CSCI 232:
Data Structures and Algorithms

Greedy Algorithms + Tractability

Reese Pearsall
Spring 2024

Announcements

Lab 12 due **Sunday** at 11:59 PM

Greedy Algorithms

Technique to solve a problem that involves making the choice the ***best helps some objective***

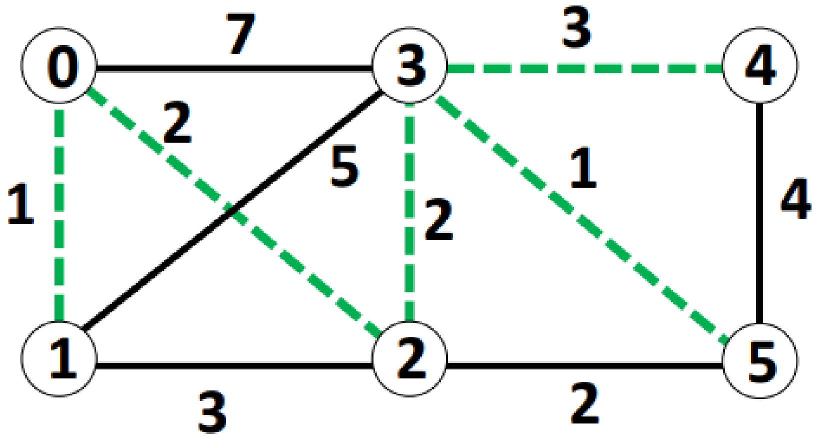
Objective = shortest cost, most profit, spend least money as possible

We (usually) do not look ahead, plan, or revisit past decisions

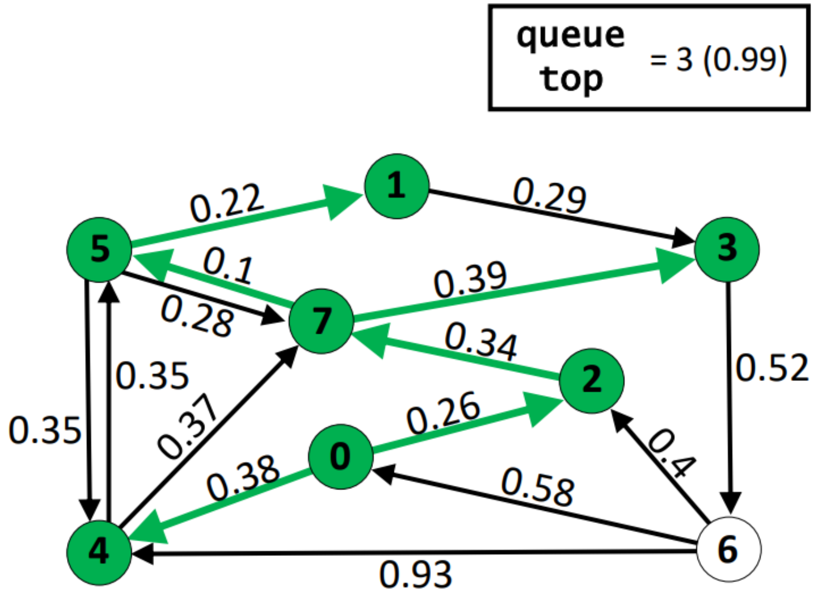
Hope that optimal local choices lead to optimal global solutions

Sometimes the greedy approach is not the best solution a problem

Greedy Algorithms



Running time O(|E|log|E|)



	Distance from 0	Previous vertex	Priority queue
0	0	-	
1	0.92	5	
2	0.26	0	
3	0.99	7	
4	0.38	0	
5	0.70	7	
6	∞		
7	0.60	2	

Running time O(|E|*log|V|)

Kruskal’s and Dijkstra’s algorithm are both examples of greedy algorithms

At each step of the algorithm, they attempt to select the edge with the **minimum** cost

(The greedy approach works fine for these, because these algorithms always return the **optimal** result)

Eating at a Buffet

Suppose you pay D dollars to enter a buffer. You can eat only N items before you get full. You know the cost of every item in the buffet

$31

$11

$16

$40

$18

$23

Our goal is to get the most “bang for our buck”,

aka. maximize
$C = (D - S)$ where S is the sum of the N items we ate at the buffet

Eating at a Buffet

Suppose you pay D dollars to enter a buffer. You can eat only N items before you get full. You know the cost of every item in the buffet

$31

$11

$16

$40

$18

$23

Our goal is to get the most “bang for our buck”,

aka. maximize
`C = (D – S)` where S is the sum of the N items we ate at the buffet

Ideas?

Eating at a Buffet

Suppose you pay D dollars to enter a buffer. You can eat only N items before you get full. You know the cost of every item in the buffet

$40

$31

$23

$18

$16

$11

Our goal is to get the most "bang for our buck",

aka. maximize

`C = (D – S)` where S is the sum of the N items we ate at the buffet

1. Sort items by their value (greatest-to-least)

Eating at a Buffet

Suppose you pay D dollars to enter a buffer. You can eat only N items before you get full. You know the cost of every item in the buffet

$40

$31

$23

$18

$16

$11

Our goal is to get the most “bang for our buck”,

aka. maximize
C = (D – S) where S is the sum of the N items we ate at the buffet

1. Sort items by their value (greatest-to-least)

2. Select the first N items in the list

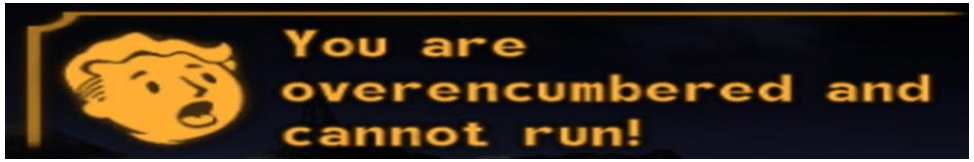
N = 3, S = $94, D = $40, C = $54

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don’t overfill knapsack

Knapsack Problem



You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (10)

Value: 10
Weight: 5

Value: 40
Weight: 4

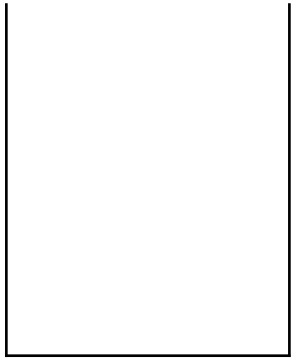
Value: 30
Weight: 6

Value: 50
Weight: 3

Suppose our knapsack can only hold 10 pounds

Ideas?

Knapsack Problem



Knapsack (0)

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Value: 10
Weight: 5

Value: 40
Weight: 4

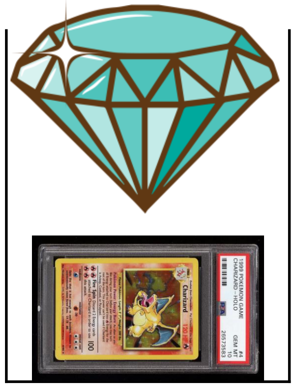
Value: 30
Weight: 6

Value: 50
Weight: 3

Suppose our knapsack can only hold 10 pounds

Stuff our knapsack with the most expensive items until we can't fit anymore

Knapsack Problem



Knapsack (7)

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Value: 10
Weight: 5

Value: 40
Weight: 4

Value: 30
Weight: 6

Value: 50
Weight: 3

Suppose our knapsack can only hold 10 pounds

Stuff our knapsack with the most expensive items until we can't fit anymore

Total value = $90

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (0)

Value: 10
Weight: 5

Value: 40
Weight: 4

Value: 30
Weight: 6

Value: 50
Weight: **10**

Suppose our knapsack can only hold 10 pounds

Stuff our knapsack with the most expensive items until we can't fit anymore

Knapsack Problem

Knapsack (10)

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Value: 10
Weight: 5

Value: 40
Weight: 4

Value: 30
Weight: 6

Value: 50
Weight: **10**

Suppose our knapsack can only hold 10 pounds

Stuff our knapsack with the most expensive items until we can't fit anymore

Taking these two items instead yields a more optimal solution!

Total value = $50

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (0)

Value: 10
Weight: 5

Value: 40
Weight: 4

Value: 30
Weight: 6

Value: 50
Weight: **10**

Suppose our knapsack can only hold 10 pounds

Any better ideas?

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (0)

Value: 10
Weight: 5

Value: 40
Weight: 4

Value: 30
Weight: 6

Value: 50
Weight: **10**

Suppose our knapsack can only hold 10 pounds

Compute the **ratio** of value/weight, and select items based on that

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (0)

Value: 10
Weight: 5
Ratio: 2

Value: 40
Weight: 4
Ratio: 10

Value: 30
Weight: 6
Ratio: 5

Value: 50
Weight: **10**
Ratio: 5

Compute the **ratio** of value/weight, and select items based on that

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (0)

Value: 40
Weight: 4
Ratio: 10

Value: 50
Weight: **10**
Ratio: 5

Value: 30
Weight: 6
Ratio: 5

Value: 10
Weight: 5
Ratio: 2

1. Sort items based on ratio

Compute the **ratio** of value/weight, and select items based on that

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (0)

Value: 40
Weight: 4
Ratio: 10

Value: 50
Weight: **10**
Ratio: 5

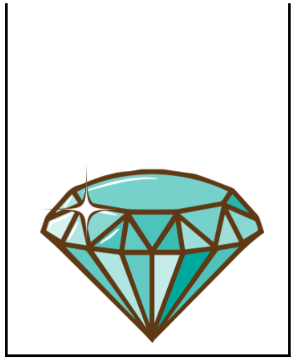
Value: 30
Weight: 6
Ratio: 5

Value: 10
Weight: 5
Ratio: 2

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

Knapsack Problem



Knapsack (4)

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Value: 40
Weight: 4
Ratio: 10

Value: 50
Weight: **10**
Ratio: 5

Value: 30
Weight: 6
Ratio: 5

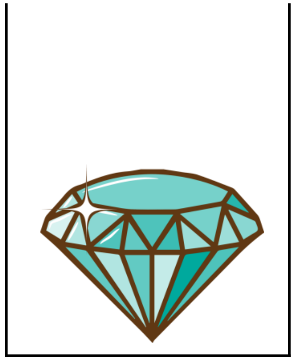
Value: 10
Weight: 5
Ratio: 2

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value



Knapsack (4)

Goal: Maximize value of items being stolen, and don't overfill knapsack

alue: 40
Weight: 4
Ratio: 10

Value: 50
Weight: **10**
Ratio: 5

Value: 30
Weight: 6
Ratio: 5

Value: 10
Weight: 5
Ratio: 2

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

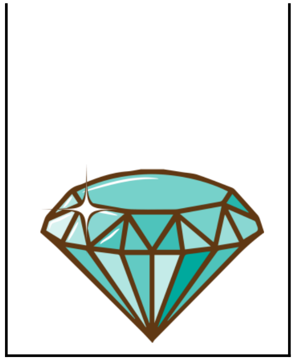
Compute the **ratio** of value/weight, and select items based on that

We cannot select this item, because it will exceed the knapsack

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack



Knapsack (4)

Value: 40
Weight: 4
Ratio: 10

Value: 50
Weight: **10**
Ratio: 5

Value: 30
Weight: 6
Ratio: 5

Value: 10
Weight: 5
Ratio: 2

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

Knapsack Problem

Knapsack (10)

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Value: 40
Weight: 4
Ratio: 10

Value: 50
Weight: **10**
Ratio: 5

Value: 30
Weight: 6
Ratio: 5

Value: 10
Weight: 5
Ratio: 2

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Knapsack (10)

Goal: Maximize value of items being stolen, and don't overfill knapsack

Value: 40
Weight: 4
Ratio: 10

Value: 50
Weight: **10**
Ratio: 5

Value: 30
Weight: 6
Ratio: 5

Value: 10
Weight: 5
Ratio: 2

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

We cannot select this item, because it will exceed the knapsack

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (10)

Value: 40
Weight: 4
Ratio: 10

Value: 50
Weight: **10**
Ratio: 5

Value: 30
Weight: 6
Ratio: 5

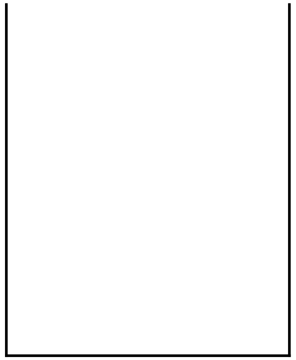
Value: 10
Weight: 5
Ratio: 2

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

Total profit of knapsack: $40 + $30 = **$70**

Knapsack Problem



Knapsack (0)

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Value: 5.5
Weight: 4
Ratio: 1.38

Value: 4
Weight: **3**
Ratio: 1.33

Value: 4
Weight: 3
Ratio: 1.33

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

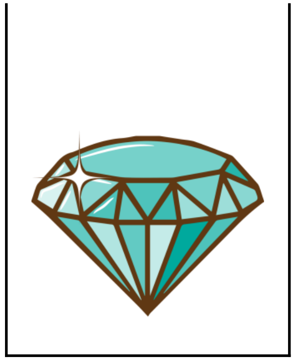
Let N = 6

Given these new prices, weights, and ratios, will our algorithm still work?

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack



Knapsack (4)

Value: 5.5
Weight: 4
Ratio: 1.38

Value: 4
Weight: **3**
Ratio: 1.33

Value: 4
Weight: 3
Ratio: 1.33

We can't add these items, because they would exceed our knapsack capacity

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

Let N = 6

Given these new prices, weights, and ratios, will our algorithm still work?

Total profit = 5.5

Knapsack Problem

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack

Knapsack (4)

Value: 5.5
Weight: 4
Ratio: 1.38

Value: 4
Weight: **3**
Ratio: 1.33

Value: 4
Weight: 3
Ratio: 1.33

1. Sort items based on ratio
2. Add items to knapsack if they will not exceed the knapsack
3. Repeat step 2 until we've checked every item

Compute the **ratio** of value/weight, and select items based on that

Let N = 6

Given these new prices, weights, and ratios, will our algorithm still work?

Total profit = 5.5

Optimal solution= 8

Knapsack Problem

You

a st

k (4)

This is the 0/1 knapsack problem, which means that we either take the item, or we don't. We can't take "half" of a watch to fill the remaining empty space of our knapsack

The greedy approach **does not** always yield the optimal solution ☹

Com
value
items based on that

algorithm still work?

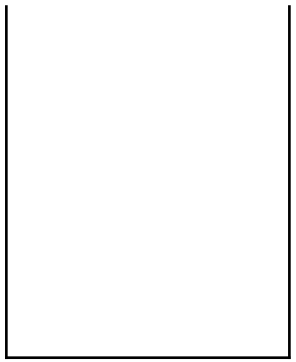
Total profit = 5.5

Optimal solution= 8

Knapsack Problem (Fractional)

You are a thief with a knapsack that can hold up to **N** weight. You are robbing a store, where each item has a weight, and a value

Goal: Maximize value of items being stolen, and don't overfill knapsack



Knapsack (n)

Fractional variant = we can take a fraction of an item

Knapsack Problem (Fractional)

Value: 40
Weight: 5

Value: 100
Weight: 20

Value: 13
Weight: 10

Value: 52
Weight: 8

Value: 88
Weight: 12

Value: 13
Weight: 8

Value: 20
Weight: 1

Knapsack (n)

Knapsack capacity: 35

Knapsack Problem (Fractional)

Value: 40
Weight: 5
Ratio: 8

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 10:
Ratio 1.3

Value: 52
Weight: 8
Ratio: 6.5

Value: 88
Weight: 12
Ratio: 7.33

Value: 13
Weight: 8
Ratio: 1.625

Value: 20
Weight: 1
Ratio: 20

Knapsack (n)

Knapsack capacity: 35

1. Sort items based on ratio

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack (n)

Knapsack capacity: 35

1. Sort items based on ratio

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack (n)

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
    1. If adding item will not exceed the capacity, add it
    2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack (0)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

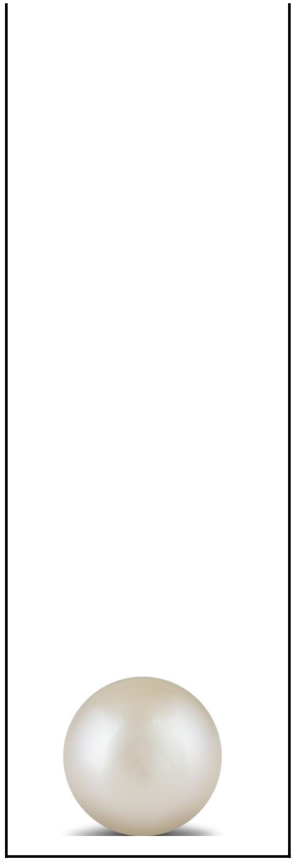
Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
    1. If adding item will not exceed the capacity, add it
    2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack



Knapsack (1)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

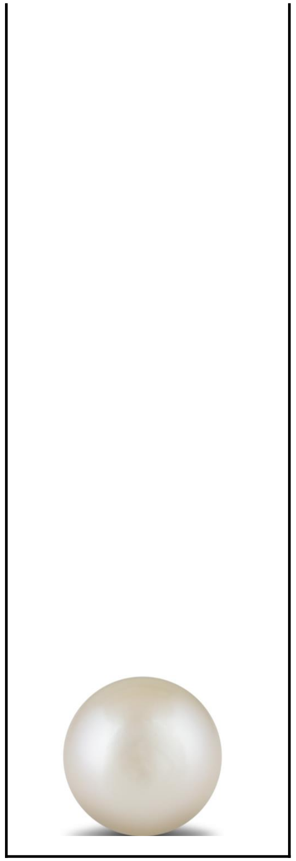
Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
    1. If adding item will not exceed the capacity, add it
    2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack



Knapsack (1)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack (6)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack (6)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
    1. If adding item will not exceed the capacity, add it
    2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack (18)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
    1. If adding item will not exceed the capacity, add it
    2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack (18)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack (26)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack (26)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

Knapsack (26)

1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

We cannot take the full 20 pounds of the gold bar, but **we can** take a fraction of it

Cut off 9 pounds of the gold bar, and place it in out knapsack

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Knapsack (35)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Pearl: 20
Emerald: 40
Silver: 88
Sapphire: 52
9/20 of a gold bar: ???

Knapsack (35)

Knapsack Problem (Fractional)

Value: 20
Weight: 1
Ratio: 20

Value: 40
Weight: 5
Ratio: 8

Value: 88
Weight: 12
Ratio: **7.33**

Value: 52
Weight: 8
Ratio: 6.5

Value: 100
Weight: 20
Ratio: 5

Value: 13
Weight: 8
Ratio: 1.625

Value: 13
Weight: 10:
Ratio 1.3

Knapsack capacity: 35

1. Sort items based on ratio
2. Iterate through sorted list
    1. If adding item will not exceed the capacity, add it
    2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Pearl: 20
Emerald: 40
Silver: 88
Sapphire: 52
the ratio
9/20 of a gold bar: 9 * 5 = 45

Knapsack (35)

Knapsack Problem (Fractional)

Value: 20	Value: 40	Value: 88	Value: 52	Value: 100	Value: 13	Value: 13
Weight: 1	Weight: 5	Weight: 12	Weight: 8	Weight: 20	Weight: 8	Weight: 10:
Ratio: 20	**Ratio: 8**	Ratio: **7.33**	**Ratio: 6.5**	**Ratio: 5**	**Ratio: 1.625**	**Ratio 1.3**

Knapsack capacity: 35

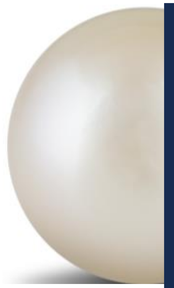
1. Sort items based on ratio
2. Iterate through sorted list
   1. If adding item will not exceed the capacity, add it
   2. If adding item will exceed the capacity, take a fraction of it to fill the remaining space of knapsack

Pearl: 20
Emerald: 40
Silver: 88
Sapphire: 52
the ratio
9/20 of a gold bar: 9 * 5 = 45

Total profit: $245

Knapsack (35)

Knapsack Problem (Fractional)



Value
Weigh
Ratio:

Knap

1. So
2. Ite
   1.
   2.
   of it to fill the remaining space of knapsack

In the **fractional** knapsack problem, the greedy approach **will** guarantee an optimal solution

Total profit: $245

Knapsack (35)

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

3 min 7 min 15 min 1 min 5 min

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

3 min   7 min   15 min   1 min   5 min

Any ideas to achieve our goal?

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

3 min   7 min   15 min   1 min   5 min

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

3 min

7 min

15 min

5 min

Select the customer that would take the least time

Being a Cashier at Walmart

3 min

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

7 min

15 min

5 min

Select the customer that would take the least time

Being a Cashier at Walmart

3 min

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

2 min

7 min

15 min

5 min

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

2 min 7 min 15 min 5 min

Select the customer that would take the least time

Being a Cashier at Walmart

2 min

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

7 min

15 min

5 min

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

2 min

6 min

7 min

15 min

8 min

5 min

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

5 min

6 min

7 min

15 min

8 min

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

6 min | 7 min | 15 min | 8 min

Select the customer that would take the least time

Being a Cashier at Walmart

6 min

7 min

15 min

8 min

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Select the customer that would take the least time

Being a Cashier at Walmart

7 min

15 min

8 min

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

7 min

11 min

15 min

8 min

Select the customer that would take the least time

And repeat this until your shift is over!

Being a Cashier at Walmart

7 min

11 min

15 min

8 min

Is this algorithm good?

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Select the customer that would take the least time

Being a Cashier at Walmart

Customer are constantly arriving to a check out line. Walmart is open 24/7

Instead of first-in-first-out, you want to serve **as many customers as possible** in your 2-hour shift

You can assume that there will always be several waiting in line, and you know how many minutes it will take to check them out

Select the customer that would take the least time

Optimal, but not Fair!

This customer has a longer service time, and they may potentially wait **a very long** time until they are served

~~Being a Cashier at Walmart~~ CPU Job Scheduling

This problem is very relevant in the world of **operating systems**.

There are many processes/software running on your computer all at the same time

Process A	Process B	Process C	Process D
11 nanoseconds	15 nanoseconds	8 nanoseconds	7 nanoseconds

Each process needs to use the hardware on the computer to do its job.

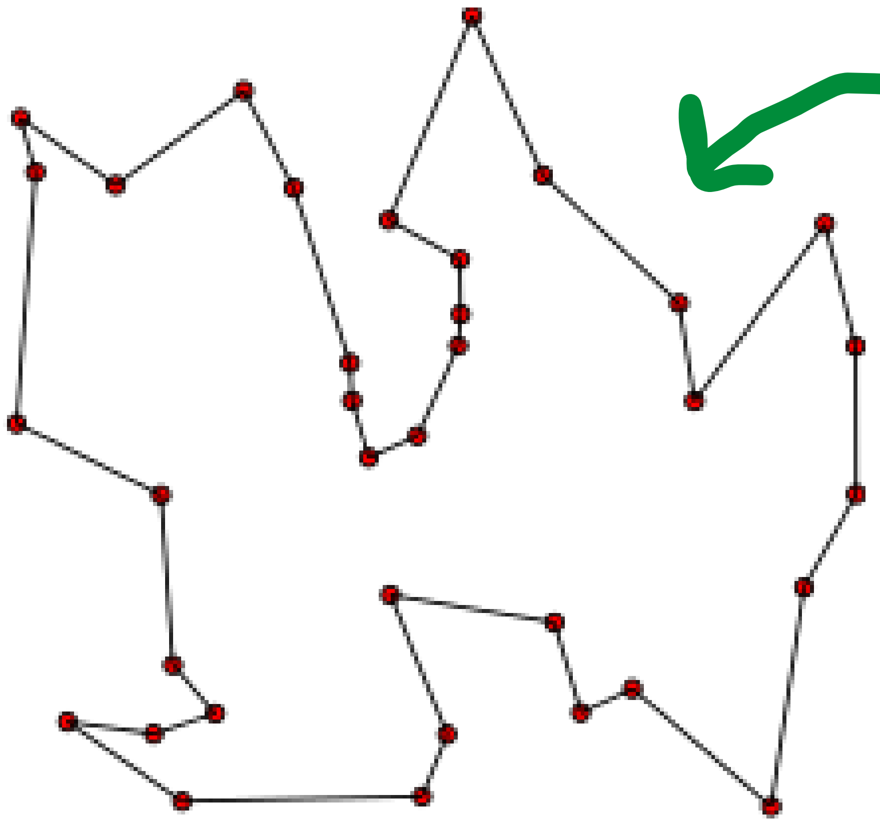
OS oversees selecting which job will be processed by the CPU next

Ideally, we want a fair approach ☺

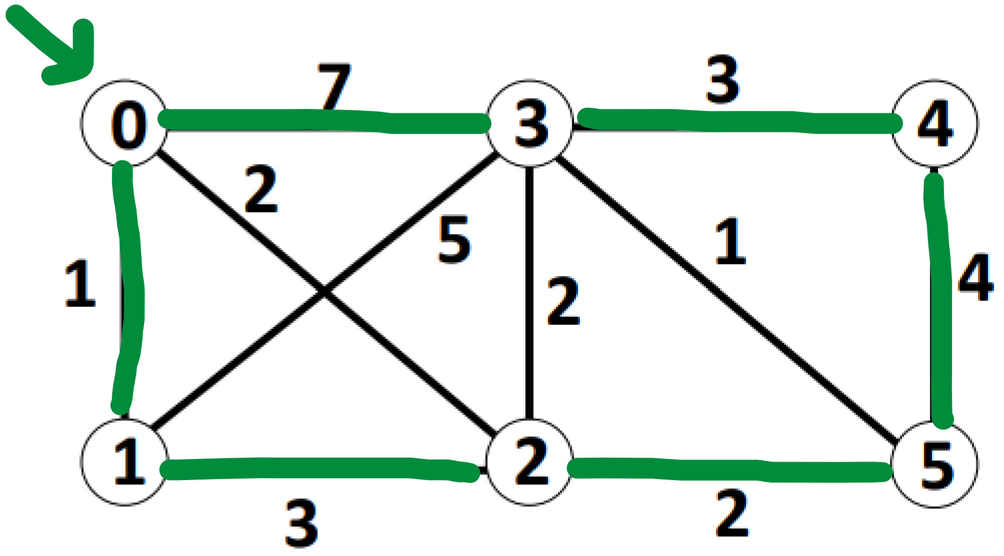
Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node, without visiting the same node twice**

<u>Hamiltonian cycle</u>

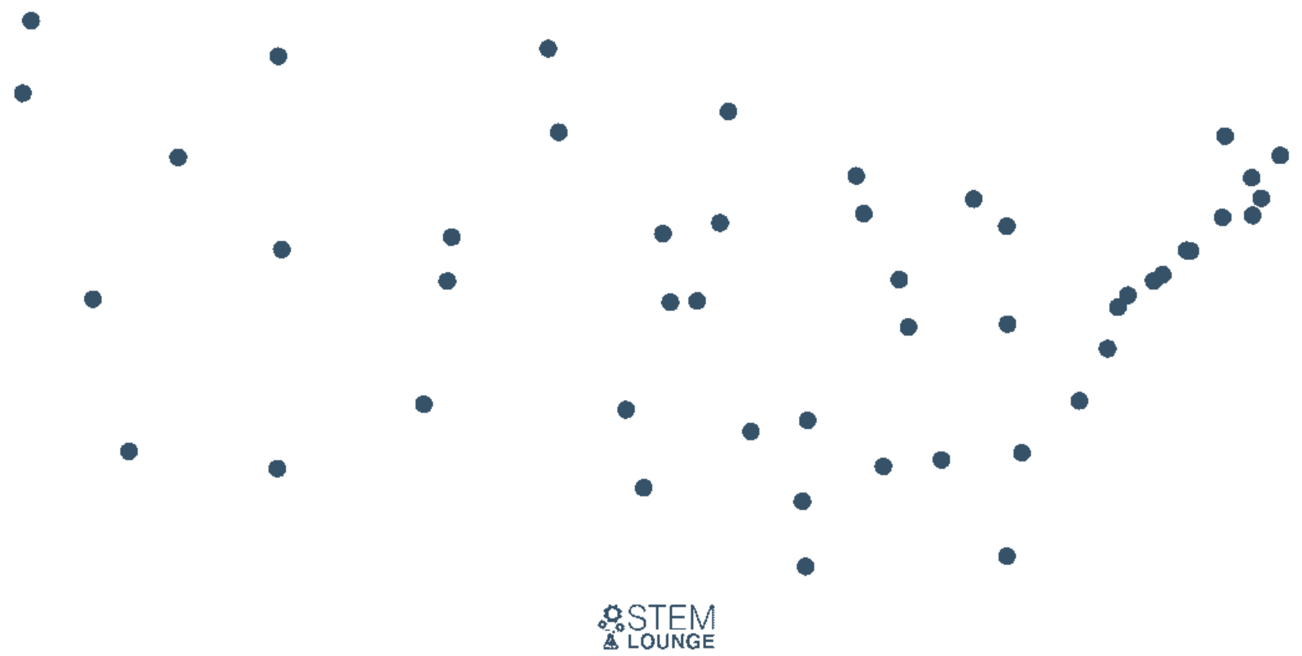


Shortest possible <u>**loop** that **connects**</u> <u>**every red dot**, and **does revisit any dot**</u>



Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**

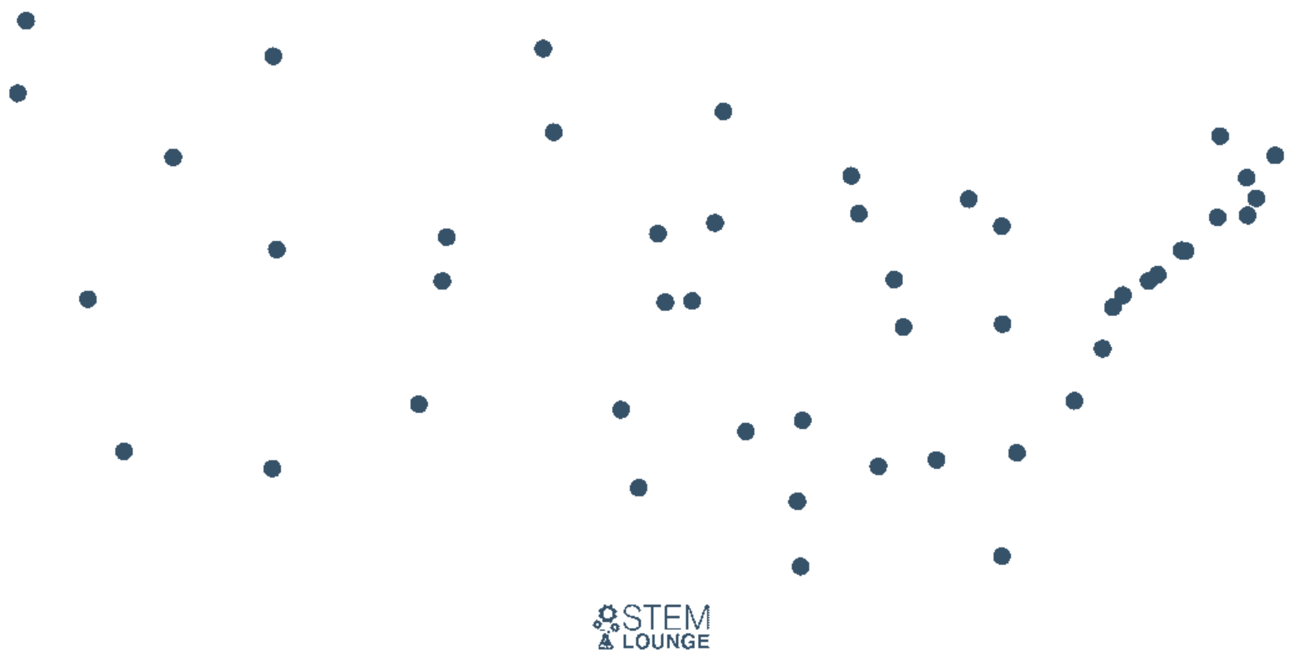


Given the nodes of major cities in the US, what would the greedy algorithm look like for the TSP problem?

You can assume every node as a direct path to every other node

Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**

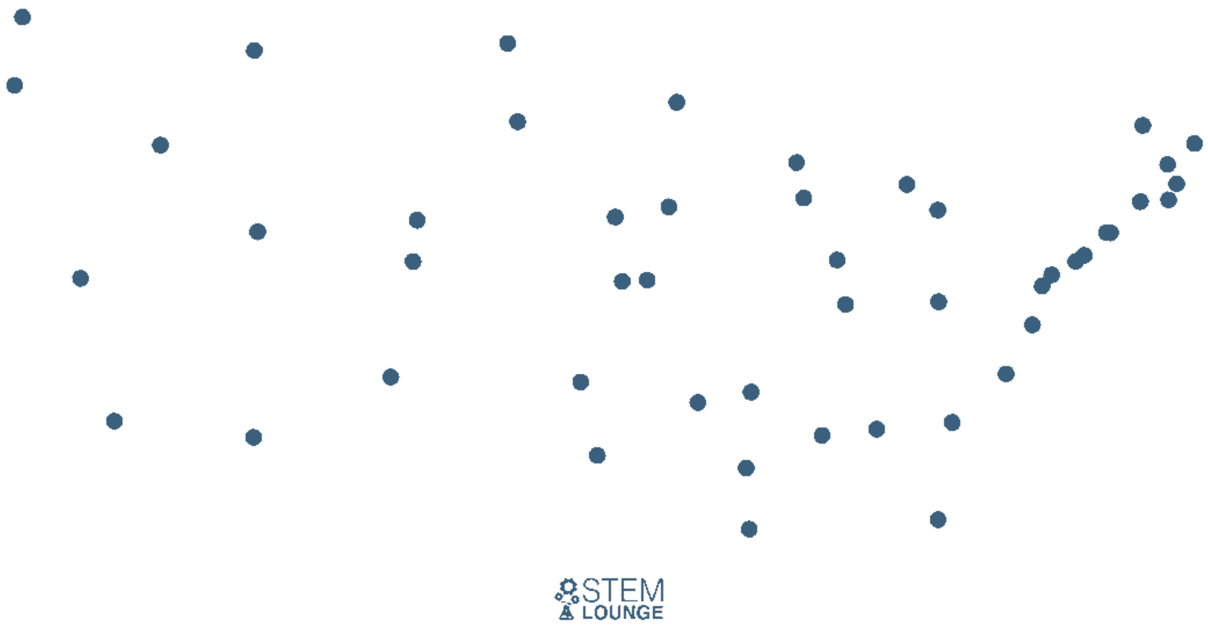


Given the nodes of major cities in the US, what would the greedy algorithm look like for the TSP problem?

Nearest neighbor: always travel to the nearest unvisited neighbor, and then travel to their nearest neighbor, and then travel to their nearest neighbor…
Once we have visited all nodes, travel back to starting node

Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**

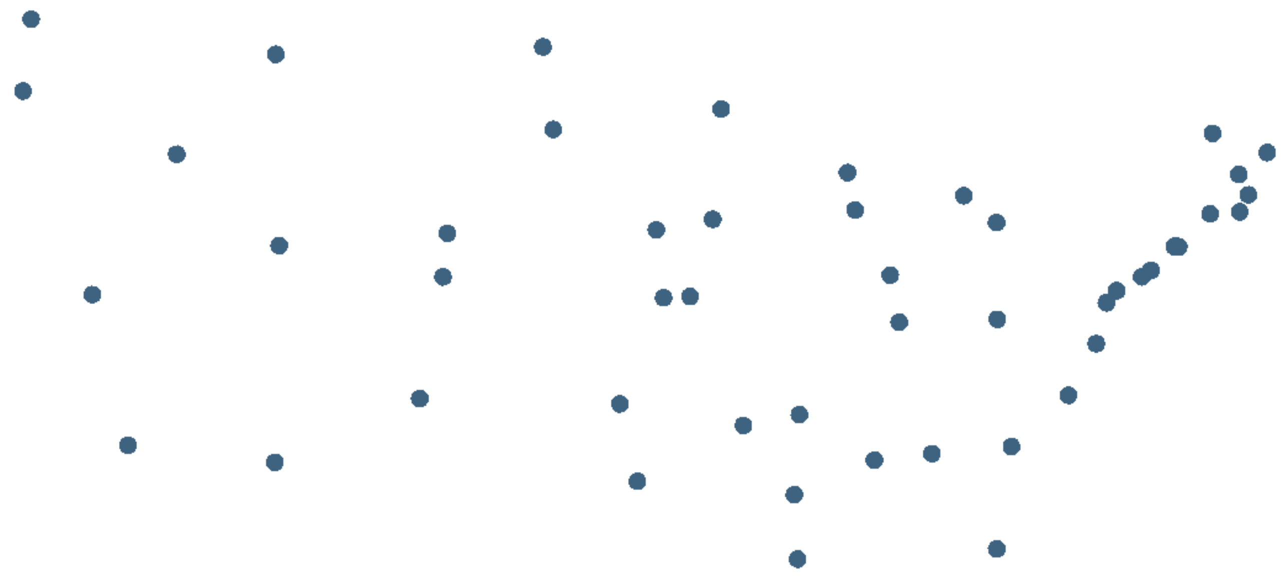


Given the nodes of major cities in the US, what would the greedy algorithm look like for the TSP problem?

Nearest neighbor: always travel to the nearest unvisited neighbor, and then travel to their nearest neighbor, and then travel to their nearest neighbor…
Once we have visited all nodes, travel back to starting node

Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**



Given the nodes of major cities in the US, what would the greedy algorithm look like for the TSP problem?

Lowest edge cost: add the shortest edge that will neither create a vertex with more than 2 edges, nor a cycle with less than the total number of cities until we have a cycle

Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**

Given the nodes of major cities in the US, what would the greedy algorithm look like for the TSP problem?

Nearest Insertion

Start with a cycle, keep growing the cycle by adding the city nearest to the cycle

Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**

Unfortunately, the greedy algorithms for TSP **do not** guarantee an optimal solution

Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**

Unfortunately, the greedy algorithms for TSP **do not** guarantee an optimal solution

The traveling salesman problem is a difficult problem… in fact, it is one of the **most difficult** problems in computer science

We do not know of an algorithm that can solve TSP in **polynomial time**

O(1) O(logn)

O(N^3)

O(n) O(N^2)

O(n log n)

Polynomial time

Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**

Unfortunately, the greedy algorithms for TSP **do not** guarantee an optimal solution

The traveling salesman problem is a difficult problem… in fact, it is one of the **most difficult** problems in computer science

We do not know of an algorithm that can solve TSP in **polynomial time**

O(1) O(logn)

O(N^3)

O(n) O(N^2)

O(n log n)

Polynomial time

The algorithms we currently have for solving TSP run in **exponential** or **factorial** time, which are infeasible for large input sizes

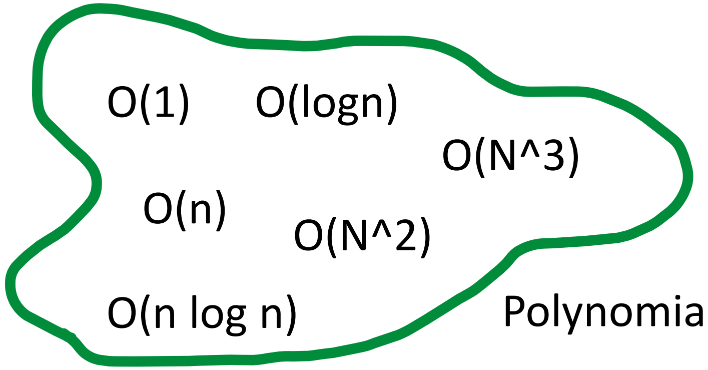
Traveling Salesman

Given a graph with edge weights and a starting node, what is the **shortest** path that **will visit every node**, and **start and end at the starting node**, **without visiting the same node twice**

Unfortunately, the greedy algorithms for TSP **do not** guarantee an optimal solution

The traveling salesman problem is a difficult problem… in fact, it is one of the **most difficult** problems in computer science

We do not know of an algorithm that can solve TSP in **polynomial time**



Polynomial time

If you can solve TSP in polynomial time, you will become a millionaire (literally)

Tractability

Tractability refers to the problems we can solve and not solve in polynomial time

P

sorting

Shortest path

Fractional knapsack

Closest pair of points

Rod cutting

Change making

and **much** more

P is the set of all problems that we can solve in polynomial time

Tractability

Tractability refers to the problems we can solve and not solve in polynomial time

P

sorting

Shortest path

Fractional knapsack

Closest pair of points

Rod cutting

Change making

and **much** more

NP

NP = set of problems that we don't know how to solve in polynomial time, but can verify in polynomial time

Tractability

Tractability refers to the problems we can solve and not solve <u>in polynomial time</u>

P

sorting

Shortest path

Fractional knapsack

Closest pair of points

Rod cutting

Change making

and **much** more

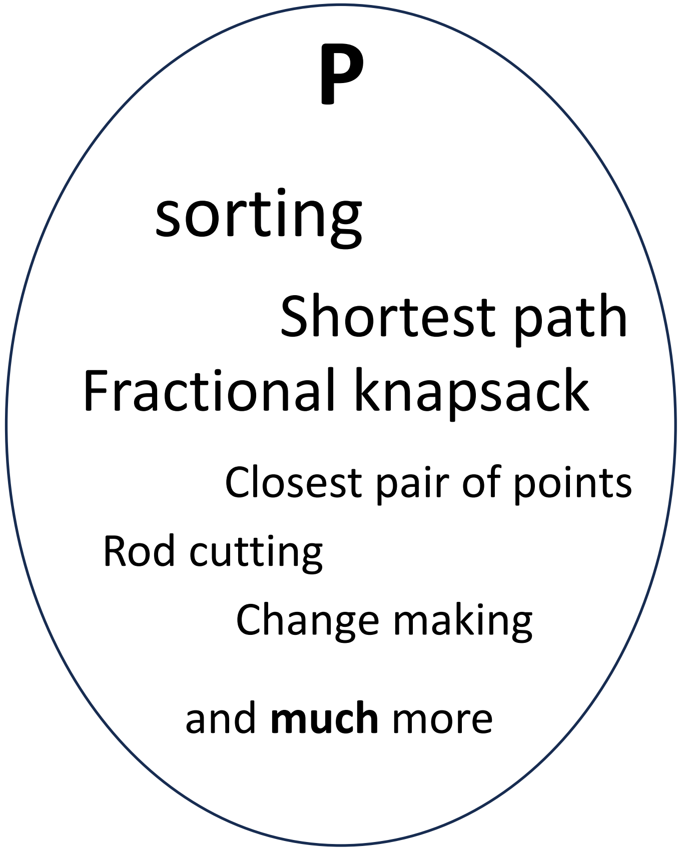
Verify = given a solution to problem, verify if it is correct/incorrect

NP

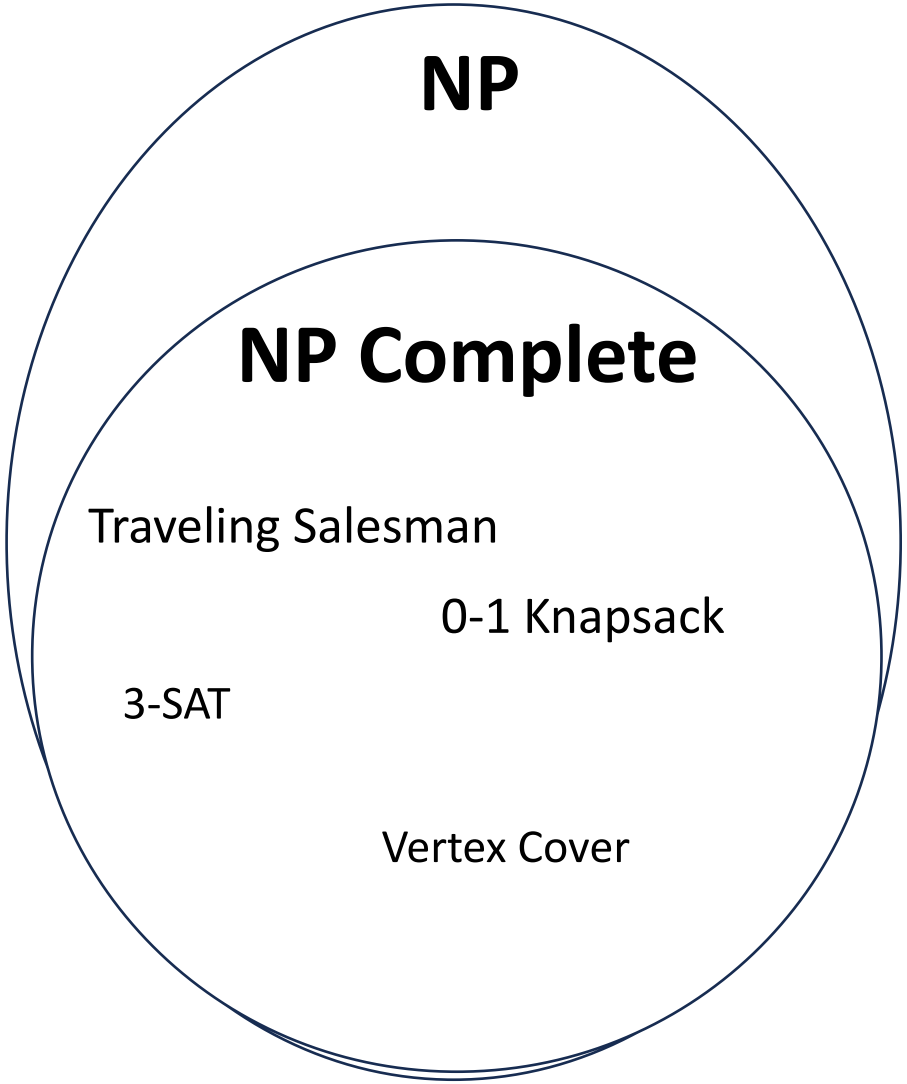
NP = set of problems that we don't know how to solve in polynomial time, but can verify in polynomial time

Tractability

Tractability refers to the problems we can solve and not solve in polynomial time

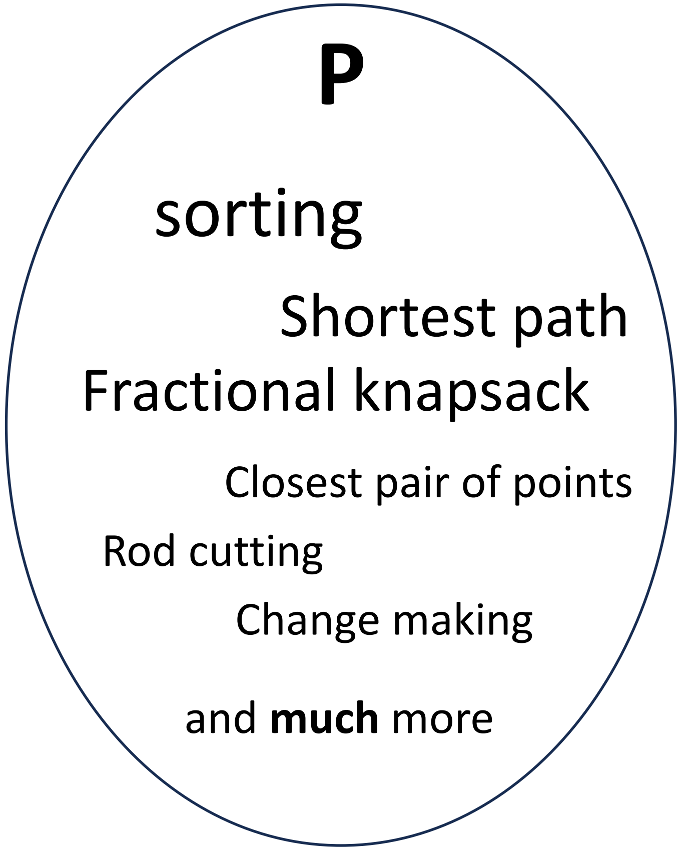


NP-Complete- The hardest problems of NP. If we can solve one NP-Complete problem, we can solve all other NP-Complete problems

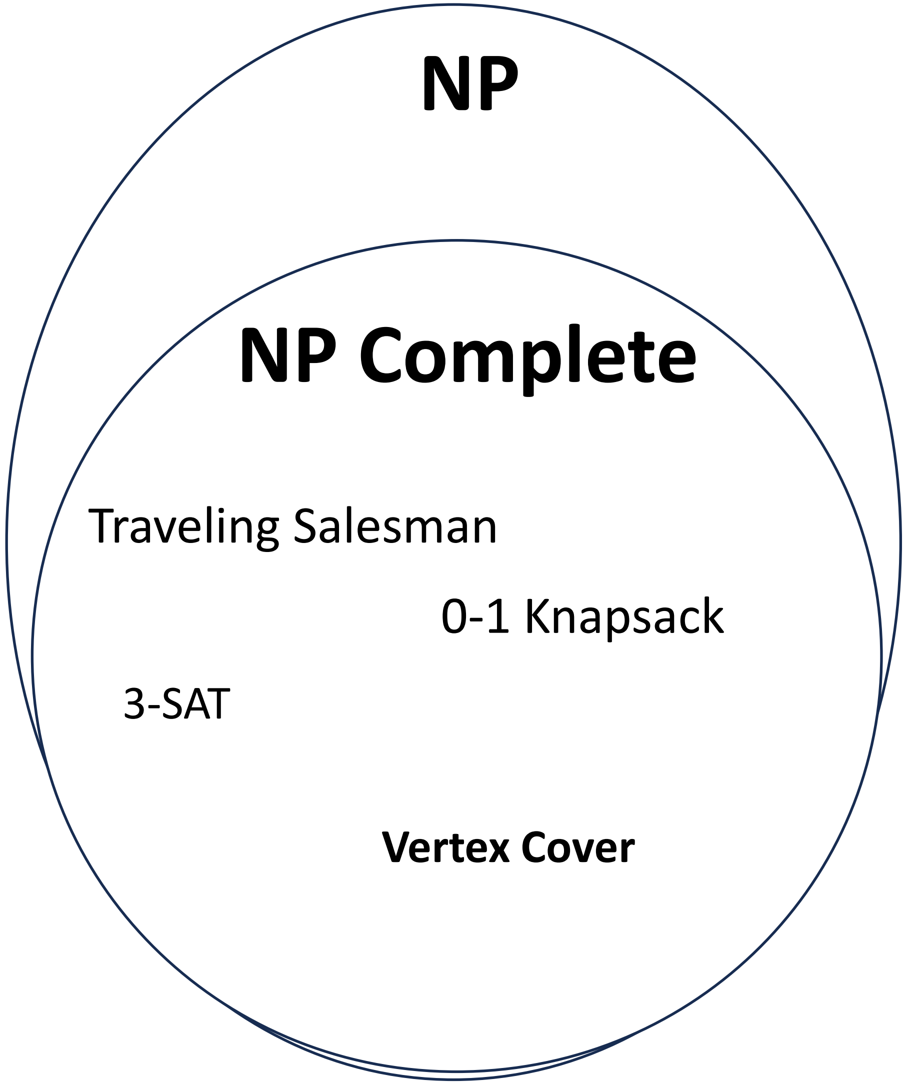


Tractability

Tractability refers to the problems we can solve and not solve <u>in polynomial time</u>



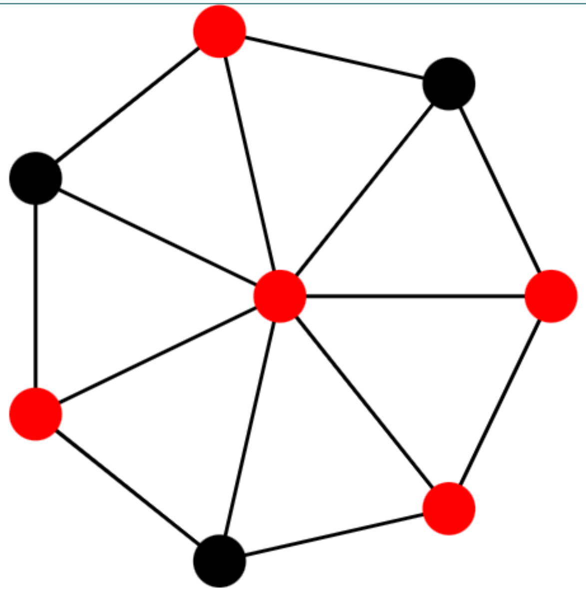
NP-Complete-
The hardest problems of NP. If we can solve one NP-Complete problem, we can solve all other NP-Complete problems



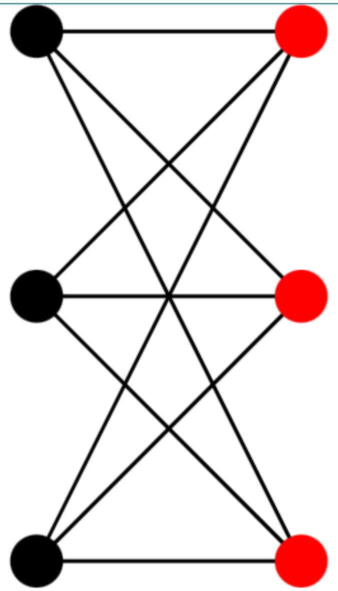
Vertex Cover = Given a graph, compute a set of vertices S that include at least one endpoint of every edge.

Tractability

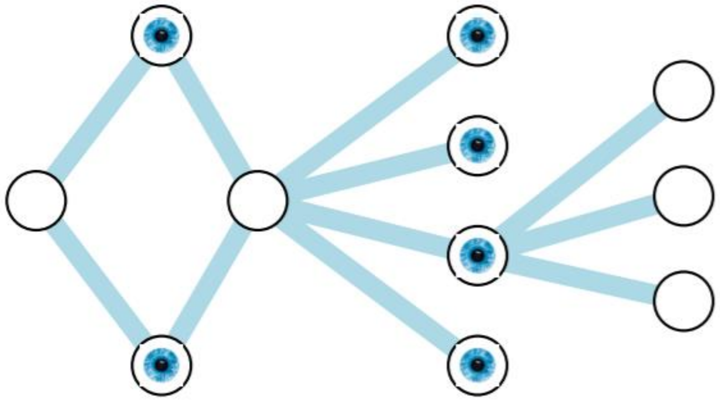
Vertex Cover = Given a graph, compute a set of vertices S that include at least one endpoint of every edge.



VC Size = 5



VC Size = 3



A vertex cover of size 6

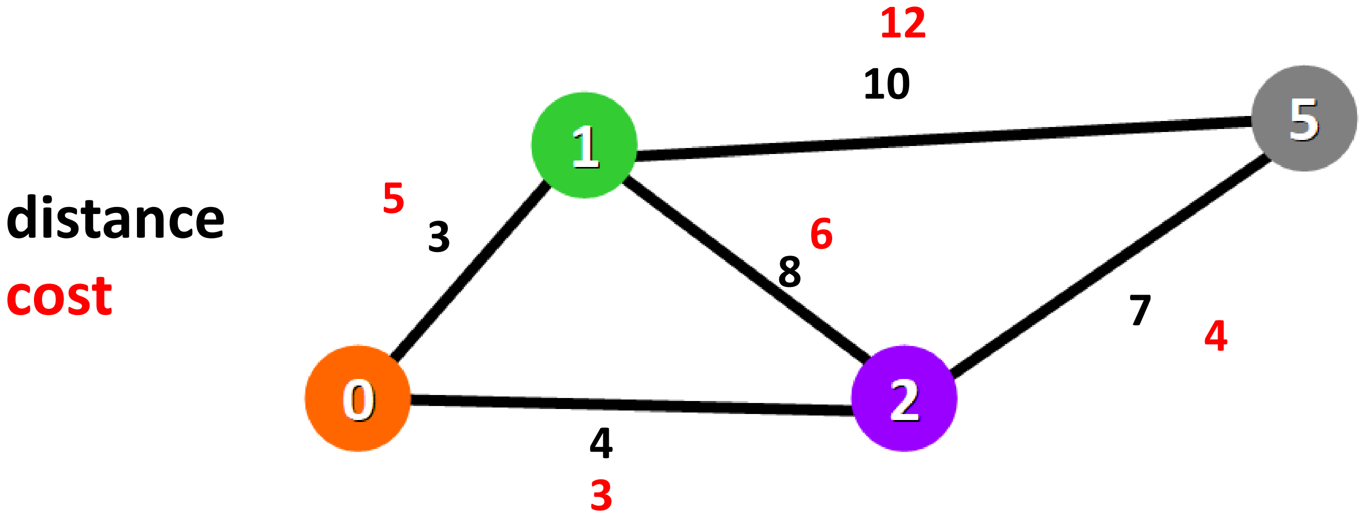
Typically, we are concerned about finding the *minimum* vertex cover

Finding shortest paths in a distance-weighted graph can be done in polynomial time.

Dijkstra's Algorithm

Tractability

Finding shortest paths that cost at most 20 in a distance-and-cost-weighted graph can be done in polynomial time.

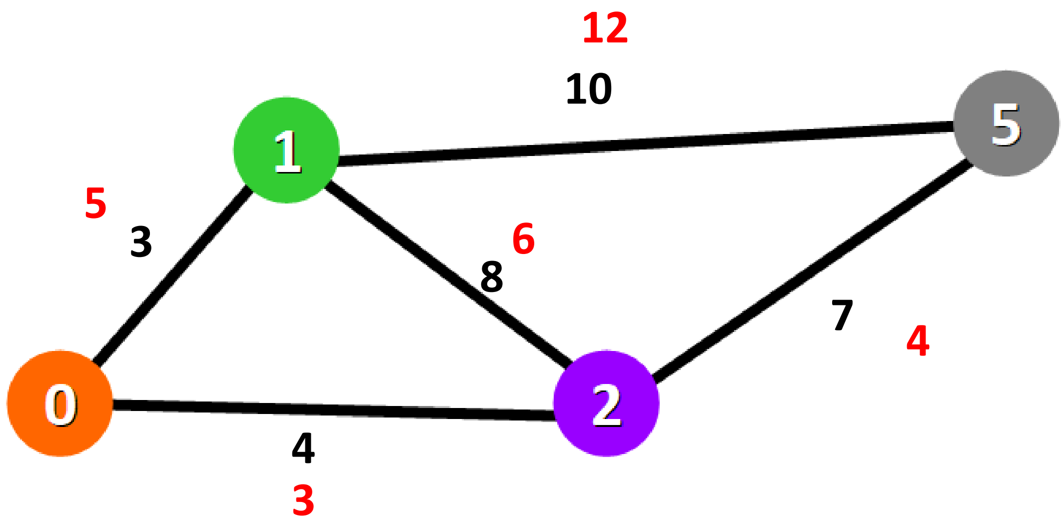


Tractability

Finding shortest paths that cost at most 20 in a distance-and-cost-weighted graph can be done in polynomial time.

distance

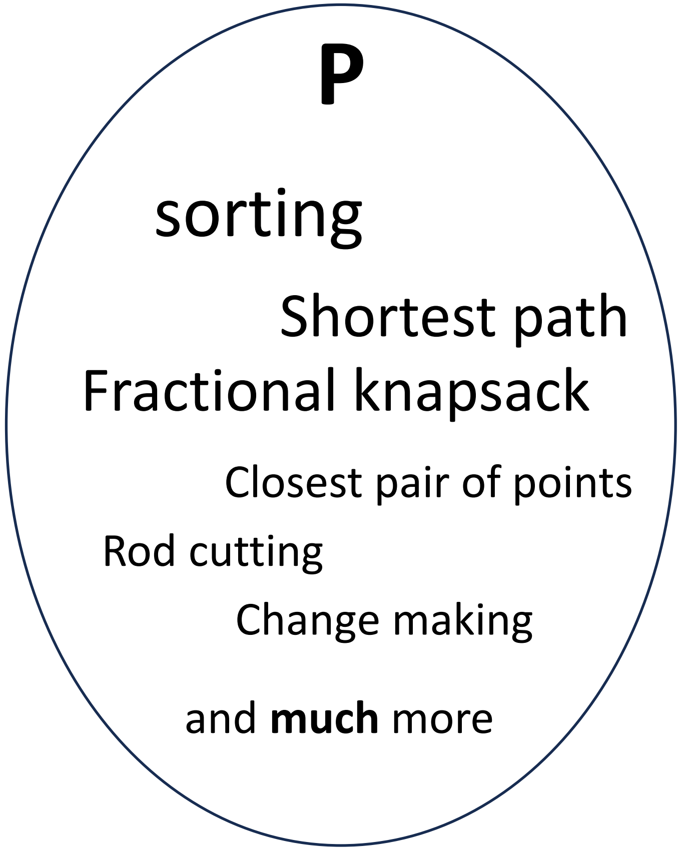
cost



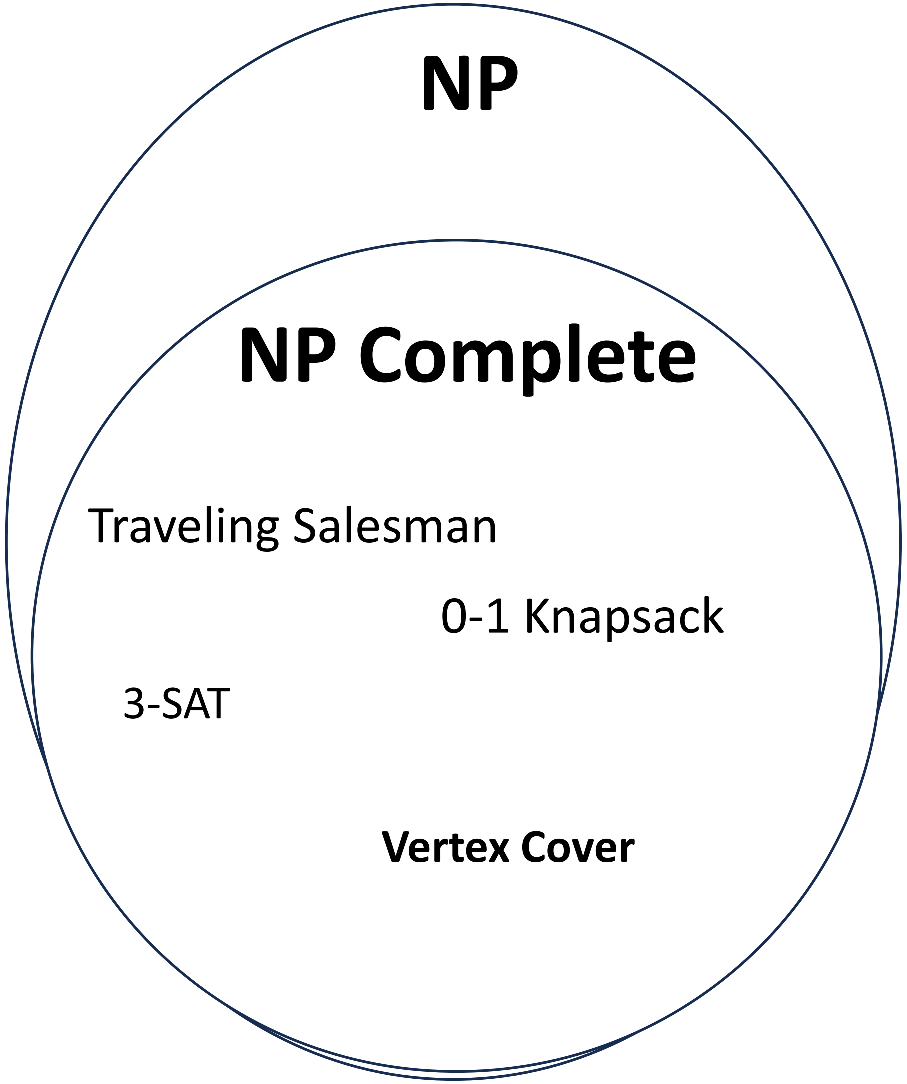
By making a small tweak to this problem, this problem actually becomes *much* more challenging (NP-C)

Tractability

Tractability refers to the problems we can solve and not solve <u>in polynomial time</u>

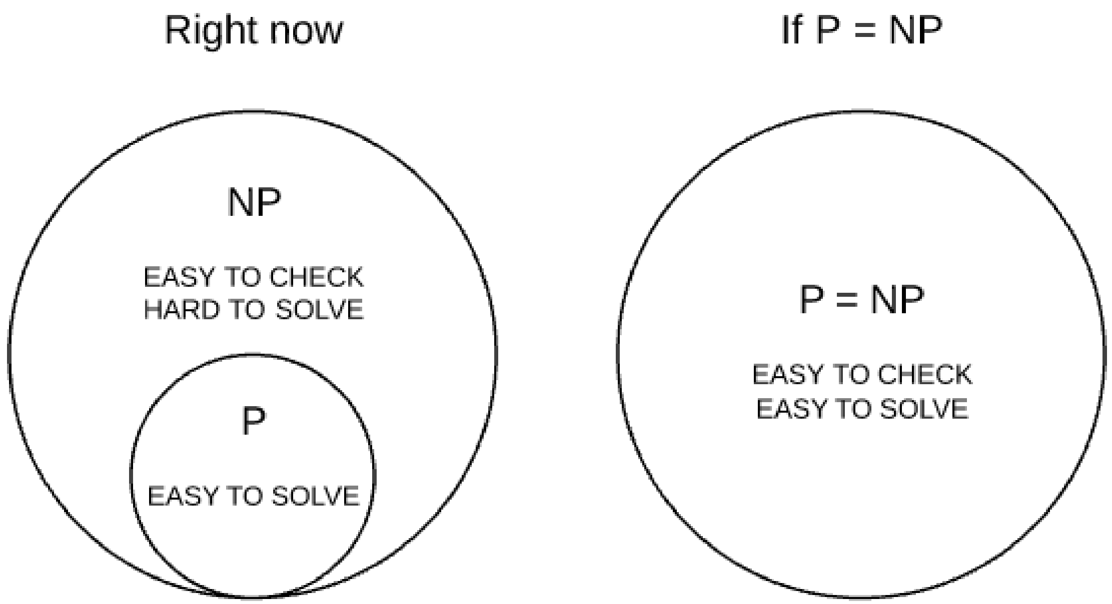


NP-Complete-
The hardest problems of NP. If we can solve one NP-Complete problem, we can solve all other NP-Complete problems



Vertex Cover = Given a graph, compute a set of vertices S that include at least one endpoint of every edge.

Tractability



On May 24, 2000, Clay Mathematics Institute came up with seven mathematical problems, for which, the solution for any of the problem will earn US $1,000,000 reward for the solver. Famously known as the Millennium Problems, so far, only one of the seven problems is solved till date.

Wanna make a million dollar, try solving one from this list. These are the problems listed for a million dollar prize reward.

1. Yang–Mills and Mass Gap
2. Riemann Hypothesis
3. P vs NP Problem
4. Navier–Stokes Equation

If you can solve TSP in polynomial time, you can win a million dollars, probably become a tenured CS professor, and also break cybersecurity