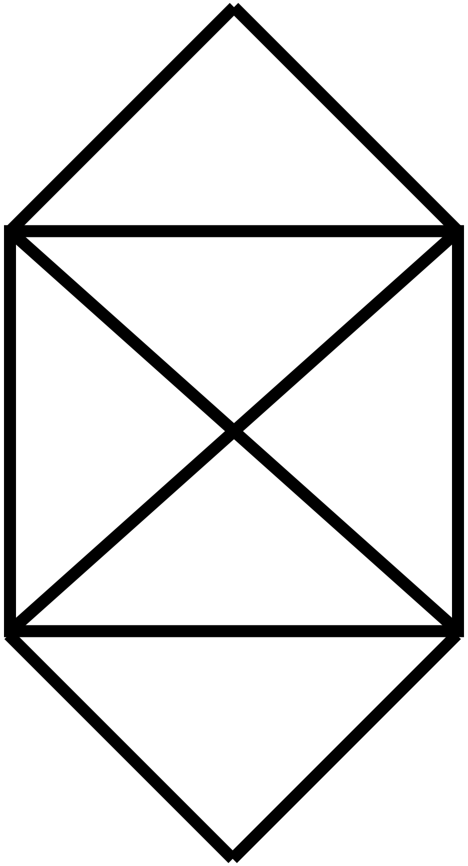
P, NP, NP-Complete

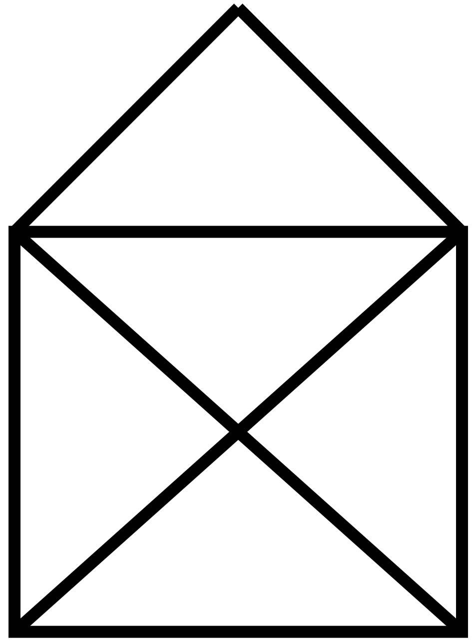
Ruth Anderson

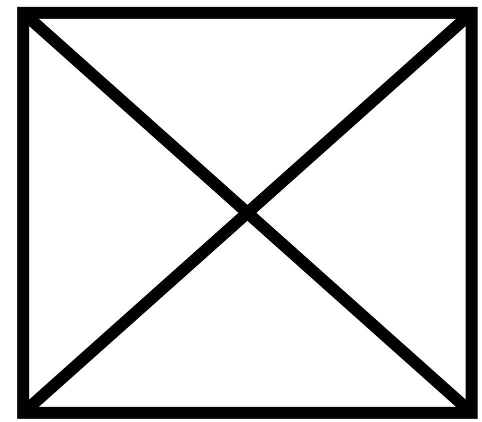
Today's Agenda

- A Few Problems:
  - Euler Circuits
  - Hamiltonian Circuits
- Intractability: P and NP
- NP-Complete
- What now?

Try it!







Which of these can you draw (trace all edges) without lifting your pencil, drawing each line only once?
Can you start and end at the same point?

Your First Task

- Your company has to inspect a set of roads between cities by driving over each of them.
- Driving over the roads costs money (fuel), and there are a lot of roads.

- Your boss wants you to figure out how to *drive over each road exactly once,* returning to your starting point.

Euler Circuits

- Euler circuit: a path through a graph that *visits each* ***edge*** *exactly once* and *starts and ends at the same vertex*

- Named after Leonhard Euler (1707-1783), who cracked this problem and founded graph theory in 1736

- An Euler circuit exists *iff*
  - the graph is connected and
  - each vertex has even degree (= # of edges on the vertex)

The Road Inspector: Finding Euler Circuits

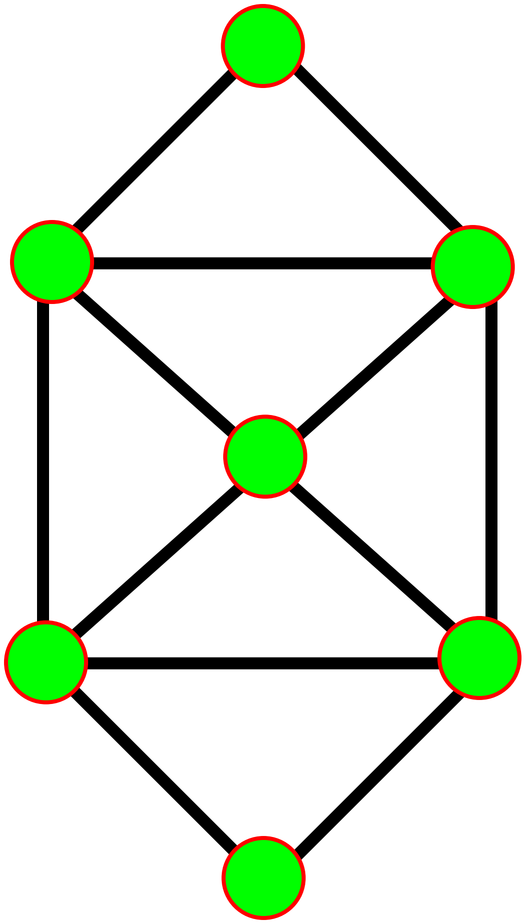
Given a graph G = (V,E), find an Euler circuit in G

Can check if one exists:
- Check if all vertices have even degree

Basic Euler Circuit Algorithm:

1. Do a depth first search from a start vertex until you are back to the start vertex.
   - You never get stuck because of the even degree property.
2. "Remove" the walk, leaving several components each with the even degree property.
   - Recursively find Euler circuits for these.
3. Splice all these circuits into an Euler circuit

Running time?



The Road Inspector: Finding Euler Circuits

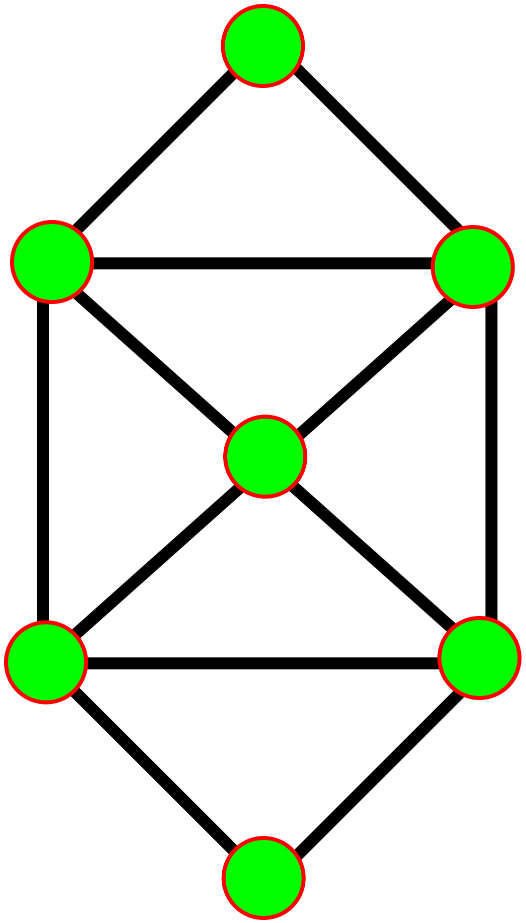
Given a graph G = (V,E), find an Euler circuit in G

Can check if one exists: (in O(|V|+|E|))

- Check if all vertices have even degree

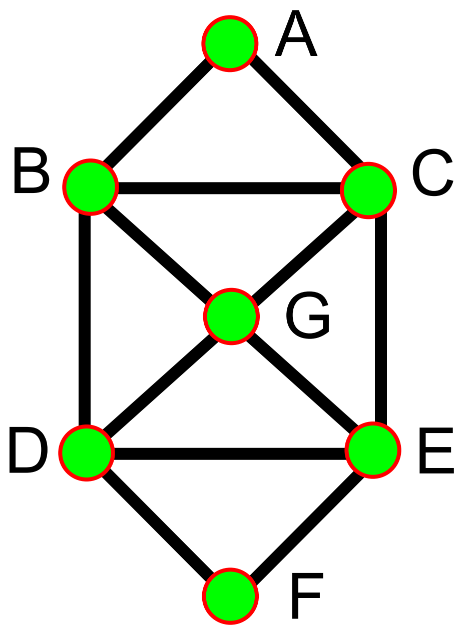
Basic Euler Circuit Algorithm:

1. Do a depth first search from a start vertex until you are back to the start vertex.
   - You never get stuck because of the even degree property.
2. "Remove" the walk, leaving several components each with the even degree property.
   - Recursively find Euler circuits for these.
3. Splice all these circuits into an Euler circuit



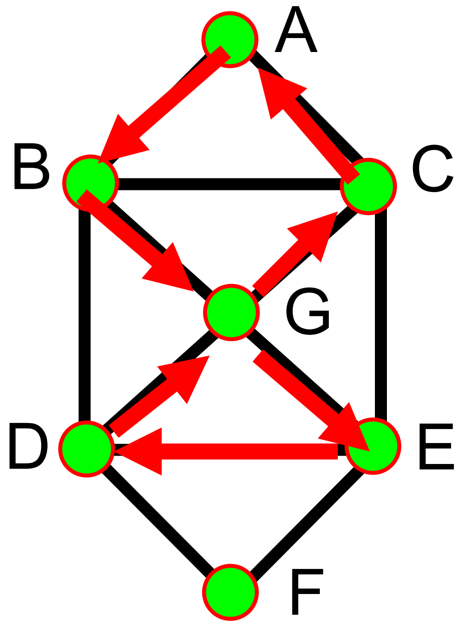
Running time? O(|V|+|E|)

Euler Circuit Example



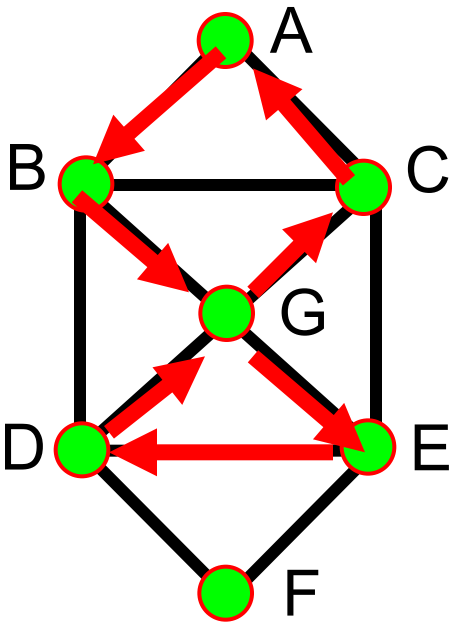
Euler(A) :

Euler Circuit Example

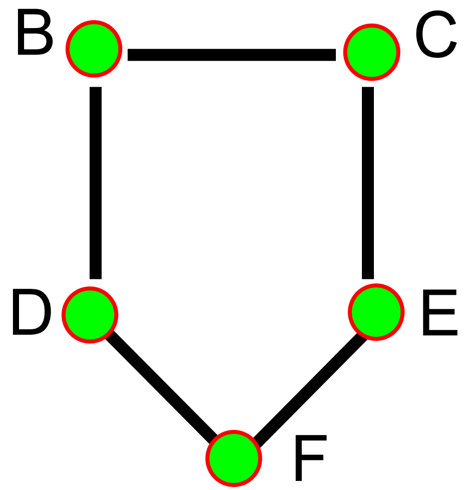


Euler(A) :
A B G E D G C A

Euler Circuit Example

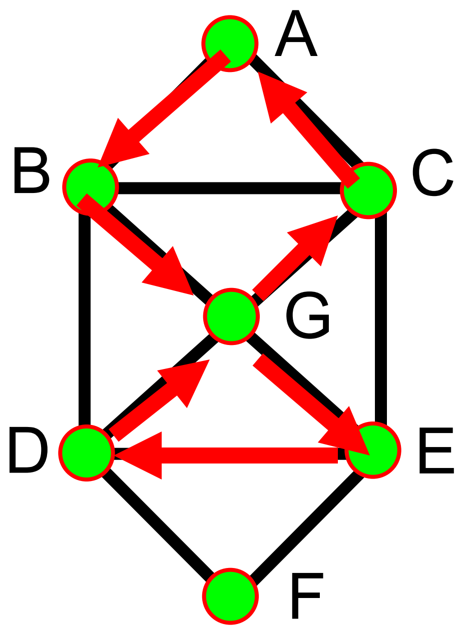


Euler(A) :
A B G E D G C A



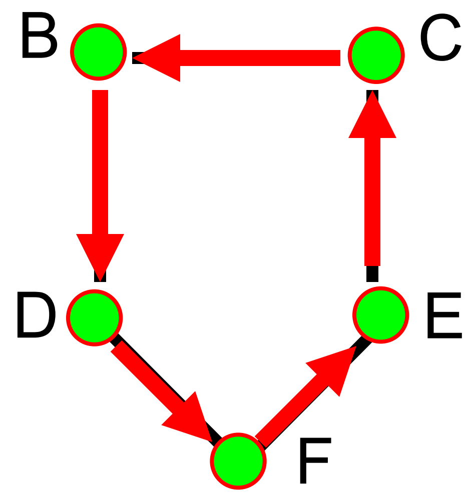
Euler(B)

Euler Circuit Example



Euler(A) :

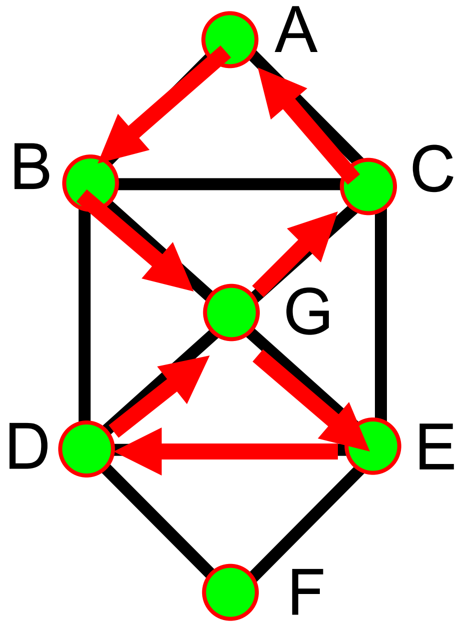
A <u>B</u> G E D G C A

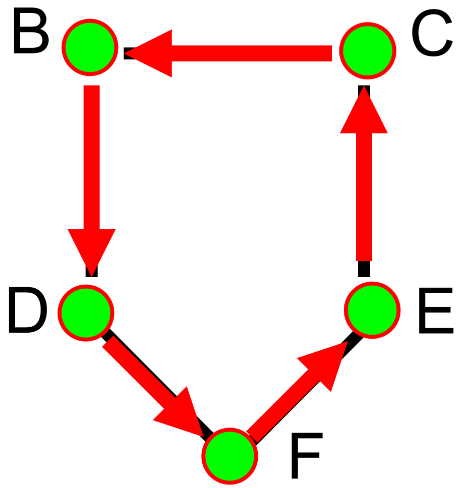


Euler(B):

B D F E C B

Euler Circuit Example





Euler(A) :
A B G E D G C A

Euler(B):
B D F E C B

Splice

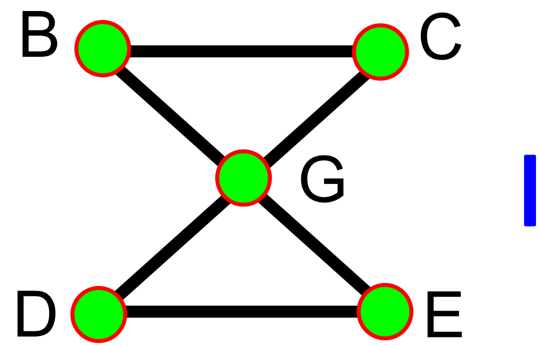
A B D F E C B G E D G C A

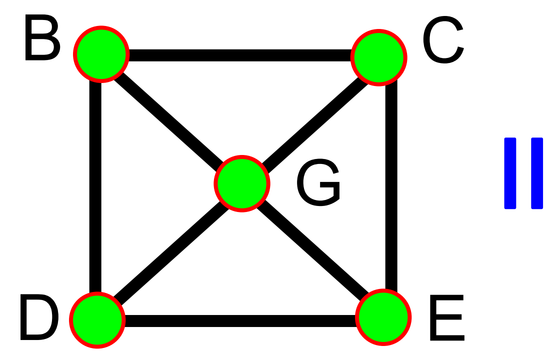
Your Second Task

- Your boss is pleased…and assigns you a new task.
- Your company has to send someone by car to a set of cities.
- The primary cost is the exorbitant toll going into each city.
- Your boss wants you to figure out *how to drive to each city exactly once*, *returning in the end to the city of origin*.

Hamiltonian Circuits

- Euler circuit: A cycle that goes through each *edge* exactly once
- Hamiltonian circuit: A cycle that goes through each *vertex* exactly once
- Does graph **I** have:
  - An Euler circuit?
  - A Hamiltonian circuit?
- Does graph **II** have:
  - An Euler circuit?
  - A Hamiltonian circuit?





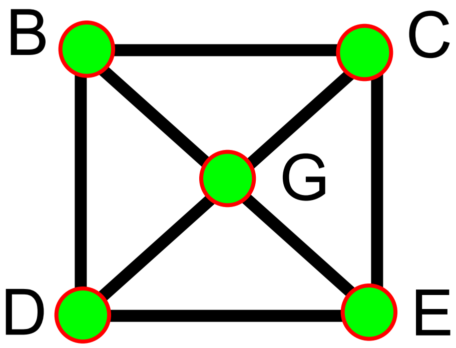
Finding Hamiltonian Circuits

- **Problem**: Find a Hamiltonian circuit in a graph G

- One solution: Search through *all paths* to find one that visits each vertex exactly once
  - Can use your favorite graph search algorithm to find paths
- This is an *exhaustive search* ("brute force") algorithm

- Worst case: need to search all paths
  - How many paths??

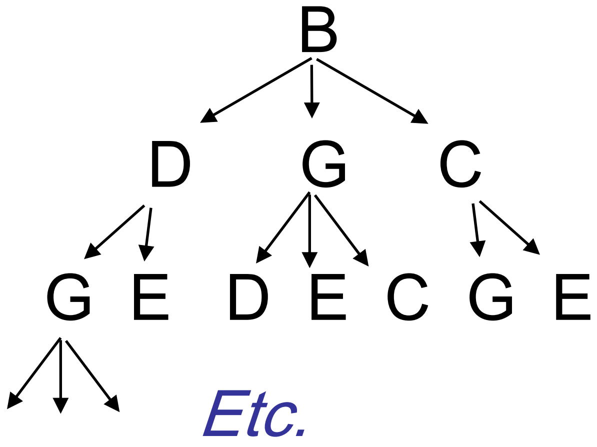
Analysis of Exhaustive Search Algorithm



Worst case: need to search all paths

- How many paths?

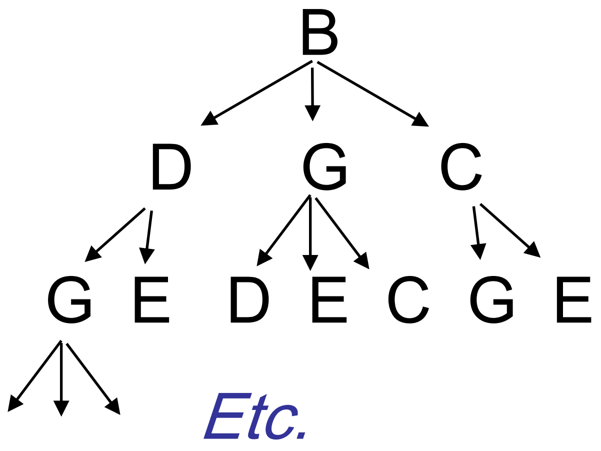
Can depict these paths as a *search tree:*



Search tree of paths from B

Analysis of Exhaustive Search Algorithm

- Let the *average* branching factor of each node in this tree be b
- |V| vertices, each with ≈ b branches
- Total number of paths ≈ b·b·b … ·b
- Worst case →



Search tree of paths from B

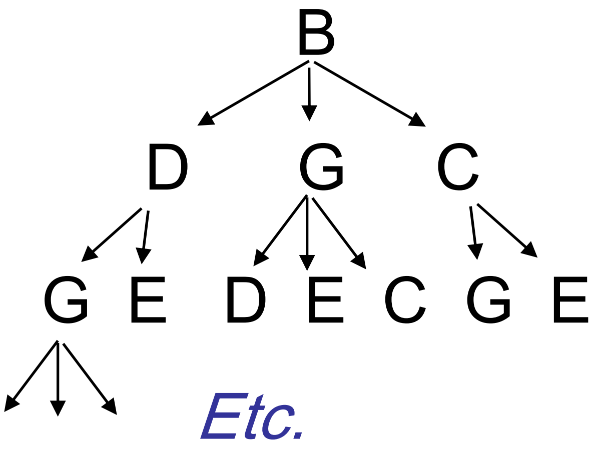
Analysis of Exhaustive Search Algorithm

- Let the *average* branching factor of each node in this tree be b

- |V| vertices, each with ≈ b branches

- Total number of paths ≈ b·b·b … ·b
  = $O(b^{|V|})$

- Worst case → Exponential time!



Search tree of paths from B

Running Times

TABLE 2 **The Computer Time Used by Algorithms.**

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s	10^{-8} s	3×10^{-7} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s	4×10^{11} yr	*
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s	*	*
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s	*	*
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s	*	*
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min	*	*

Time needed to solve problems of various sizes with an algorithm using the indicated number *n* of bit operations, assuming that each bit operation takes 10^{-11} seconds, a reasonable estimate of the time required for a bit operation using the fastest computers available today. **Times of more than 10^{100} years are indicated with an asterisk.** In the future, these times will decrease as faster computers are developed.

From Rosen, *Discrete Mathematics and Its Applications, 2012*

Polynomial vs. Exponential Time

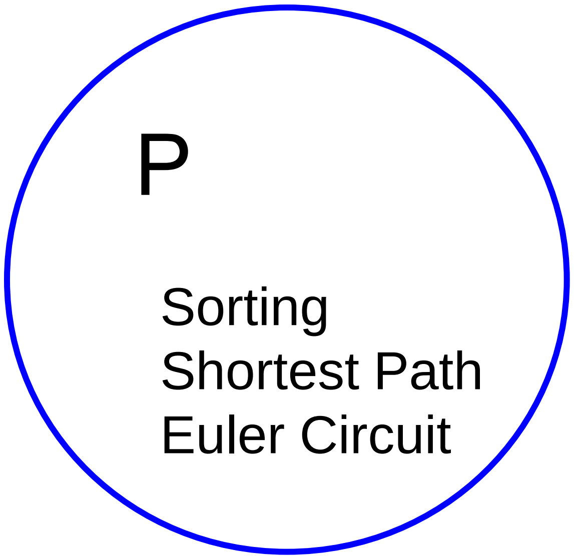
- All of the algorithms we have discussed in this class have been **polynomial time** algorithms:
  - Examples: $O(\log N)$, $O(N)$, $O(N \log N)$, $O(N^2)$
  - Algorithms whose running time is $O(N^k)$ for some $k > 0$

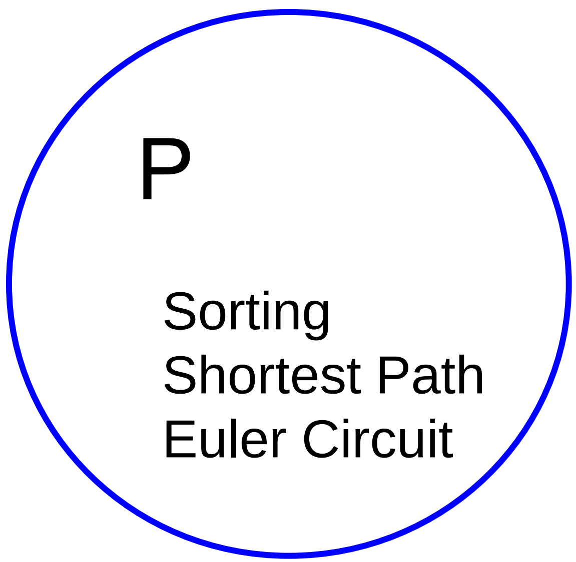
- **Exponential time** b^N is asymptotically worse than any polynomial function N^k for any k

The Complexity Class P

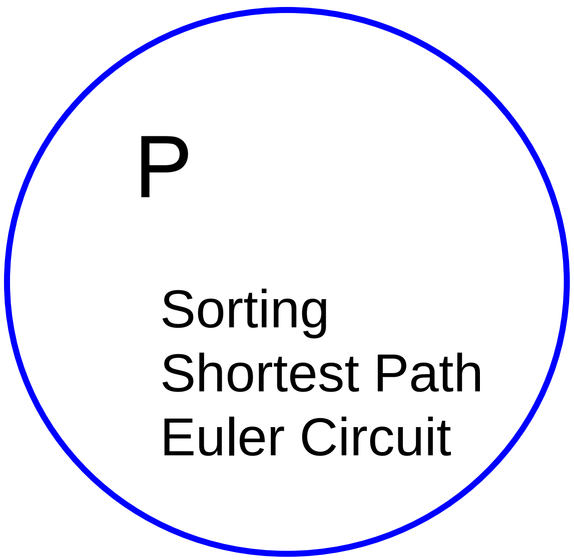
- P is the set of all problems that can be solved in *polynomial time*
  - All *problems* that have some *algorithm* whose running time is $O(N^k)$ for some k

- Examples of problems in P:
  sorting, shortest path, Euler circuit, *etc*.


P
Sorting
Shortest Path
Euler Circuit



Hamiltonian Circuit



Hamiltonian Circuit
Satisfiability (SAT)
Vertex Cover
Travelling Salesman

Satisfiability

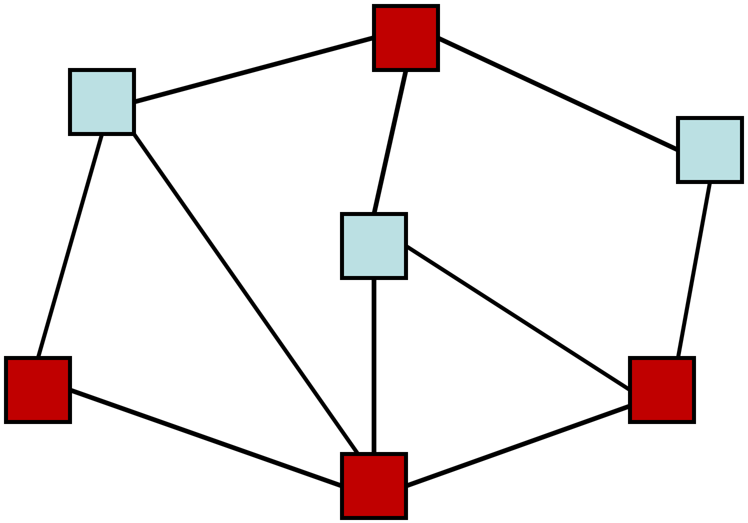
$$(\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee \neg x_5)$$

Input: a logic formula of size **m** containing **n** variables

Output: An assignment of Boolean values to the variables in the formula such that the formula is true

$O(\mathbf{m} * 2^{n})$ algorithm: Try every variable assignment

Vertex Cover:



Input: A graph (**V**,**E**) and a number **m**

Output: A subset **S** of **V** such that *for every edge* (**u**,**v**) in **E**, at least one of **u** or **v** is in **S** and **|S|=m** (if such an **S** exists)

$O(2^m)$ algorithm: Try every subset of vertices of size **m**

Traveling Salesman

Input: A *complete* *weighted* graph (**V**,**E**) and a number **m**

Output: A circuit that visits each vertex exactly once and has total cost < **m** if one exists

O(**|V|!**) algorithm: Try every path, stop if find cheap enough one

A Glimmer of Hope

- If given a candidate solution to a problem, we can **check if that solution is correct in polynomial-time**, then **maybe** a polynomial-time solution exists?

- Can we do this with Hamiltonian Circuit?
  - Given a candidate path, is it a Hamiltonian Circuit?

A Glimmer of Hope

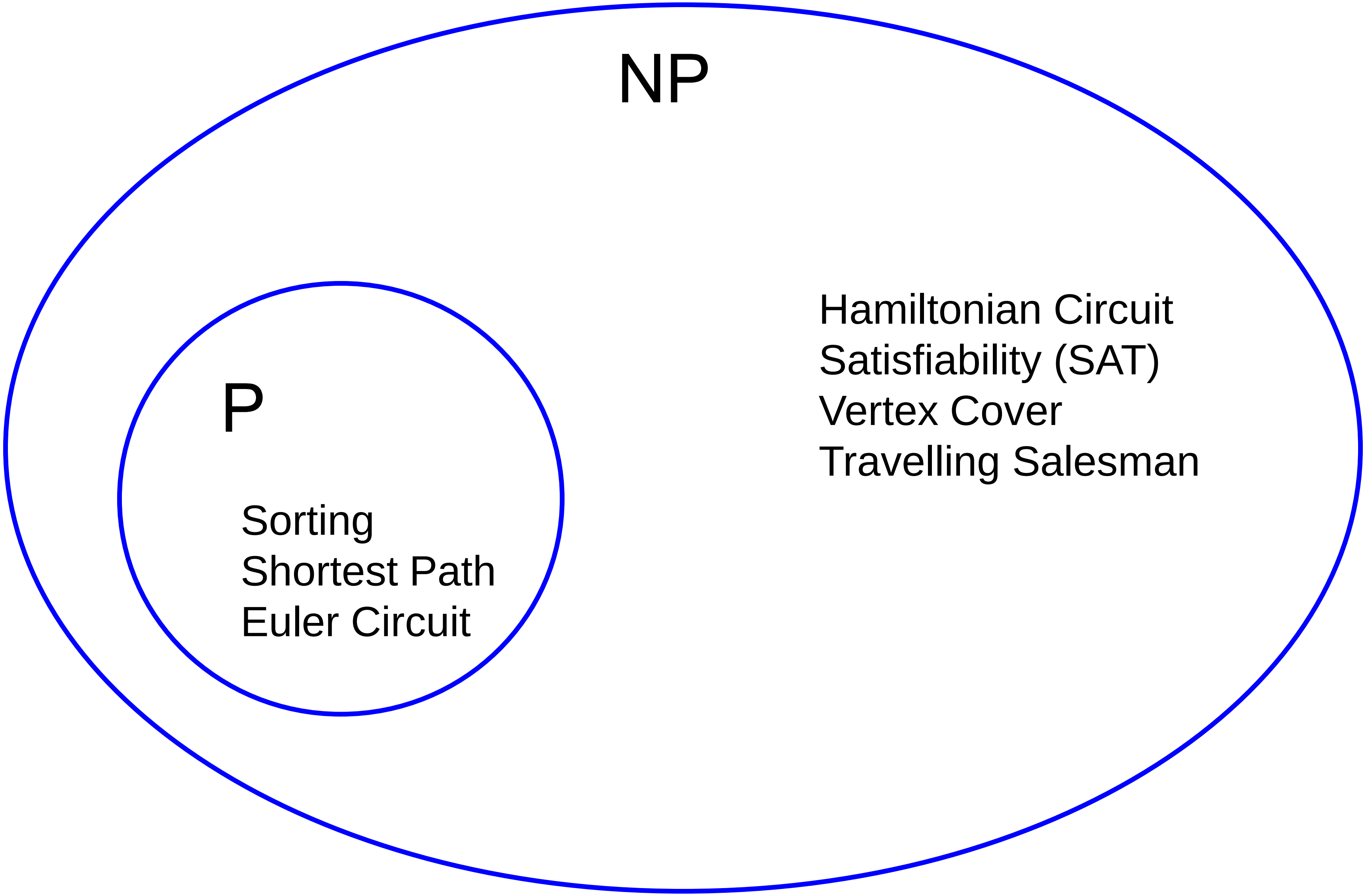
- If given a candidate solution to a problem, we can **check if that solution is correct in polynomial-time**, then **maybe** a polynomial-time solution exists?

- Can we do this with Hamiltonian Circuit?
  - Given a candidate path, is it a Hamiltonian Circuit? just check if all vertices are visited exactly once in the candidate path

The Complexity Class NP

- *Definition*: NP is the set of all problems for which a given *candidate solution* can be *tested* in polynomial time

- Examples of problems in NP:
  - *Hamiltonian circuit:* Given a candidate path, can test in linear time if it is a Hamiltonian circuit
  - *Satisfiability:* Given a circuit made out of AND, OR, NOT gates, and an assignment of values, is the output "1"?
  - *All problems that are in P   (why?)*


NP
P
Sorting
Shortest Path
Euler Circuit
Hamiltonian Circuit
Satisfiability (SAT)
Vertex Cover
Travelling Salesman

Why do we call it "NP"?

- NP stands for *Nondeterministic Polynomial time*
  - Why "nondeterministic"? Corresponds to algorithms that can guess a solution (if it exists), the solution is then verified to be correct in polynomial time
  - Can also think of as allowing a special operation that allows the algorithm to magically guess the right choice at each branch point.
  - Nondeterministic algorithms don't exist – purely theoretical idea invented to understand how hard a problem could be

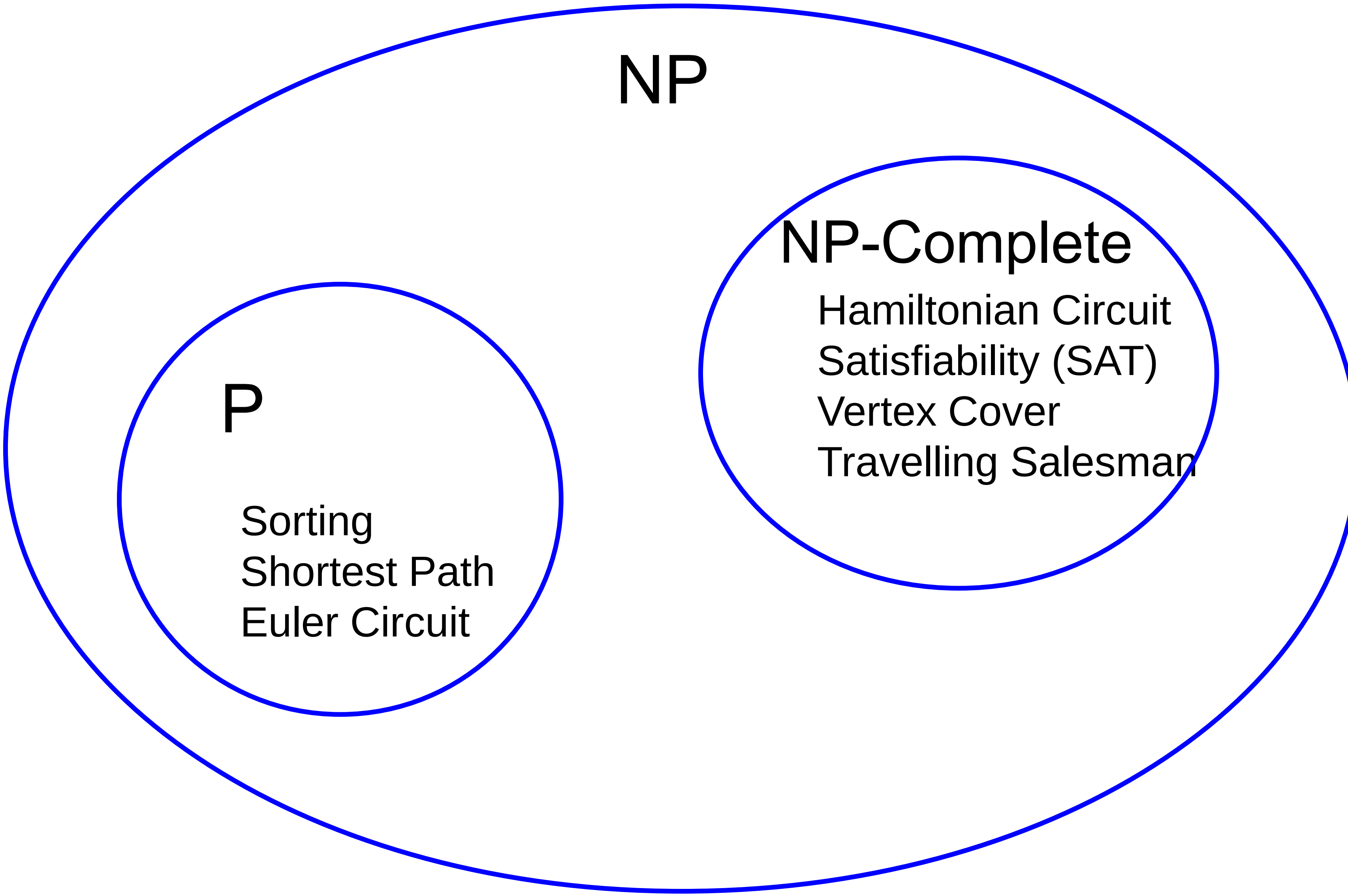
Your Chance to Win a Turing Award!

It is generally believed that $P \neq NP$,
i.e. there are problems in NP that are **not** in P

- But no one has been able to show even one such problem!
- This is the fundamental open problem in theoretical computer science
- Nearly everyone has given up trying to prove it. Instead, theoreticians prove theorems about what follows once we assume $P \neq NP$!

NP-completeness

- Set of problems in NP that (we are pretty sure) ***cannot*** be solved in polynomial time.
- These are thought of as the **hardest** problems in the class NP.
- **Interesting fact:** If any one NP-complete problem could be solved in polynomial time, then ***all*** NP-complete problems could be solved in polynomial time.
- Even more: If any NP-complete problem is in P, then all of NP is in P


NP
NP-Complete
Hamiltonian Circuit
Satisfiability (SAT)
Vertex Cover
Travelling Salesman
P
Sorting
Shortest Path
Euler Circuit

Saving Your Job

- Try as you might, every solution you come up with for the Hamiltonian Circuit problem runs in exponential time…..
- You have to report back to your boss.
- Your options:
  - Keep working
  - Come up with an alternative plan…

In general, what to do with a Hard Problem

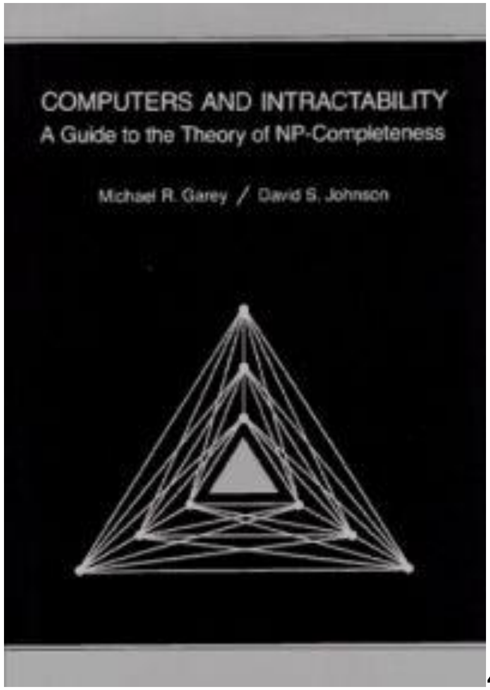
- Your problem seems really hard.
- If you can transform an NP-complete problem into the one you're trying to solve, then you can stop working on your problem!

What do we do about it?

- Approximation Algorithm:
  - Can we get an efficient algorithm that guarantees something *close* to optimal?
- Heuristics:
  - Can we get something that seems to work well *most* of the time?
- Restrictions:
  - Maybe you have stated your problem too generally. Many hard problems are easy for restricted inputs.

Great Quick Reference

- *Computers and Intractability: A Guide to the Theory of NP-Completeness*, by Michael S. Garey and David S. Johnson



EXTRA SLIDES

EXTRA SLIDES

Your Third Task

- Your boss buys your story that others couldn’t solve the last problem.
- Again, your company has to send someone by car to a set of cities.
- The primary cost is distance traveled (which translates to fuel costs).
- Your boss wants you to figure out *how to drive to each city exactly once*, then returning to the first city, while *staying within a fixed mileage budget C*.

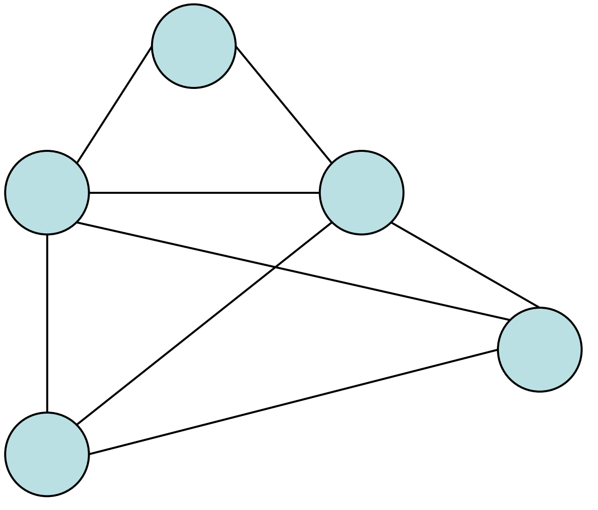
Travelling Salesman Problem (TSP)

- Your third task is basically TSP:
  - Given complete weighted graph G, integer k.
  - Is there a cycle that visits all vertices with cost <= k?
- One of the canonical problems.
- Note difference from Hamiltonian cycle:
  - graph is complete
  - we care about weight.

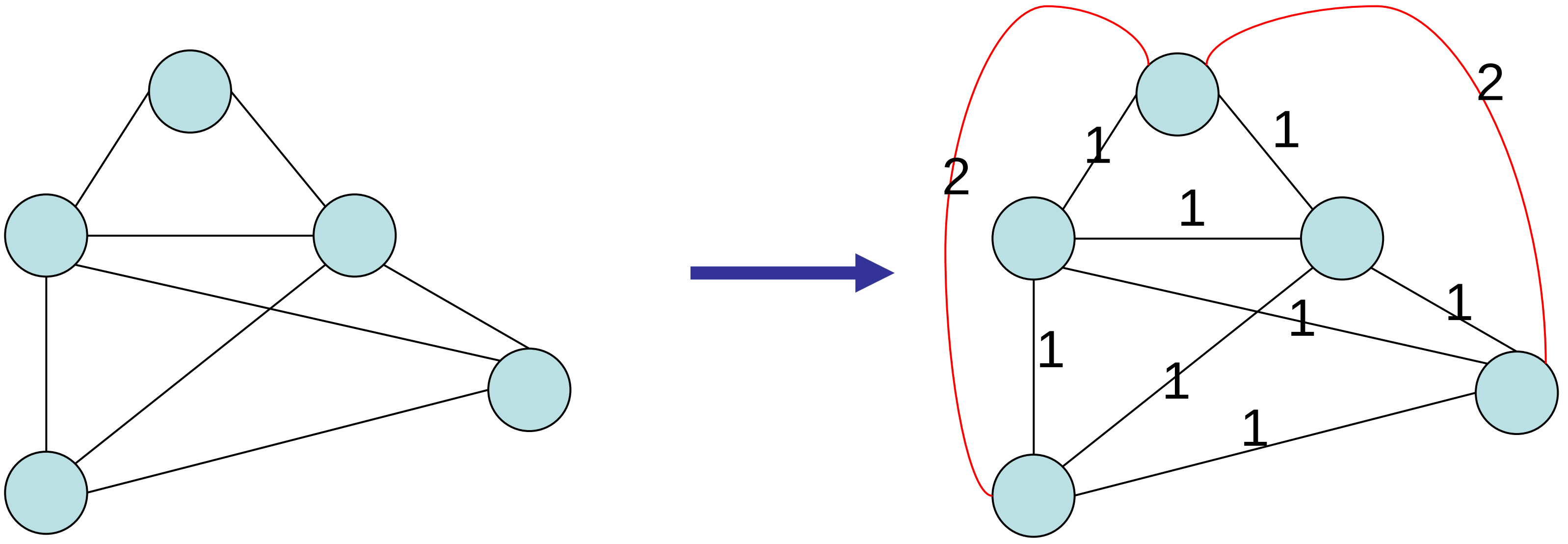
Transforming Hamiltonian Cycle to TSP

- We can reduce Hamiltonian Cycle to TSP.
- Given graph G=(V, E):
  - Assign weight of 1 to each edge
  - Augment the graph with edges until it is a complete graph G’=(V, E’)
  - Assign weights of 2 to the new edges
  - Let k = |V|.

Example



Example



Polynomial-time transformation

- G' has a TSP tour of weight |V| iff (if and only if) G has a Hamiltonian Cycle.
- What was the cost of transforming HC into TSP?

- In the end, because there is a polynomial time transformation from HC to TSP, we say *TSP is "at least as hard as" Hamiltonian cycle.*

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long