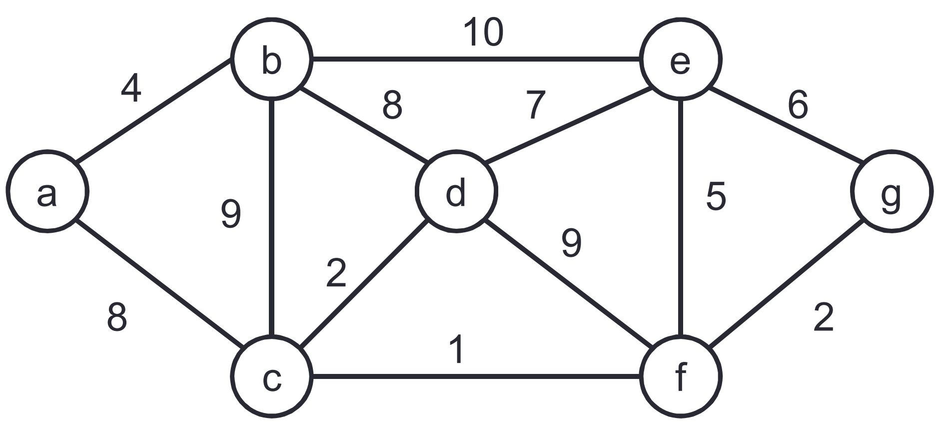
CSE 100: DISJOINT SET, MST, NP-COMPLETENESS

Announcements

- PA3
  - Final submission deadline 11:59pm on Thursday, December 6 (slip days allowed)
- HW5
  - Due next Wednesday!

Minimum spanning trees: Kruskal's algorithm

- Sort edges from smallest to largest
- Initially place each node into its own subset
- Repeat until all nodes are connected:
  - Select the smallest edge where the endpoints are in different subsets and include that edge in the MST



What is the worst case running time of Kruskal's algorithm?

A. O(|E|)
B. O(|E| log(|E|))
C. O(|V| * |E|)
D. Other

Assume the graph is *connected* (i.e. no nodes are "floating")

Greedy Algorithm

- A "greedy algorithm" is one that always selects the locally largest step toward the goal.

Greedy Algorithm

- A "greedy algorithm" is one that always selects the locally largest step toward the goal.

Kruskal's algorithm
Sort edges from smallest to largest
Initially place each node into its own subset
Repeat until all nodes are connected:
    Select the smallest edge where the endpoints are in different subsets
        and include that edge in the MST

Is Kruskal's algorithm a greedy algorithm?
A. Yes
B. No
C. Sometimes

Greedy Algorithm

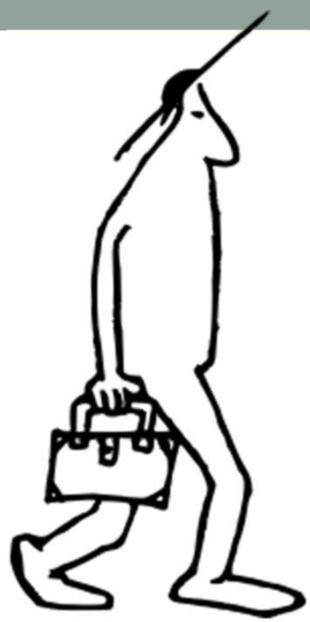
- A "greedy algorithm" is one that always selects the locally largest step toward the goal.

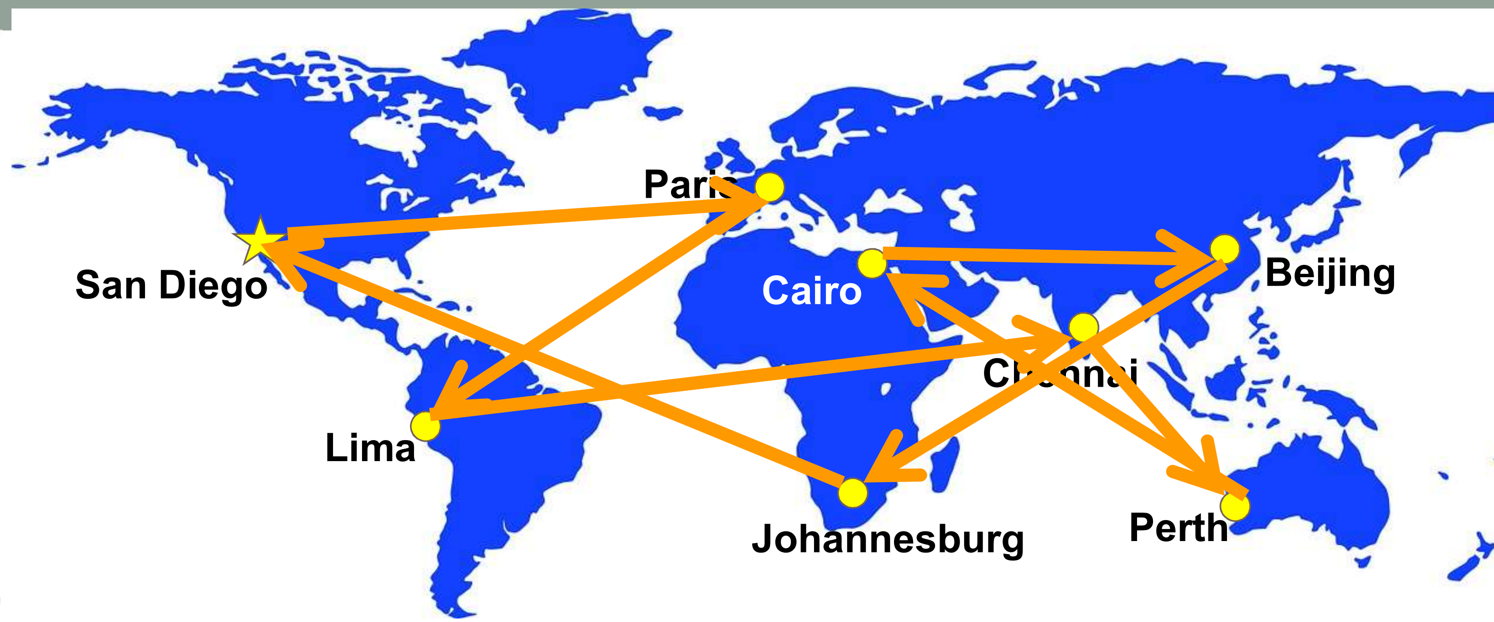
Kruskal's algorithm
Sort edges from smallest to largest
Initially place each node into its own subset
Repeat until all nodes are connected:
    Select the smallest edge where the endpoints are in different subsets
        and include that edge in the MST

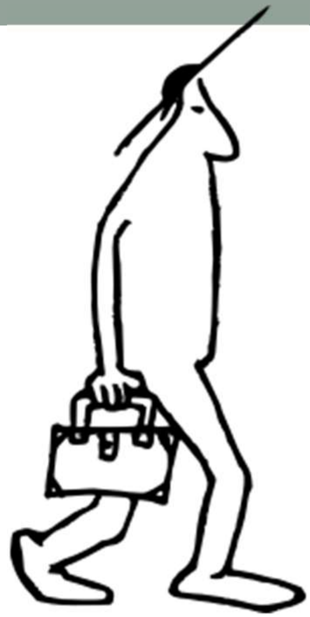
Greedy algorithms are simple and fast, but for MANY problems they do not return the optimal solution.

Is Kruskal's algorithm a greedy algorithm?
A. Yes
B. No
C. Sometimes

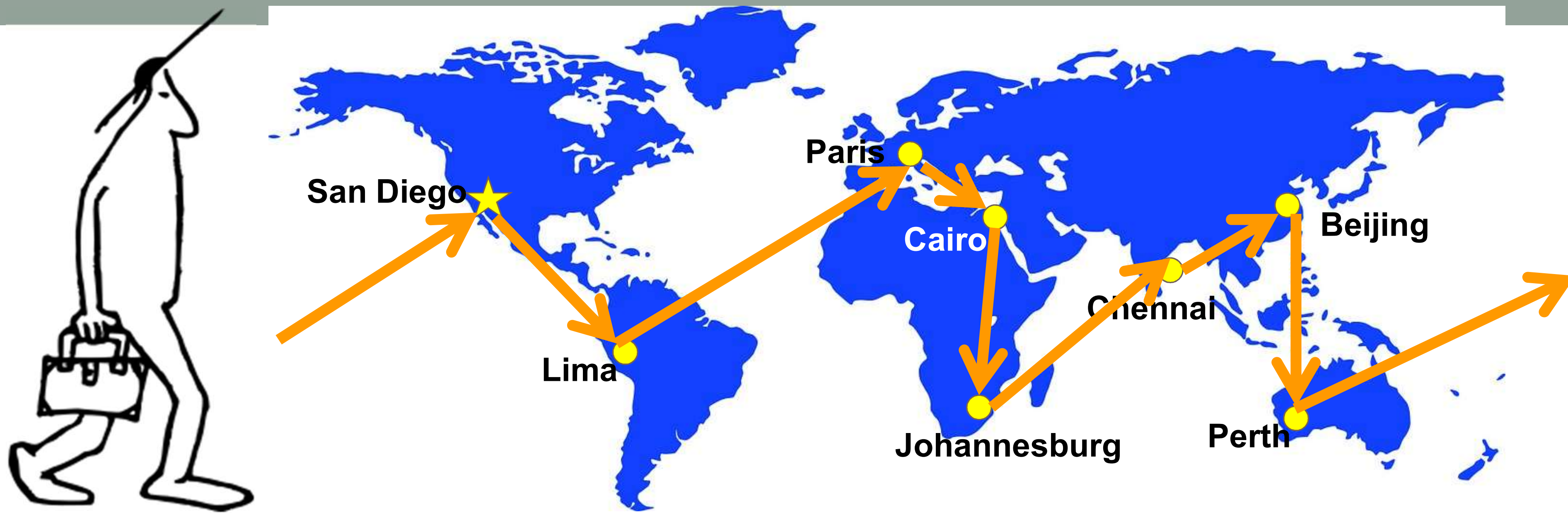

Paris
San Diego
Cairo
Beijing
Chennai
Lima
Johannesburg
Perth

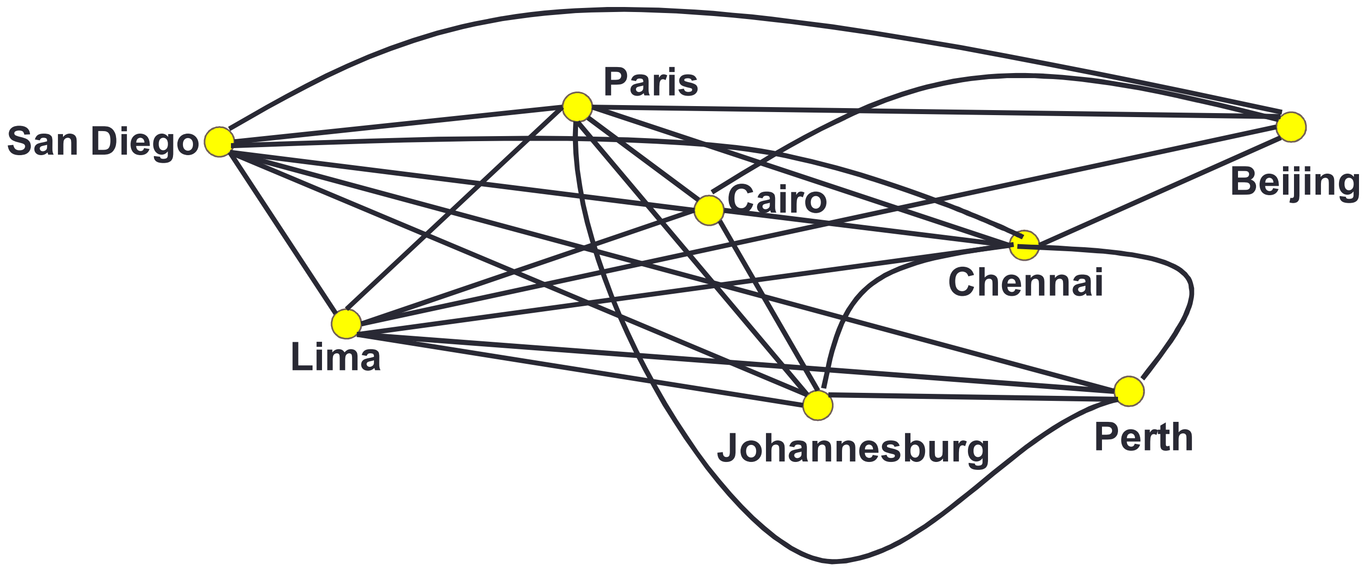



San Diego
Cairo
Beijing
Lima
Johannesburg
Perth



San Diego
Paris
Cairo
Chennai
Beijing
Perth
Johannesburg
Lima


San Diego
Lima
Paris
Cairo
Johannesburg
Chennai
Beijing
Perth



Is this graph sparse or dense?

A. Sparse

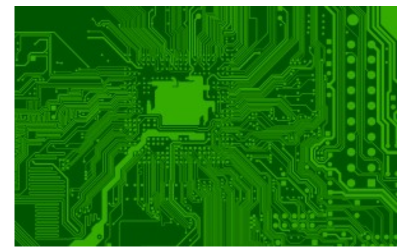
B. Dense

	SD	Lima	Paris	Chen.	Cairo	Perth	Beij.	J'berg
SD	0	6,091	9,144	14,587	12,276	15,078	10,234	16,575
Lima	6,091	0	10,248	17,540	12,414	14,924	16,637	10,872
Paris	9,144	10,248	0	8,031	3210	14,269	8,212	8,295
Chen.	14,587	17,540	8,031	0	5,360	6,276	4,615	7,133
Cairo	12,276	12,414	3210	5,360	0	11,258	7,540	6,260
Perth	15,078	14,924	14,269	6,276	11,258	0	7,985	8,308
Beij.	10,234	16,637	8,212	4,615	7,540	7,985	0	11,699
J'berg	16,575	10,872	8,295	7,133	6,260	8,308	11,699	0

The Traveling Salesperson Problem: Given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Lots of applications!

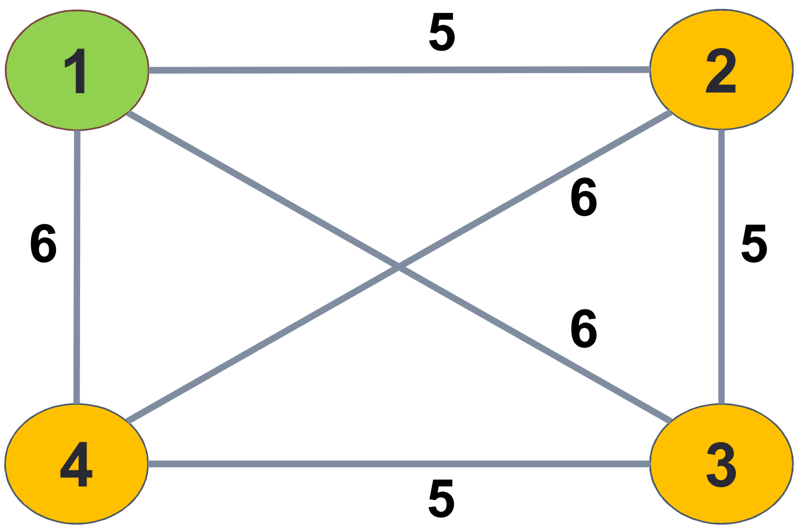


http://www.math.uwaterloo.ca/tsp/index.html

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Greedy algorithm: pick best next choice

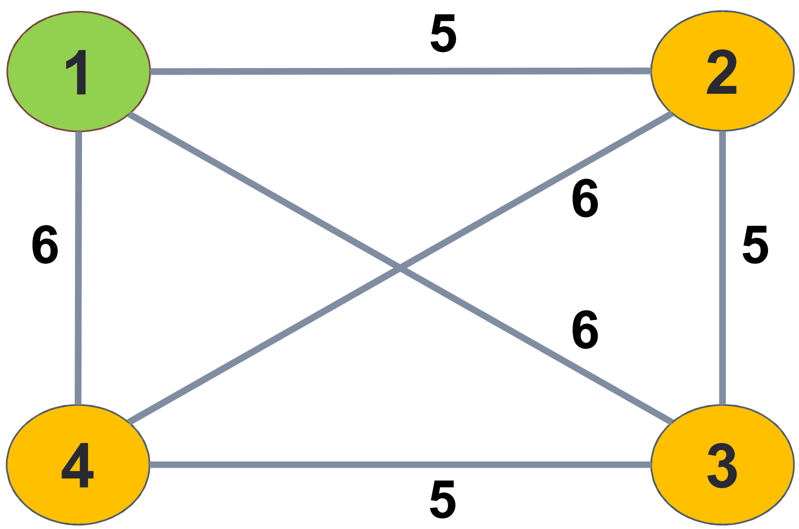
Warmup: What tour does the Greedy algorithm construct for this graph?



A. 1 → 2 → 3 → 4 → 1
B. 1 → 3 → 2 → 4 → 1
C. 1 → 4 → 3 → 2 → 1
D. 1 → 2 → 4 → 3 → 1

Greedy algorithm: pick best next choice

Is this the best possible tour for this graph?



1 → 2 → 3 → 4 → 1

A. Yes
B. No

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

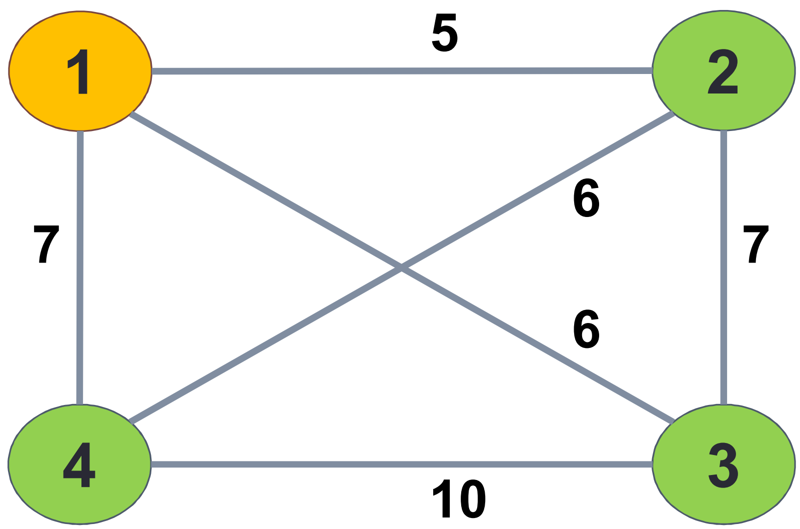
Greedy algorithm: pick best next choice

Will the greedy algorithm always work?

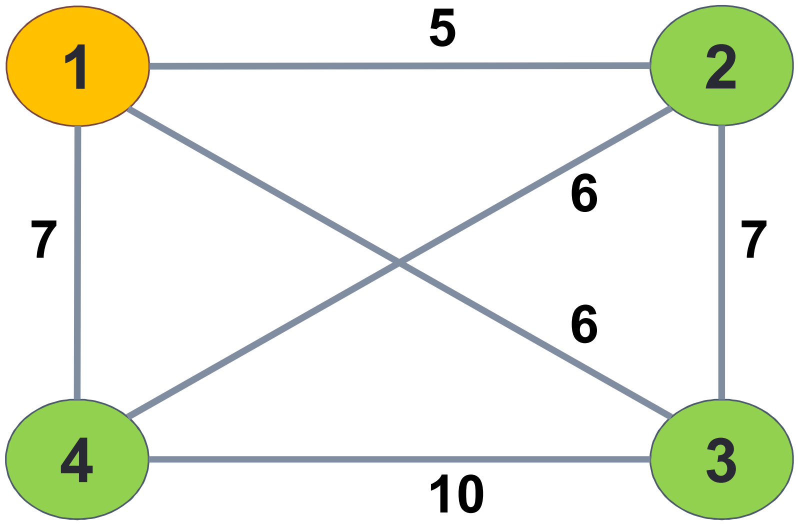
A. ~~Yes~~

B. No

If yes, why? ***If no, find counterexample.***

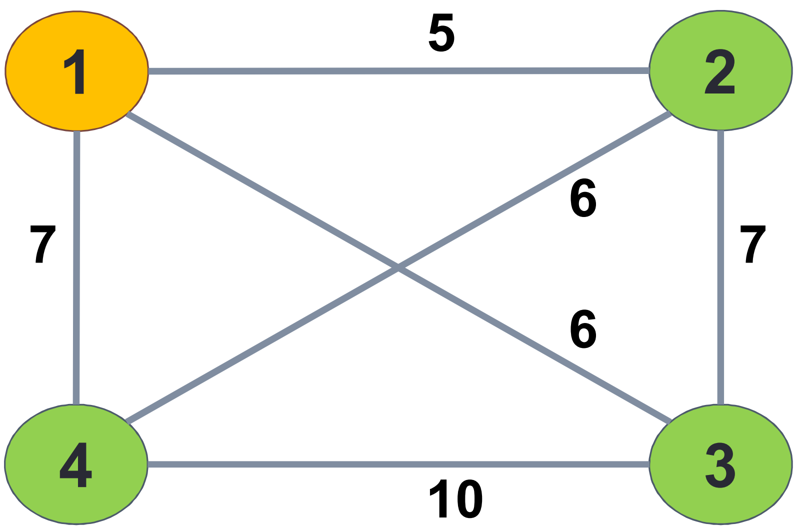


1
2
3
4
5
6
6
7
7
10



Greedy:

1 → 2 → 4 → 3 → 1



Greedy: 1 → 2 → 4 → 3 → 1 **27**

Optimal: 1 → 3 → 2 → 4 → 1

6+7+6+7 = 26

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

	SD	Lima	Paris	Chen.	Cairo	Perth	Beij.	J'berg
SD	0	6,091	9,144	14,587	12,276	15,078	10,234	16,575
Lima	6,091	0	10,248	17,540	12,414	14,924	16,637	10,872
Paris	9,144	10,248	0	8,031	3210	14,269	8,212	8,295
Chen.	14,587	17,540	8,031	0	5,360	6,276	4,615	7,133
Cairo	12,276	12,414	3210	5,360	0	11,258	7,540	6,260
Perth	15,078	14,924	14,269	6,276	11,258	0	7,985	8,308
Beij.	10,234	16,637	8,212	4,615	7,540	7,985	0	11,699
J'berg	16,575	10,872	8,295	7,133	6,260	8,308	11,699	0

Just try all paths and choose the shortest!

Brute force approach

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

- **SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego**

  6,091 + 10,248 + 3210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

★ **SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego**

6,091 + 10,248 + 3210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

SD → Lima → Paris → Cairo → Perth → Beijing → Chennai → Johannesburg → San Diego

6,091 + 10,248 + 3210 + 11,258 + 7,985 + 4,615 + 7,133 + 16,575 = 67,115km

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

★ **SD → Lima → Paris → Cairo → Perth → Beijing → Chennai → Johannesburg → San Diego**

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 4,615 + 7,133 + 16,575 = 67,115km

SD → Lima → Paris → Cairo → Perth → Johannesburg → Beijing → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 8,308 + 11,699 + 4,615 + 14,587 = 70,016km

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

SD → Lima → Paris → Cairo → Perth → Beijing → Chennai → Johannesburg → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 4,615 + 7,133 + 16,575 = 67,115km

SD → Lima → Paris → Cairo → Perth → Johannesburg → Beijing → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 8,308 + 11,699 + 4,615 + 14,587 = 70,016km

. . .

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Perth → Beijing → Johannesburg → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 11,699 + 7,133 + 14,587 = 72,211km

SD → Lima → Paris → Cairo → Perth → Beijing → Chennai → Johannesburg → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 7,985 + 4,615 + 7,133 + 16,575 = 67,115km

SD → Lima → Paris → Cairo → Perth → Johannesburg → Beijing → Chennai → San Diego

6,091 + 10,248 + 3,210 + 11,258 + 8,308 + 11,699 + 4,615 + 14,587 = 70,016km

. . .

It works!!

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

SD → Lima → Paris → Cairo → Johannesburg → Perth → Chennai → Beijing → San Diego

6,091 + 10,248 + 3,210 + 6,260 + 8,308 + 6,276 + 4,615 + 10,234 = 55,242km

But how long does it take…?

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

Permutations coming next

```
bestPath = null, bestDist = +Infinity
for each permutation of cities, starting and ending in Hometown:
    calculate distance of current permutation
    if (distance < bestDist)
        bestPath = current permutation, bestDist = distance

return bestPath
```

O(n)

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

```
bestPath = null, bestDist = +Infinity
for each permutation of cities, starting and ending in Hometown:
    calculate distance of current permutation
    if (distance < bestDist)
        bestPath = current permutation, bestDist = distance

return bestPath
```

O(n)

But how many permutations?!?

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

How many permutations for a TSP starting with San Diego?

San Diego
Cairo
Johannesburg
Chennai
Lima
Paris
Beijing
Perth

How many permutations are there for the tour?

A. 7!
B. 7^n
C. 2^7
D. 2*8

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

How many permutations?

San Diego	**How many choices for the first city? 1 (San Diego)**
Cairo	**How many choices for the next city? 7**
Johannesburg	**How many choices for the next city? 6**
Chennai	**How many choices for the next city? 5**
Lima	**How many choices for the next city? 4**
Paris	**How many choices for the next city? 3**
Beijing	**How many choices for the next city? 2**
Perth	**How many choices for the next city? 1**
	How many choices for the last city? 1 (San Diego)

In general we have (n-1)! permutations to try!

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Brute force algorithm: Generate all paths and choose the shortest

(n-1)! permutations

```
bestPath = null, bestDist = +Infinity
for each permutation of cities, starting and ending in Hometown:
    calculate distance of current permutation
    if (distance < bestDist)
        bestPath = current permutation, bestDist = distance

return bestPath
```

O(n)

(n-1)! * n = O(n!)

N	N!
10	~3.6 million
19	1.22×10^{17} (the age of the universe)
23	# of stars in the universe
59	# of atoms in the universe

In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Greedy algorithm: pick best next choice

```
bestPath = []
current = Hometown
cities to visit = all other cities
while (more cities to visit)
    select city closest to current and add to bestPath
    remove current city from cities to visit
    current = selected city
return bestPath
```

What is the running time of the greedy algorithm?

A. O(n)
B. $O(n^2)$
C. $O(n^3)$
D. O(n!)

TSP Brute Force

N	N!
10	~3.6 million
19	1.22×10^{17} (the age of the universe)
23	# stars in the universe
59	# of atoms in the universe

Yikes!

What do we do now?

Think really hard about a faster solution?

Complexity Theory

Classifies problems by their inherent difficulty

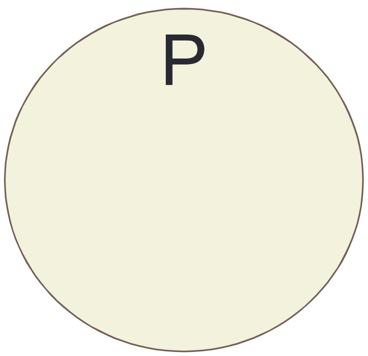
Searching a Linked List – O(n)

Sorting an Array – O(n log n)

n x n Matrix-Matrix Multiply– $O(n^{\sim 2.37})$

Complexity Theory

Classifies problems by their inherent difficulty



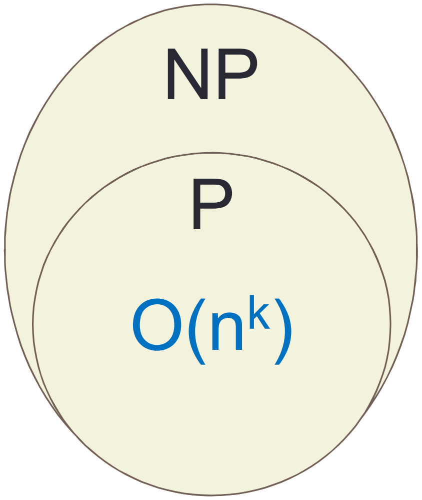
Searching a Linked List – O(n)

Sorting an Array – O(n log n)

n x n Matrix-Matrix Multiply– $O(n^{\sim 2.37})$

Complexity Theory

Classifies problems by their inherent difficulty



Running Times

(or why people worry about algorithm complexity)

problem size →

complexity	n = 10	100	1,000	10,000	100,000	1,000,000
log n	3.3219	6.6438	9.9658	13.287	16.609	19.931
$\log^2 n$	10.361	44.140	99.317	176.54	275.85	397.24
sqrt n	3.162	10	31.622	100	316.22	1000
n	**10**	**100**	**1000**	**10000**	**100000**	**1000000**
n log n	33.219	664.38	9965.8	132877	$1.66*10^6$	$1.99*10^7$
$n^{1.5}$	31.6	10^3	$31.6*10^4$	10^6	$31.6*10^7$	10^9
n^2	100	10^4	10^6	10^8	10^{10}	10^{12}
n^3	1000	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	1024	10^{30}	10^{301}	10^{3010}	10^{30103}	10^{301030}
n!	3 628 800	$9.3*10^{157}$	10^{2567}	10^{35659}	10^{456573}	$10^{5565710}$

O

Running times

Running times of different big-O algorithms for larger and larger inputs.

P ?= NP How to get rich and famous

CMI

ABOUT PROGRAMS MILLENNIUM PROBLEMS

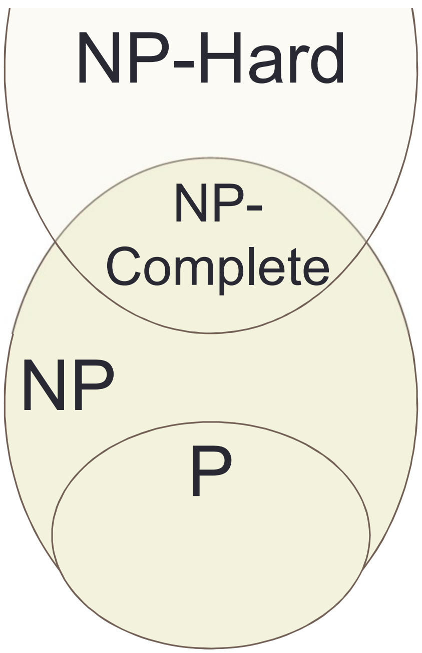
The Millennium Prize Problems

Following the decision of the Scientific Advisory Board, the Board of Directors of CMI designated a $7 million prize fund for the solutions to these problems, with $1 million allocated to the solution of each problem.

, with $1 million allocated to the solution of each problem.

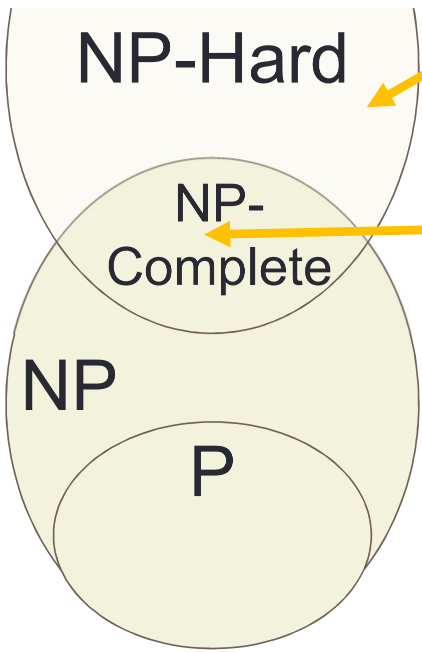
http://www.claymath.org/millennium-problems

Complexity Theory



(Hierarchy if P != NP)

Complexity Theory



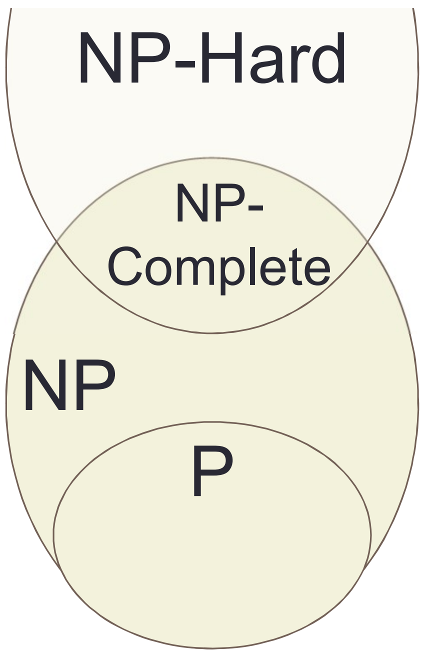
(Hierarchy if P != NP)

NP-Hard: Problems are *at least* as difficult to solve as hardest problems in NP

NP-Complete: No known polynomial time algorithm to find a solution, but can check a solution in polynomial time

A polynomial time solution for *any* NP-Complete problem would solve *all* NP-Complete problems

Complexity Theory

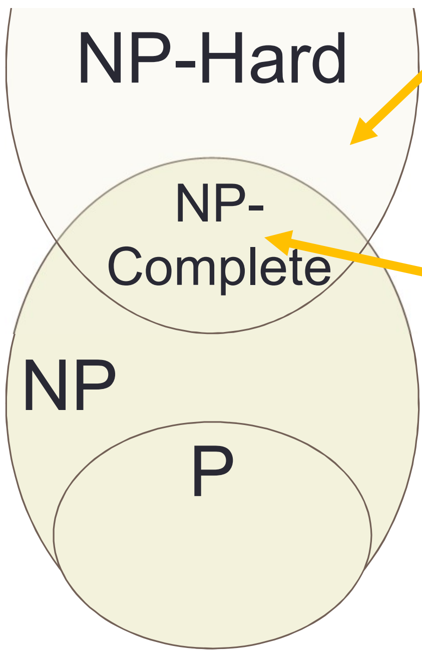


TSP "optimization": given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

Where does (do you think) the TSP optimization problem fits into this diagram?

A. In P (there is a polynomial time way to find a solution)
B. In NP/NP-Complete (we might not know a polynomial time way to find a solution, but if someone gives us a proposed solution, we can verify whether or not it's correct)
C. NP-Hard (neither of the above is true)
D. I have no idea! I'm so confused!

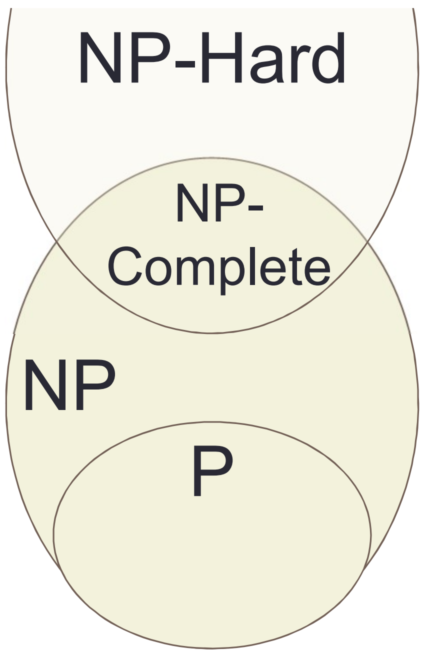
Complexity Theory



TSP "optimization": given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

TSP "decision": given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has a distance less than L.

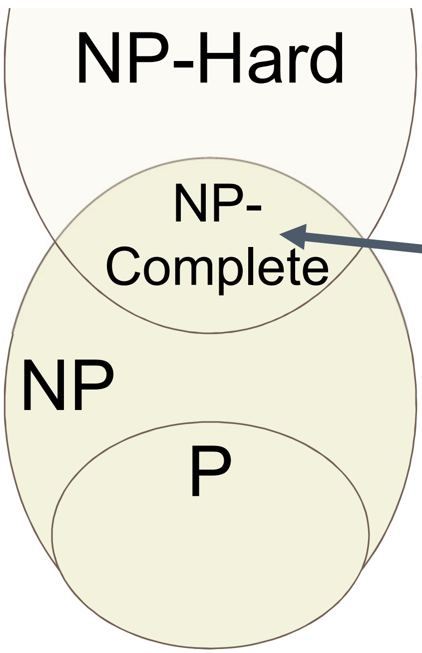
Complexity Theory



Since TSP (both versions) is NP-Hard, solving it in polynomial time may be difficult (if not impossible)

Next time… how to prove a problem is NP-Hard.

Complexity Theory



To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. **Prove it is NP-Hard**

Proving a Problem is in NP

To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. **Prove it is NP-Hard**

For all problems in NP, there exists an algorithm that can verify whether a solution to the problem is correct or not in polynomial time.

To show a problem is in NP, you must give such an algorithm.

TSP-decision is in NP

To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. **Prove it is NP-Hard**

Given a path $v_0 \ldots v_n$, you can verify whether it visits each node and has distance less than L in polynomial time:

dist = 0
For k = 0…n-1:
    If v_k previously marked as visited, fail
    Mark v_k as visited
    dist += weight (v_k, v_{k+1})
If v_0 != v_n fail
If any vertices are not marked as visited, fail
If dist >= L, fail

Else Success

Is TSP-decision NP-Hard?

To prove a problem is NP-Complete you must do two things:
1. **Prove it is in NP**
2. **Prove it is NP-Hard**

To prove a problem is NP-Hard, you must prove it is *at least* as hard as any problem in NP.

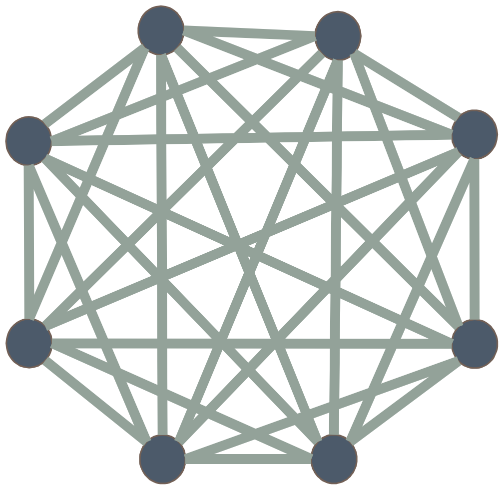
So we should choose a problem that we know is "as hard as any other problem in NP" and show that TSP-decision is at least as hard.

We'll choose a problem called "Hamiltonian Cycle"

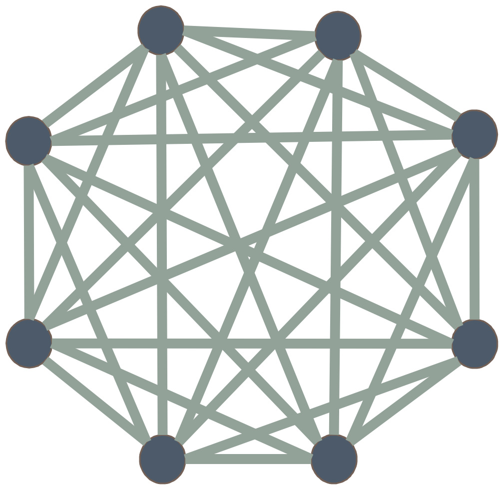
(You have to trust me that Hamiltonian Cycle is "as hard as any other problem in NP". It's somewhat involved to prove it.)

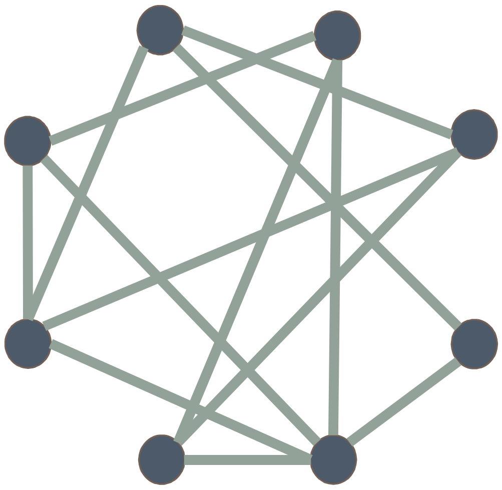
In TSP, given n cities with one Hometown and all pairwise distances, plan a tour starting and ending at Hometown that visits every city exactly once and has minimum distance.

~~In TSP,~~ given n cities with ~~one Hometown and~~ all pairwise distances, plan a tour that visits every city exactly once ~~and has minimum distance~~.

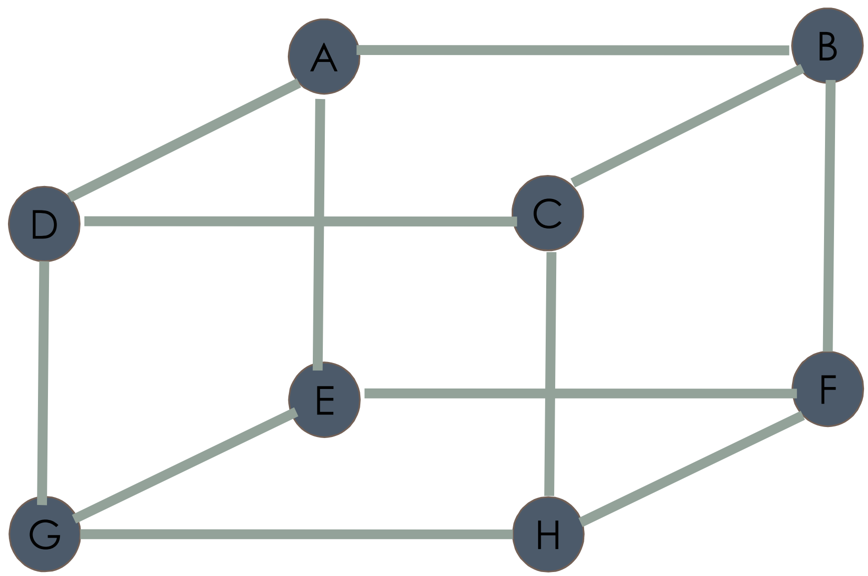


~~In TSP,~~ given n cities ~~with one Hometown and all pairwise distances~~, plan a tour that visits every city exactly once ~~and has minimum distance~~.

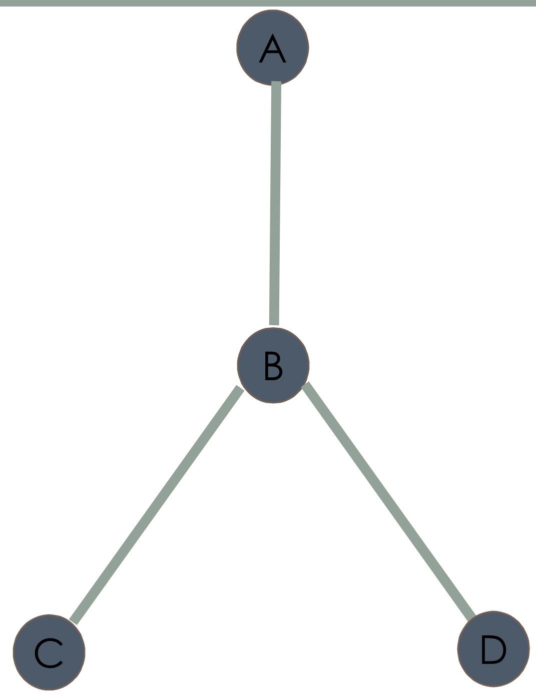




A graph is Hamiltonian if there is a path (or cycle) through the graph which visits each vertex exactly once.



Does this graph have a Hamiltonian cycle?
A. Yes (If yes, what is it?)
B. No



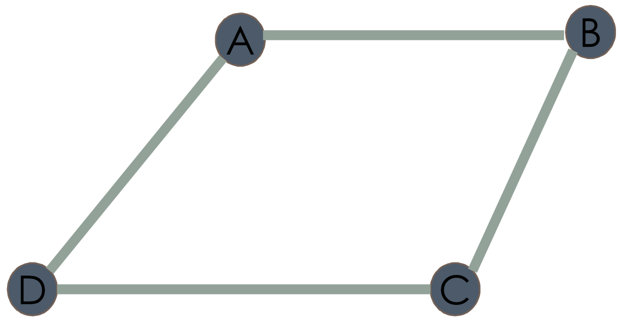
Does this graph have a Hamiltonian cycle?

A. Yes (If yes, what is it?)

B. No

Is TSP-decision NP-Hard?

To prove a problem is NP-Complete you must do two things:
1. **Prove it is in NP**
2. **Prove it is NP-Hard**



To prove a problem is NP-Hard, you must prove it is *at least* as hard as any problem in NP.

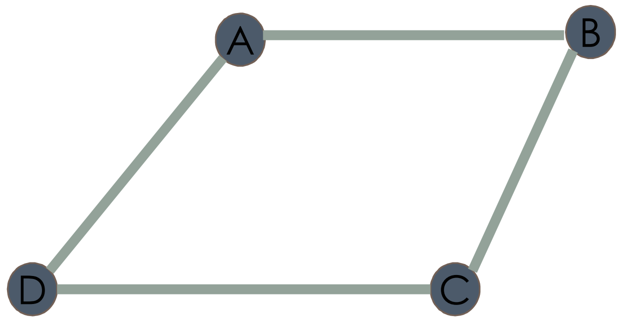
So we should choose a problem that we know is "as hard as any other problem in NP" and show that TSP-decision is at least as hard.

We'll choose a problem called "Hamiltonian Cycle"

(You have to trust me that Hamiltonian Cycle is "as hard as any other problem in NP". It's somewhat involved to prove it.)

Is TSP-decision NP-Hard?

To prove a problem is NP-Complete you must do two things:
1. **Prove it is in NP**
2. **Prove it is NP-Hard**



Goal: Show TSP-decision is at least as hard as Hamiltonian Cycle.

But how?

We will use something called a reduction where we reduce Hamiltonian Cycle to a version of TSP-decision in polynomial time.

$$\text{Hamiltonian Cycle} \le_P \text{TSP-decision}$$

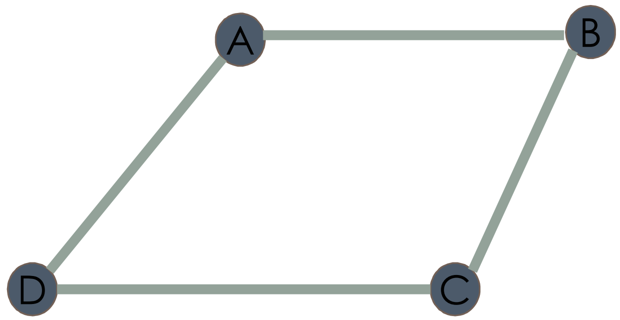
"Hamiltonian Cycle is polynomial time reducible to TSP-decision"

“TSP-Decision is at least as hard as Hamiltonian Cycle”
“A solution to TSP-Decision is powerful enough to solve Hamiltonian Cycle in polynomial time.”

Is TSP-decision NP-Hard?

To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. **Prove it is NP-Hard**



Goal: Reduce Hamiltonian Cycle TSP-decision in polynomial time

$$\text{Hamiltonian Cycle} \leq_P \text{TSP-decision}$$

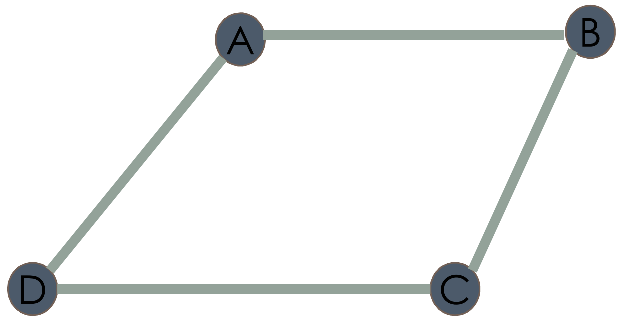
Assume you have a black box for solving TSP-decision.
Use that black box to solve Hamiltonian Cycle.

Then if the black box turns out to be polynomial time,
The solution for Hamiltonian Cycle will be polynomial time too!

Is TSP-decision NP-Hard?

To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. $\boxed{\textbf{Prove it is NP-Hard}}$



Hamiltonian Cycle $\leq_P$ TSP-decision

Assume you have a black box for solving TSP-decision.
Use that black box to solve Hamiltonian Cycle.

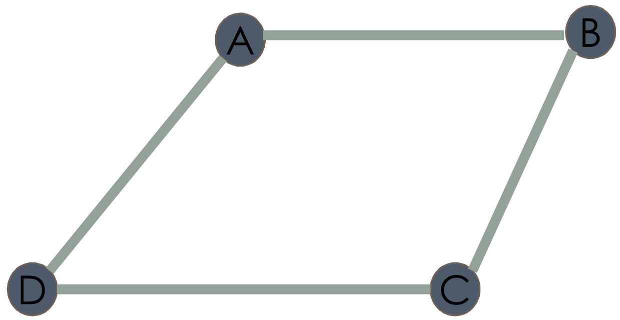
Can you directly pass this graph into TSP-decision?

A. Yes
B. No

Is TSP-decision NP-Hard?

To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. $\boxed{\textbf{Prove it is NP-Hard}}$



Goal: Reduce Hamiltonian Cycle TSP-decision in polynomial time

$$\text{Hamiltonian Cycle} \leq_P \text{TSP-decision}$$

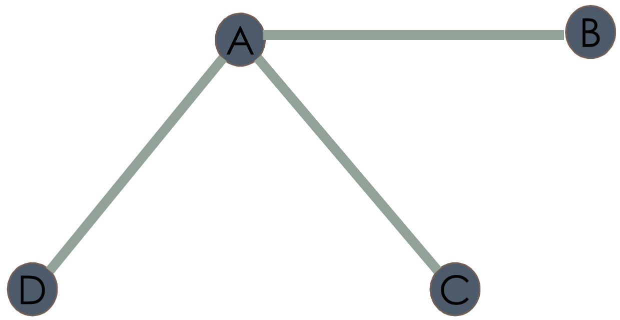
Assume we have TSP-decision(Graph, Hometown, L) that returns true if there is a tour starting and ending in Hometown with total distance < L and false if not.

```
HamiltonianCycle(G):
   Let L = |V| + 1
   Construct G', a complete graph with vertices from G
   and with edge weights as follows:
        if edge exists in G, edge weight is 1
        otherwise edge weight is INF
   Choose an arbitrary vertex v in G' as the hometown
   return TSP-decision(G', v, L)
```

Is TSP-decision NP-Hard?

To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. $\boxed{\textbf{Prove it is NP-Hard}}$



Goal: Reduce Hamiltonian Cycle TSP-decision in polynomial time

$$\text{Hamiltonian Cycle} \leq_P \text{TSP-decision}$$

Assume we have TSP-decision(Graph, Hometown, L) that returns true if there is a tour starting and ending in Hometown with total distance < L and false if not.

```
HamiltonianCycle(G):
    Let L = |V| + 1
    Construct G', a complete graph with vertices from G
    and with edge weights as follows:
        if edge exists in G, edge weight is 1
        otherwise edge weight is INF
    Choose an arbitrary vertex v in G' as the hometown
    return TSP-decision(G', v, L)
```

Clicker question break

- Reducing problem X to problem Y in polynomial time means:
  A. Assume you have a black box for solving X. Devise a polynomial time algorithm that uses that black box to solve Y.
  B. Assume you have a black box for solving Y. Devise a polynomial time algorithm that uses that black box to solve X.

Clicker question break

- Assume you know that problem C is NP-Complete. You want to show that problem D is NP-Hard. Should you:

A. Assume you have a black box for problem C and use it to solve problem D

B. Assume you have a black box for problem D and use it to solve problem C

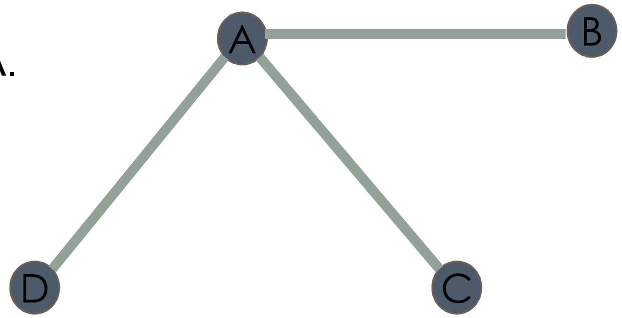
Clicker question break

- If we know problem A is "at least as hard as any problem in NP" then we can show problem B is NP hard by:
    A. Reducing problem A to problem B ($A \leq_P B$)
    B. Reducing problem B to problem A ($B \leq_P A$)

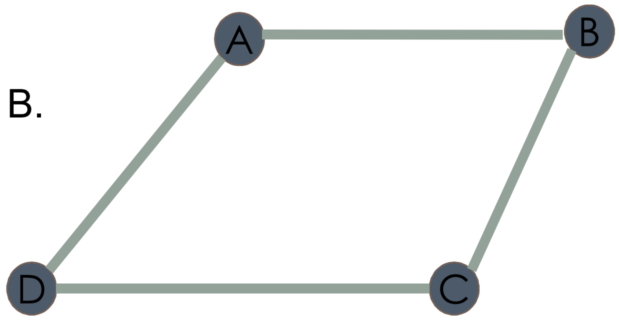
Hamiltonian Path

A Hamiltonian path is a path that visits each vertex exactly once, but doesn't return to the starting vertex
Which of the following graphs have Hamiltonian Paths?

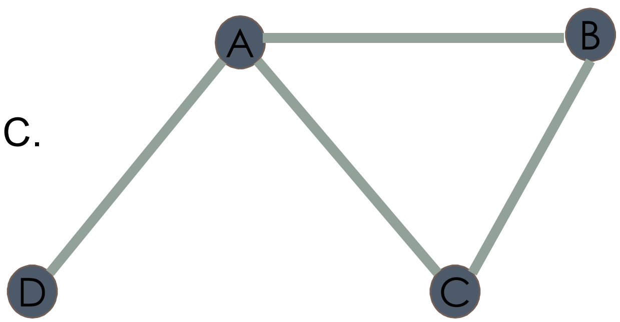
A.



B.



C.

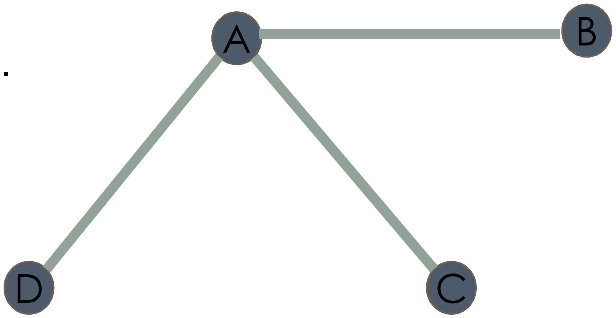


D. B&C
E. All of them

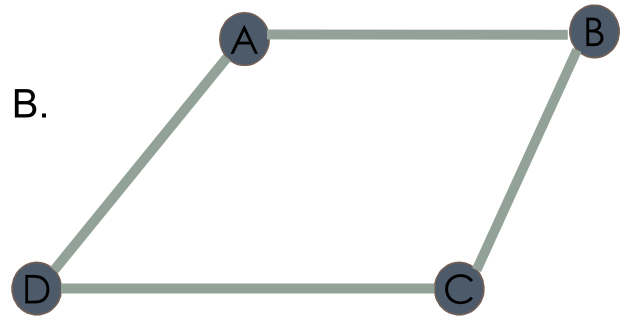
Hamiltonian Path vs. Hamiltonian Cycle

Which of the following graphs have Hamiltonian Cycles?

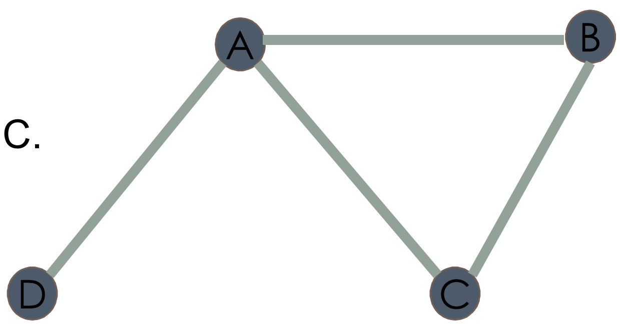
A.



B.



C.

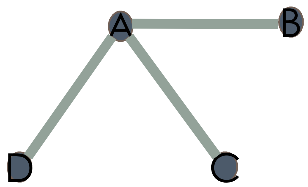


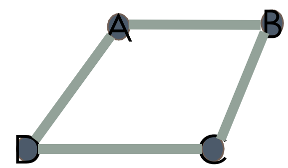
D. B&C
E. All of them

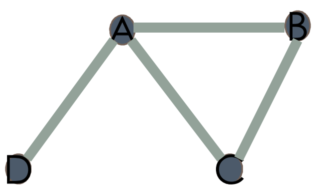
Prove Hamiltonian Path is NP-Complete

A Hamiltonian path is a path that visits each vertex exactly once, but doesn't return to the starting vertex.

Your goal, convert an instance of a Hamiltonian path problem to an instance of a Hamiltonian Cycle (in poly-time)







To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. **Prove it is NP-Hard**

Prove Hamiltonian Path is NP-Complete

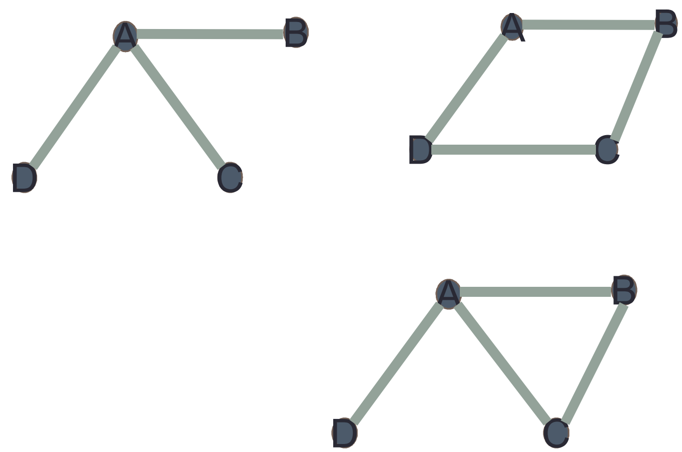
Hamiltonian Cycle $\leq_P$ Hamiltonian Path

"Hamiltonian Cycle reduces to Hamiltonian Path"

```
HamiltonianCycle(G):
    Construct G', with vertices and edges from G
    Arbitrarily choose a vertex v and make a copy v' such that
    v' has all the same edges to all the same vertices as v.
    Then add two new vertices to G', w and w' such that there
    is a single edge between w and v and a single edge
    between w' and v'.
    return HamiltonianPath(G').
```

If G has a Ham cycle, then G' has a Ham Path that starts in w and ends in w'.

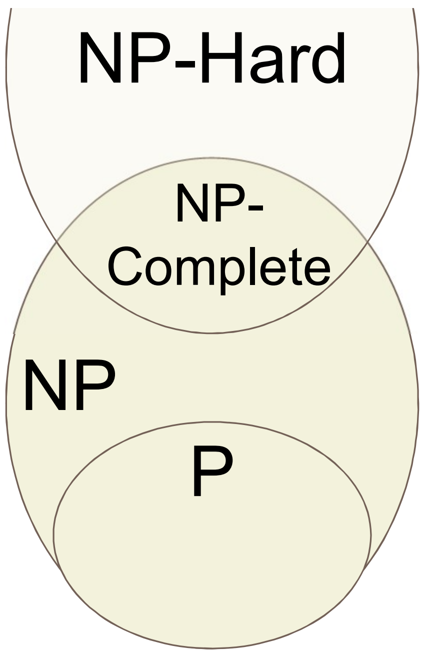
If G did not have a Ham cycle, then G' cannot have a Ham path. Any Ham path in G' must start in w and end in w' (or vice versa). This means it must get from v to v' via a Hamiltonian path, which is only possible if there was a Hamiltonian cycle in the original graph.



To prove a problem is NP-Complete you must do two things:

1. **Prove it is in NP**
2. **Prove it is NP-Hard**

Complexity Theory



How you might use this?

You are given a new problem.

You want to know if it's hard to solve…

Is it similar to known NP-complete problems?

Satisfiability problem (SAT)

Given a set of boolean values $x_{1,}$ …, x_n, does there exist an interpretation which satisfies a Boolean formula N?

Example formula:
$(x_1 \lor x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3 \lor \neg x_4)$

Satisfiability problem (SAT)

Example formula:

$(x_1 \vee x_2) \wedge$
$(\neg x_1 \vee \neg x_2) \wedge$
$(x_1 \vee \neg x_2) \wedge$
$(\neg x_1 \vee x_2 \vee x_3) \wedge$
$(\neg x_2 \vee \neg x_3)$

Is there a solution to the equation above?

A. Yes
B. No

Satisfiability problem (SAT) -> 3SAT

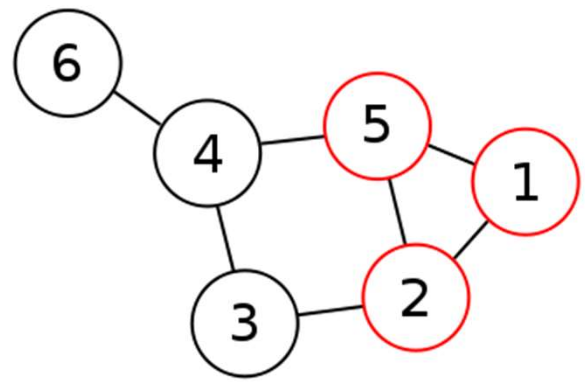
Notice each clause has 3 literals?

$(x_1 \vee x_2) \wedge$
$(\neg x_1 \vee \neg x_2) \wedge$
$(x_1 \vee \neg x_2) \wedge$
$(\neg x_1 \vee x_2 \vee x_3) \wedge$
$(\neg x_2 \vee \neg x_3)$

Any more than 3 literal problem can be converted to 3 literal.

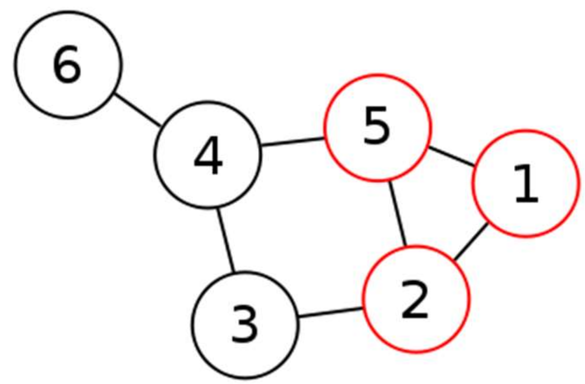
Clique

A clique in a graph G is a complete subgraph of G. It is a subset K of the vertices such that every two vertices in K are connected by an edge in G.



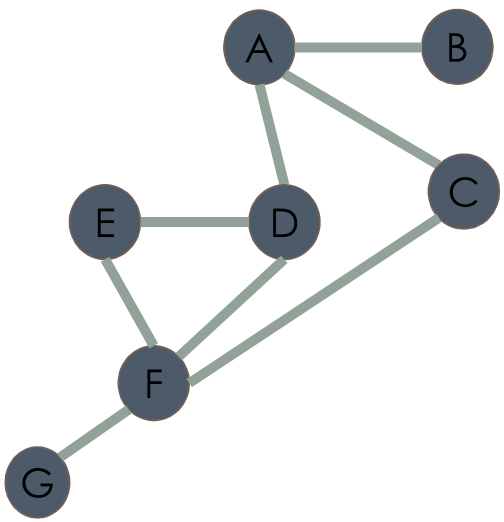
Vertex Cover

Given a graph G, what is the smallest set of vertices such that each edge in the graph is adjacent to at least one vertex in the set.



Vertex Cover

Given a graph G, what is the smallest set of vertices such that each edge in the graph is adjacent to at least one vertex in the set.



In this example above, what is the smallest vertex cover?

Set Cover

Given a set of elements {1,2,...,n} (the universe) and a collection S of m sets whose union equals the universe, the set cover problem is to identify the smallest sub-collection of S whose union equals the universe.

For example, consider the universe U={1,2,3,4,5,6} and the collection of sets

S={ {1,2,3}, {2,3,4,5}, {4,5,6}, {3,4,5} }

What is the smallest sub-collection whose union is the universe?

Knapsack

Given a set of items, each with a value and weight, determine the elements to add to the collection such that the weight is below a limit w and the value is as large as possible.

A: 1 lbs, $2
B: 12 lbs, $14
C: 4 lbs, $9
D: 2 lbs, $1
E: 1 lbs, $1
F: 3 lbs, $4

Assuming you have as many of each item in set A-F as you desire, what is the best solution to keep the weight at <= 19 lbs?

Subset Sum

Given a set of integers, is there a non-empty subset of those integers whose sum is 0?

-20, -8, -4, -2, 5, 9

Is there a solution to this problem above?

A. Yes
B. No

Sudoku

Given an n x n matrix, is there a solution?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

3x3 puzzle

If you can reduce to 3-SAT…

There are powerful known 3SAT solvers. If they succeed, they can solve "hard" problems quickly. But if they can't, runtimes explode.

Many, many more problems

https://en.wikipedia.org/wiki/List_of_NP-complete_problems