Dynamic Programming

2/6/25

Announcements

- Homework 4 due March 18th
- Quiz 3 + Quiz 2 retake today
  - I was surprised more people didn’t retake Quiz 1 last time
- Spring Break!

Harder Example

Write a recursive algorithm to find the minimum number of coins to make change.

To be more specific...

GIVEN:

D: A list of denominations of coins (e.g. 5, 10, 12, 15)

l: the length of the denominations list (number of possible coins)

a: An amount of money you want to make change for (integer value)

RETURN:

m: The minimum number of coins to make change for *a*

We tried this with a greedy algorithm (choose largest coin)

It worked for some denominations:

E.g. 1, 5, 10, 25, 50

It did not work for others.

E.g. 1, 3, 6, 12, 24, 30

Simpler example: 1, 4, 6

Let’s try recursion for a different approach.

1. What's the simplest possible case?

Denominations: 1, 4, 6

1. What's the simplest possible case?

Denominations: 1, 4, 6

Making change for 0 cents: return 0 coins

2. Play around and visualize!

Denominations: 1 4 6

0:

1: 1

2: 1 1

3: 1 1 1

4: 4

5: 4 1

6: 6

7: 6 1

8: 4 4

9: 4 4 1

10: 6 4

11: 6 4 1

12: 6 6

13: 6 6 1

14: 6 4 4

15: 6 4 4 1

16: 6 6 4

17: 6 6 4 1

18: 6 6 6

2. Play around and visualize! (consider non-optimal solutions too!)

Denominations: 1 4 6

0:

1: 1

2: 1 1

3: 1 1 1

4: 4

5: 4 1

6: 6

7: 6 1

8: 4 4

9: 4 4 1

10: 6 4

11: 6 4 1

12: 6 6

13: 6 6 1, 4 4 1 1

14: 6 4 4, 6 6 1 1, 4 4 1 1 1

15: 6 4 4 1 , 6 6 1 1 1

16: 6 6 4, 6 4 4 1 1

17: 6 6 4 1 , 6 4 4 1 1 1

18: 6 6 6, 6 6 4 1 1, 6 4 4 4

Note: these examples are not exhaustive

3. Relate hard cases to simpler cases

Denominations: 1 4 6

0:

1: 1

2: 1 1

3: 1 1 1

4: 4

5: 4 1

6: 6

7: 6 1

8: 4 4

9: 4 4 1

10: 6 4

11: 6 4 1

12: 6 6

13: 6 6 1

14: 6 4 4

15: 6 4 4 1

16: 6 6 4

17: 6 6 4 1

18: 6 6 6

3. Relate hard cases to simpler cases

Denominations: 1 4 6

0:

1: 1

2: 1 1

3: 1 1 1

4: 4

5: 4 1

6: 6

7: 6 1

8: 4 4

9: 4 4 1

10: $\boxed{6\ 4}$

11: $\boxed{6\ 4}$ 1

12: 6 6

13: 6 6 1

14: $\boxed{6\ 4}$ 4

15: $\boxed{6\ 4}$ 4 1

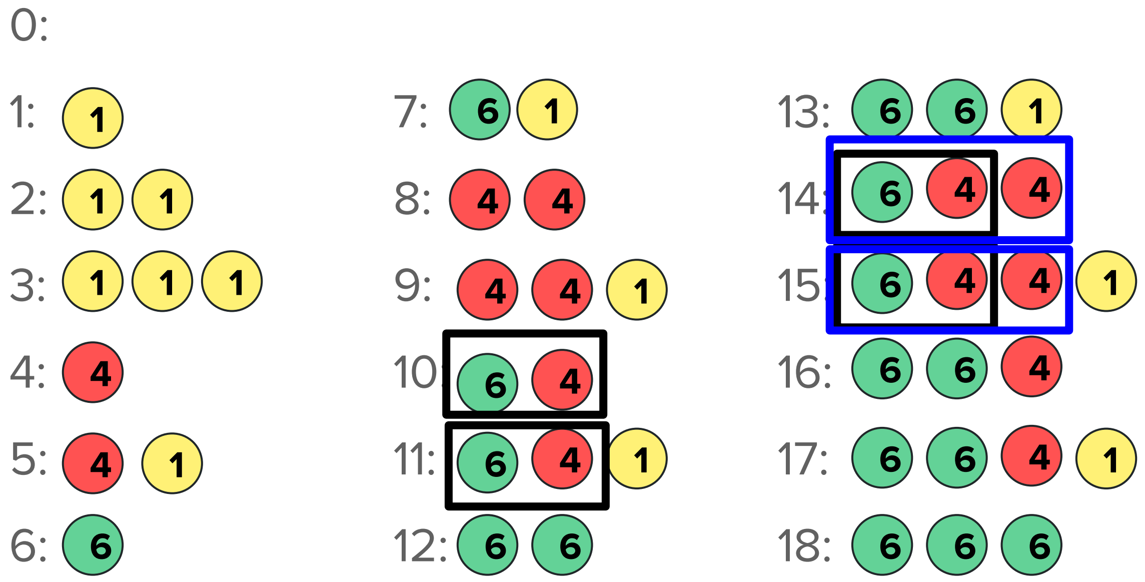
16: 6 6 4

17: 6 6 4 1

18: 6 6 6

3. Relate hard cases to simpler cases

Denominations: 1 4 6



4. Generalize the pattern

4. Generalize the pattern

If our denominations are D_0 through D_n, we can make change for amount X by adding coin D_i to a solution for amount X - D_i

Recall that we are minimizing the number of coins used.

So: change(X) = 1 + min([change[X - D_i] for i = 0 to n])

More formally

Denominations: 1, 4, 6

```
  F(n) = 1 + min(F(n-1), F(n-4), F(n-6))
```

5. Write code

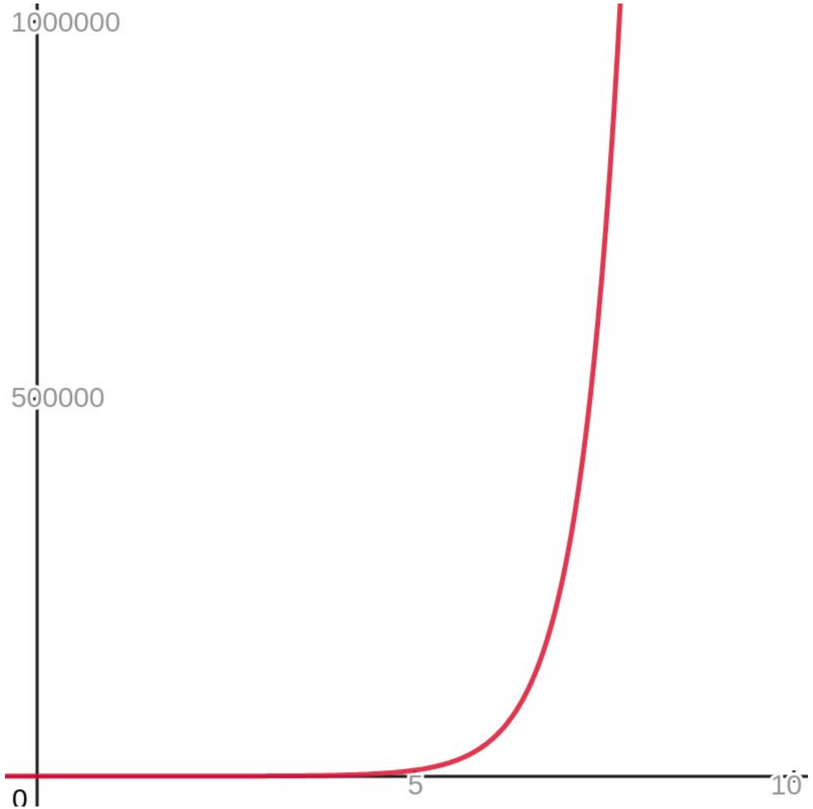
Dynamic programming

What is the time complexity of this algorithm?

Denominations: 1, 3, 6, 12, 24, 30

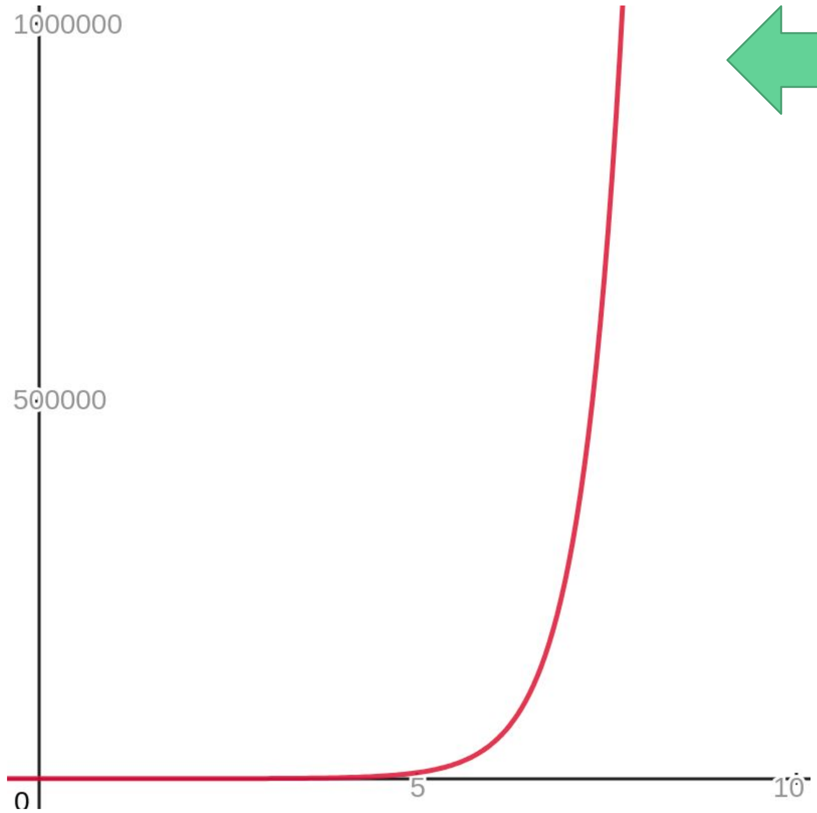
```
def change(amt, num_coins, denom_list):
    if amt == 0:
        return 0

    # subtract the largest possible value out of the amount
    # calculate previous amount
    result = math.inf
    for i in range(num_coins):
        if denom_list[i] <= amt:
            new_amt = amt - denom_list[i]
            result_for_this_coin = change(new_amt, num_coins, denom_list)
            if result_for_this_coin + 1 < result:
                result = result_for_this_coin + 1
    return result
```

6^n



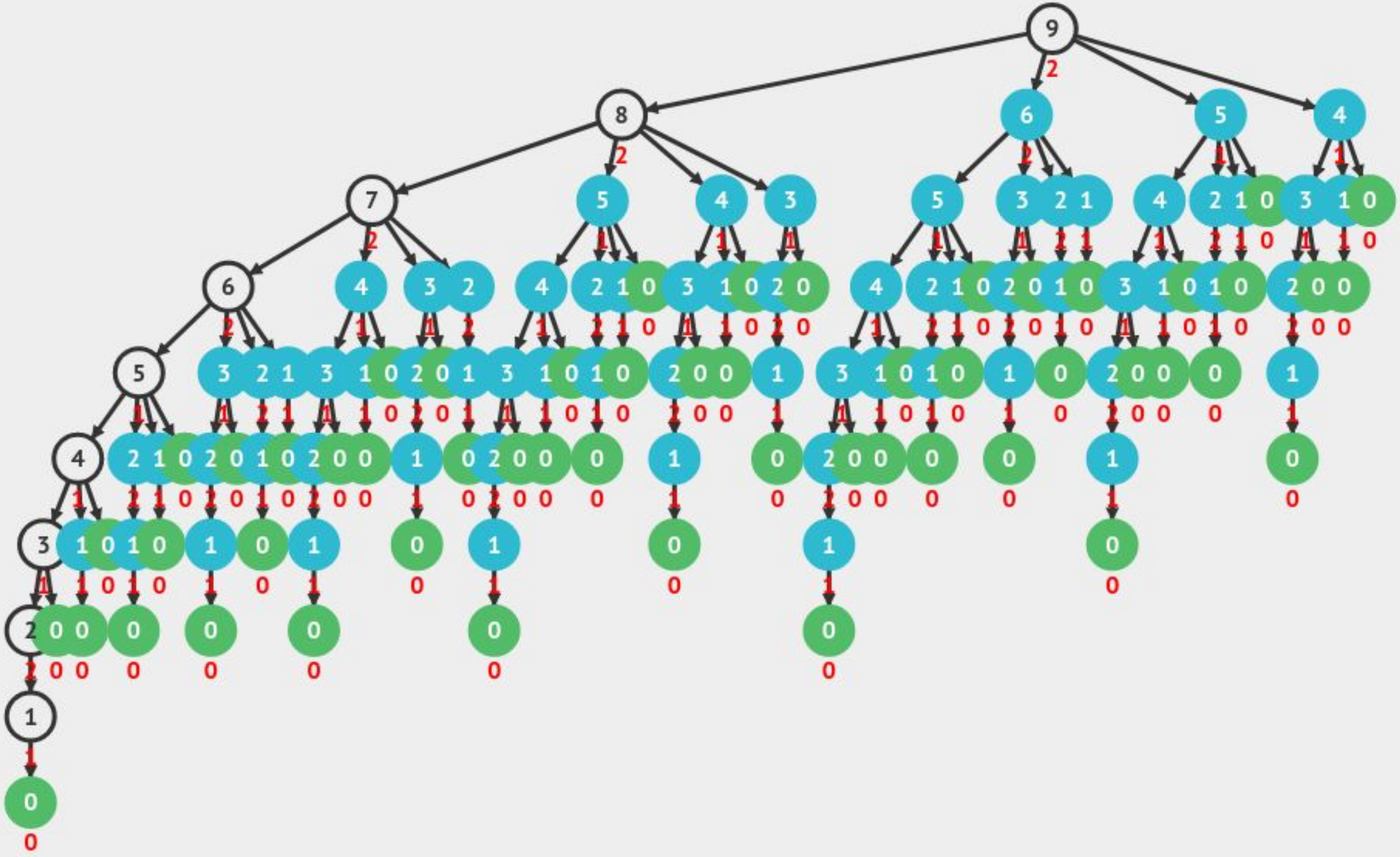
https://www.desmos.com/calculator/bqxwpkdv8u

6^n



Really bad

https://www.desmos.com/calculator/bqxwpkdv8u https://visualgo.net/en/recursion



How can we improve?

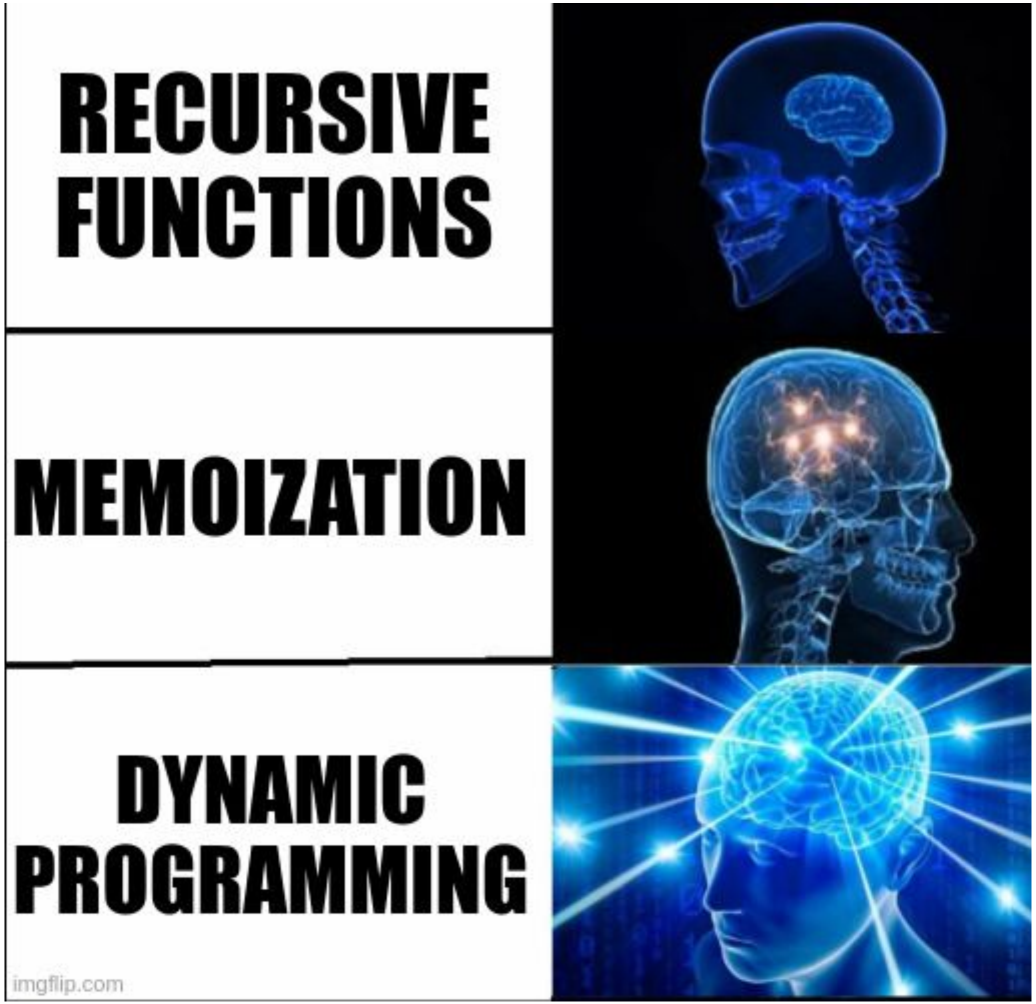
Memoization

- Keep a hash table mapping inputs to results
- What is the time complexity?

Memoization

- Keep a hash table mapping inputs to results
- O(n*m)-ish time complexity ➜ huge improvement!
- Still a fair amount of overhead
  - Recursive calls
  - Cache performance
  - Hash collisions

Can we do even better?


RECURSIVE FUNCTIONS
MEMOIZATION
DYNAMIC PROGRAMMING
imgflip.com

Dynamic programming

- Yes, the name is super uninformative
- Basic idea: store answers in a table
- Populate table in an order that ensures you always have the necessary answers to smaller problems
- Works on the same principles as recursion, but without recursion

Conceptual steps in dynamic programming

1. Formulate your problem recursively
2. Show that the number of different instances of your recurrence is bounded by a polynomial.
3. Specify an order of evaluation for the recurrence so you always have what you need.

Implementation steps

1. Base case ➜ initialization of array
2. Body of function ➜ body of loop (arguments ➜ loop variables)
3. Recursive calls ➜ array lookups
4. Returns ➜ store in array

Equivalence between recursion and dynamic programming

1. **Base case → initialization of array**
2. **Body of function → body of loop**
    - **(arguments → loop variables)**
3. **Recursive calls → array lookups**
4. **Returns → store in array**

```
def fib(n):
    if n <= 1 :
        return n
  | return fib(n - 1) + fib(n - 2)
```

```
def fib(n):
    table = [] * (n + 1)
    table[0] = 0
    table[1] = 1

    for i in range(2, n+1):
    |   table[i] = table[i-1] + table[i-2]
    return table[n]
```

Demo

Table representation of dynamic programming

N	0	1	2	3	4	5	6	7	8	9
Answer										

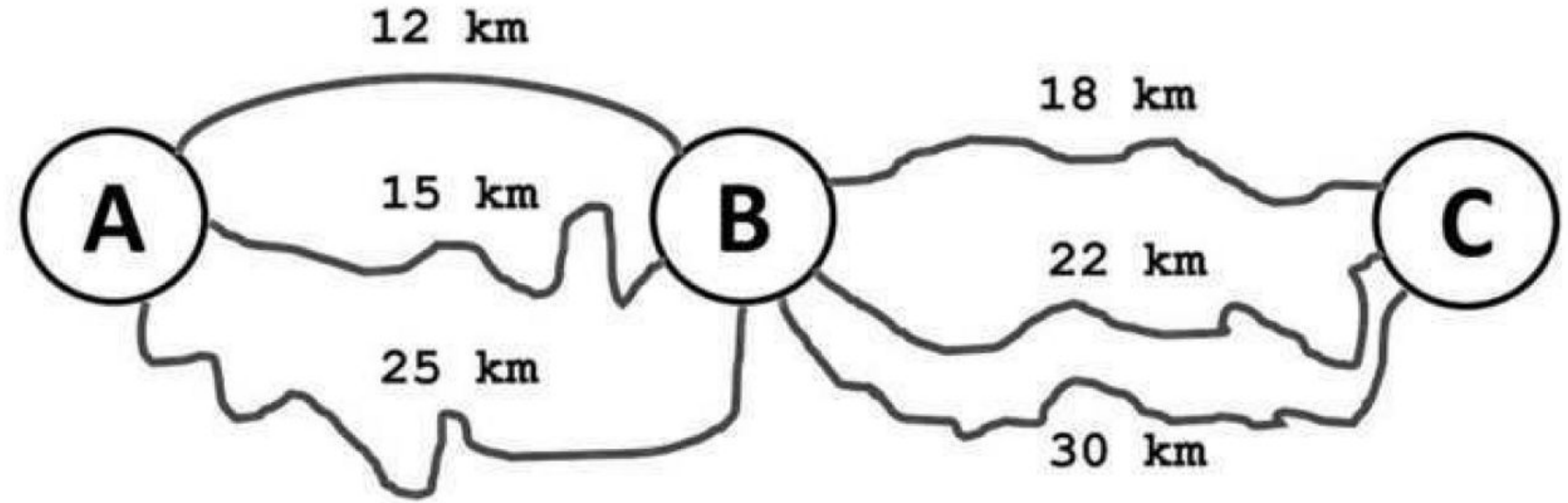
Denominations: 1, 2, 4

Practice problem: fill out the table for these denominations

N	0	1	2	3	4	5	6	7	8	9
Answer										

Denominations: 1, 3, 5

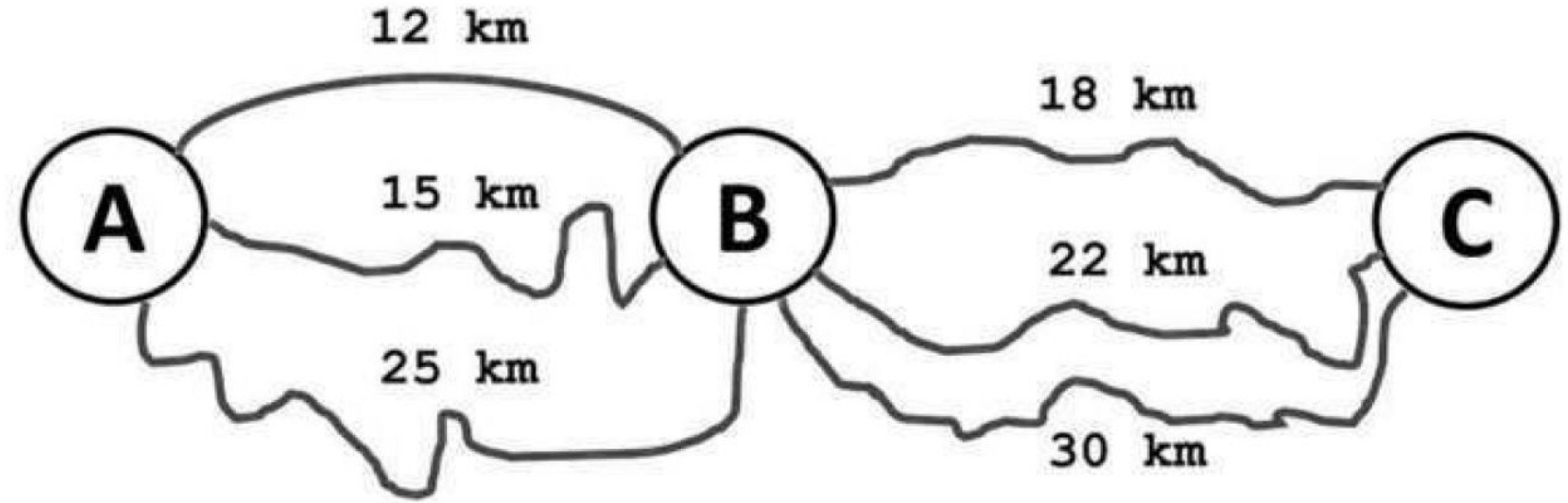
When can we use dynamic programming?



Optimal substructure: The details of our past solutions won't affect our current solution.

Image source: Tarun Kumar

When can we use dynamic programming?



The principle of optimality: “An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.“

When can we use dynamic programming?

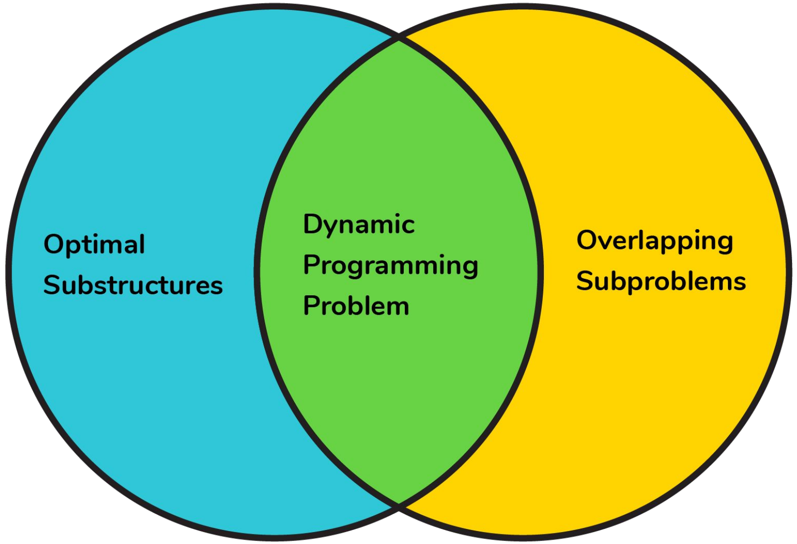


Image source: interviewbit

When can we use dynamic programming?

Intuitively, these circumstances often occur when we are dealing with ordered sequences

Intuition building

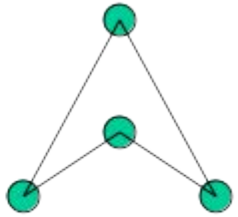
Can we use dynamic programming to calculate N!?

Intuition building

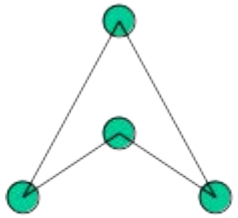
Can we use dynamic programming for the coin change problem?

Intuition building

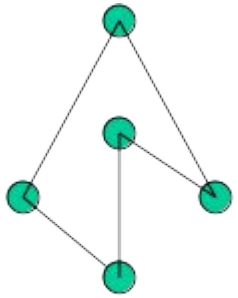
Can we use dynamic programming for the following problem: given a set of points on an Cartesian plane, find the shortest path that visits all points?



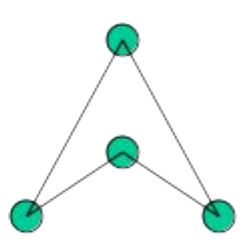
T(4)

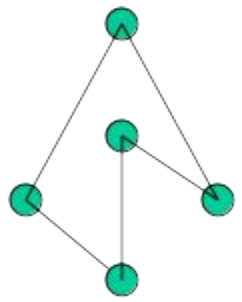


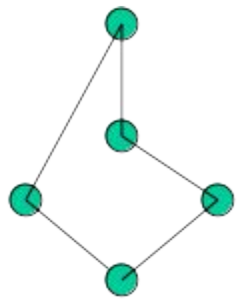
T(4)



T(5)


T(4)


T(5)


Shortest Tour

In-class problem

Convert our recursive coin change implementation from last class into dynamic programming

```
def change(amt, num_coins, denom_list):
    if amt == 0:
        return 0

    # subtract the largest possible value out of the amount
    # calculate previous amount
    result = math.inf
    for i in range(num_coins):
        if denom_list[i] <= amt:
            new_amt = amt - denom_list[i]
            result_for_this_coin = change(new_amt, num_coins, denom_list)
            if result_for_this_coin + 1 < result:
                result = result_for_this_coin + 1
    return result
```

Intuition building

Recall the problem where Little Red Riding Hood is choosing fruits to bring to her grandmother.

- Her basket has a limited capacity. Within this capacity, she wants to choose the fruits her grandmother will like the most.
- If she can cut the fruit into arbitrarily small units, a greedy algorithm will work.
- If she cannot, a greedy algorithm will not work.

Will dynamic programming work on variant where fruits cannot be cut?

The knapsack problem

The classic Knapsack problem is typically put forth as:

A thief breaks into a store and wants to fill their knapsack with as much value in goods as possible before making their escape. Given the following list of items available, what should they take?

- Item A: weight = w_A , value = v_A
- Item B, weight = w_B , value = v_B
- Item C, weight = w_C , value = v_C

  …

The Simplest Versions…

Can items be divided up such that only a portion is taken?

The thief can hold 5 pounds and has to choose from:

3 pounds of gold dust at $379.22/pound

6 pounds of silver dust at $188.89/pound

1/9 pound of platinum dust at $433.25/pound

The Simplest Versions…

Can items be divided up such that only a portion is taken?

The thief can hold 5 pounds and has to choose from:

3 pounds of gold dust at $379.22/pound

6 pounds of silver dust at $188.89/pound

1/9 pound of platinum dust at $433.25/pound

Are all of the weights or total values identical?

The thief breaks into a ring shop where all of the rings weigh 1oz. He can hold 12 ounces; which should he take?

A Deceptively Hard Version…

What if each item has the same price/pound?

A Deceptively Hard Version…

What if each item has the same price/pound?

This problem reduces to the bin-packing problem: we want to fit as many pounds of material into the knapsack as possible.

How can we approach this problem?

Example

The thief breaks into a gold refinery; he can steal from a selection of raw gold nuggets, each of the same value per pound. If he can carry 50 pounds, what selection would maximize the amount he carries out?

47.3 pounds	6.0 pounds	5.2 pounds
36.7 pounds	5.6 pounds	5.2 pounds
25.5 pounds	5.6 pounds	5.0 pounds
16.7 pounds	5.4 pounds	3.2 pounds
8.8 pounds	5.3 pounds	0.25 pounds

An Easier Version...

What if all of the sizes we are working with are *relatively small* integers? For example, if we could fit 10 pounds and:

Object A is 2 pounds and worth $40
Object B is 3 pounds and worth $50
Object C is 1 pound and worth $100
Object D is 5 pounds and worth $95
Object E is 3 pounds and worth $30

We can use dynamic programming!

First step: formulate the problem recursively

What are the subproblems? How can we make this problem simpler?

First step: formulate the problem recursively

What are the subproblems? How can we make this problem simpler?

$$F(i, w) = \max([F(i - 1, w), V_i + F(i - 1, w - w_i), F(i, w - 1)])$$

The solution...

$w_A = 2$ $v_A = \$40$

$w_B = 3$ $v_B = \$50$

$w_C = 1$ $v_C = \$100$

$w_D = 5$ $v_D = \$95$

$w_E = 3$ $v_E = \$30$

Items

Weight

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

The solution...

$w_A = 2 \quad v_A = \$40$

$w_B = 3 \quad v_B = \$50$

$w_C = 1 \quad v_C = \$100$

$w_D = 5 \quad v_D = \$95$

$w_E = 3 \quad v_E = \$30$

Items

Weight

	A	B	C	D	E
1	$0	$0	$100	$100	$100
2	$40	$40	$100	$100	$100
3	$40	$50	$140	$140	$140
4	$40	$50	$150	$150	$150
5	$40	$90	$150	$150	$150
6	$40	$90	$190	$195	$195
7	$40	$90	$190	$195	$195
8	$40	$90	$190	$235	$235
9	$40	$90	$190	$245	$245
10	$40	$90	$190	$245	$245

Discussion Problem

Draw the dynamic programming table for the following instance of the knapsack problem:

$W_A = 4$	$V_A = 2$
$W_B = 1$	$V_B = 1$
$W_C = 2$	$V_C = 3$

Weight limit: 5

	A	B	C
1			
2			
3			
4			
5			

The hardest situation...

What if our problem just isn't so neat?

$w_A = 2$	$v_A = \$40$
$w_B = \pi$	$v_B = \$50$
$w_C = 1.98$	$v_C = \$100$
$w_D = 5$	$v_D = \$95$
$w_E = 3$	$v_E = \$30$

We have to resort to brute force….

Discussion problem

You are a given a sentence. You need to choose words from it such that you get the most possible letters but no two adjacent words are selected.

How can you solve this problem with dynamic programming?