DTS203TC
Design and Analysis of Algorithms

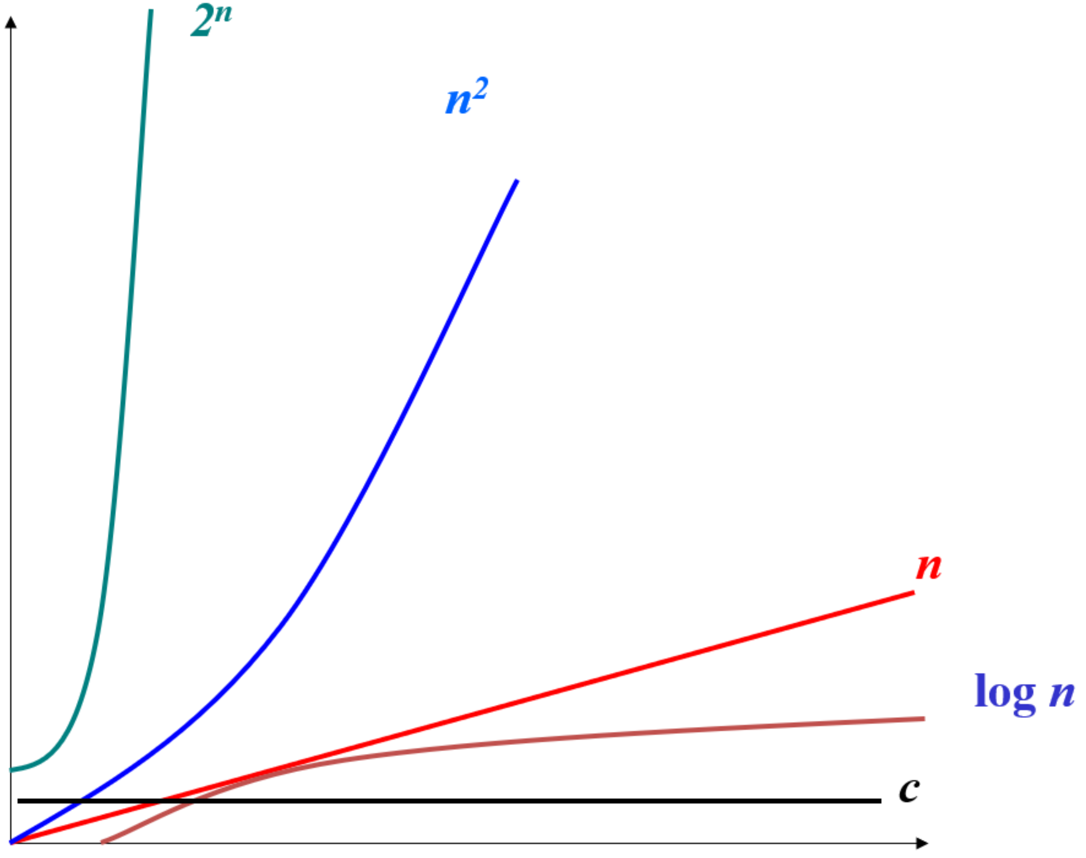
Lecture 20: NP-Completeness

Dr. Qi Chen
School of AI and Advanced Computing

Learning Outcome

- Computational Complexity Theory
- The classes P and NP
- Polynomial-time reduction
- NP Hard and NP Completeness
- NP completeness problems
  - Hamiltonian circuit
  - SAT (satisfiability)
  - 0/1 knapsack
  - 3-Coloring
  - K-Clique
  - Vertex cover

Growth rate



1 log n n n^2 n^3 ... n^k ... 2^n

constant logarithmic polynomial exponential

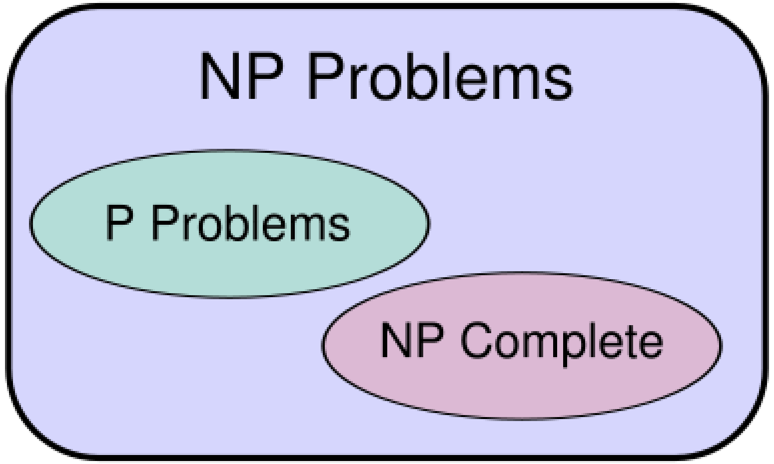
Computational Complexity

- Polynomial Time
  - Linear Search - $O(n)$
  - Binary Search - $O(\log n)$
  - Insertion Sort - $O(n^2)$
  - MergeSort - $O(n\log n)$
  - …
- Exponential Time
  - 0/1 knapsack - $O(2^n)$
  - Hamiltonian circuit - $O(2^n)$
  - Traveling Salesman Problem - $O(2^n)$
  - Circuit-SAT - $O(2^n)$
  - …

Hard Computational Problems

- An algorithm is efficient if its running time is bounded by a ***polynomial*** of its input size.
- Some computational problems seems ***hard*** to solve.
  - Despite numerous attempts we ***do not know any efficient*** algorithms for these problems
- We are also far away from ***proving*** these problems are indeed hard to solve
- In more formal language, we don't know whether ***NP = P or NP ≠ P***. This is an important and fundamental question in theoretical computer science!

Hard Computational Problems



Circuit-SAT
Hamiltonian circuit problem
0/1 Knapsack problem

P = NP?

https://www.claymath.org/millennium/p-vs-np/

- Named as one of the seven "**Millennium Problems**" by the Clay Institute
- can earn you $1 million for its solution (and a place in mathematical and computer science history).
- Check https://www.claymath.org/millennium-problems/ to read more about this (select the "P vs NP" link).

Examples

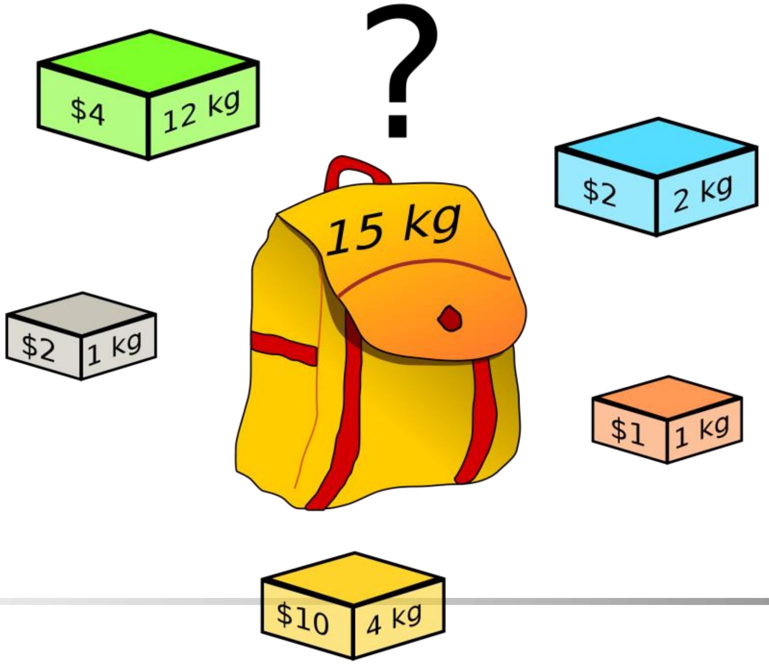
- We have seen two examples of these difficult problems where no efficient (polynomial) algorithm is known
  - 0/1 Knapsack problem
  - Hamiltonian circuit problem
  - One more: Circuit-SAT

- All we know is to exhaust all possible solutions to find the best one

0-1 Knapsack Problem

Input: Given n items with integer weights $w_1, w_2, \ldots, w_n$ and integer values $v_1, v_2, \ldots, v_n$, and a knapsack with capacity W.

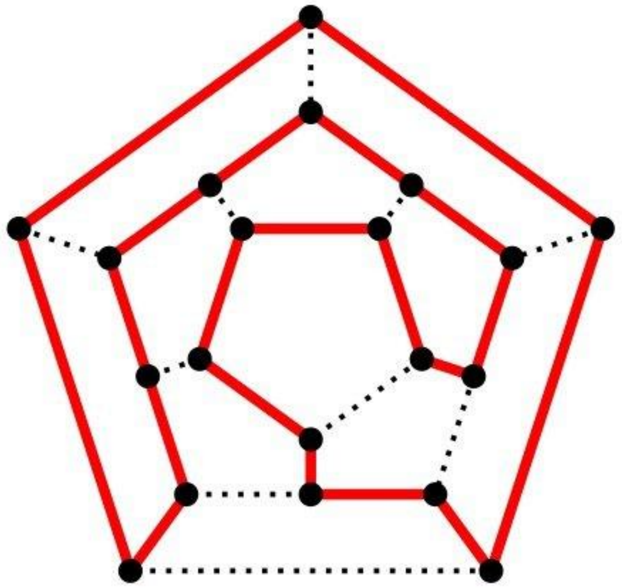
Problem (optimization version): Find a subset of items whose total weight does not exceed W and that maximizes the total value.

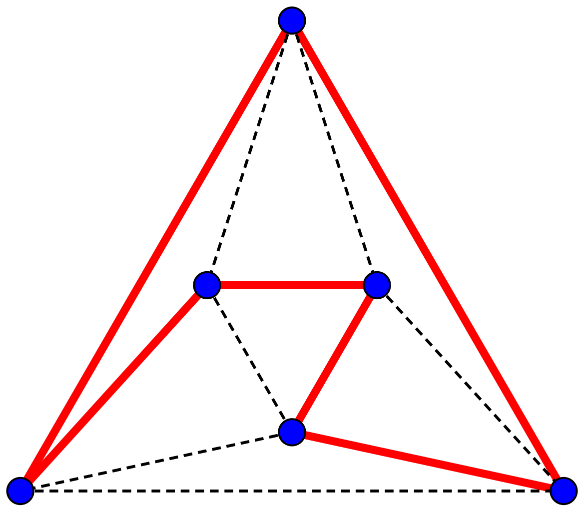


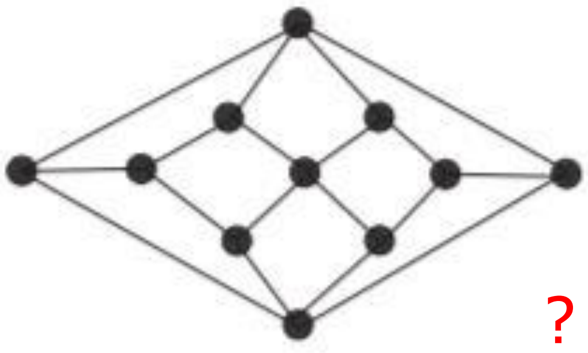
Hamiltonian Circuit

Input: A connected graph G

Question: Does G have a Hamiltonian circuit?
i.e., does G have a circuit that passes through every vertex exactly once, except for the starting and ending vertex?







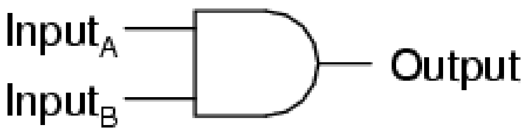
Boolean Circuit

A **Boolean Circuit** is a directed graph where each vertex, called a *logic gate* corresponds to a simple Boolean function, one of **AND, OR, or NOT**.

Incoming edges: inputs for its Boolean function

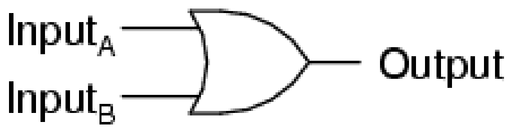
Outgoing edges: outputs

2-input AND gate



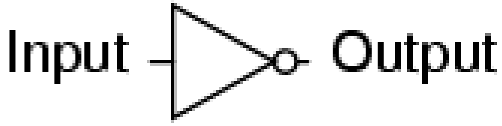
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

2-input OR gate



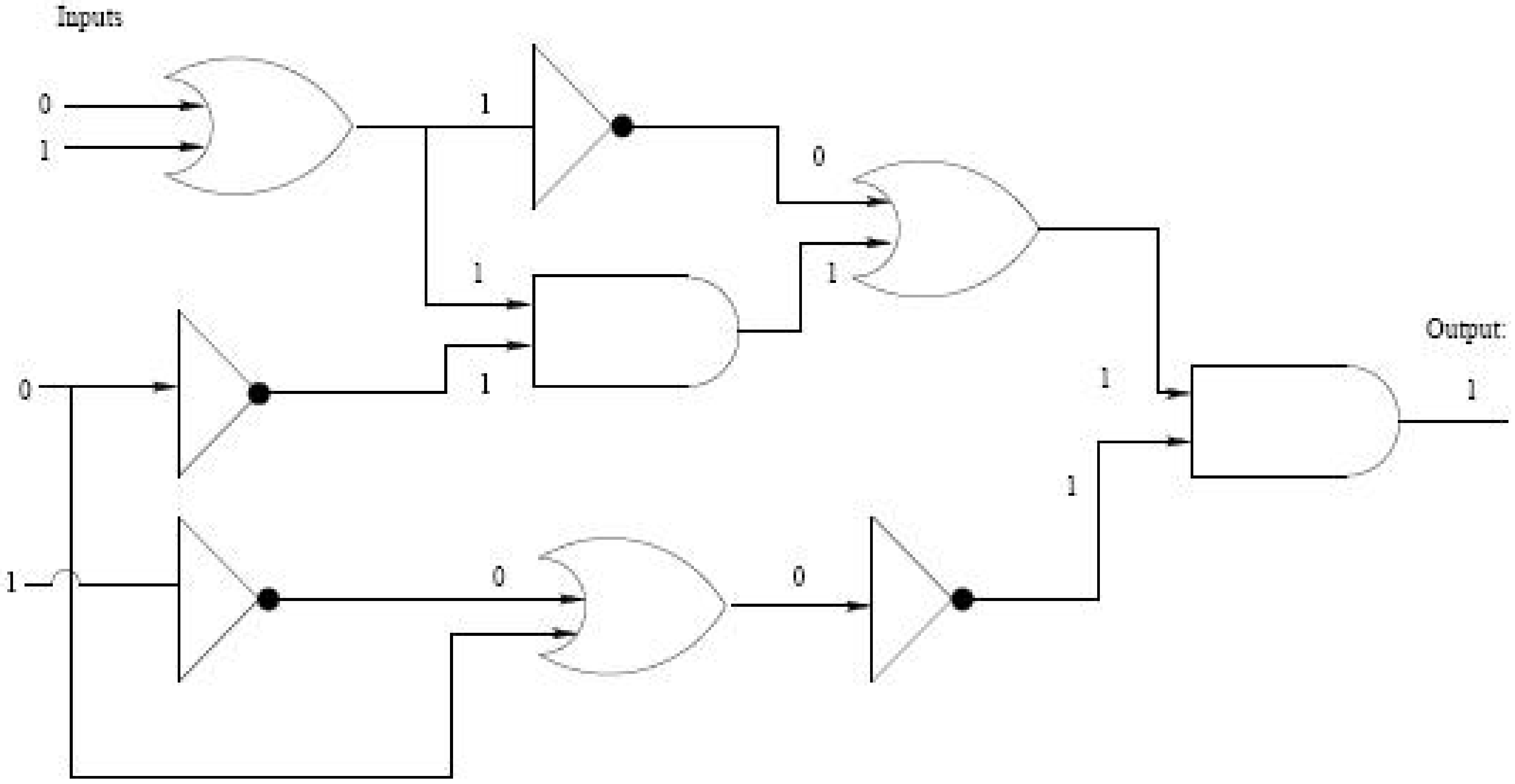
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate truth table



Input	Output
0	1
1	0

Boolean Circuit - Example



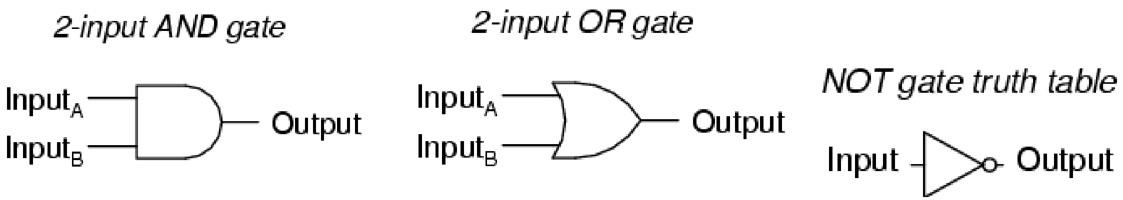
Circuit-SAT

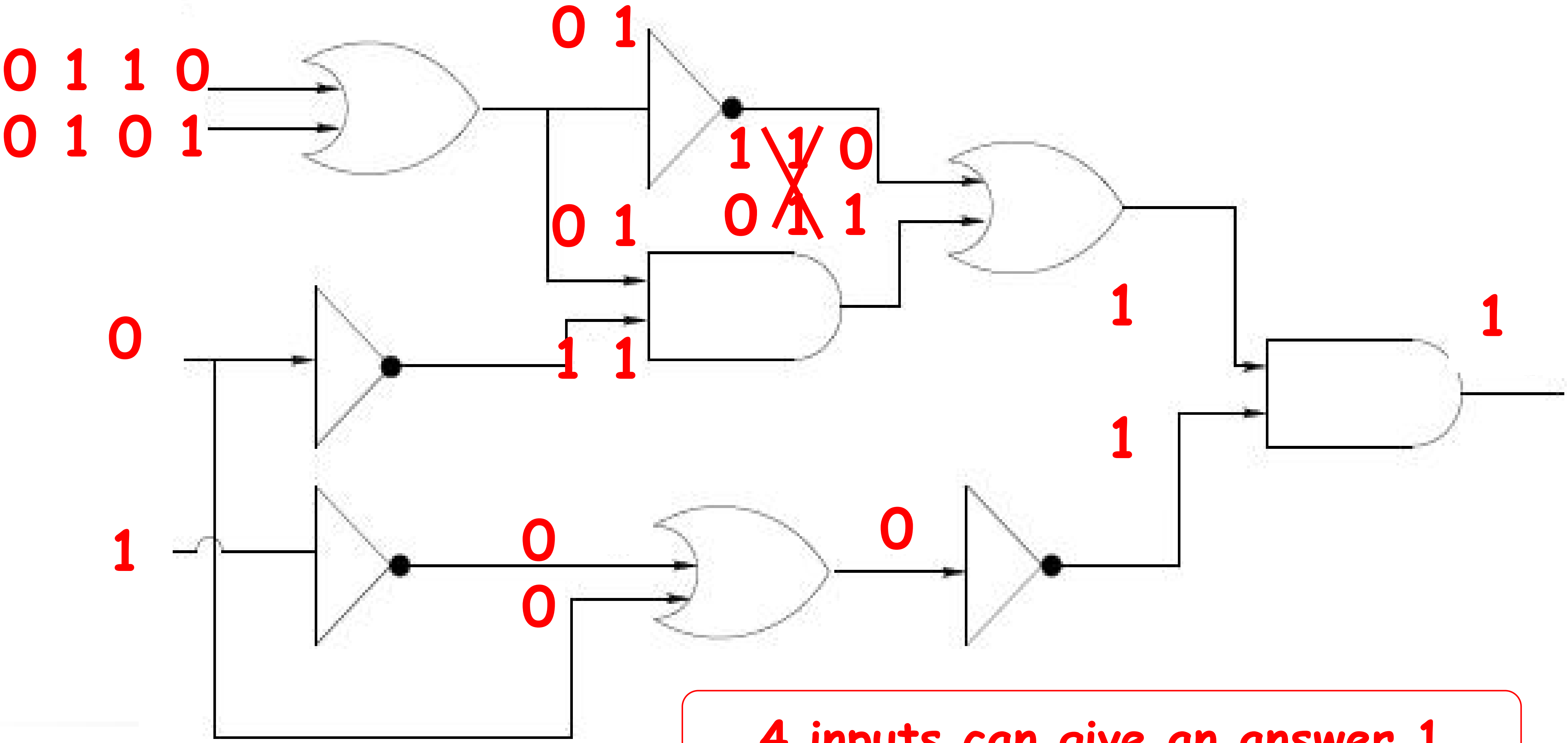
Input: a Boolean Circuit with a single output vertex

Question: is there an assignment of values to the inputs so that the output value is 1?

SAT means satisfiability

Circuit-SAT





4 inputs can give an answer 1

Decision/Optimisation problems

- A ***decision*** problem is a computational problem for which the output is either ***yes*** or ***no***.
- In an ***optimisation*** problem, we try to ***maximise*** or ***minimise*** some value.
- An optimisation problem can be ***turned into*** a decision problem if we add a ***parameter k***; and then ask whether the optimal value in the optimisation problem is ***at most*** or ***at least*** k.
- Note that if a decision problem is ***hard***, then its related optimisation version must also be ***hard***.

Example - MST

Optimisation problem: Given a graph G with integer weights on its edges. What is the weight of a ***minimum*** spanning tree (MST) in G?

Decision problem: Given a graph G with integer weights on its edges, and ***an integer k***. Does G have a MST of weight ***at most k*** ?

Example – Knapsack problem

- **Input:** Given n items with integer weights $w_1, w_2, \ldots, w_n$ and integer values $v_1, v_2, \ldots, v_n$, a knapsack with capacity W

- **Optimisation problem:** Find a subset of items whose total weight does not exceed W and that ***maximises*** the total value.

- **Decision problem:** Is there a subset of items whose total weight does not exceed W and whose total value is ***at least k*** ?

Exercise

State the decision version of the following problems

- Given a **weighted** graph G and a source vertex a, find the ***shortest*** paths from a to every other vertex

- Given a weighted graph G, a source vertex a and ***a value k***, is there shortest path from a to a vertex v such that each path is of weight ***at most k*** ?

Can All Decision Problems Be Solved By Algorithms?

- The Answer is No.
- The problems can not be solved by algorithms is called undecidable problems.
- One such a problem is Halting Problem (AlanTuring 1936)
  - "Given a computer program and an input to it, determine whether the program will halt on that input or continue working indefinitely on it."

Solving/Verifying a problem

- Solving a problem is different from verifying a problem
  - solving: we are given an input, and then we have to FIND the solution
  - verifying: in addition to the input, we are given a **"certificate"** and we verify whether the certificate is indeed a solution

- We may not know how to solve a problem efficiently, but we may know how to verify whether a candidate is actually a solution

Example – Hamiltonian circuit problem

- Suppose through some (unspecified) means (like good guessing), we find a ***candidate*** for a Hamiltonian circuit, i.e. a list of vertices and edges that ***might be a Hamiltonian circuit*** in the input graph G.
- It is easy to check if this is indeed a Hamiltonian circuit. Check
  - that all the proposed edges exist in G,
  - that we indeed have a cycle, and
  - that we hit every vertex in G once.
- If the candidate solution is indeed a Hamiltonian circuit, then it is a ***certificate*** verifying that the answer to the decision problem is "Yes"

Example – 0/1 Knapsack Problem

- Consider an instance of the 0/1 Knapsack problem (decision version)
- Suppose someone proposes a subset of items, it is easy to check
  - if those items have **total weight at most W** and
  - if the **total value is at least k**
- • If both conditions are true, then the subset of items is a ***certificate*** for the decision problem
  - i.e., it verifies that the answer to the 0/1 knapsack decision problem is "Yes"

Example – Circuit-SAT

- Consider a Boolean Circuit
- Suppose someone proposes an assignment of truth values to the input, it is easy to check
  - if the input values lead to a final value of 1 in the output
  - this is done by checking every logic gate
- If the input truth values give a final value of 1, these values form a ***certificate*** for the decision problem

Complexity Classes P and NP

The complexity class P is the set of all decision problems that can be ***solved*** in worst-case ***polynomial time***.

The complexity class NP is the set of all decision problems that can be ***verified*** in ***polynomial time***.

P stands for polynomial, and
NP stands for non-deterministic polynomial.

The Class P

MST problem is in P

- Given a weighted graph G and a value k, does there exists a MST with weight at most k?
- run Kruskal's algorithm (polynomial time) and if the MST found has weight at most k, then the answer is "Yes"

Single-source-shortest-paths problem is in P

- Given a weighted graph G, a source vertex s, and a value k, does there exist shortest paths from s to every other vertex whose path length is at most k?
- run Dijkstra's algorithm (polynomial time) and if the paths found have lengths at most k, then answer is "Yes"

The Class NP

Hamiltonian circuit problem is in NP

- we can check in polynomial time if a proposed circuit is a Hamiltonian circuit

0/1 Knapsack problem is in NP

- we can check in polynomial time if a proposed subset of items whose weight is at most W and whose value is at least k

Circuit-SAT is in NP

- we can check in polynomial time if proposed values lead to a final output value of 1

P = NP ?

- Note that $P \subseteq NP$

- The (million dollar) question is that mathematicians and computer scientists do not know whether $P = NP$ or $P \neq NP$

- However, there is a common belief that P is different from NP
  - i.e., there is some problem in NP that is not in P

Polynomial-time reduction

Given any two decision problems A and B, we say that

1) A is ***polynomial time reducible*** to B, or
2) there is a polynomial time reduction from A to B

if given any input α of A, we can **construct** in ***polynomial time*** an input β of B such that
α is yes **if and only if** β is yes.

We use the notation $A \leq_P B$

Intuitively, this means that problem **B** is at least as difficult as problem ***A***

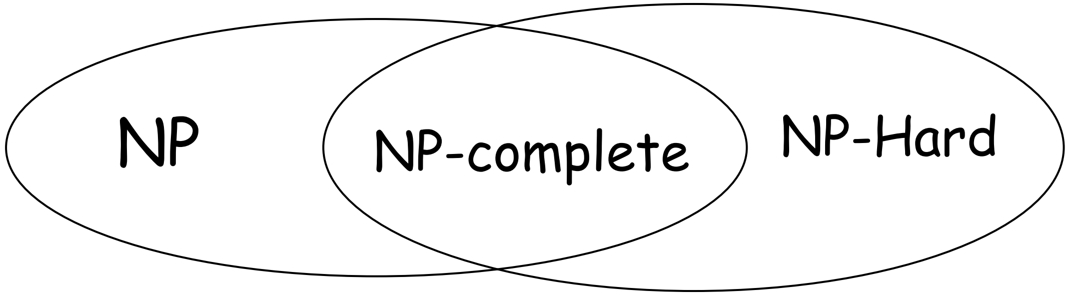
NP-hardness / NP-completeness

A problem M is said to be **NP-hard** if every other problem in NP is polynomial time reducible to M

- intuitively, this means that M is at least as difficult as all problems in NP

M is further said to be NP-complete if

1. M is in NP, and
2. M is NP-hard



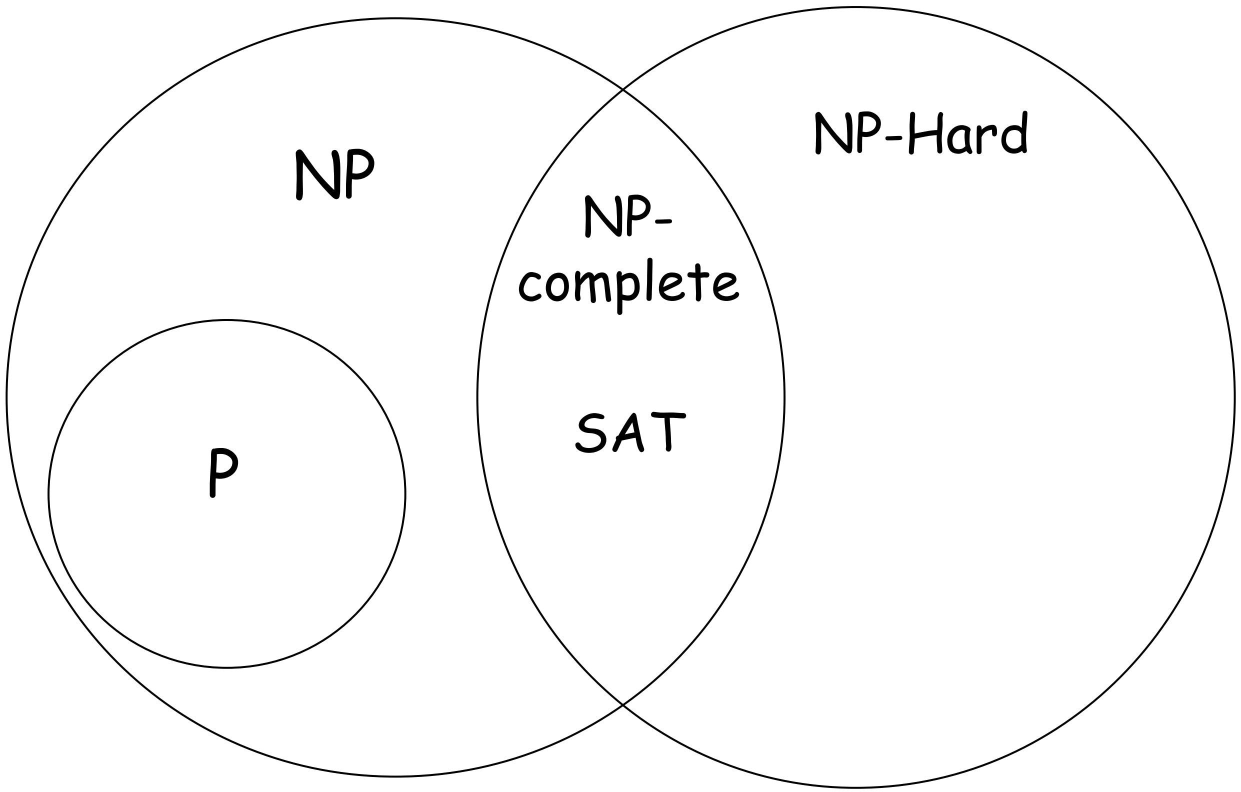
NP-complete problems are some of the hardest problems in NP

NP-Complete Problem

- The ***Cook-Levin Theorem*** states that Circuit-SAT is NP-complete (a "first" NP-complete problem)
  - if there exists a deterministic polynomial-time algorithm for solving Circuit-SAT, then P = NP.
- Using polynomial time reducibility we can show existence of other NP-complete problems

- A useful result to prove NP-completeness:
- **Lemma**
- **If L1 $\le_P$ L2 and L2 $\le_P$ L3, then L1 $\le_P$ L3**

Any thing in NP $\leq_P$ SAT



Other NP-Complete Problems

We have seen these NP-Complete Problems

- Hamiltonian Circuit Problem
- 0/1 Knapsack Problem
- Circuit-SAT

Others

- CNF-SAT and 3-SAT (conjunctive normal form satisfiability problem)
- 3-Coloring
- K-Clique
- Vertex Cover

Conjunctive normal form (CNF)

- a Boolean formula is in CNF if it is formed as a collection of ***clauses*** combined using the operator AND (∧) and each clause is formed by ***literals*** (variables or their negations) combined using the operator OR (∨)

- example: $(\overline{x1} \vee x2 \vee \overline{x4} \vee x5) \wedge (\overline{x2} \vee x1 \vee x4)$

CNF-SAT and 3-SAT

CNF-SAT

- **Input:** a Boolean formula in CNF
- **Question:** Is there an assignment of Boolean values to its variables so that the formula evaluates to ***true***? (i.e., the formula is satisfiable)

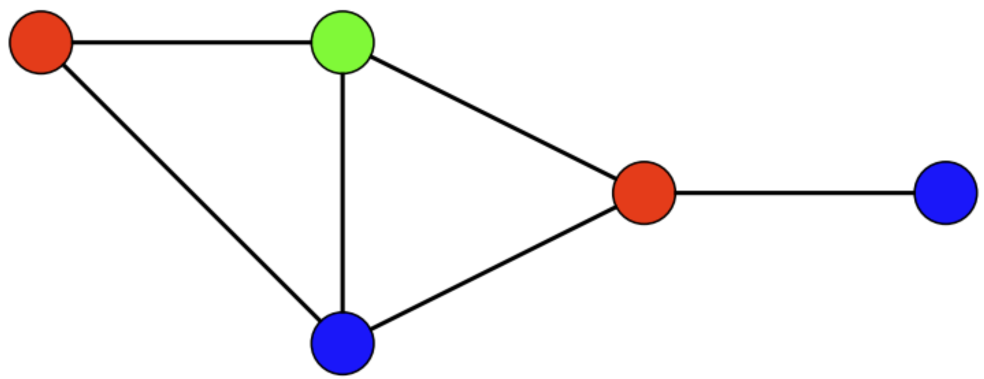
3-SAT

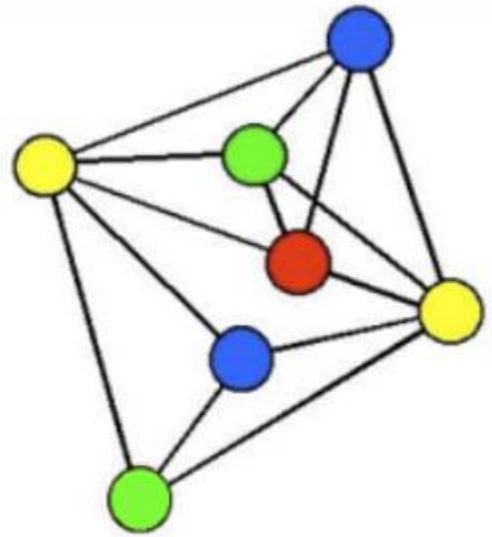
- **Input:** a Boolean formula in CNF in which each clause has exactly 3 literals

CNF-SAT and 3-SAT are NP-complete

3 Coloring

- Given an undirected graph. Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?

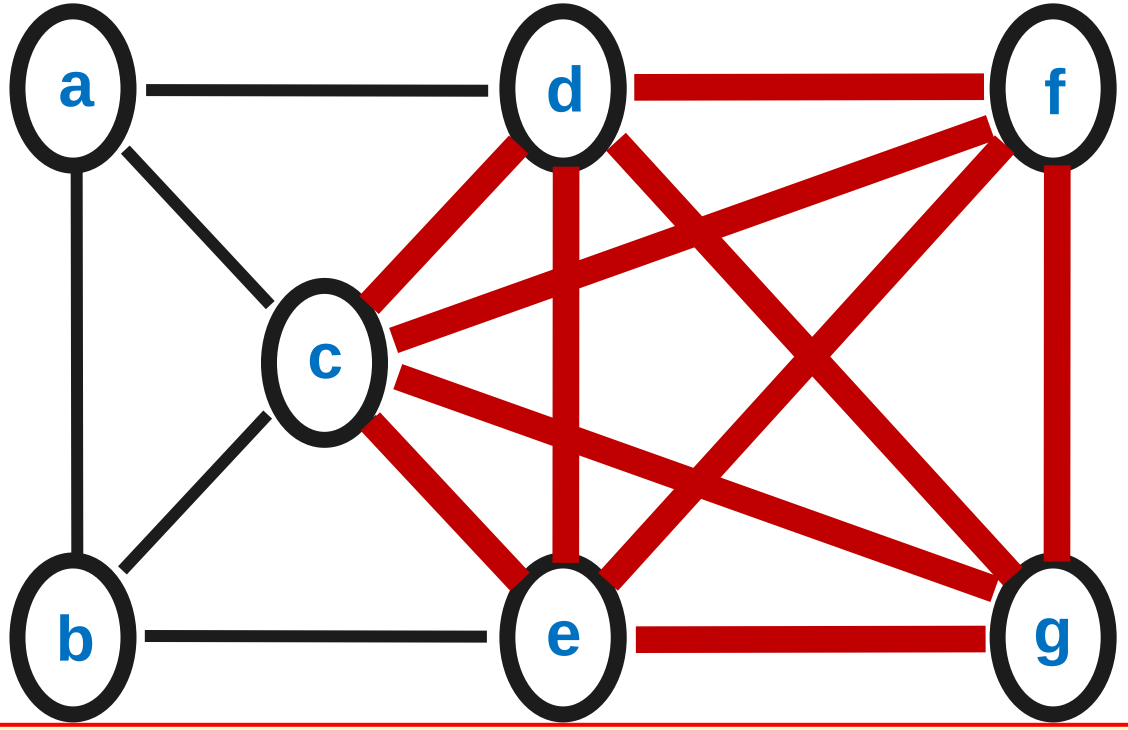




3 Coloring is NP-complete

K-Clique

- A clique is a subgraph of a graph such that all the vertices in this subgraph are connected with each.
- k-Clique: a clique of size k exists in the given graph?

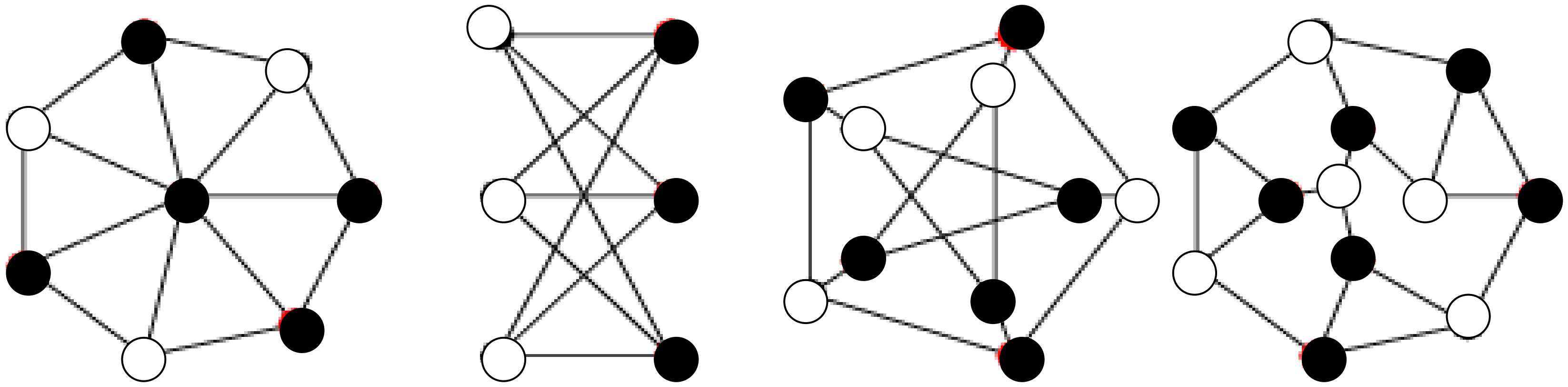


K-Clique is NP-complete

Vertex Cover

Given a graph $G = (V,E)$

A **vertex cover** is a subset $C \subseteq V$ such that for every edge (v,w) in E, **$v \in C$ or $w \in C$**



some graphs and their vertex cover (shaded vertices)

Vertex Cover

- The optimisation problem is to find as **small** a vertex cover as possible

- **Vertex Cover** is the **decision** problem that takes a graph **G** and an integer **k** and asks whether there is a vertex cover for G containing ***at most*** k vertices

Vertex Cover is NP-complete

How to prove Vertex Cover is NP-Complete?

- 1. Proof that vertex cover is in NP

- 2. Proof that vertex cover is NP Hard

Vertex Cover is in NP

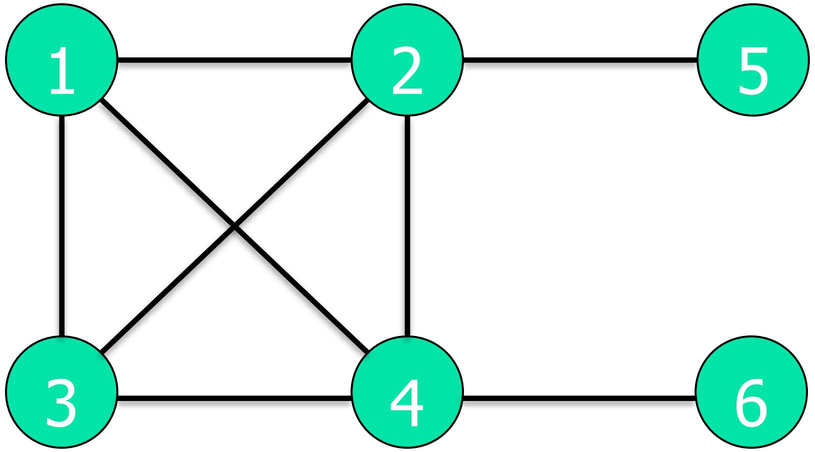
- If any problem is in NP, then given a 'certificate' (a solution) to the problem and an instance of the problem (a graph G=(V,E) and a positive integer k), we should be able to verify the certificate in polynomial time.
- The certificate for the vertex cover problem is a subset B of V.

```
Verify(G,k,B)
count = 0
for each vertex v in B
    remove all edges adjacent to v from set E
    count = count + 1
    if count <= k and E is empty
        return True
return False
```

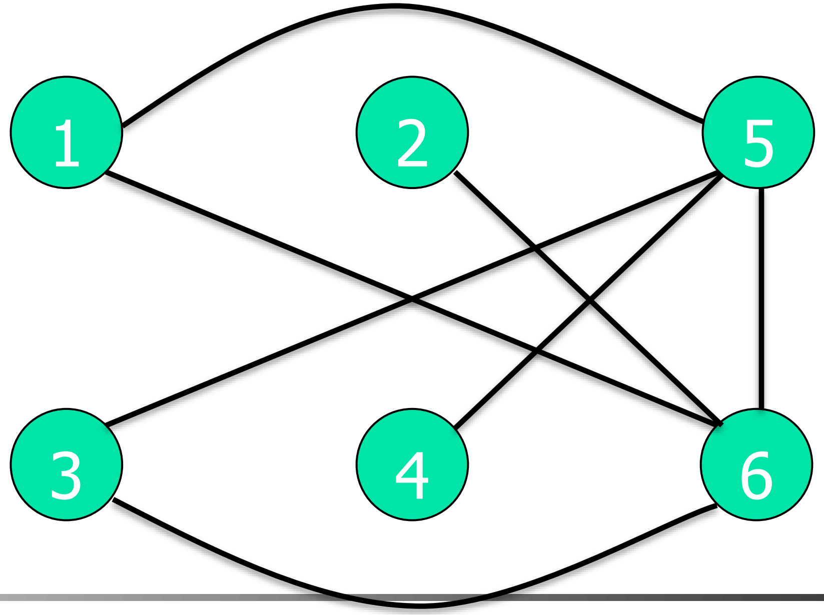
Runs in polynomial time

Vertex Cover is NP-Hard

- K-Clique is NP Complete.
- To Prove Vertex Cover is NP Hard, we use a reduction from k-Clique.
- Consider the graph G' which consists of all edges not in G, but in the complete graph using all vertices in G.



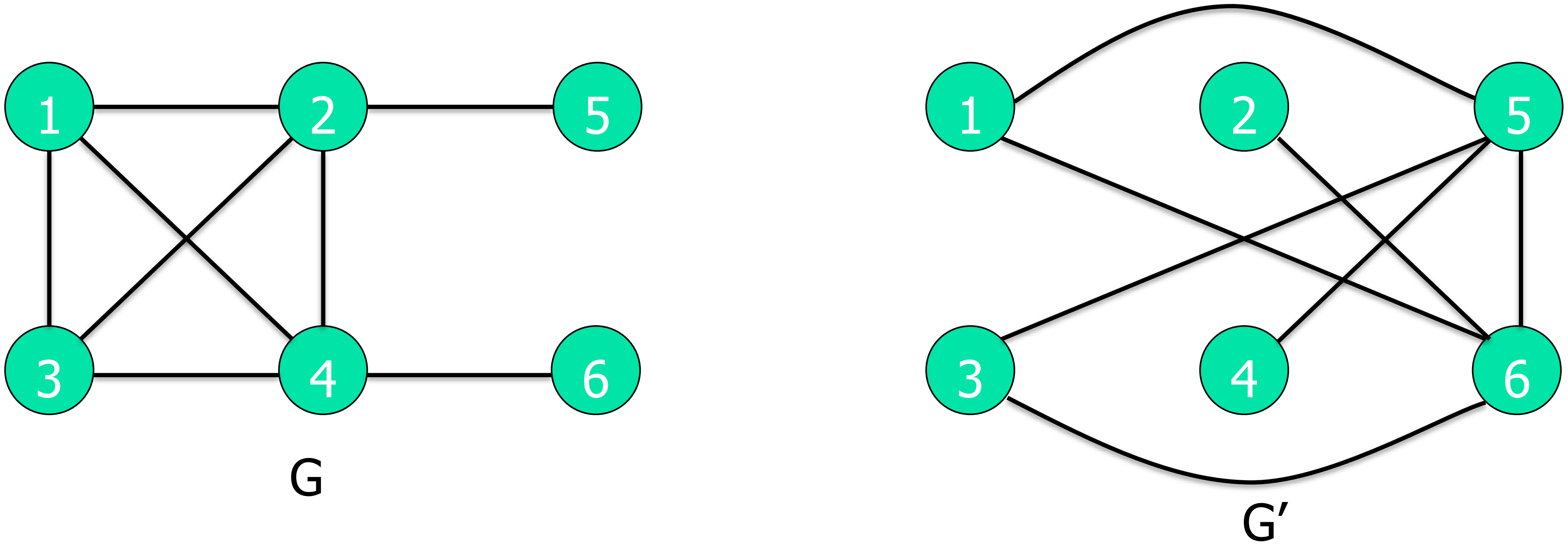
G



G′

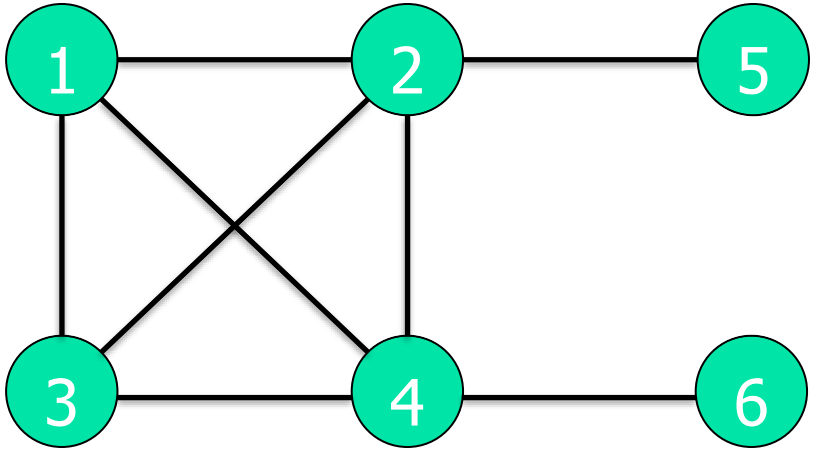
Vertex Cover is NP-Hard

- The problem of finding whether a clique of size k exists in the graph G is the same as the problem of find whether a vertex cover of size |V|-k in G'.

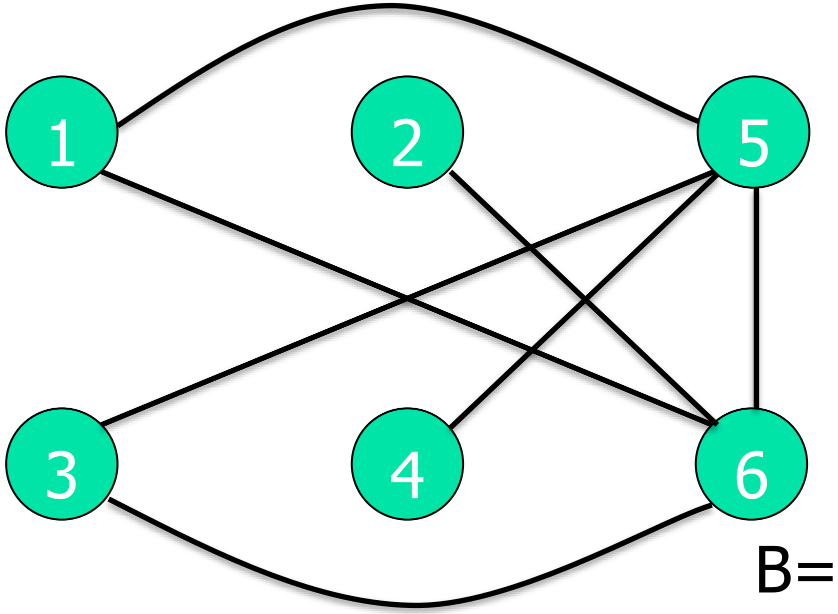


Vertex Cover is NP-Hard

- Assume that there is a clique of size k in G.
- For any edge (u, v) in G', at least one of u or v must be in the set B (which is V-A). |B| = |V|-k
- Thus, all edges in G' are covered by vertices in the set B.



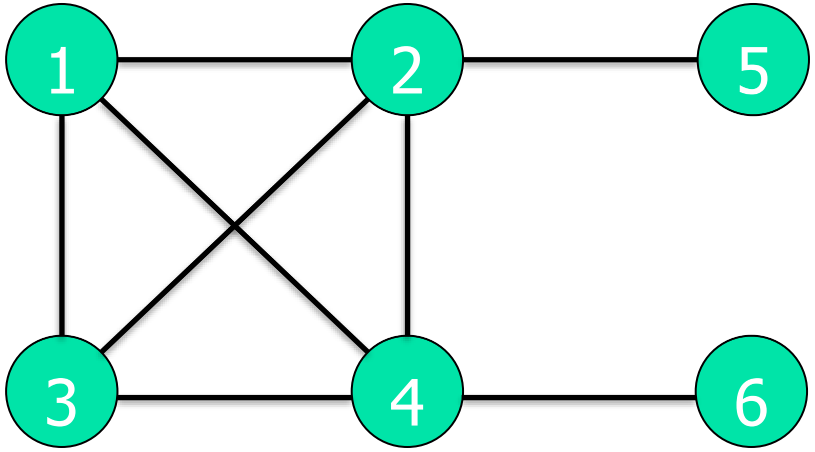
G
A={1, 2, 3, 4}



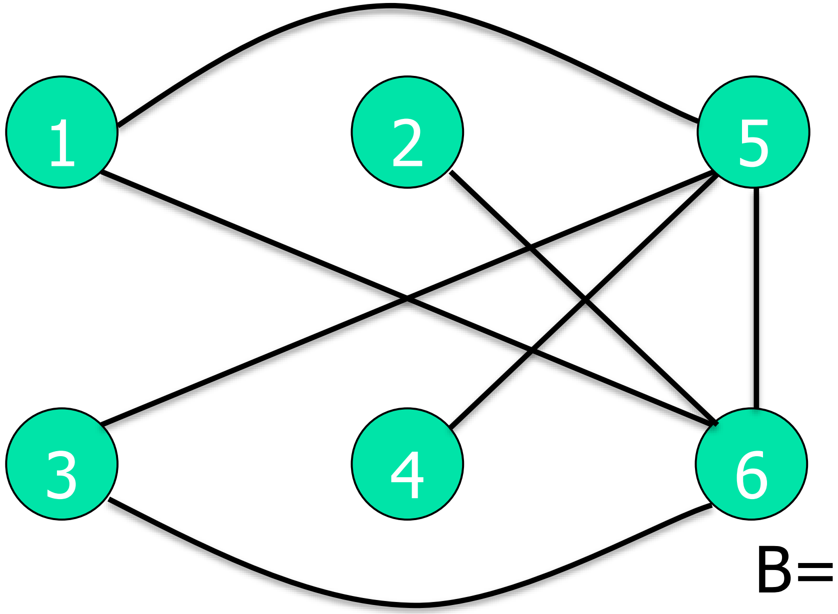
G'
B={5,6}

Vertex Cover is NP-Hard

- Assume that there is a vertex cover B of size |V|-k in G'.
- For all edge (u, v) that both u and v are not in set B are in G.
- Thus, these edges constitute a clique of size k.



G
A={1, 2, 3, 4}

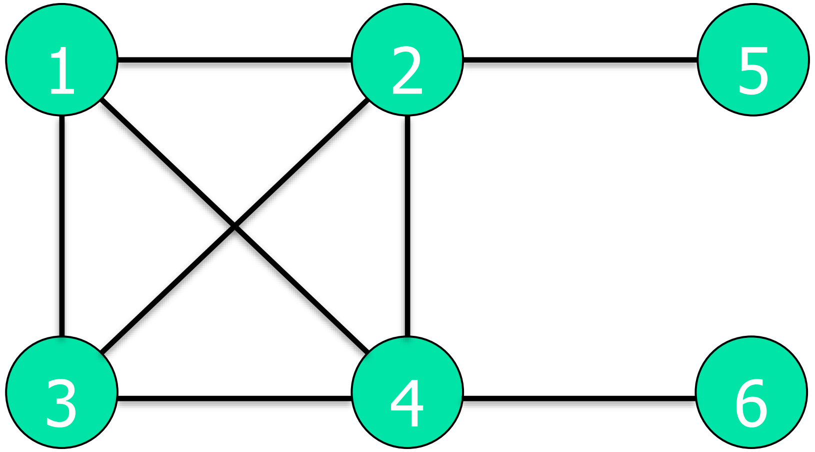


G'
B={5,6}

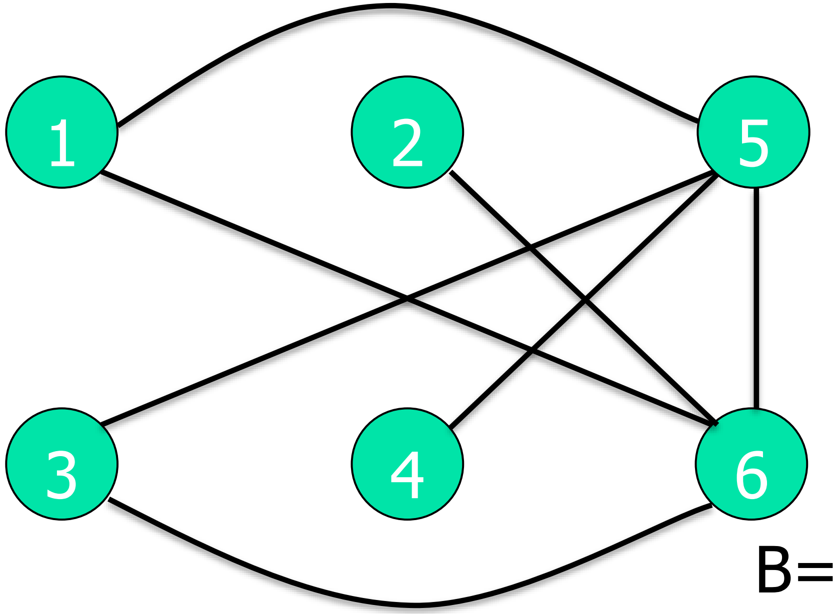
Vertex Cover is NP-Hard

Vertex Cover is in NP ✓
Vertex Cover is NP-Hard ✓
Vertex Cover is NP-complete ✓

- So, We can say that there is a clique of size k in graph G if and only if there is a vertex cover of size |V| - k in G', and any instance of the clique problem can be reduced to an instance of the vertex cover problem.



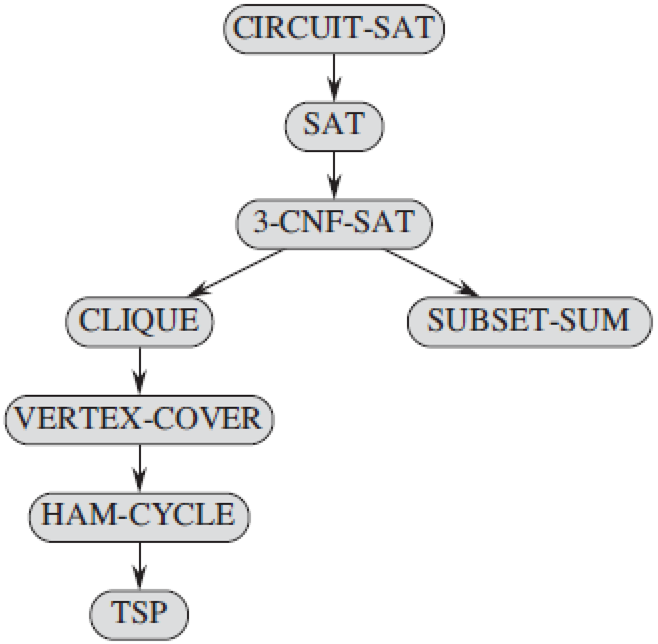
G
A={1, 2, 3, 4}



G'
B={5,6}

Optional Exercises

- Prove Clique is NP Completeness
- Prove HAM-CYCLE is NP Completeness



Learning Outcome

- Computational Complexity Theory
- The classes P and NP
- Polynomial-time reduction
- NP Hard and NP Completeness
- NP completeness problems
  - Hamiltonian cycle
  - SAT
  - 0/1 knapsack
  - 3-Coloring
  - K-Clique
  - Vertex cover