



# CSE 332: Data Structures & Parallelism

## Lecture 24: P, NP, NP-Complete (part 1)

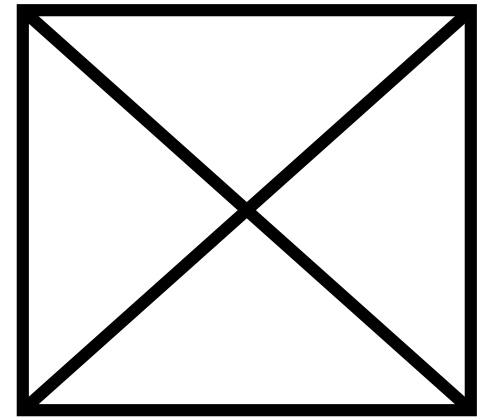
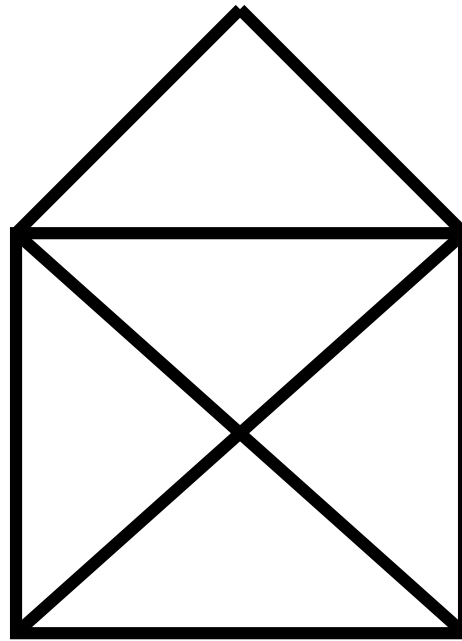
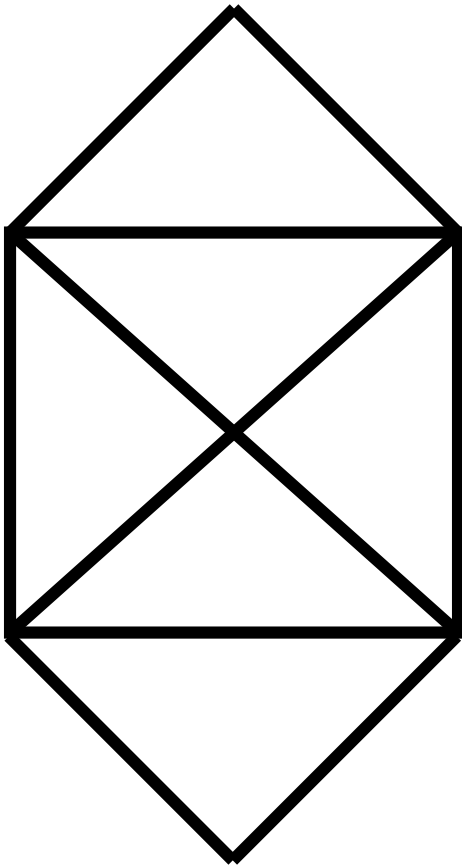
Ruth Anderson

Autumn 2019

# Agenda (for next 2 lectures)

- 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

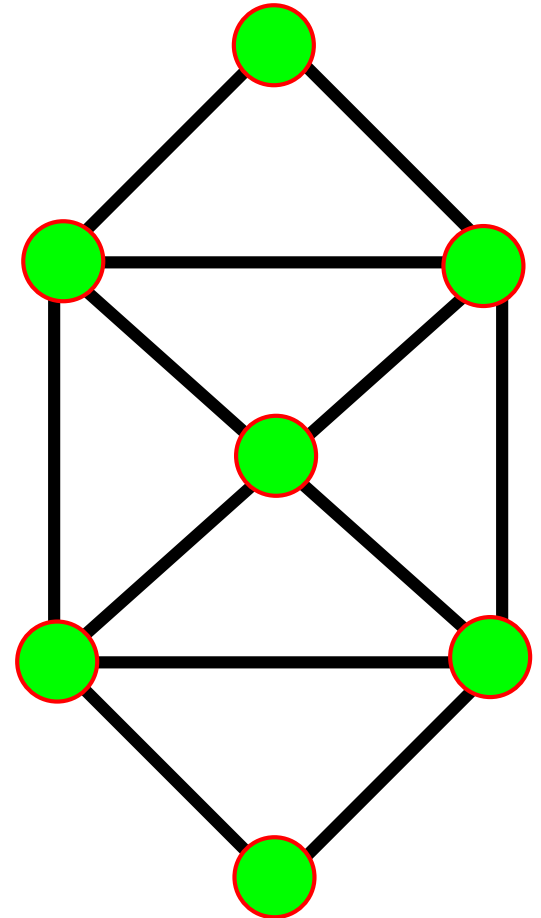
Given a connected, undirected graph  $G = (V, E)$ , find an Euler circuit in  $G$

Can check if one exists:

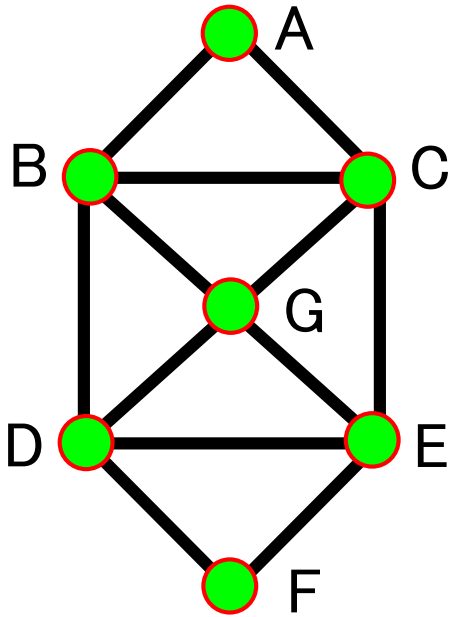
- Check if all vertices have even degree

Basic Euler Circuit Algorithm:

1. Do an edge walk 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

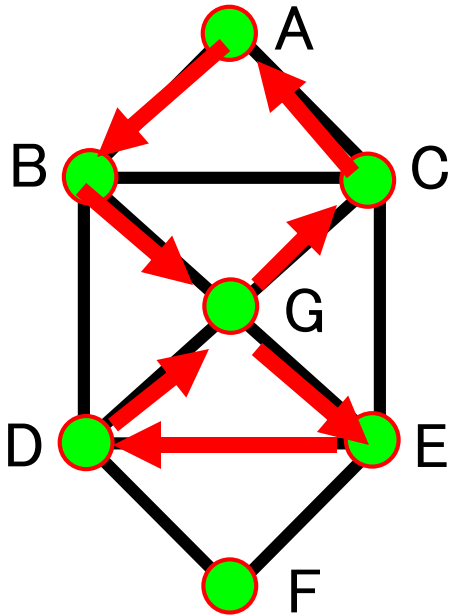


# Euler Circuit Example



Euler(A) :

# Euler Circuit Example

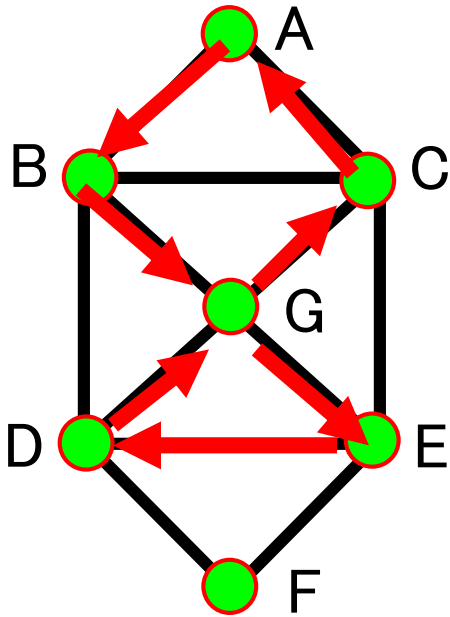


Euler(A) :

A B G E D G C A

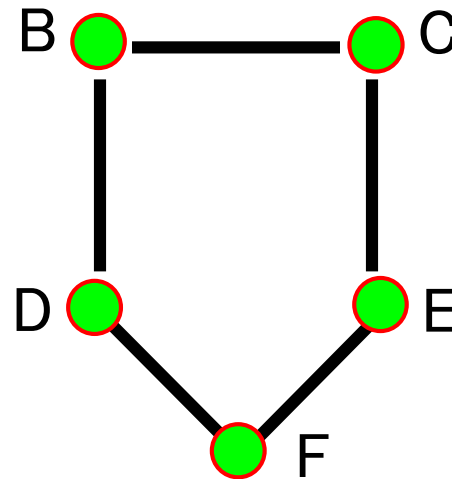


# Euler Circuit Example



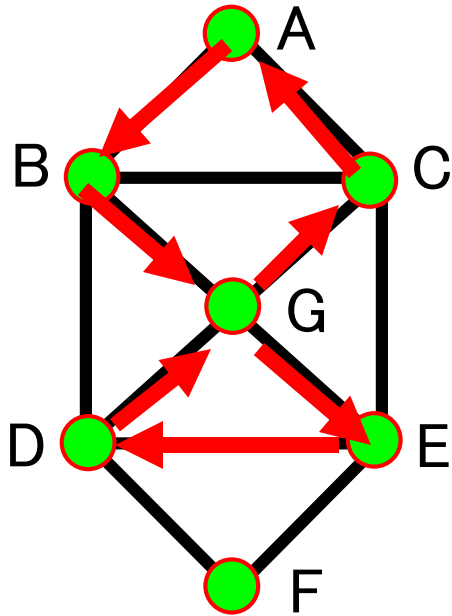
Euler(A) :

A B G E D G C A



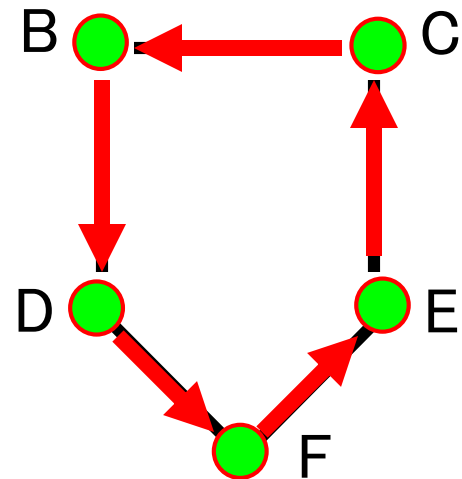
Euler(B)

# Euler Circuit Example



Euler(A) :

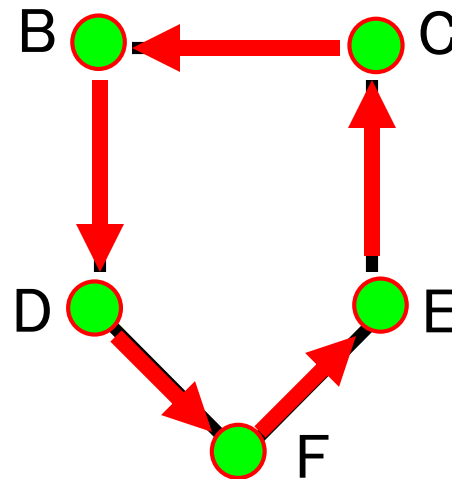
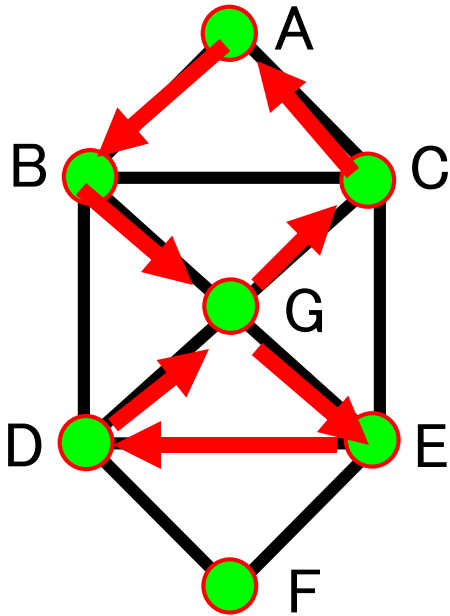
A B G E D G C A



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

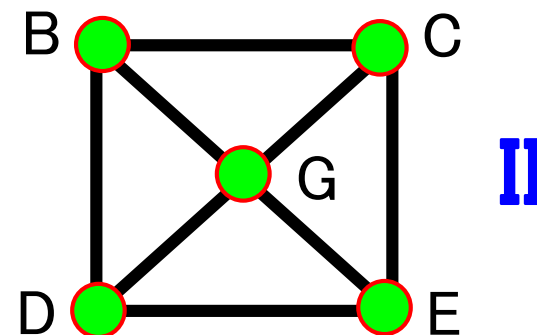
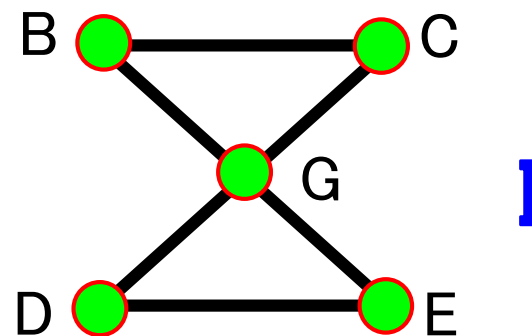
12/04/2019 A B D F E C B G E D G C A

# 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?
- Which problem sounds harder?



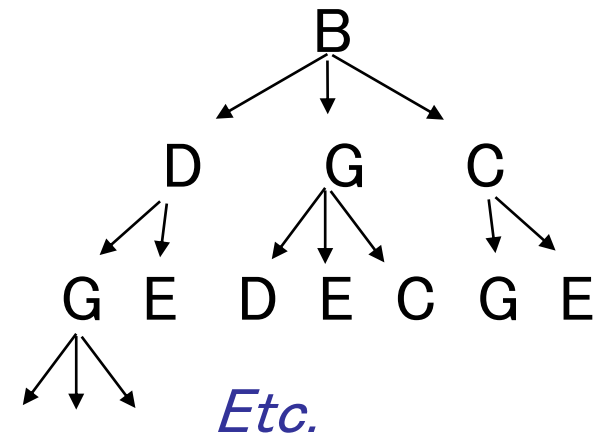
# Finding Hamiltonian Circuits

- **Problem:** Find a Hamiltonian circuit in a connected, undirected 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

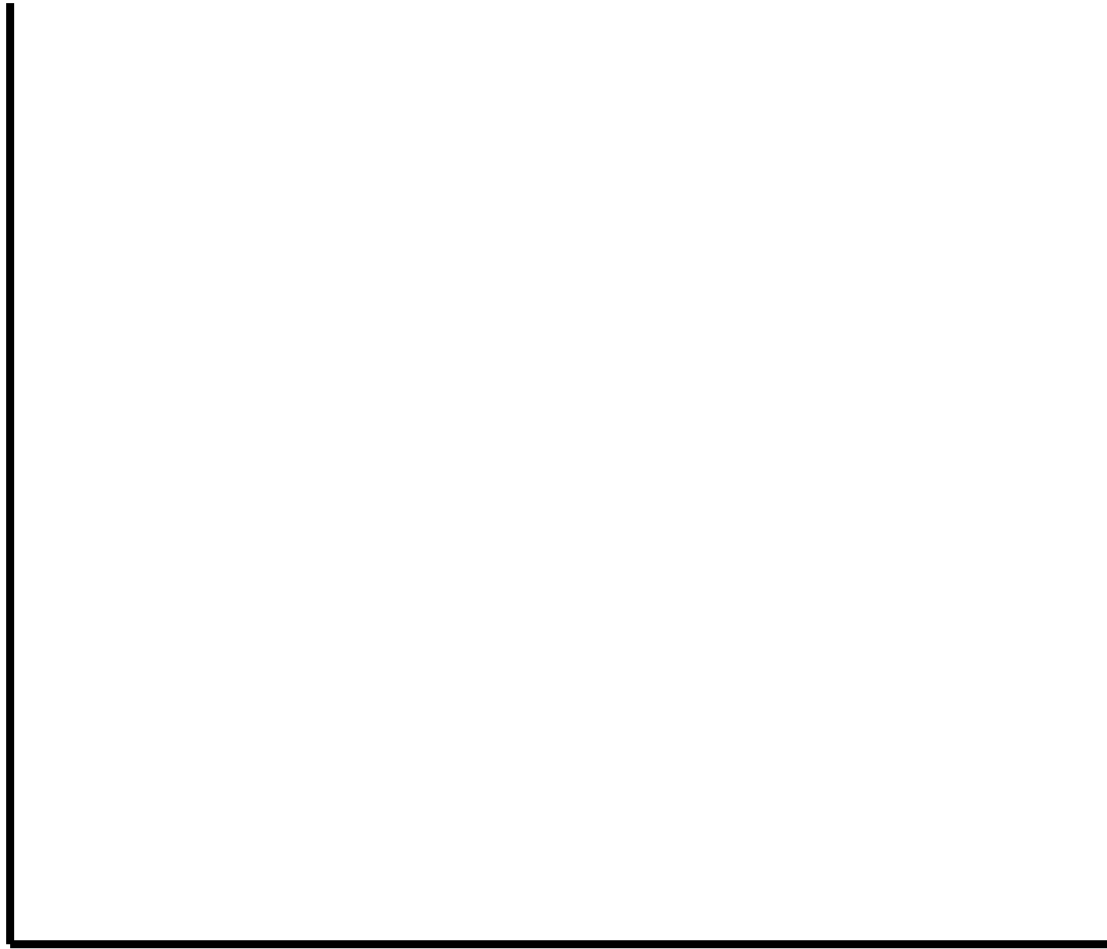
- Let the *average* branching factor of each node in this tree be  $b$
- $|V|$  vertices, each with  $\approx b$  branches
- Total number of paths  $\approx b \cdot b \cdot b \dots \cdot b$
- Worst case  $\rightarrow$



*Search tree of paths from B*



# Running Times



# More Running Times

**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

Somewhat old, from Rosen

# 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  $k$

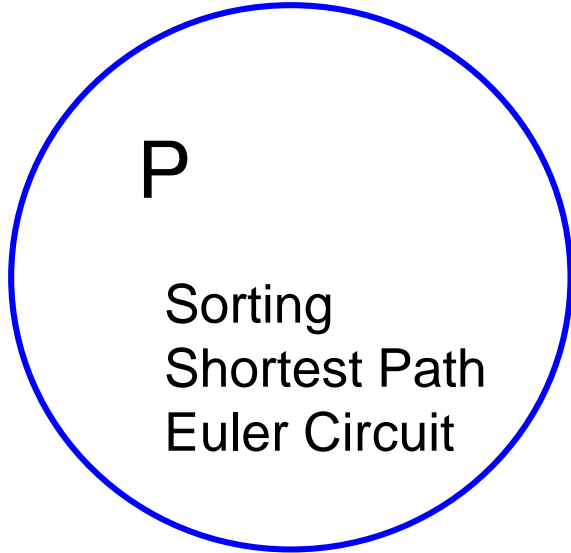
# The Complexity Class P

- P is the set of all problems that can be solved in *polynomial worst case 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.*

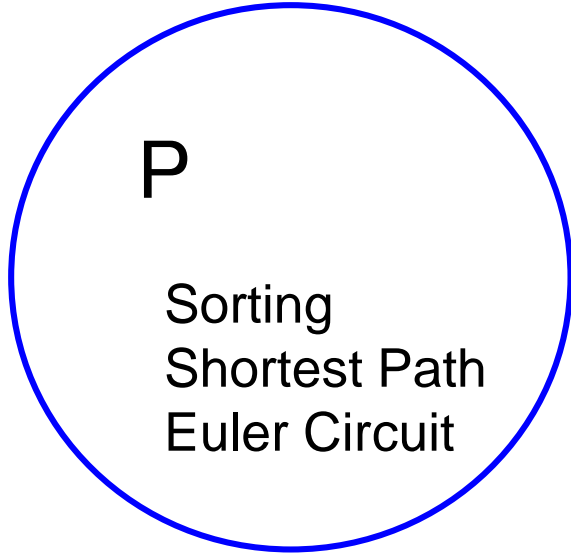


P

Sorting  
Shortest Path  
Euler Circuit



Hamiltonian Circuit



Hamiltonian Circuit  
Satisfiability (SAT)  
Vertex Cover  
Travelling Salesman

# Satisfiability

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

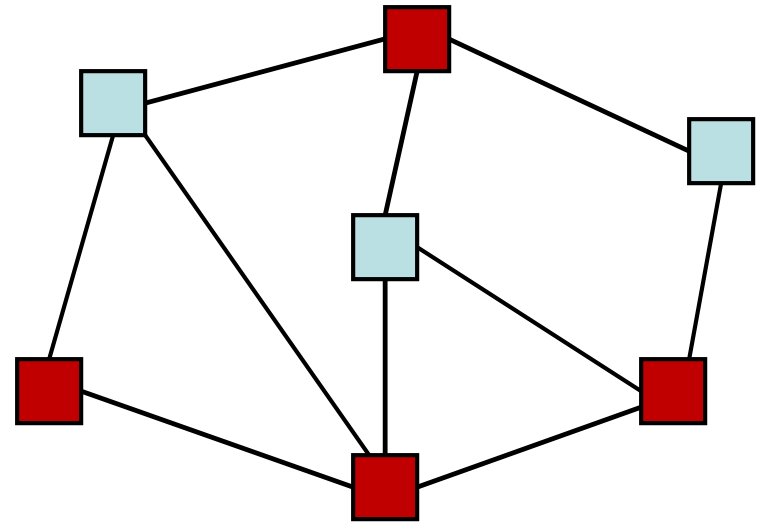
**Input:** a logic formula of size **m** containing **n** variables

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

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)

**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

Algorithm: Try every path, stop if find cheap enough one