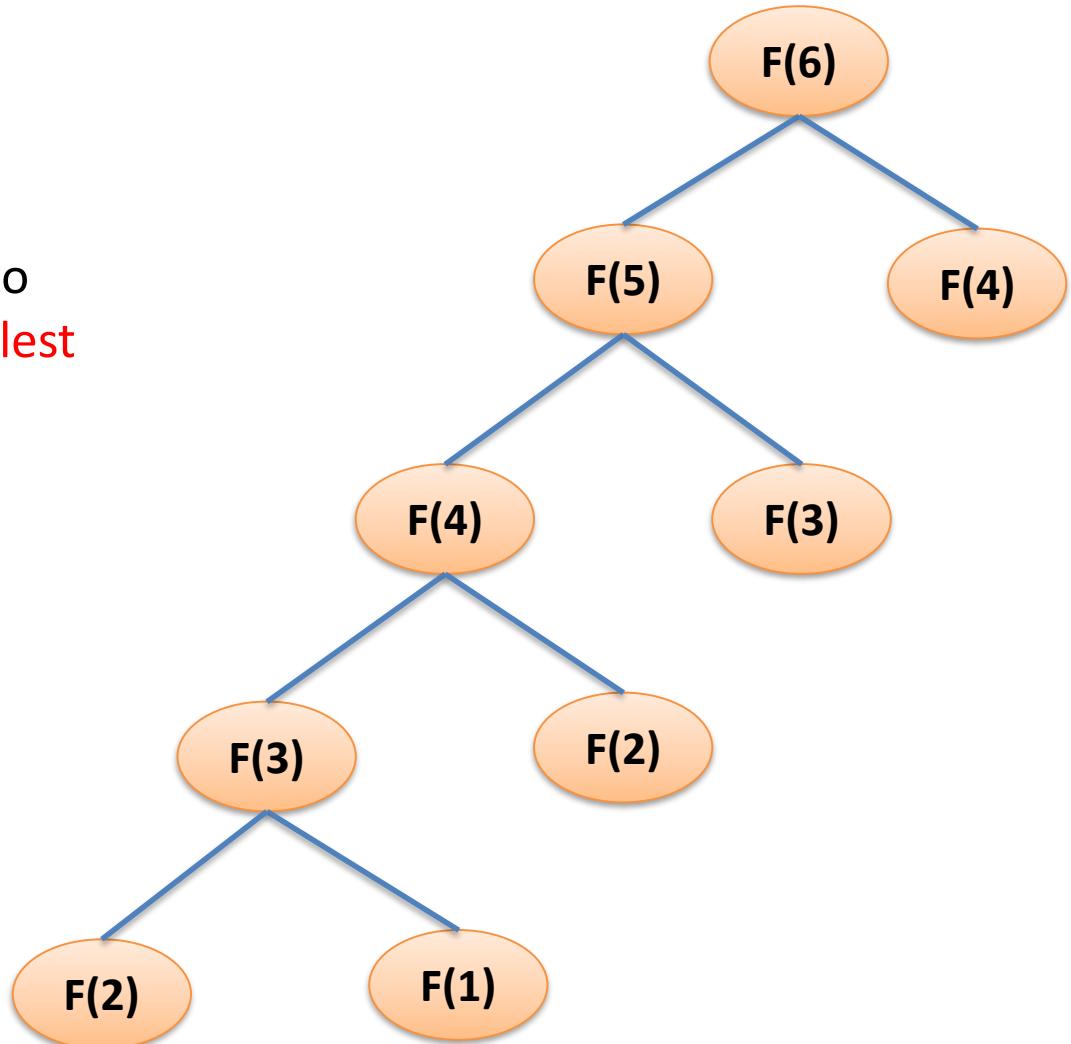
- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem



- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**



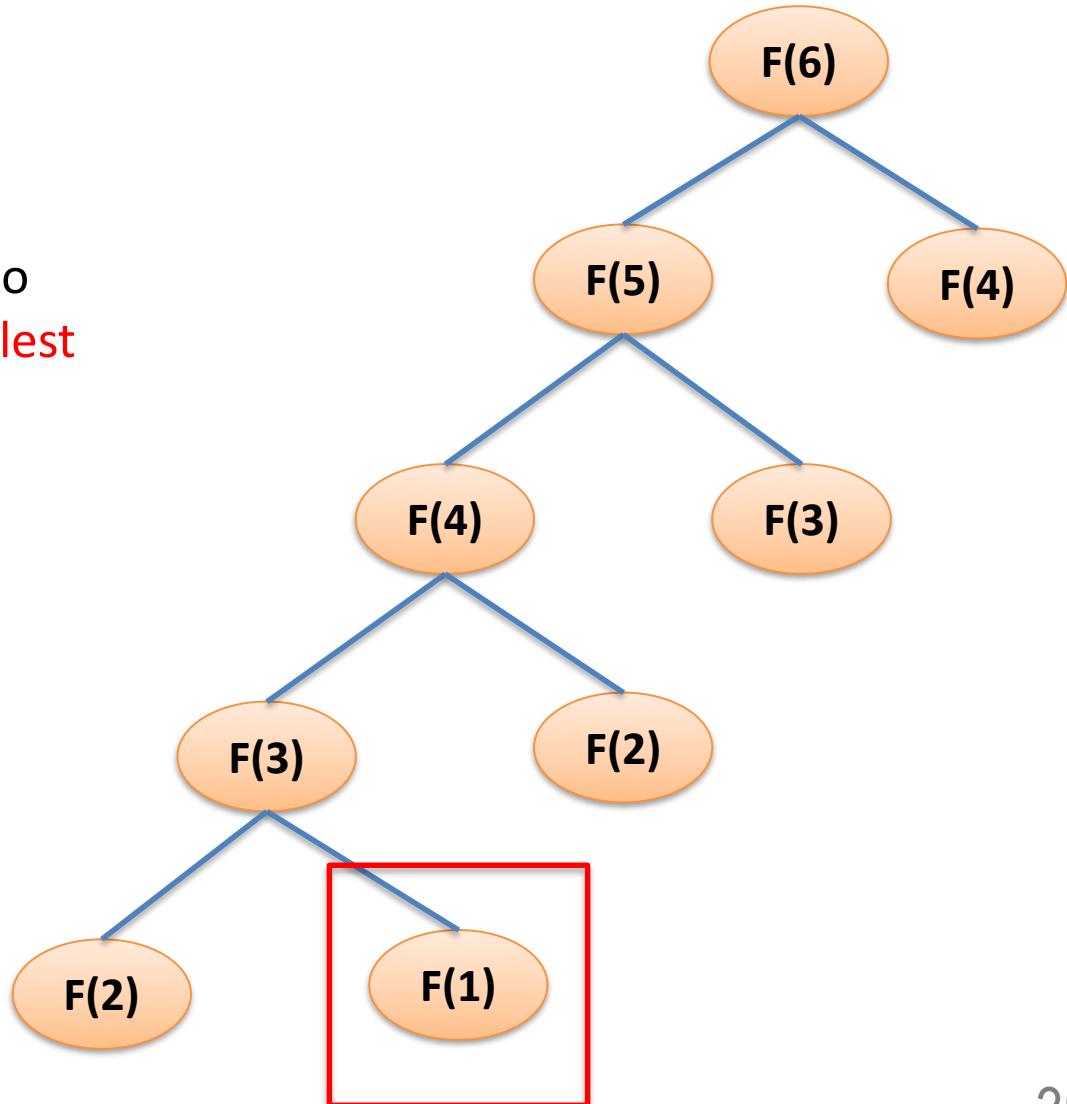
- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**
 - Then solve it, reusing the results



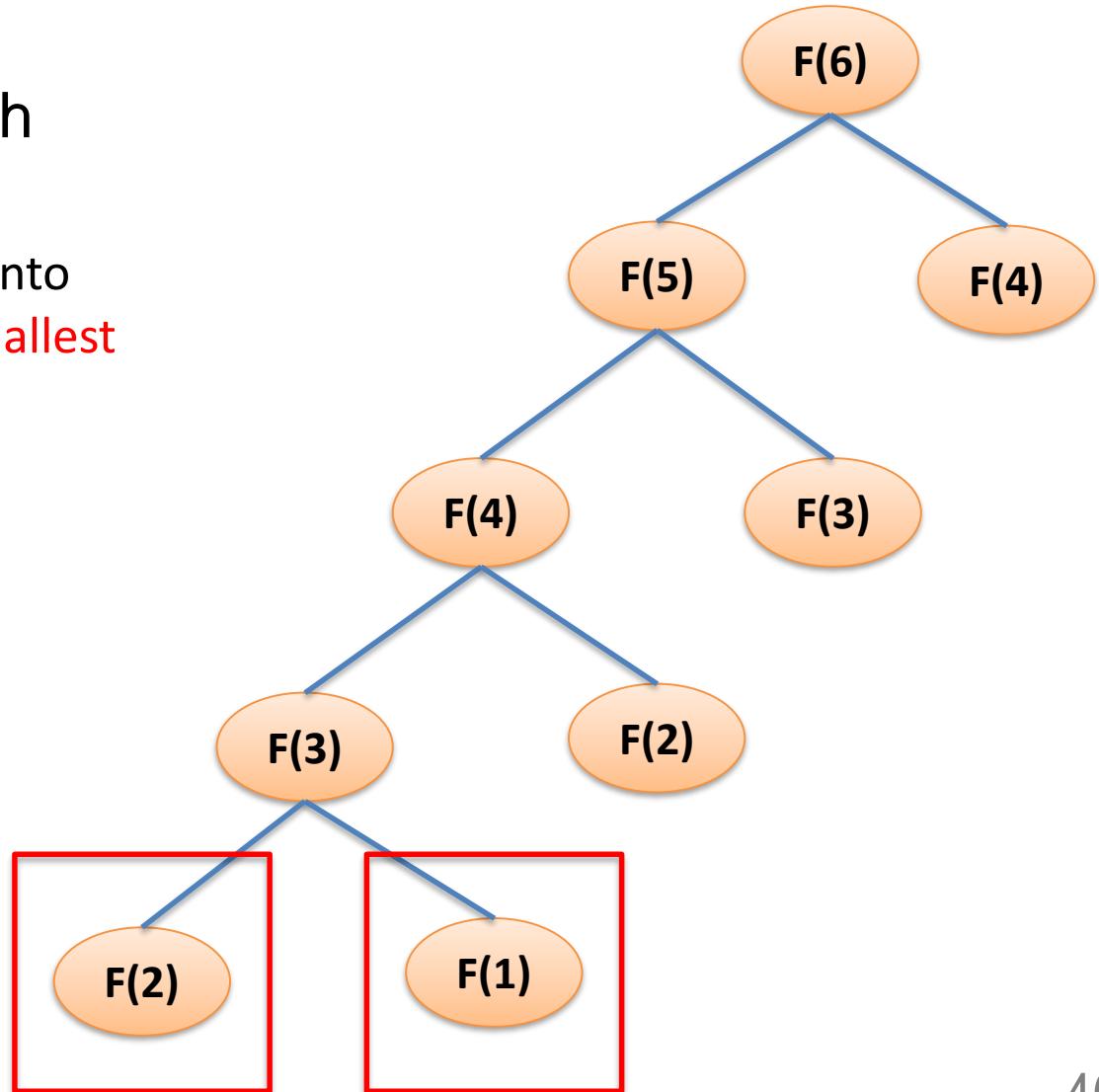
Fibonacci

n-th number in the series

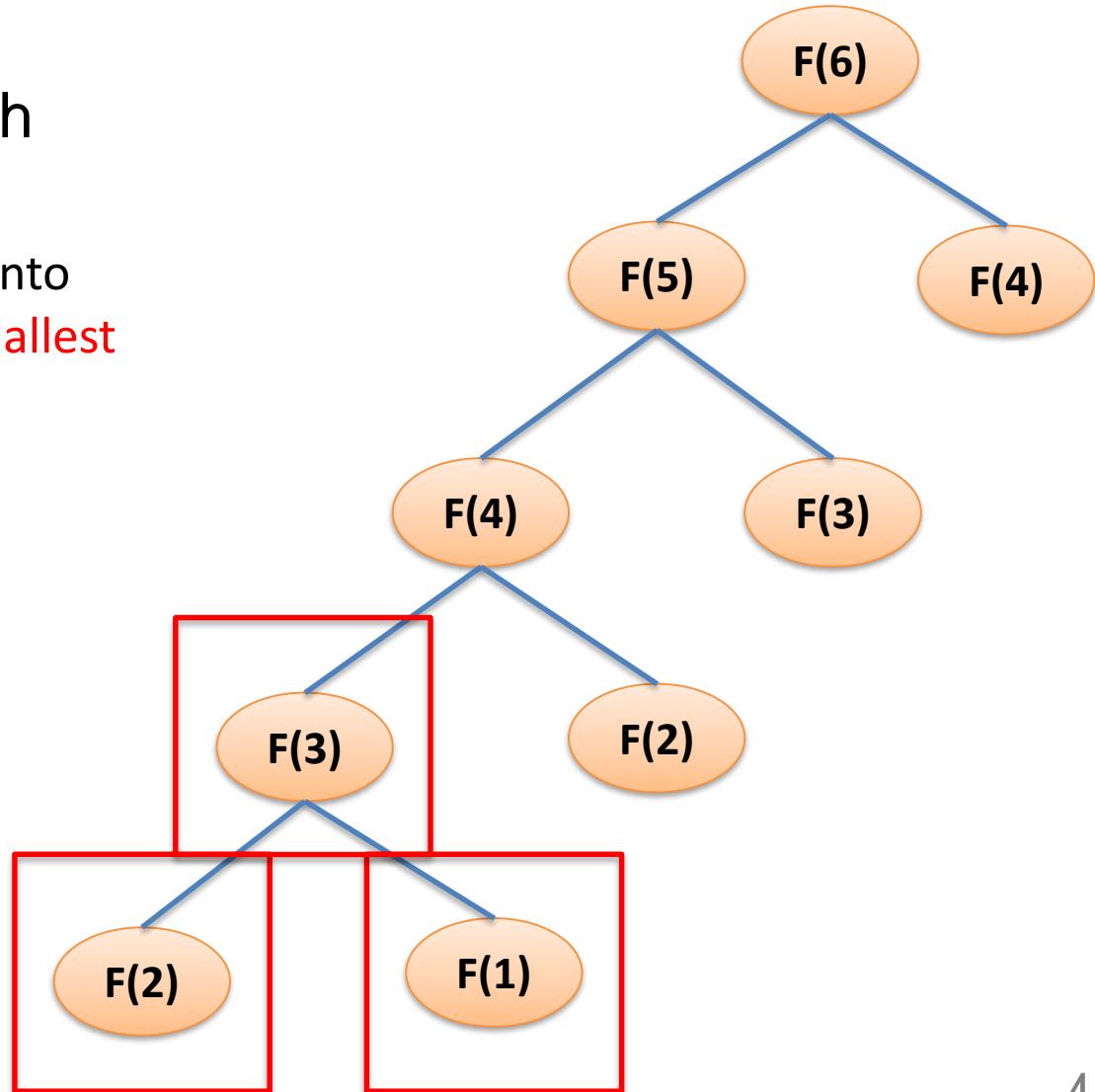
- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**
 - Then solve it, reusing the results



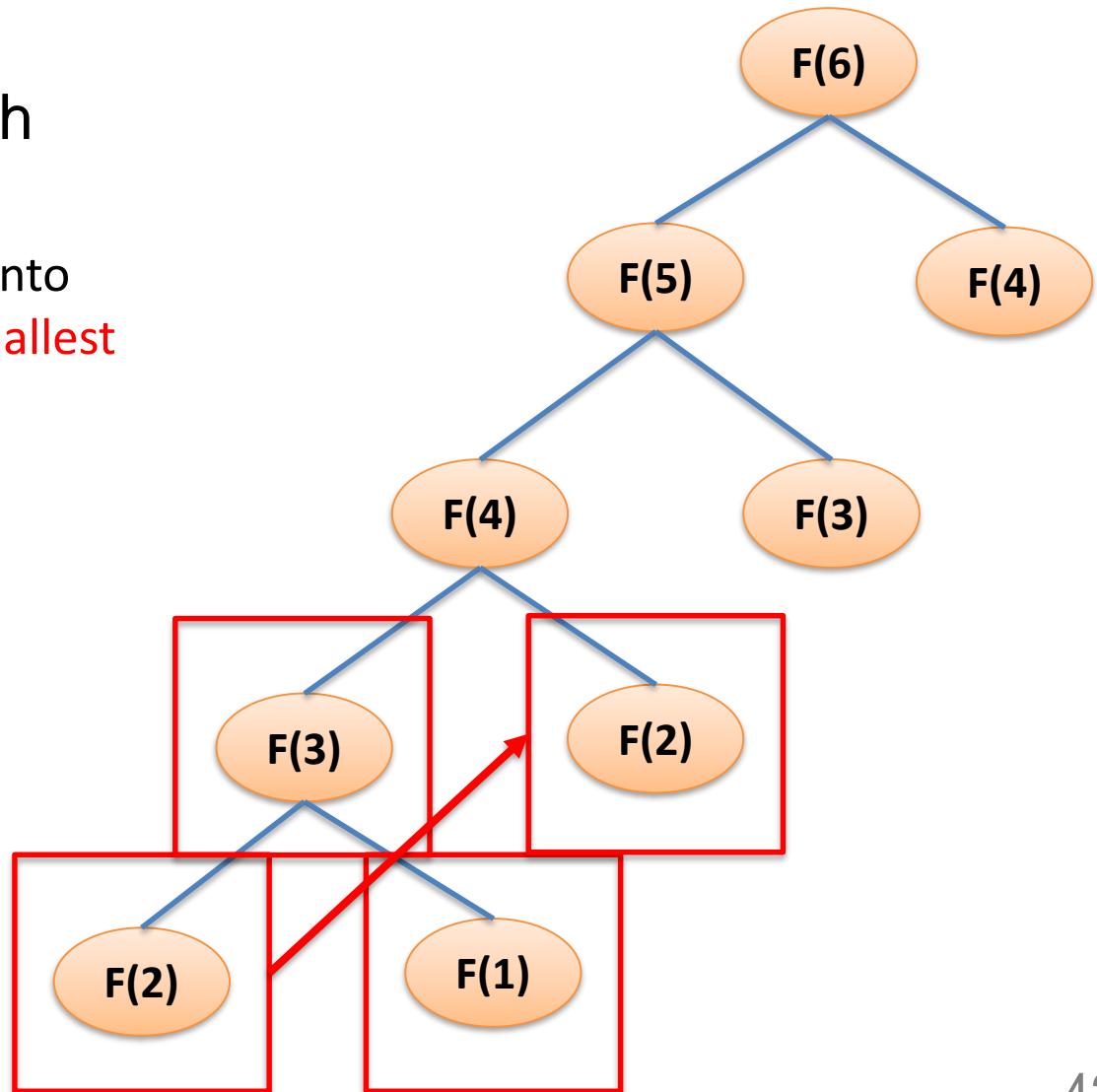
- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**
 - Then solve it, reusing the results



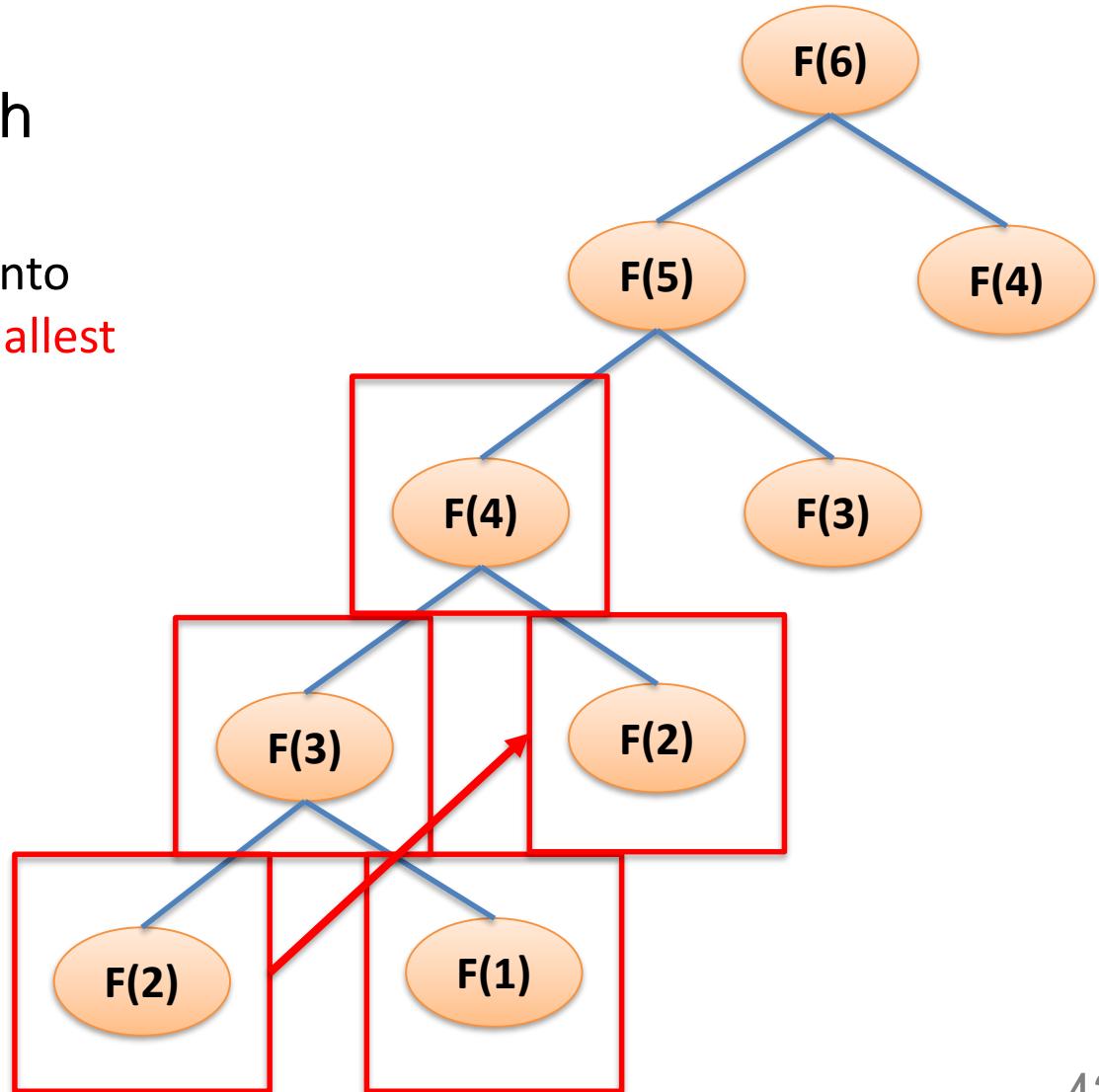
- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**
 - Then solve it, reusing the results



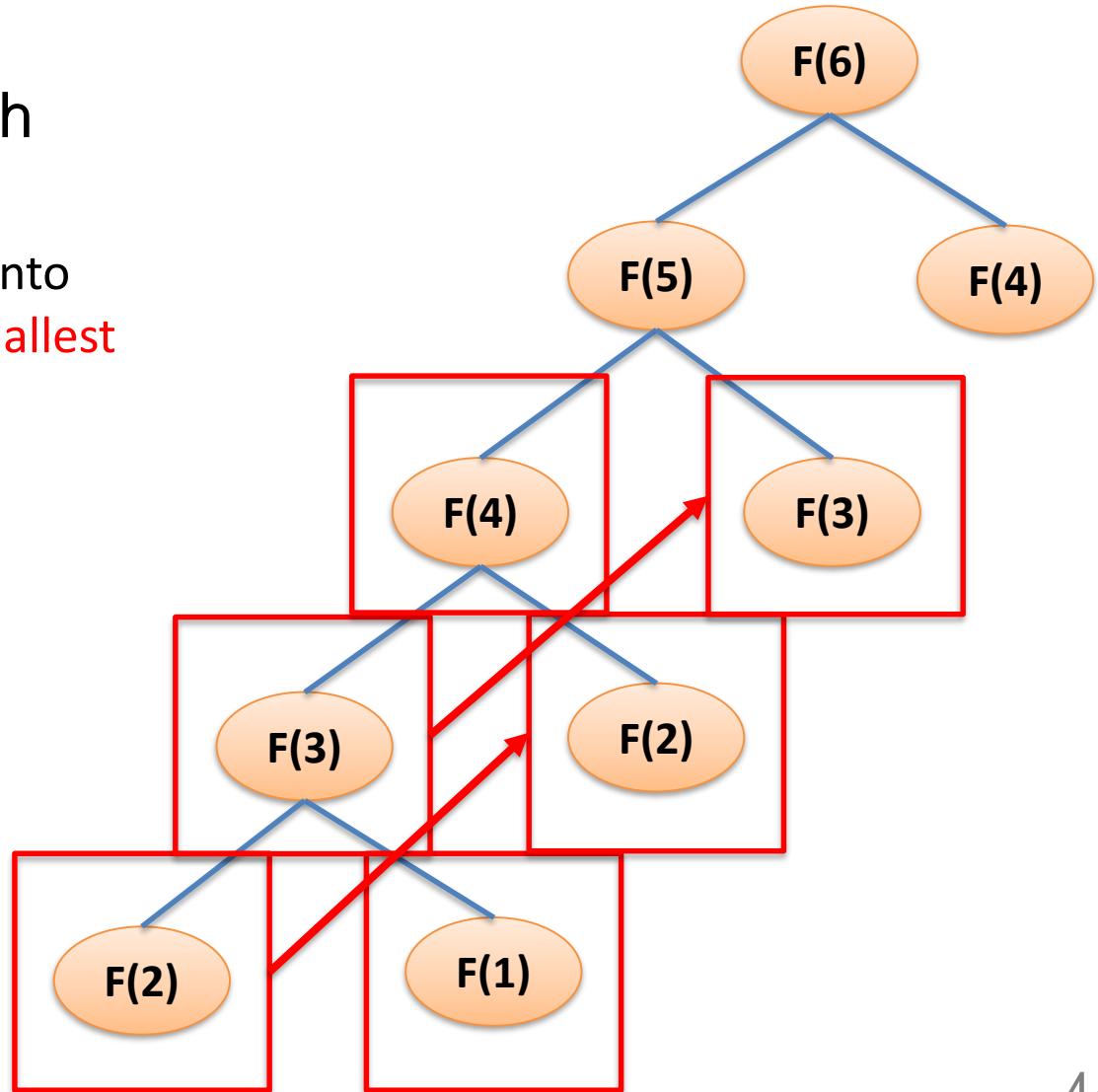
- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**
 - Then solve it, reusing the results



- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**
 - Then solve it, reusing the results



- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**
 - Then solve it, reusing the results



- Top-down approach
 - Start from top
 - Keep on breaking it into smaller problem. **Smallest is the base case!**
 - Then solve it, reusing the results

```
1     memo = [-1] * N
2     memo[0] = 0                      # in CompSci, we start from 0
3     memo[1] = 1
4
5     def fib_dpa(N):
6         if memo[N] != -1:    # if we have computed before
7             return memo[N]
8         # computer the fibonacci
9         memo[N] = fib_dpa(N-1) + fib_dpa(N-2)
10        return memo[N]
```

Questions?

Dynamic Programming

vs Divide and Conquer

- Is it the same?

Dynamic Programming

vs Divide and Conquer

- Is it the same?
 - Take a big thing
 - Divide to a smaller one
 - Solve the smaller one

Dynamic Programming

vs Divide and Conquer

- Is it the same?
 - Take a big thing
 - Divide to a smaller one
 - Solve the smaller one
 - Combine solutions up to the big thing...



Dynamic Programming

vs Divide and Conquer

- Is it the same?
 - Take a big thing
 - Divide to a smaller one
 - Solve the smaller one
 - Combine solutions up to the big thing...
- NO!... But why?

Dynamic Programming

vs Divide and Conquer

- Is it the same?
 - Take a big thing
 - Divide to a smaller one
 - Solve the smaller one.
These solutions are reusable due to overlapping sub problems
 - Combine solutions up to the big thing...
- NO!... But why?

Dynamic Programming

vs Divide and Conquer

- Is it the same?
 - Take a big thing
 - Divide to a smaller one
 - Solve the smaller one.
These solutions are reusable due to overlapping sub problems
 - Combine solutions up to the big thing...
- NO!... But why?
- Let us look at bottom-up to understand better...

Dynamic Programming

vs Divide and Conquer

- Is it the same?
 - Take a big thing
 - Divide to a smaller one
 - Solve the smaller one.
*These **OPTIMAL** solutions are reusable due to overlapping sub problems*
 - Combine solutions up to the big thing...
- NO!... But why?
- Let us look at bottom-up to understand better...

Questions?

Fibonacci

n-th number in the series

- Bottom-up approach
 - Can you explain it?

Fibonacci

n-th number in the series

- Bottom-up approach
 - Can you explain it?



Fibonacci

n-th number in the series

- Bottom-up approach
 - Start from the base case
 - Solve it
 - Use it to solve bigger case
 - Until we reach the final one...

- Bottom-up approach
 - Start from the base case
 - Solve it
 - Use it to solve bigger case
 - Until we reach the final one...

```
1  memo = [-1] * N
2  memo[0] = 0                      # in CompSci, we start from 0
3  memo[1] = 1
4
5  for i in range(2,N):
6      memo[i] = memo[i-1] + memo[i-2]
```

Questions?

Fibonacci

n-th number in the series

- What is the complexity?
 - Bottom-up
 - Top-down

Fibonacci

n-th number in the series

- What is the complexity?

- Bottom-up
- Top-down

```
1     memo = [-1] * N
2     memo[0] = 0                      # in CompSci, we start from 0
3     memo[1] = 1
4
5     def fib_dpa(N):
6         if memo[N] != -1:    # if we have computed before
7             ...
8             return memo[N]
9         # computer the fibonacci
10        memo[N] = fib_dpa(N-1) + fib_dpa(N-2)
11        return memo[N]
```

```
1     memo = [-1] * N
2     memo[0] = 0                      # in CompSci, we start from 0
3     memo[1] = 1
4
5     for i in range(2,N):
6         memo[i] = memo[i-1] + memo[i-2]
```

Fibonacci

n-th number in the series

- What is the complexity?

- Bottom-up
- Top-down



Suvashish Chakraborty shared Nondeterministic Memes for NP

Complete Teens's photo.

5 hrs ·

#relatabe #ADSstuff

When you use Dynamic Programming to solve a naively exponential time problem in polynomial time



Questions?

Dynamic Programming

The superpower

- Sounds easy right?

Dynamic Programming

The superpower

- Sounds easy right?
- Can be used to solve a lot of problem
 - Especially finding combinations

Dynamic Programming

The superpower

- Sounds easy right?
- Can be used to solve a lot of problem
 - Especially finding combinations
 - Popular in interviews

Dynamic Programming

The superpower

- Sounds easy right?
- Can be used to solve a lot of problem
 - Especially finding combinations
 - Popular in interviews
 - Very popular in programming competitions!

Dynamic Programming

The superpower

- Sounds easy right?
 - It isn't that easy however... Why?
- Can be used to solve a lot of problem
 - Especially finding combinations
 - Popular in interviews
 - Very popular in programming competitions!

Dynamic Programming

The superpower

- Sounds easy right?
 - It isn't that easy however... Why? Not easy to break problems down



- Can be used to solve a lot of problem
 - Especially finding combinations
 - Popular in interviews
 - Very popular in programming competitions!

Dynamic Programming

The superpower

- Sounds easy right?
 - It isn't that easy however... Why? Not easy to break problems down



- So how?



Dynamic Programming

The superpower

- Sounds easy right?
 - It isn't that easy however... Why? Not easy to break problems down



- So how?
 - Practice



Dynamic Programming

The superpower

- Sounds easy right?
 - It isn't that easy however... Why? Not easy to break problems down
- So how?
 - Coin change
 - Knapsack
 - Edit-distance

Questions?

Coin Change problem

The less number of coins...

- Consider the following scenario

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}



Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?



Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
 - $2 \times 50 + 1 \times 10$
 - 11×10
 - ... and many more



Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
 - **2x50 + 1x10 (3 coins)**
 - 11x10
 - ... and many more
 - But we want the **smallest** number of coins!



Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
 - **2x50 + 1x10 (3 coins)**
 - 11x10
 - ... and many more
 - But we want the **smallest** number of coins!
- Let us now explore the possible solutions...



Questions?

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Brute force?

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Brute force?
 - Try every combination!

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Brute force?
 - Try every combination!
 - Choose the smallest number of coin...

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Brute force?
 - Try every combination!
 - Choose the smallest number of coin...
 - Will it work? Of course!

Questions?

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Greedy solution?

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Greedy solution?
 - Try with the biggest number of coin
 - Then fill the balance with the rest #ez

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Greedy solution?
 - Try with the biggest number of coin
 - Then fill the balance with the rest #ez
 - Doesn't always work (**greed is not good**)

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Greedy solution?
 - Try with the biggest number of coin
 - Then fill the balance with the rest #ez
 - Doesn't always work (**greed is not good**)
{1,5,6,9} and I want 12

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Greedy solution?
 - Try with the biggest number of coin
 - Then fill the balance with the rest #ez
 - Doesn't always work (**greed is not good**)
{1,5,6,9} and I want 12
 - $1 \times 9 + 3 \times 1 = 12$ for 4 coins

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Greedy solution?
 - Try with the biggest number of coin
 - Then fill the balance with the rest #ez
 - Doesn't always work (**greed is not good**)
{1,5,6,9} and I want 12
 - $1 \times 9 + 3 \times 1 = 12$ for 4 coins
 - $2 \times 6 = 12$ for 2 coins...

Questions?

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,10,50}
 - I want 110 doge coin, so what is the possible coin combination?
- Dynamic solution?
 - Let us try a smaller problem (easier for me to visualize)

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - I want 12 doge coin value, so what is the possible coin combination?

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - I want 12 doge coin value, so what is the possible coin combination?

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins													

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - I want 12 doge coin value, so what is the possible coin combination?

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0												

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - I want 12 doge coin value, so what is the possible coin combination?

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	inf											

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will **loop** through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	inf											

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	inf											

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

$$0+1 = 1$$



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	inf										

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

$$0+1 = 1$$

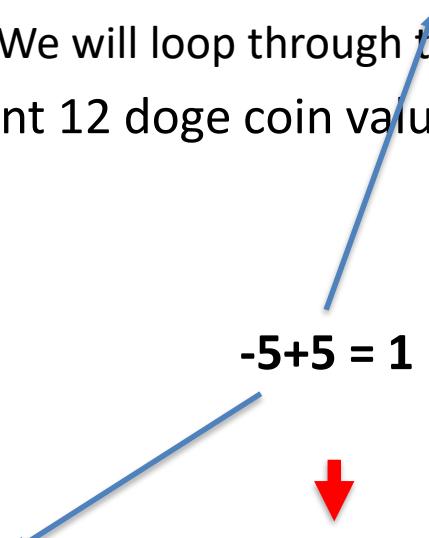


Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	inf										

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?



$-5+5 = 1$

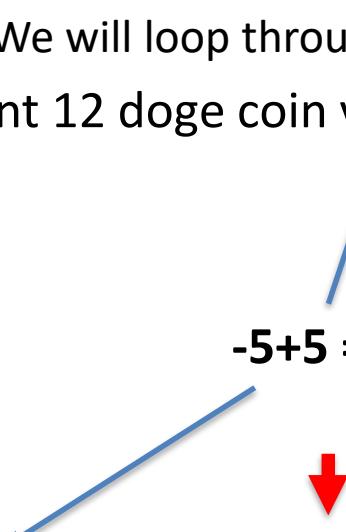
Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	inf										

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

-5+5 = 1, and so on....



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	inf										

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	inf										

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

Repeat the process...



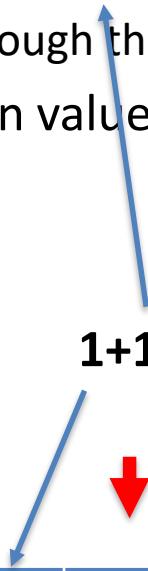
Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	inf										

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1,5,6,9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

$$1+1=2$$



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	inf										

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1,5,6,9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

1+1=2, 2 coins as well

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	inf									

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	inf									

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	inf							

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1,5,6,9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

4+1=5, for 5 coins

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	inf							

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1, 5, 6, 9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

$4+1=5$, for 5 coins

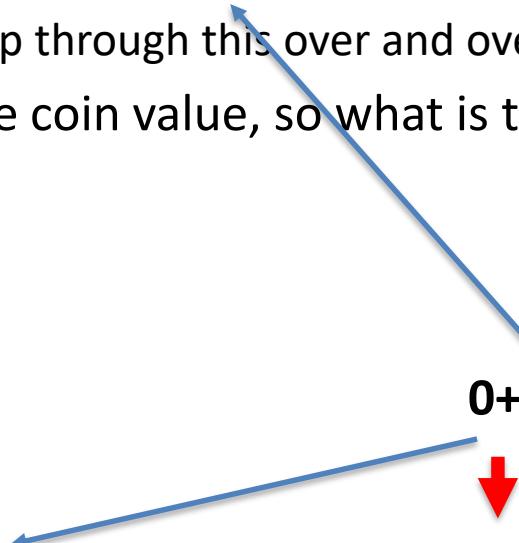
Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	5	inf						

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1, 5, 6, 9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

$0+5=5$, for 1 coin



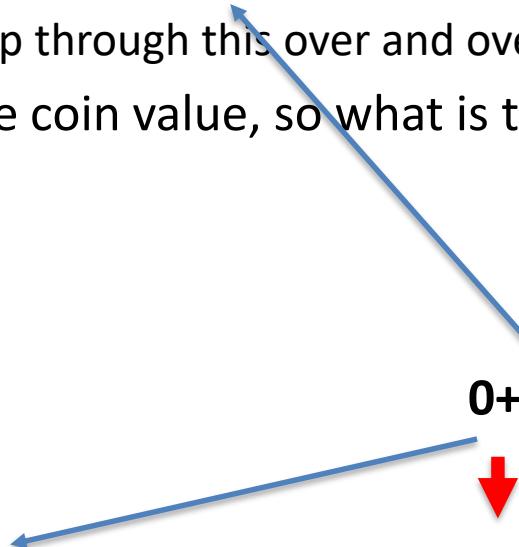
Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	5	inf						

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1, 5, 6, 9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

$0+5=5$, for 1 coin



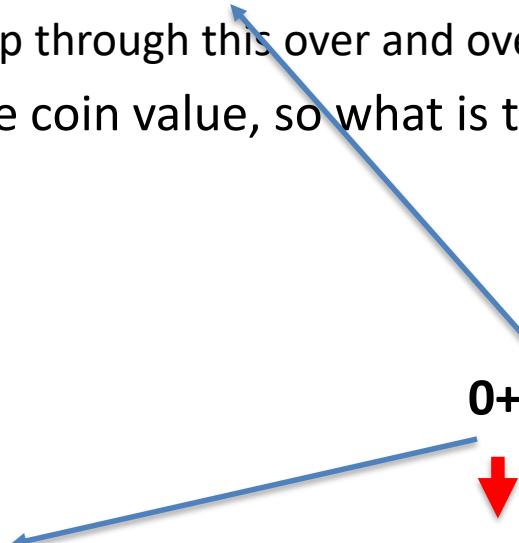
Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	5	inf						
						vs 1							

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1, 5, 6, 9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

$0+5=5$, for 1 coin



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	inf						

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1, 5, 6, 9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

5+1=6, for 2 coins

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	inf						

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1,5,6,9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

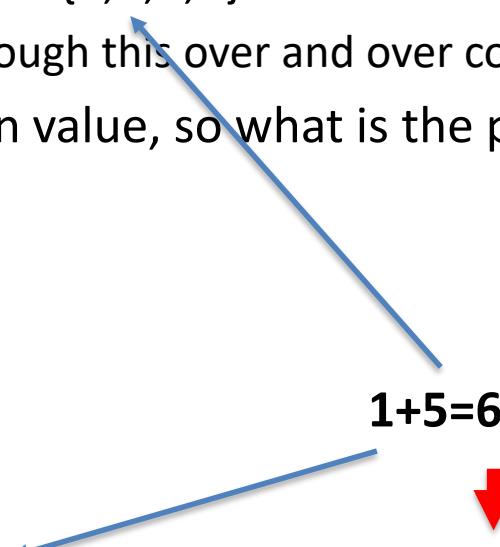
5+1=6, for 2 coins

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	2	inf	inf	inf	inf	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?



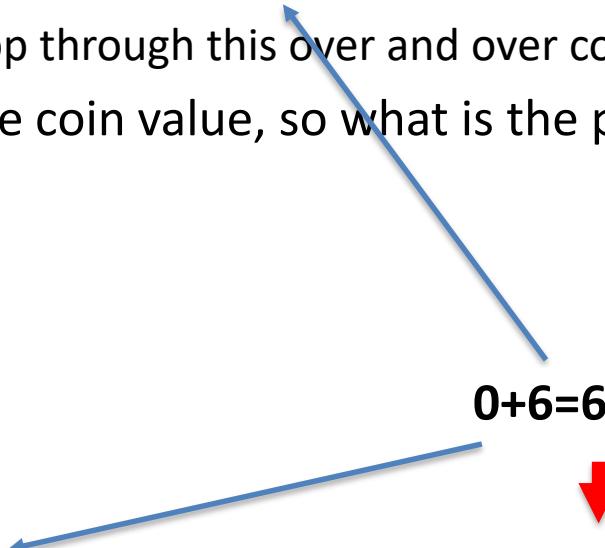
Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	2	inf	inf	inf	inf	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is $\{1, 5, 6, 9\}$
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?

$0+6=6$, for 1 coin



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	inf	inf	inf	inf	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	inf	inf	inf	inf	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	inf	inf	inf	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	inf	inf	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	4	inf	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	?	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	inf	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	inf

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2

Questions?

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done
 - Complexity?



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	4	2	2	2

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done
 - Complexity? O(NM)



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	4	2	2	2

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done
 - Complexity? $O(NM)$ still much faster than brute force...



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	4	2	2	2

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done
 - Complexity? $O(NM)$ still much faster than brute force...
 - Note: If the list is sorted, we can terminate earlier on the smaller values



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	4	2	2	2

Questions?

Coin Change problem

The less number of coins...

- Consider the following scenario
 - Dogecoin currency is {1,5,6,9}
 - We will loop through this over and over considering the coins...
 - I want 12 doge coin value, so what is the possible coin combination?
 - So keep on running it and eventually we would be done
 - Complexity? $O(NM)$ still much faster than brute force...
 - Can you **code** it?



Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	4	2	2	2

Coin Change problem

The less number of coins...

- I'll give the algorithm here

```
1  memo = [inf] * (N+1)
2  memo[0] = 0
3  for value in range(1,N+1):
4      for j in range(M):                      # this is to go through coins
5          if coin[j] <= value
6              balance = value - coin[j]
7              count = 1 + memo[balance]
8              if count < memo[value]:           # if we have new optimal
9                  memo[value] = count
10
11 return memo[N]
```

Coin Change problem

The less number of coins...

- I'll give the algorithm here
 - Is this top-down or bottom-up?

```
1     memo = [inf] * (N+1)
2     memo[0] = 0
3     for value in range(1,N+1):
4         for j in range(M):                      # this is to go through coins
5             if coin[j] <= value
6                 balance = value - coin[j]
7                 count = 1 + memo[balance]
8                 if count < memo[value]:        # if we have new optimal
9                     memo[value] = count
10    return memo[N]
```

Coin Change problem

The less number of coins...

- I'll give the algorithm here
 - Is this top-down or bottom-up?

```
1     memo = [inf] * (N+1)
2     memo[0] = 0
3     for value in range(1,N+1):
4         for j in range(M):                      # this is to go through coins
5             if coin[j] <= value
6                 balance = value - coin[j]
7                 count = 1 + memo[balance]
8                 if count < memo[value]:        # if we have new optimal
9                     memo[value] = count
10    return memo[N]
```

Coin Change problem

The less number of coins...

- I'll give the algorithm here
 - Is this top-down or **bottom-up**?
 - The following is the top-down from Aamir (usually recursive)

```
1  memo = [inf] * (N+1)
2  memo[0] = 0
3  for value in range(1,N+1):
4      for j in range(M):                      # this is to go through coins
5          if coin[j] <= value
6              balance = value - coin[j]
7              count = 1 + memo[balance]
8              if count < memo[value]:           # if we have new optimal
9                  memo[value] = count
10
11 return memo[N]
```

Top-down Solution

Initialize Memo[] to contain -1 for all indices # -1 indicates the solution for this index has not been computed yet

```
Memo[0] = 0  
Function CoinChange(value)  
    if Memo[value] != -1: \\\ DISCUSS infinity is incorrect  
        return Memo[value]  
    else:  
        minCoins = Infinity  
        for i=1 to N  
            if Coins[ i ] <= value  
                c = 1 + CoinChange(value - Coins[ i ])  
                if c < minCoins  
                    minCoins = c  
        Memo[value] = minCoins  
    return Memo[value]
```

Bottom up solution:

1 + Memo[value – Coins[i]]

Coin Change problem

The less number of coins...

- Top-down vs Bottom up
 - Top-down might save some computations
 - Bottom-up might save space especially since no recursion



Coin Change problem

The less number of coins...

- Top-down vs Bottom up
 - Top-down might save some computations
 - Bottom-up might save space especially since no recursion
 - I only use bottom-up

Coin Change problem

The less number of coins...

- Top-down vs Bottom up
 - Top-down might save some computations
 - Bottom-up might save space especially since no recursion
 - I only use bottom-up
 - But some problems could be easier with top-down as it is more intuitive
 - Technically both are interchangeable...

Questions?

Kapsack Problem

Min-max like a boss

- A problem that can be applicable to a lot of real life scenario

Kapsack Problem

Min-max like a boss

- A problem that can be applicable to a lot of real life scenario
 - Given a limitation (cost)

Kapsack Problem

Min-max like a boss

- A problem that can be applicable to a lot of real life scenario
 - Given a limitation (cost)
 - Optimize something (profit)

Kapsack Problem

Min-max like a boss

- A problem that can be applicable to a lot of real life scenario
 - Given a limitation (cost)
 - Optimize something (profit)
 - **18th most popular** algorithmic problem

Kapsack Problem

Min-max like a boss

- A problem that can be applicable to a lot of real life scenario
 - Given a limitation (cost)
 - Optimize something (profit)
 - **18th most popular** algorithmic problem
- Given a capacity C and a set of items with weights and value... can you find a combination of item such as the total weight $< C$ but the total value is maximized?

Kapsack Problem

Min-max like a boss

- A problem that can be applicable to a lot of real life scenario
 - Given a limitation (cost)
 - Optimize something (profit)
 - **18th most popular** algorithmic problem
- Given a capacity C and a set of items with weights and value... can you find a combination of item such as the total weight $< C$ but the total value is maximized?
 - Unbounded = items are unlimited
 - Bounded = each item can only be taken once

Kapsack Problem

Min-max like a boss

- Let say you have these items

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

Kapsack Problem

Min-max like a boss



- Let say you have these items

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

- And you only have 12 kg... what would you loot?

Kapsack Problem

Min-max like a boss



- Let say you have these items

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

- And you only have 12 kg... what would you loot?
 - $2 \times B + 2 \times D = \780

Kapsack Problem

Min-max like a boss



- Let say you have these items

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

- And you only have 12 kg... what would you loot?
 - $2 \times B + 2 \times D = \780
 - So how do we code it?
This is very similar to the coin change!

Questions?

Kapsack Problem

Unbounded

- Let us try the bottom-up approach

Kapsack Problem

Unbounded

- Let us try the bottom-up approach
 - Start with 0 weight till 12 weight

Kapsack Problem

Unbounded

- Let us try the bottom-up approach
 - Start with 0 weight till 12 weight
 - Find the optimal for each weight...

Kapsack Problem

Unbounded

- Let us try the bottom-up approach
 - Start with 0 weight till 12 weight
 - Find the optimal for each weight...
 - Use the earlier optimal for the new weight

Kapsack Problem

Unbounded

- Let us try the bottom-up approach
 - Start with 0 weight till 12 weight
 - Find the **optimal** for each weight...
Maximum profit!
 - Use the earlier **optimal** for the new weight



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0												

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	0	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



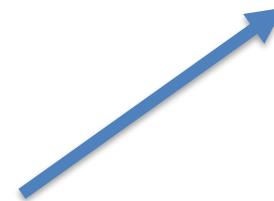
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	0	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



9kg, too heavy



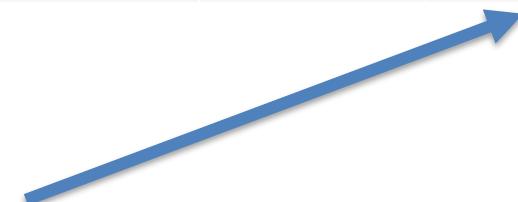
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	0	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



5kg, too heavy



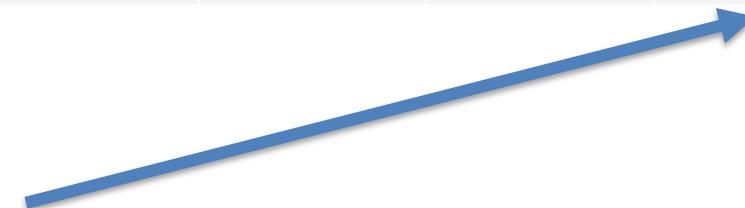
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	0	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



6kg, too heavy



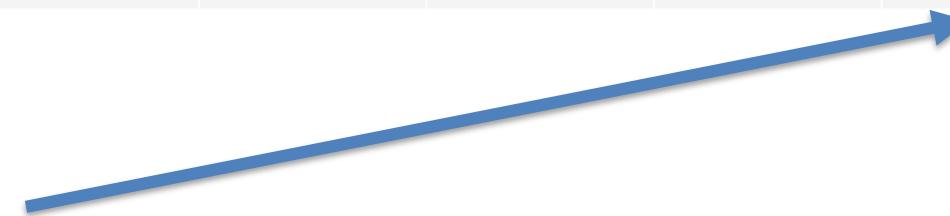
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	0	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg, can fit



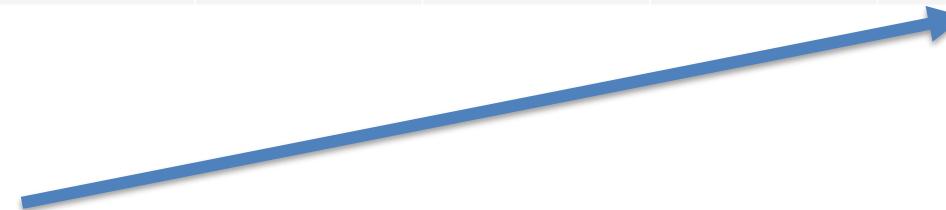
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	0	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg, can fit... so we find the optimal of 0kg + 1kg



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	0	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg, can fit... so we find the optimal of 0kg + 1kg



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



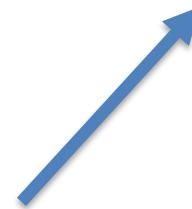
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



9kg too heavy



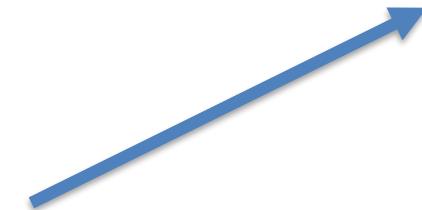
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



5kg too heavy



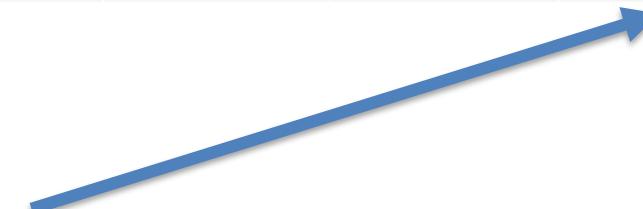
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



6kg too heavy



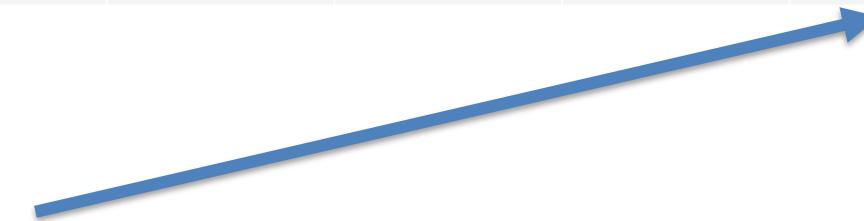
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg is OK



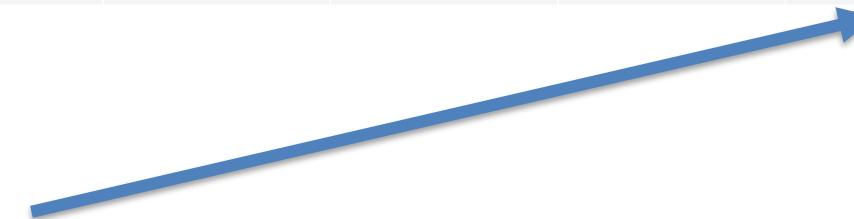
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg is OK... optimal of 1kg + profit from this item



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg is OK... optimal of 1kg + profit from this item



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	0	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg is OK... optimal of 1kg + profit from this item



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	0	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

We just repeat the process



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	0	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

We just repeat the process



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

Now let us try here



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



9kg too heavy...



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



5kg can fit. Optimal 0kg + current item



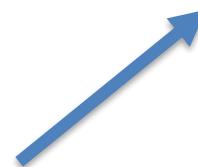
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	0	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



5kg can fit. Optimal 0kg + current item



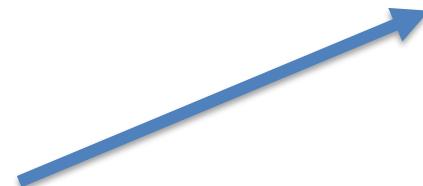
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



6kg too heavy



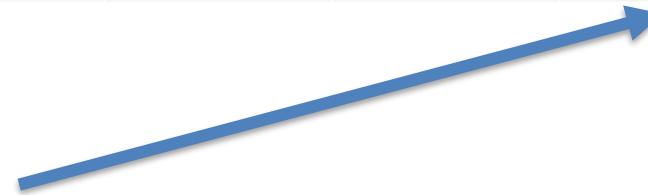
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg is OK. Optimal 4kg + current item



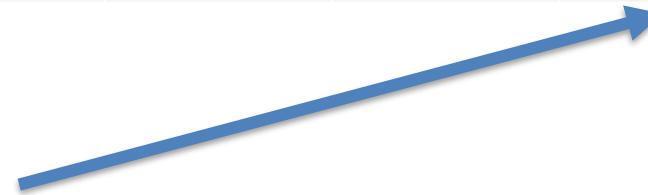
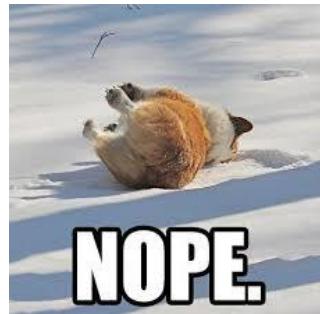
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



1kg is OK. Optimal 4kg + current item but it is only 200



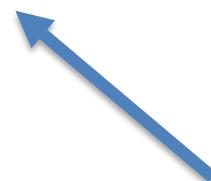
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



Moving on....



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



heavy



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	0	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

Optimal 1kg + current item



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	0	0	0	0	0	0	0

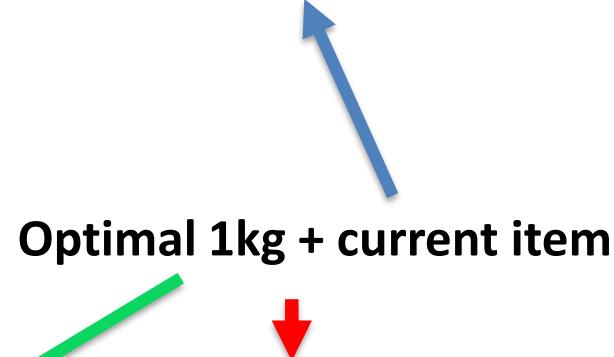
Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

Optimal 1kg + current item



A blue arrow points from the text "Optimal 1kg + current item" to the value of item D (\$40) in the table above. A red arrow points from the text to the weight of item D (1kg) in the table above.

Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	390	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



Optimal 0kg + current item = 180 only



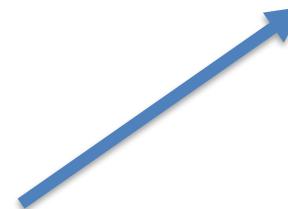
Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	390	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40



Optimal 5kg + current item = same 390

Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	390	0	0	0	0	0	0

Kapsack Problem

Unbounded

- Let us run through it

Item	A	B	C	D
Weight	9kg	5kg	6kg	1kg
Value	\$550	\$350	\$180	\$40

Eventually....



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	390	430	470	550	700	740	780

Questions?

Have a break!

Kapsack Problem

Unbounded

- So what is our algorithm?
 - Let us try to produce it now as part of the class activity!

Kapsack Problem

Unbounded

- So what is our algorithm?
 - Very similar to the coin change except
 - Finding maximum instead of minimum
 - Initialized to 0

Kapsack Problem

Unbounded

- So what is our algorithm?
 - Very similar to the coin change except
 - Finding maximum instead of minimum
 - Initialized to 0
 - You have the 1 array, called items
 - N number of items
 - Items[i].weight for the weight
 - Items[i].profit for the profit

Questions?

Kapsack Problem

Unbounded

- So what is our algorithm?
 - Very similar to the coin change except
 - Finding maximum instead of minimum
 - Initialized to 0

```
1  memo = [0] * (N+1)                                # N is the total weight
2  memo[0] = 0
3  for bag_weight in range(1,N+1):
4      for j in range(M):                            # this is to go through items
5          if items[j].weight <= bag_weight:
6              balance = bag_weight - items[j].weight
7              profit = item[j].profit + memo[balance]
8              if profit > memo[bag_weight]:    # if we have new optimal
9                  memo[bag_weight] = profit
10
11 return memo[N]
```

Kapsack Problem

Unbounded

- So what is our algorithm?
 - Very similar to the coin change except
 - Finding maximum instead of minimum
 - Initialized to 0
 - Complexity?

```
1  memo = [0] * (N+1)                                # N is the total weight
2  memo[0] = 0
3  for bag_weight in range(1,N+1):
4      for j in range(M):                            # this is to go through items
5          if items[j].weight <= bag_weight:
6              balance = bag_weight - items[j].weight
7              profit = item[j].profit + memo[balance]
8              if profit > memo[bag_weight]:    # if we have new optimal
9                  memo[bag_weight] = profit
10
11 return memo[N]
```

Kapsack Problem

Unbounded

- So what is our algorithm?
 - Very similar to the coin change except
 - Finding maximum instead of minimum
 - Initialized to 0
 - Complexity? $O(NM)$

```
1  memo = [0] * (N+1)                                # N is the total weight
2  memo[0] = 0
3  for bag_weight in range(1,N+1):
4      for j in range(M):                            # this is to go through items
5          if items[j].weight <= bag_weight:
6              balance = bag_weight - items[j].weight
7              profit = item[j].profit + memo[balance]
8              if profit > memo[bag_weight]:    # if we have new optimal
9                  memo[bag_weight] = profit
10
11 return memo[N]
```

Kapsack Problem

Unbounded

- So what is our algorithm?
 - Very similar to the coin change except
 - Finding maximum instead of minimum
 - Initialized to 0
 - Complexity?
 - Top-down? See Nathan's slides

Questions?

Kapsack Problem

Unbounded

- But is the code correct?

```
1  memo = [0] * (N+1)                                # N is the total weight
2  memo[0] = 0
3  for bag_weight in range(1,N+1):
4      for j in range(M):                            # this is to go through items
5          if items[j].weight <= bag_weight:
6              balance = bag_weight - items[j].weight
7              profit = item[j].profit + memo[balance]
8              if profit > memo[bag_weight]:    # if we have new optimal
9                  memo[bag_weight] = profit
10
11 return memo[N]
```

Kapsack Problem

Unbounded

- But is the code correct?

- Currently, we assume the maximum value is when you find items with a total weight of 12 kg

Eventually....


Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	390	430	470	550	700	740	780

```
1 memo = [0] * (N+1)                                # N is the total weight
2 memo[0] = 0
3 for bag_weight in range(1,N+1):
4     for j in range(M):                            # this is to go through items
5         if items[j].weight <= bag_weight:
6             balance = bag_weight - items[j].weight
7             profit = item[j].profit + memo[balance]
8             if profit > memo[bag_weight]:    # if we have new optimal
9                 memo[bag_weight] = profit
10
11 return memo[N]
```

Kapsack Problem

Unbounded

- But is the code correct?

- Currently, we assume the maximum value is when you find items with a total weight of 12 kg
- What if we can't reach 12 kg?
- What if the optimal is at 10 kg instead of 12 kg?

Eventually....



Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	40	80	120	160	350	390	430	470	550	700	740	780

```
1 memo = [0] * (N+1)                                # N is the total weight
2 memo[0] = 0
3 for bag_weight in range(1,N+1):
4     for j in range(M):                            # this is to go through items
5         if items[j].weight <= bag_weight:
6             balance = bag_weight - items[j].weight
7             profit = item[j].profit + memo[balance]
8             if profit > memo[bag_weight]:    # if we have new optimal
9                 memo[bag_weight] = profit
10
11 return memo[N]
```

Kapsack Problem

Unbounded

- But is the code correct?

- Currently, we assume the maximum value is when you find items with a total weight of 12 kg
- What if we can't reach 12 kg?
- What if the optimal is at 10 kg instead of 12 kg?
- So what must we change?

```
1  memo = [0] * (N+1)                                # N is the total weight
2  memo[0] = 0
3  for bag_weight in range(1,N+1):
4      for j in range(M):                            # this is to go through items
5          if items[j].weight <= bag_weight:
6              balance = bag_weight - items[j].weight
7              profit = item[j].profit + memo[balance]
8              if profit > memo[bag_weight]:    # if we have new optimal
9                  memo[bag_weight] = profit
10
11 return memo[N]
```

Kapsack Problem

Unbounded

- But is the code correct?

- Currently, we assume the maximum value is when you find items with a total weight of 12 kg
 - What if we can't reach 12 kg?
 - What if the optimal is at 10 kg instead of 12 kg?
 - So what must we change?
 - `memo[i] = memo[i-1]` # copy the previous optimal

```
1  memo = [0] * (N+1)                                # N is the total weight
2  memo[0] = 0
3  for bag_weight in range(1,N+1):
4      for j in range(M):                            # this is to go through items
5          if items[j].weight <= bag_weight:
6              balance = bag_weight - items[j].weight
7              profit = item[j].profit + memo[balance]
8              if profit > memo[bag_weight]:    # if we have new optimal
9                  memo[bag_weight] = profit
10
11 return memo[N]
```

Kapsack Problem

Unbounded

- But is the code correct?

- Currently, we assume the maximum value is when you find items with a total weight of 12 kg
- What if we can't reach 12 kg?
- What if the optimal is at 10 kg instead of 12 kg?
- So what must we change?
 - `memo[i] = memo[i-1]` # copy the previous optimal
 - or, linear search through memo for the maximum

```
1  memo = [0] * (N+1)                                # N is the total weight
2  memo[0] = 0
3  for bag_weight in range(1,N+1):
4      for j in range(M):                            # this is to go through items
5          if items[j].weight <= bag_weight:
6              balance = bag_weight - items[j].weight
7              profit = item[j].profit + memo[balance]
8              if profit > memo[bag_weight]:    # if we have new optimal
9                  memo[bag_weight] = profit
10
11 return memo[N]
```

Questions?

Kapsack Problem

0/1 items

- Same problem, but you can't repeat the item

Kapsack Problem

0/1 items

- Same problem, but you can't repeat the item
 - So how would we solve it?

Kapsack Problem

0/1 items

- Same problem, but you can't repeat the item
 - So how would we solve it?
- This is where we can see the growing of problems...

Kapsack Problem

0/1 items

- Same problem, but you can't repeat the item
 - So how would we solve it?
- This is where we can see the growing of problems...
 - Grow from 0 weight to N weight

Kapsack Problem

0/1 items

- Same problem, but you can't repeat the item
 - So how would we solve it?
- This is where we can see the growing of problems...
 - Grow from 0 weight to N weight
 - Grow from a set of 0 items till M items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

Weight	0	1	2	3	4	5	6	7	8	9	10	11	12
Profit	0	?	?	?	?	?	?	?	?	?	?	?	?

Kapsack Problem

0/1 items

- We use a matrix

Increasing weight



	0	1	2	3	4	5	6	7	8	9	10	11	12
0													
A													
B													
C													
D													

Kapsack Problem

0/1 items

- We use a matrix

Increasing weight

Increasing items in set

	0	1	2	3	4	5	6	7	8	9	10	11	12
0													
A													
B													
C													
D													

Kapsack Problem

0/1 items

- We use a matrix

Increasing weight

Increasing items in set

	0	1	2	3	4	5	6	7	8	9	10	11	12
{}													
{A}													
{A,B}													
{A,B,C}													
{A,B,C,D}													

Kapsack Problem

0/1 items

- We use a matrix
 - So we fill up the matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12
0													
A													
B													
C													
D													

Kapsack Problem

0/1 items

- We use a matrix
 - Base cases

	0	1	2	3	4	5	6	7	8	9	10	11	12
0													
A													
B													
C													
D													

Kapsack Problem

0/1 items

- We use a matrix
 - Base cases
 - No item to choose from...

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A													
B													
C													
D													

Kapsack Problem

0/1 items

- We use a matrix
 - Base cases
 - No item to choose from...
 - Max weight is 0....

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0												
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix
 - So row by row...
 - Start with item A

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0												
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0												
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0							
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	?						
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230						
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	?					
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230					
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230				
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230			
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0												
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40											
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40										
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40							
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	40						
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	40	?					
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	40	270					
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	40	270	270				
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?
 - Is this correct?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	40	270	270	270	270	270	270
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?
 - Is this correct? Here, we can choose not to include B, having only A in bag

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	40	270	270	270	270	270	270
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?
 - Is this correct? Here, we can choose not to include B, having only A in bag

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0												
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	?											
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	?											
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40											
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40								
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350							
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390						
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	40	350	390	390				
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0												

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	?											

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390				

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550			

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590		

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	40	350	390	390	390	390	580	620
D	0	40	40	40	40	40	350	390	390	390	550	590	?

Kapsack Problem

0/1 items

- We use a matrix

- So row by row...
 - Start with item A
 - Should we add it?

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	40	350	390	390	390	390	580	620
D	0	40	40	40	40	40	350	390	390	390	550	590	620

Questions?

Kapsack Problem

0/1 items

- We use a matrix
 - What is the algorithm/ code?

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

Kapsack Problem

0/1 items

- We use a matrix
 - What is the algorithm/ code?

```
1      # for every row (item)
2      for i=1 to M:
3          # for every column (weight)
4          for j=1 to N:
5              # get the excluded at current weight (row above)
6              exclude = memo[i-1][j]
7              # calculate the include
8              include = 0
9              if item[i].weight <= j:
10                  include = item[i].value + memo[i-1][j-item[i].weight]
11              memo[i][j] = max(exclude,include)
```

Kapsack Problem

0/1 items

- We use a matrix
 - What is the algorithm/ code?
 - Complexity?

```
1      # for every row (item)
2      for i=1 to M:
3          # for every column (weight)
4          for j=1 to N:
5              # get the excluded at current weight (row above)
6              exclude = memo[i-1][j]
7              # calculate the include
8              include = 0
9              if item[i].weight <= j:
10                  include = item[i].value + memo[i-1][j-item[i].weight]
11              memo[i][j] = max(exclude,include)
```

Kapsack Problem

0/1 items

- We use a matrix
 - What is the algorithm/ code?
 - Complexity?
 - $O(NM)$ time from filling matrix
 - $O(NM)$ space for the matrix

```
1      # for every row (item)
2      for i=1 to M:
3          # for every column (weight)
4          for j=1 to N:
5              # get the excluded at current weight (row above)
6              exclude = memo[i-1][j]
7              # calculate the include
8              include = 0
9              if item[i].weight <= j:
10                 include = item[i].value + memo[i-1][j-item[i].weight]
11             memo[i][j] = max(exclude,include)
```

Kapsack Problem

0/1 items

- We use a matrix
 - What is the algorithm/ code?
 - Complexity?
 - $O(NM)$ time from filling matrix
 - $O(NM)$ space for the matrix... we can reduce this however

```
1      # for every row (item)
2      for i=1 to M:
3          # for every column (weight)
4          for j=1 to N:
5              # get the excluded at current weight (row above)
6              exclude = memo[i-1][j]
7              # calculate the include
8              include = 0
9              if item[i].weight <= j:
10                 include = item[i].value + memo[i-1][j-item[i].weight]
11             memo[i][j] = max(exclude,include)
```

Kapsack Problem

0/1 items

- We use a matrix
 - What is the algorithm/ code?
 - Complexity?
 - $O(NM)$ time from filling matrix
 - $O(NM)$ space for the matrix... we can reduce this however
We realize we do not need the entire matrix! We get the current value by looking at the row above only. So we can just store the latest 2 row...

Kapsack Problem

0/1 items

- We use a matrix
 - What is the algorithm/ code?
 - Complexity?
 - $O(NM)$ time from filling matrix
 - $O(NM)$ space for the matrix... we can reduce this however
 - We realize we do not need the entire matrix! We get the current value by looking at the row above only. So we can just store the latest 2 row...
 - Reducing complexity to $O(2N+M)$

Kapsack Problem

0/1 items

- We use a matrix
 - What is the algorithm/ code?
 - Complexity?
 - $O(NM)$ time from filling matrix
 - $O(NM)$ space for the matrix... we can reduce this however
 - We realize we do not need the entire matrix! We get the current value by looking at the row above only. So we can just store the latest 2 row...
 - Reducing complexity to $O(2N+M)$
 - But in reality, we can't do this space saving... because we need it to reconstruct the solution...

Questions?

Take a break!

Kapsack Problem

0/1 items

- So what are the items?

Kapsack Problem

0/1 items

- So what are the items?

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

Kapsack Problem

0/1 items

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive).

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

Kapsack Problem

0/1 items

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive).

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	390	550	590	620

Kapsack Problem

0/1 items

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). If **same value**, means we **do not include...**

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	390	550	590	620

Kapsack Problem

0/1 items

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). If **same value**, means we **do not include...**

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	390	550	590	620

- {}

Kapsack Problem

0/1 items

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). If **different value**, means we **include...**

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C}

Kapsack Problem

0/1 items

- So what are the items?

- Recall that we compare the current value (inclusive) with the value in the row above (exclusive). If **different value**, means we **include...**
- Then we update to the suitable weight of the included item

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C}

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?

- Recall that we compare the current value (inclusive) with the value in the row above (exclusive). If **different value**, means we **include**...
- Then we update to the suitable weight of the included item

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C}

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?

- Recall that we compare the current value (inclusive) with the value in the row above (exclusive). If **different value**, means we **include**...
- Then we update to the suitable weight of the included item

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C}

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). So now do the same...

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C}

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). So now do the same...

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C,B}

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). So now do the same...

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C,B}

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). So now do the same...

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C,B}

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). So now do the same...

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C,B,A}

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). So now do the same...

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- {C,B,A} makes up the item for total value of 620

Kapsack Problem

0/1 items

Item	A	B	C	D
Weight	6kg	1kg	5kg	9kg
Value	\$230	\$40	\$350	\$550

- So what are the items?
 - Recall that we compare the current value (inclusive) with the value in the row above (exclusive). So now do the same...

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	230	230	230	230	230	230	230
B	0	40	40	40	40	40	230	270	270	270	270	270	270
C	0	40	40	40	40	350	390	390	390	390	390	580	620
D	0	40	40	40	40	350	390	390	390	550	590	590	620

- $\{C, B, A\}$ makes up the item for total value of 620
- This is what we call **backtracking!**

Questions?

Backtracking

Reconstruction solution

- We often need to reconstruct solutions

Backtracking

Reconstruction solution

- We often need to reconstruct solutions
 - Coin change = what are the coins?

Backtracking

Reconstruction solution

- We often need to reconstruct solutions
 - Coin change = what are the coins?
 - Knapsack = what are the items?

Backtracking

Reconstruction solution

- We often need to reconstruct solutions
 - Coin change = what are the coins?
 - Knapsack = what are the items?
 - Edit distance = what are the modifications made (insert/delete/replace)

Backtracking

Reconstruction solution

- We often need to reconstruct solutions
 - Coin change = what are the coins?
 - Knapsack = what are the items?
 - Edit distance = what are the modifications made (insert/delete/replace)
 - When we update the optimal value, we can store the decision we made...

Backtracking

Reconstruction solution

- We often need to reconstruct solutions
 - Coin change = what are the coins?
 - Knapsack = what are the items?
 - Edit distance = what are the modifications made (insert/delete/replace)
 - When we update the optimal value, we can store the decision we made... But this **waste a lot of memory** as we need to store decisions/combinations at every step!

Backtracking

Reconstruction solution

- We often need to reconstruct solutions
 - Coin change = what are the coins?
 - Knapsack = what are the items?
 - Edit distance = what are the modifications made (insert/delete/replace)
 - When we update the optimal value, we can store the decision we made... But this **waste a lot of memory** as we need to store decisions/combinations at every step!
 - So, we leave bread crumbs to backtrack

Backtracking

Reconstruction solution

- We often need to reconstruct solutions
 - Coin change = what are the coins?
 - Knapsack = what are the items?
 - Edit distance = what are the modifications made (insert/delete/replace)
 - When we update the optimal value, we can store the decision we made... But this **waste a lot of memory** as we need to store decisions/combinations at every step!
 - So, we leave **bread crumbs to backtrack** or only the final **decision made!**

Questions?

Backtracking

Coin change

- Let say we store our decisions...

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2

Backtracking

Coin change

- Let say we store our decisions...

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2
The coins	{}	1	1, 1	1, 1	1, 1	5	6	6, 1	6, 1	9	5, 5	6, 5	6, 6

Backtracking

Coin change

- Let say we store our decisions...
 - Space complexity?

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2
The coins	{}	1	1, 1	1, 1	1, 1	5	6	6, 1	6, 1	9	5, 5	6, 5	6, 6

Backtracking

Coin change

- Let say we store our decisions...
 - Space complexity? Can be $O(N^2)$

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2
The coins	{}	1	1, 1	1, 1	1, 1	5	6	6, 1	6, 1	9	5, 5	6, 5	6, 6

Backtracking

Coin change

- Let say we store our decisions...
 - Space complexity? Can be $O(N^2)$
 - Improve it further?

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2
The coins	{}	1	1, 1	1, 1	1, 1	5	6	6, 1	6, 1	9	5, 5	6, 5	6, 6

Backtracking

Coin change

- Let say we store our decisions...
 - Space complexity? Can be $O(N^2)$
 - Improve it further? Remember the last coin you added

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2
The coins	{}	1	1	1	1	5	6	1	1	9	5	6	6

Backtracking

Coin change

- Let say we store our decisions...
 - Space complexity? Can be $O(N^2)$
 - Improve it further? Remember the last coin you added
So space complexity now?

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2
The coins	{}	1	1	1	1	5	6	1	1	9	5	6	6

Backtracking

Coin change

- Let say we store our decisions...
 - Space complexity? Can be $O(N^2)$
 - Improve it further? Remember the last coin you added
So space complexity now? $O(N)$

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2
The coins	{}	1	1	1	1	5	6	1	1	9	5	6	6

Backtracking

Coin change

- Let say we store our decisions...
 - Space complexity? Can be $O(N^2)$
 - Improve it further? Remember the last coin you added
So space complexity now? $O(N)$
- Coins = {6,6}

Value	0	1	2	3	4	5	6	7	8	9	10	11	12
Number of coins	0	1	2	3	4	1	1	2	3	1	2	2	2
The coins	{}	1	1	1	1	5	6	1	1	9	5	6	6



Questions?

Backtracking

vs Decision Array

- Backtracking save space
- Decision array save time

Backtracking

vs Decision Array

- Backtracking save space
 - Less auxiliary space
 - Same space complexity
- Decision array save time

Backtracking

vs Decision Array

- Backtracking save space
 - Less auxiliary space
 - Same space complexity
- Decision array save time
 - Faster
 - But time complexity lies in finding the solution still...

Questions?

Edit Distance

Cost to convert string

- Edit-distance
- We will skip this since this is similar to the Knapsack really...

Edit Distance

Cost to convert string

- Edit-distance
- We will skip this since this is similar to the Knapsack really...
 - Will be covered in the tutorial, linking it up with the longest common subsequence (LCS) problem there

- Edit-distance
- We will skip this since this is similar to the Knapsack really...
 - Will be covered in the tutorial, linking it up with the longest common subsequence (LCS) problem there
 - Note: Nathan's slide at the end

Questions?

Summary

Dynamic Programming Algorithm (DPA)

- Take problem
- Break it down

Summary

Dynamic Programming Algorithm (DPA)

- Take problem
- Break it down
 - Sub-problem has optimal solution
 - Sub-problem overlap (thus can be reused)

Summary

Dynamic Programming Algorithm (DPA)

- Take problem
- Break it down
 - Sub-problem has optimal solution
 - Sub-problem overlap (thus can be reused)
 - Store these solutions (memoization) for faster computing

Summary

Dynamic Programming Algorithm (DPA)

- Take problem
- Break it down
 - Sub-problem has optimal solution
 - Sub-problem overlap (thus can be reused)
 - Store these solutions (memoization) for faster computing
- Reconstruct solution
 - Decision array
 - Backtracking

Summary

Dynamic Programming Algorithm (DPA)

- Take problem
- Break it down
 - Sub-problem has optimal solution
 - Sub-problem overlap (thus can be reused)
 - Store these solutions (memoization) for faster computing
- Reconstruct solution
 - Decision array
 - Backtracking

Questions?

Thank You

Edit Distance

- The words **computer** and **commuter** are very similar, and a change of just one letter, **p** → **m**, will change the first word into the second.
- The word **sport** can be changed into **sort** by the deletion of **p**, or equivalently, **sort** can be changed into **sport** by the insertion of **p'**.
- Notion of **editing** provides a simple and handy formalisation to compare two strings.
- The goal is to convert the first string (i.e., sequence) into the second through a series of edit operations
- The permitted edit operations are:
 1. **insertion** of a symbol into a sequence.
 2. **deletion** of a symbol from a sequence.
 3. **substitution** or replacement of one symbol with another in a sequence.

Edit Distance

Edit distance between two sequences

- Edit distance is the **minimum number of edit operations** required to convert one sequence into another

For example:

- Edit distance between **computer** and **commuter** is 1
- Edit distance between **sport** and **sort** is 1.
- Edit distance between **shine** and **sings** is ?
- Edit distance between **d_na_sgivethis** and **d_en_ts_gnawstrims** is ?

Some Applications of Edit Distance

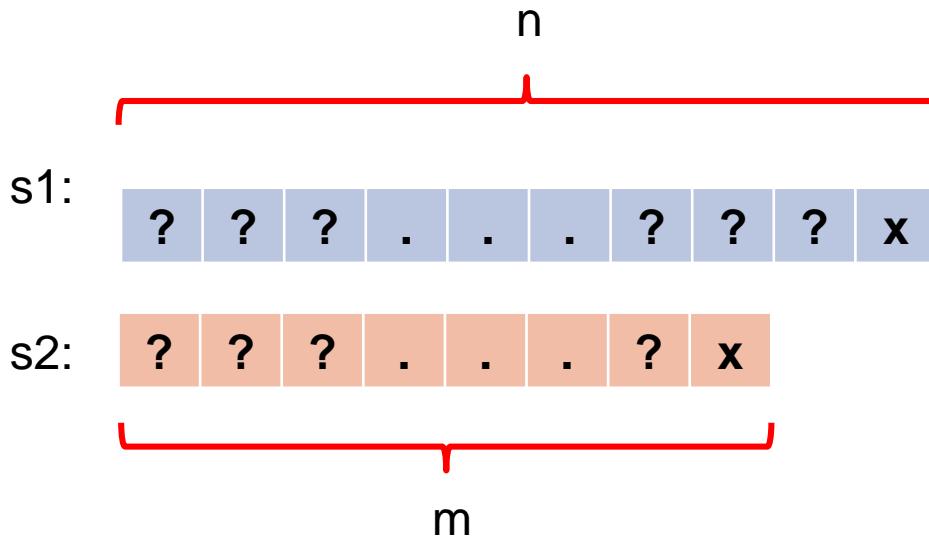
- Natural Language Processing
 - Auto-correction
 - Query suggestions
- Bioinformatics
 - DNA/Protein sequence alignment

Computing Edit Distance

We want to convert s_1 to s_2 containing n and m letters, respectively

To gain an intuition for this problem, lets look at some situations we might run into

This is a good technique in general, try playing around with the problem and see what happens



How much does it cost to turn s_1 into s_2 if the last characters are **the same?**

We can leave the last character, and just convert the front part of one string into the front part of the other

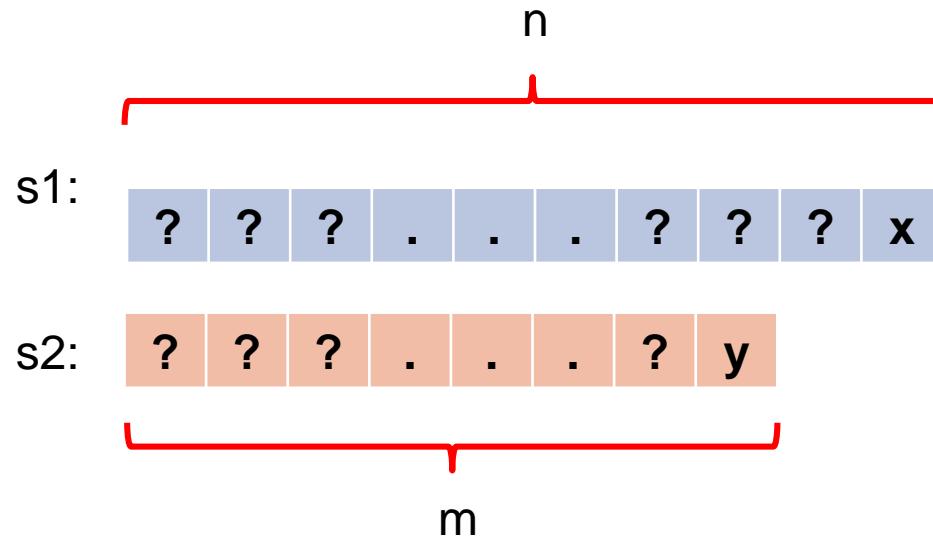
$$\begin{aligned} \text{edit}(s_1[1..n], s_2[1..m]) \\ =\text{edit}(s_1[1..n-1], s_2[1..m-1]) \end{aligned}$$

Computing Edit Distance

We want to convert s_1 to s_2 containing n and m letters, respectively

To gain an intuition for this problem, lets look at some situations we might run into

This is a good technique in general, try playing around with the problem and see what happens



How much does it cost to turn s_1 into s_2 if the last characters are **different**?

We have some options

Computing Edit Distance

- Remember! We can assume that we have solved ALL subproblems already. In other words, we know
- $\text{Edit}(s1[1..i], s2[1..j])$ for all $i \leq n, j \leq m$ **BUT NOT** when $i = n$ AND $j = m$ (since this is the exact problem we are trying to solve)
- Alternatively, we could think about it visually
- In this table, $\text{cell}[i][j]$ is the cost of turning $s1[1..i]$ into $s2[1..j]$

	1	2	3	.	.	.	m
1							
2							
3							
.							
.							
.							
n							

Known:

Unknown:



Computing Edit Distance

We know:

$\text{Edit}(s1[1..i], s2[1..j])$ for all $i \leq n, j \leq m$

BUT NOT $i = n$ AND $j = m$

Equivalently, we know all the **blue** cells

From the bottom right corner, there are three things we can do:

- Go up
- Go left
- Go left and up

	1	2	3	.	.	.	m
1							
2							
3							
.							
.							
.							
n							

Computing Edit Distance

We know:

$\text{Edit}(s1[1..i], s2[1..j])$ for all $i \leq n, j \leq m$

BUT NOT $i = n$ AND $j = m$

Equivalently, we know all the blue cells

	1	2	3	.	.	.	m
1							
2							
3							
.							
.							
.							
n							

From the bottom right corner, there are three things we can do:

- Go up
- Go left
- Go left and up

Going up:

Subproblem: $\text{edit}(s1[1..n-1], s2[1..m])$

- First delete $s1[n]$
- Then turn $s1[1..n-1]$ into $s2[1..m]$

Total cost:

$\text{cost(delete)} + \text{edit}(s1[1..n-1], s2[1..m])$

Computing Edit Distance

We know:

$\text{Edit}(s1[1..i], s2[1..j])$ for all $i \leq n$, $j \leq m$

BUT NOT $i = n$ AND $j = m$

Equivalently, we know all the blue cells

	1	2	3	.	.	.	m
1							
2							
3							
.							
.							
.							
n							

From the bottom right corner, there are three things we can do:

- Go up
- Go left
- Go left and up

Going left:

Subproblem: $\text{edit}(s1[1..n], s2[1..m-1])$

- First turn $s1[1..n]$ into $s2[1..m-1]$
- First insert $s2[m]$ at the end of $s2$

Total cost:

$\text{edit}(s1[1..n], s2[1..m-1]) + \text{cost(insert)}$

Computing Edit Distance

We know:

$\text{Edit}(s1[1..i], s2[1..j])$ for all $i \leq n$, $j \leq m$

BUT NOT $i = n$ AND $j = m$

Equivalently, we know all the blue cells

	1	2	3	.	.	.	m
1							
2							
3							
.							
.							
.							
n							

From the bottom right corner, there are three things we can do:

- Go up
- Go left
- Go left and up

Going left:

Subproblem: $\text{edit}(s1[1..n-1], s2[1..m-1])$

- Replace $s1[n]$ with $s2[m]$
- Turn $s1[1..n-1]$ into $s2[1..m-1]$

Total cost:

$\text{edit}(s1[1..n-1], s2[1..m-1]) + \text{cost}(\text{replace})$

Computing Edit Distance

Base cases?

- When one string is empty, the cost is just the length of the other string
- we would have to insert each character in the other string, starting from nothing
- So $\text{edit}(s1[], s2[1..j]) = j$
- $\text{edit}(s1[1..i], s2[]) = i$

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \end{cases}$$

Computing Edit Distance

If the last characters are the same:

- $\text{edit}(s_1[1..n-1], s_2[1..m-1])$

If the last characters are different, three options:

- cost(delete) + $\text{edit}(s_1[1..n-1], s_2[1..m])$
- $\text{edit}(s_1[1..n], s_2[1..m-1]) + \text{cost(insert)}$
- $\text{edit}(s_1[1..n-1], s_2[1..m-1]) + \text{cost(replace)}$

We want the minimum cost, so our optimal substructure will be (if all costs are 1)

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ						
S						
I						
N						
G						
S						

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S						
I						
N						
G						
S						

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1					
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1					
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0				
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0				
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1			
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1			
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2		
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2					
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1				
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	A choice?			
N	3					
G	4					
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1			
N	3					
G	4		Now	you	Try!	
S	5					

Example

$$\text{Dist}[i, j] = \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min \begin{cases} \text{Dist}[i - 1, j - 1] + 1_{S_1[i] \neq S_2[j]} \\ \text{Dist}[i - 1, j] + 1 \\ \text{Dist}[i, j - 1] + 1 \end{cases} & \text{otherwise} \end{cases}$$

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Outline

1. Introduction to Dynamic Programming
2. Coins Change
3. Unbounded Knapsack
4. 0/1 Knapsack
5. Edit Distance
6. Constructing Optimal Solution

Constructing optimal solutions

- The algorithms we have seen determine optimal values, e.g.,
 - Minimum number of coins
 - Maximum value of knapsack
 - Edit distance
- How do we construct optimal solution that gives the optimal value, e.g.,
 - The coins to give the change
 - The items to put in knapsack
 - Converting one string to the other
- There may be multiple optimal solutions and our goal is to return just one solution!
- Two strategies can be used.
 1. Create an additional array recording decision at each step
 2. Backtracking

Decision Array

- Make a second array of the same size
- Each time you fill in a cell of the memo array, record your decision in the decision array
- Remember, going right (or coming from the left) is insert
- Going down (or coming from up) is delete
- Going down and right (or coming from up and left) is replace OR do nothing

	Φ	S	H	I	N	E
Φ	0					
S						
I						
N						
G						
S						

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S				
S						
I						
N						
G						
S						
	Φ	S	H	I	N	E
Φ	0	1				
S						
I						
N						
G						
S						

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H			
S						
I						
N						
G						
S						
	Φ	S	H	I	N	E
Φ	0	1	2			
S						
I						
N						
G						
S						

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I		
S						
I						
N						
G						
S						
	Φ	S	H	I	N	E
Φ	0	1	2	3		
S						
I						
N						
G						
S						

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	
S						
I						
N						
G						
S						
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	
S						
I						
N						
G						
S						

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S						
I						
N						
G						
S						
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S						
I						
N						
G						
S						

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S					
I	Delete I					
N	Delete N					
G	Delete G					
S	Delete S					
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1					
I	2					
N	3					
G	4					
S	5					

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing				
I	Delete I					
N	Delete N					
G	Delete G					
S	Delete S					
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0				
I	2					
N	3					
G	4					
S	5					

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H			
I	Delete I					
N	Delete N					
G	Delete G					
S	Delete S					
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1			
I	2					
N	3					
G	4					
S	5					

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I		
I	Delete I					
N	Delete N					
G	Delete G					
S	Delete S					
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2		
I	2					
N	3					
G	4					
S	5					

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	
I	Delete I					
N	Delete N					
G	Delete G					
S	Delete S					
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	
I	2					
N	3					
G	4					
S	5					

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I					
N	Delete N					
G	Delete G					
S	Delete S					
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2					
N	3					
G	4					
S	5					

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I				
N	Delete N					
G	Delete G					
S	Delete S					
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1				
N	3					
G	4					
S	5					

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Choice?	Choice?	Delete G	replace G, E
S	Delete S	Delete S	Choice?	Choice?	Delete S	Choice?

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S
	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Decision Array

	Φ	S	H	I	N	E
Φ	null	Insert S	Insert H	Insert I	Insert N	Insert E
S	Delete S	Do nothing	Insert H	Insert I	Insert N	Insert E
I	Delete I	Delete I	replace I,H	Do nothing	Insert N	Insert E
N	Delete N	Delete N	replace N,H	Delete N	Do nothing	Insert E
G	Delete G	Delete G	Delete G	Delete G	Delete G	replace G, E
S	Delete S	Delete S	Delete S	Delete S	Delete S	Delete S

Sequence of operations:

Delete S (from position 5)

replace G with E (at position 4)

insert H (at position 2)

- SINGS
- SING
- SINE
- SHINE

Backtracking

- Start in bottom right
- Are the letters the same?

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Start in bottom right
- Are the letters the same?
- No

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Start in bottom right
- Are the letters the same?
- No
- From recurrence...
- We know that our current value (3) was obtained from any of the three previous cells by adding 1

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Start in bottom right
- Are the letters the same?
- No
- From recurrence...
- We know that our current value (3) was obtained from any of the three previous cells by adding 1
- So our options are up or up-and-left
- Choose one arbitrarily

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue from our new cell (but remember the path)

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue from our new cell (but remember the path)
- Letters are different

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue from our new cell (but remember the path)
- Letters are different
- Must have come from the cell above

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Letters are the same
- From our recurrence, we know
 - IF our value is the same as the up-and-left cell, then we could have come from there
 - IF our value is one more than the up cell or the left cell, then we could have come from there
- In this case, we came from up-and-left

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Letters are the same
- From our recurrence, we know
 - IF our value is the same as the up-and-left cell, then we could have come from there
 - IF our value is one more than the up cell or the left cell, then we could have come from there
- In this case, we came from up-and-left

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way
- When you reach the top left cell, you are done
- So the sequence of operations is

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

The diagram illustrates a backtracking path through a grid of cells. The grid has columns labeled Φ, S, H, I, N, and E, and rows labeled Φ, S, I, N, G, and S. The cells contain numerical values in green. Red arrows show the sequence of moves: starting at (S, 3), moving to (N, 2), then (G, 2), then (S, 3), then (I, 1), then (H, 1), then (S, 0), and finally (Φ, 0).

Backtracking

- Continue in this way
- When you reach the top left cell, you are done
- So the sequence of operations is
- Replace(5, E)

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way
- When you reach the top left cell, you are done
- So the sequence of operations is
- Replace(5, E), delete[4]

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way
- When you reach the top left cell, you are done
- So the sequence of operations is
- Replace(5, E), delete[4], nothing

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way
- When you reach the top left cell, you are done
- So the sequence of operations is
- Replace(5, E), delete[4], nothing, nothing

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

The diagram illustrates a backtracking search path on a 7x7 grid. The grid contains values representing the cost or state of each cell. Red arrows indicate the direction of the search path. The cells are colored based on their value: blue for 0, red for 1, light blue for 2, green for 3, and yellow for 4 or 5. The path starts at the bottom-right cell (S) and moves towards the top-left cell (Phi).

Backtracking

- Continue in this way
- When you reach the top left cell, you are done
- So the sequence of operations is
- Replace(5, E), delete[4], nothing, nothing, insert(2, H)

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

The diagram shows a 6x7 grid of numbers. Red arrows indicate a path from the bottom-right cell (5, E) to the top-left cell (0, Φ). The path starts at (5, E), moves up to (3, N), then right to (2, G), up to (1, I), right to (0, S), up to (1, Φ), and finally right to (0, Φ).

Backtracking

- Continue in this way
- When you reach the top left cell, you are done
- So the sequence of operations is
- Replace(5, E), delete[4], nothing, nothing, insert(2, H), nothing

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3

Backtracking

- Continue in this way
- When you reach the top left cell, you are done
- So the sequence of operations is
- Replace(5, E), delete[4], nothing, nothing, insert(2, H), nothing
- SINGS
- SINGE (replace position 5 with E)
- SINE (delete G in position 4)
- SHINE (insert H at position 2)

	Φ	S	H	I	N	E
Φ	0	1	2	3	4	5
S	1	0	1	2	3	4
I	2	1	1	1	2	3
N	3	2	2	2	1	2
G	4	3	3	3	2	2
S	5	4	4	4	3	3