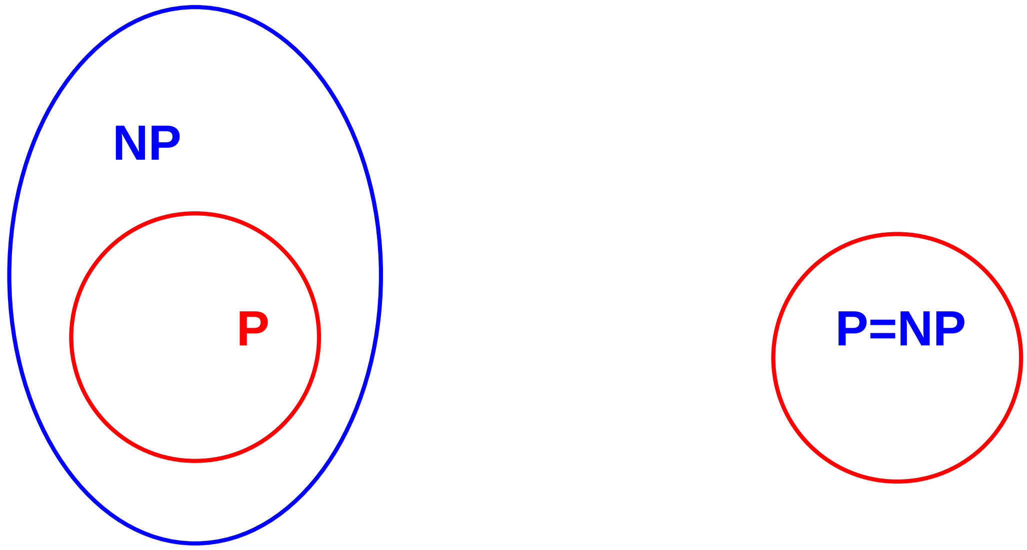


Whether $P = NP$ is one of the great unsolved mysteries in contemporary mathematics.

- most computer scientists believe they are not equal
- most bogus proofs show them equal

Remarks

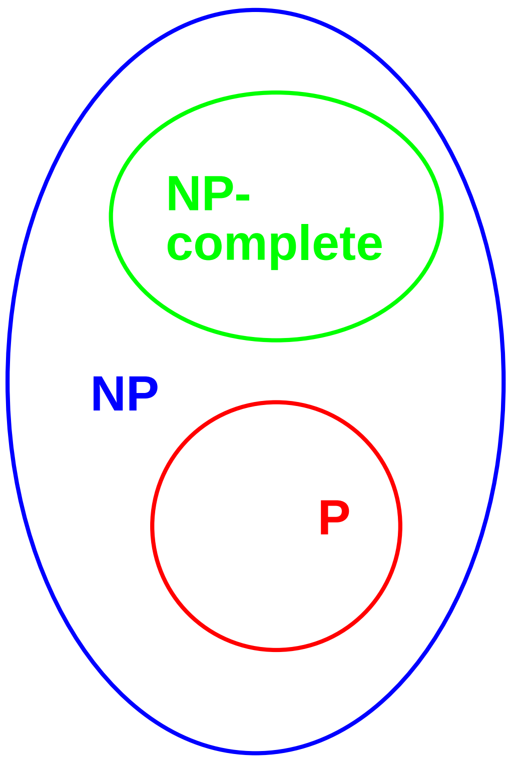
If P differs from NP, then the distinction between P and $NP - P$ meaningful and important.

- languages in P tractable
- languages in $NP - P$ intractable

Until we can prove that $P \neq NP$, there is no hope of proving that a language lies in $NP - P$.

Nevertheless, we *can* prove statements of the form "If $P \neq NP$ then $A \in NP - P$."

Remarks



The class of *NP-complete* languages are

- hardest" languages in NP
- "least likely" to be in P
- If any NP-complete $A \in P$, then $NP = P$.

Cook-Levin

Theorem: A language $S \in NP$ exists where $S \in P$ if and only if $P = NP$.

This theorem establishes the class of NP-complete languages

Satisfiability

A **boolean variable** assumes values

- true (written 1), and
- false (written 0).

Boolean operations:

- and: $\wedge$
- or: $\vee$
- not: $\neg$ or overbar.

Examples:

$$
\begin{aligned}
0 \wedge 1 &= 0 \\
0 \vee 1 &= 1 \\
\overline{0} &= 1
\end{aligned}
$$

Satisfiability

A *boolean formula* is an expression involving boolean variables and operations.

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

Definition: A formula is *satisfiable* if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

Satisfiability

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

is satisfiable by

$$\begin{aligned} x &= 0 \\ y &= 1 \\ z &= 0 \end{aligned}$$

This assignment *satisfies* ϕ.

Satisfiability

Define

$$\text{SAT} = \{\langle\phi\rangle | \phi \text{ is satisfiable Boolean formula}\}$$

Satisfiability

Also useful to consider special version.

- a *literal* is a variable or negated variable: x or $\overline{x}$.
- a *term* is several literals joined by $\vee$s: $(x_1 \vee \overline{x_2} \vee \overline{x_3})$
- a Boolean formula is
- conjunctive normal form (cnf) if it comprises terms connected with $\wedge$s.

$$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$$

Satisfiability

In a *3cnf formula*, all terms have three literals.

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

Define

$$\text{3SAT} = \{\langle\phi\rangle | \phi \text{ is satisfiable 3cnf formula}\}$$

Claim: In a satisfiable 3cnf formula, each clause must contain at least one literal assigned 1.

Poly-Time Computable Functions

A polynomial-time computable function is

$$f : \Sigma^* \longrightarrow \Sigma^*$$

if some poly-time TM

- starts with input w, and
- halts with only $f(w)$ on tape.

Poly-Time Reducibility

Definition: Language A is *polynomial-time mapping reducible* to B, written

$$A \leq_P B,$$

if there is a poly-time computable function

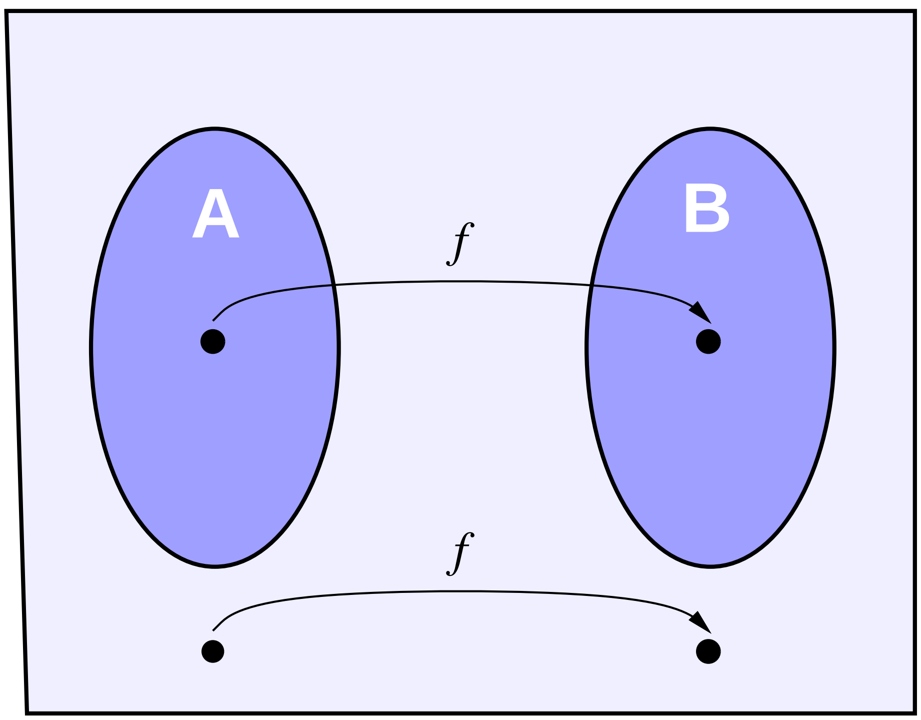
$$f : \Sigma^* \longrightarrow \Sigma^*$$

such that, for every w,

$$w \in A \Longleftrightarrow f(w) \in B.$$

Function f is called the *polynomial-time reduction* of A to B.

Computable Functions



Converts questions about membership in A to membership in B, but does it *efficiently*.

Computable Functions

Theorem: If $A \leq_P B$ and $B \in P$ then $A \in P$.

Proof: Let

- M be the poly-time decider for B, and
- f the reduction from A to B.

Define

N: on input w

1. compute $f(w)$

2. run M on input $f(w)$ and output whatever M outputs.

Analysis:

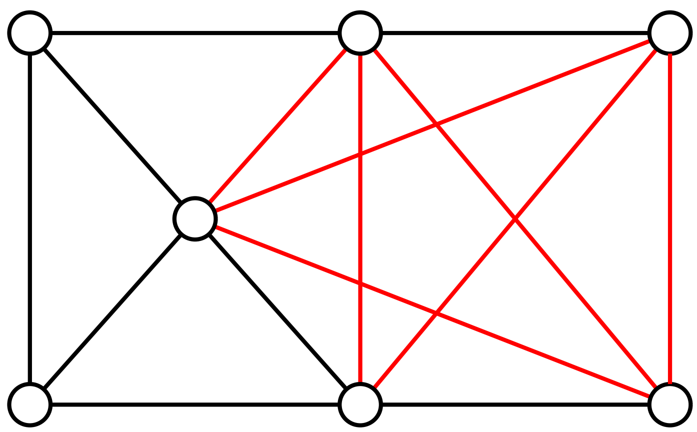
- each stage runs once
- stage 1 is polynomial
- stage 2 is polynomial

Reductions

Claim: 3SAT is polynomial time reducible to CLIQUE.

Construct a graph representation of the formula.

Examples: Clique



A *clique* in a graph is a subgraph where every two nodes are connected by an edge.

A k-*clique* is a clique of size k.

3SAT vs. CLIQUE

Let ϕ be a formula with k clauses.

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

We define a graph G as follows.

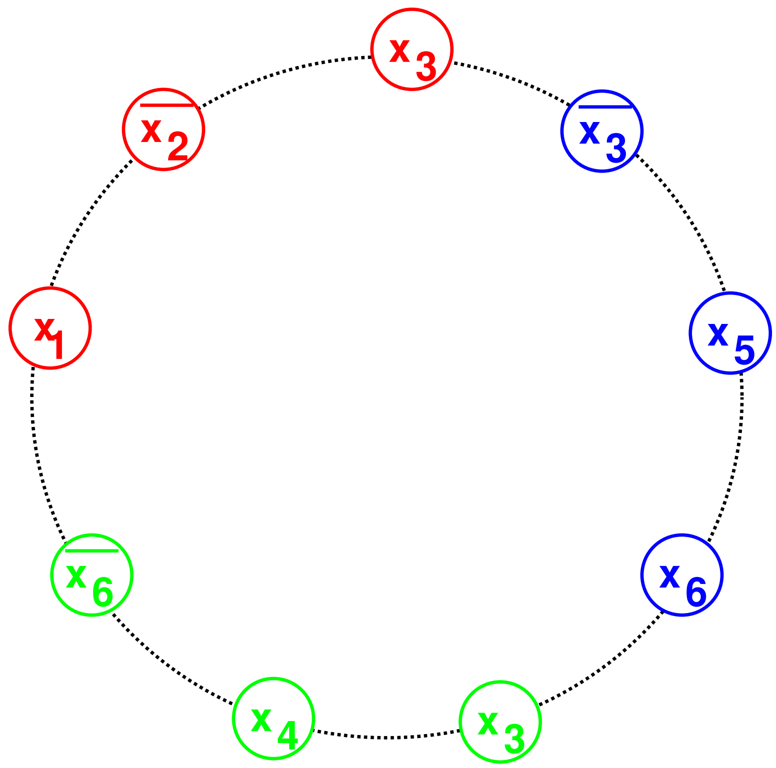
3SAT vs. CLIQUE

We define a graph G as follows.

- nodes in G are organized into *triples* $t_1, ldots, t_k$.
- each triple corresponds to a term of ϕ
- each node in triple corresponds to literal

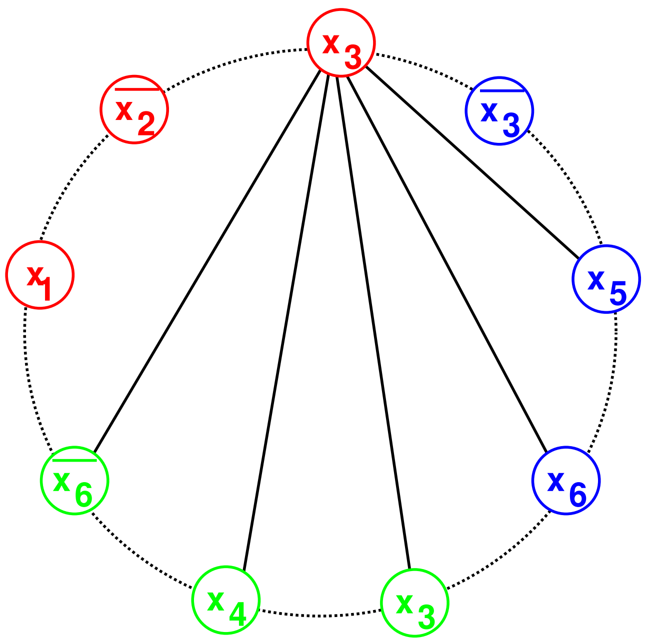
3SAT vs. CLIQUE

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$



3SAT vs. CLIQUE

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$



332

Add edges between all vertex pairs, except

- within same triple
- between contradictory literals

3SAT vs. CLIQUE

Claim: If ϕ is satisfiable, G has a k-clique.

Suppose ϕ is satisfiable.

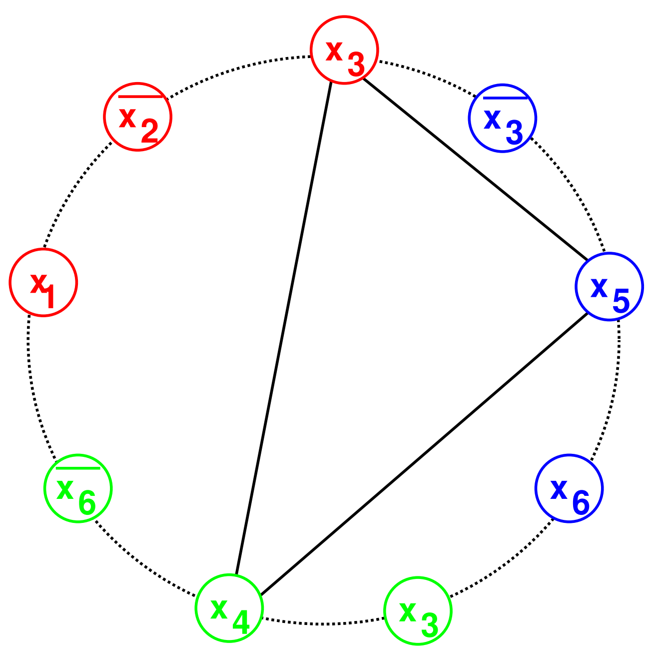
- at least one literal is true in every term
- in every tuple, select one true literal
- they can be joined by edges
- yielding a k-clique

3SAT vs. CLIQUE

Claim: If ϕ is satisfiable, G has a k-clique.

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$

1 1 1

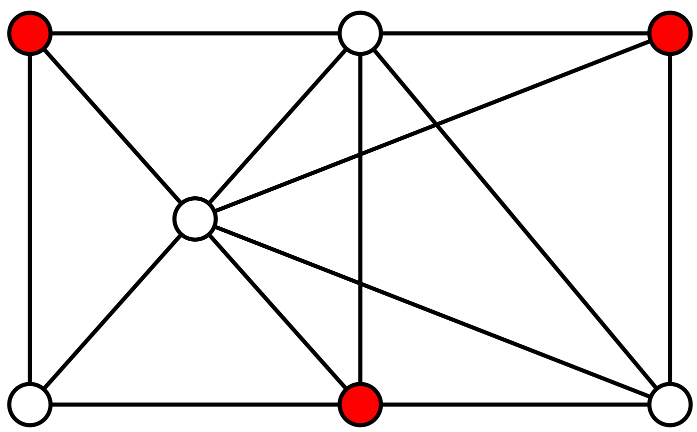


3SAT vs. CLIQUE

Claim: If G has a k-clique, ϕ is satisfiable.

- No two of the cliques nodes are in the same triple.
- k vertexes and k terms, so
- Each triple has one clique node.
- Assign 1 to each node in clique
- no contradictions.

Independent Set



An *independent* in a graph is a set of vertexes, no two of which are linked by an edge.

The *independent set* problem asks whether there exists an independent set of size k.

Independent Set

Define

INDEPENDENT-SET $= \{\langle G, V\rangle | V$ is an independent. set for $G\}$

Claim: INDEPENDENT-SET is polynomial time reducible to CLIQUE, and vice-versa.

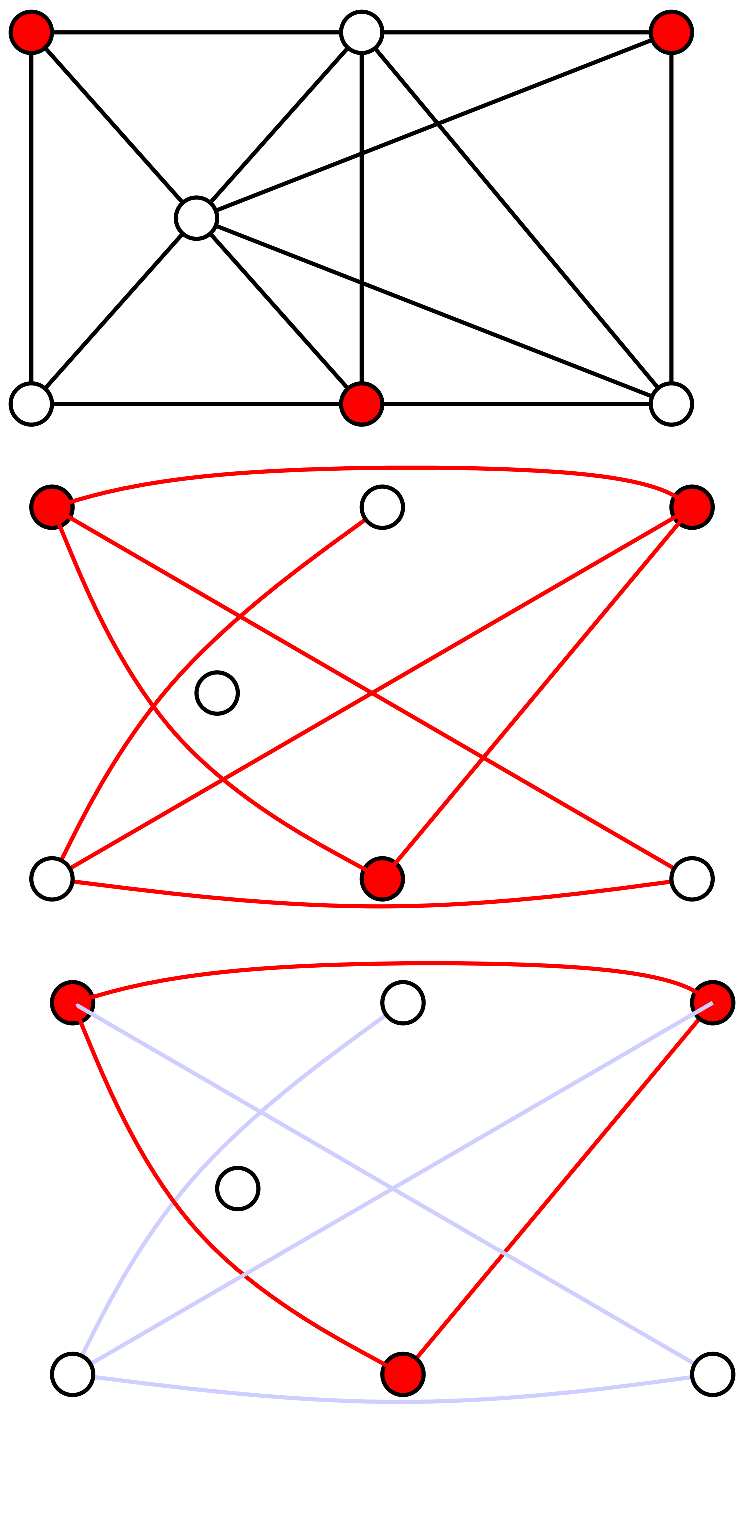
Independent Set

Definition: The *complement* of $G = (V, E)$ is a graph $G^c = (V, E^c)$, where
$E^c = \{v_1, v_2 | v_1, v_2 \in V \text{ and } (v_1, v_2) \notin E\}$.

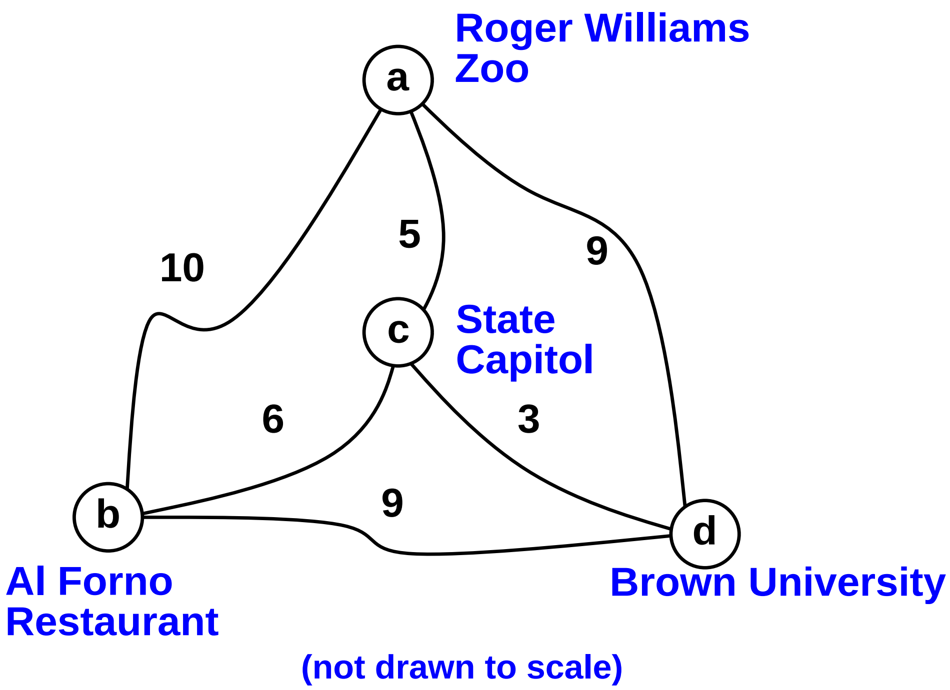
Claim: If V is an independent set for G, then V is a clique in G^c.

'nuff said.

Independent Set



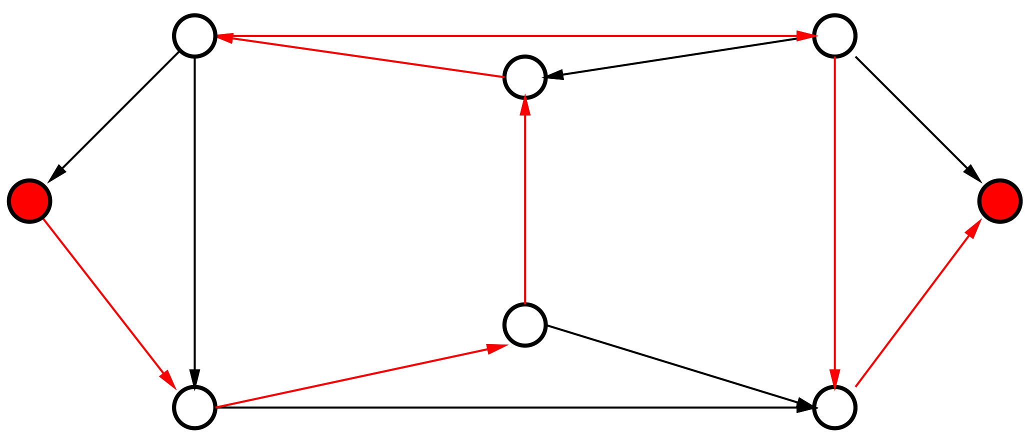
Traveling Salesman



Parameters:

- set of *cities* C
- set of inter-city *distances* D
- goal k

Hamiltonian Path

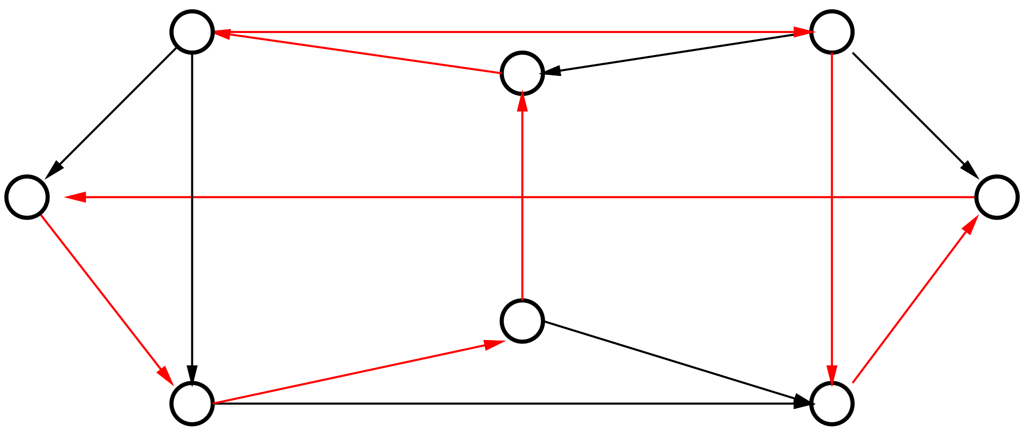


A Hamiltonian path in a directed G visits each note once.

Hamiltonian Path

$$\text{HAMPATH} = \{\langle G, s, t\rangle | G \text{ has Hamiltonian path from } s \text{ to } t\}$$

Hamiltonian Circuit



A Hamiltonian circuit

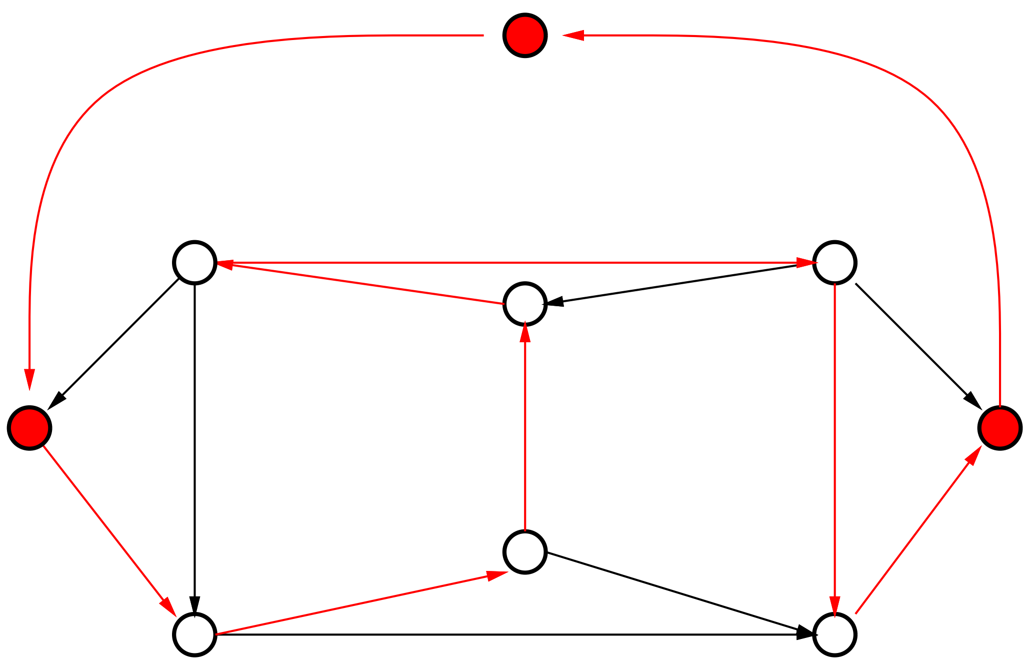
- visits each note once.
- ends up where it started

Hamiltonian Circuit

HAMPATH $= \{\langle G, s\rangle | G$ has Hamiltonian circuit from $s\}$

Reduction

Theorem: HAMPATH is polynomial-time reducible to HAMCIRCUIT.



Reduction

Theorem: HAMCIRCUIT is polynomial-time reducible to HAMPATH.

To decide whether $\langle G, s\rangle \in$ HAMCIRCUIT, call HAMPATH with inpu $\langle G, s, s\rangle$.

Traveling Salesman

Define

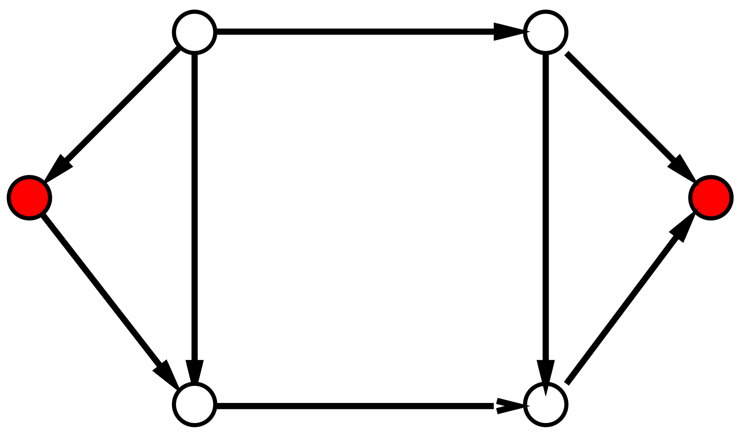
$$\text{TRAVELING-SALESMAN} = \{\langle C, D, k\rangle | (C, D, k) \text{ there is a tour of total distance } \leq k\}$$

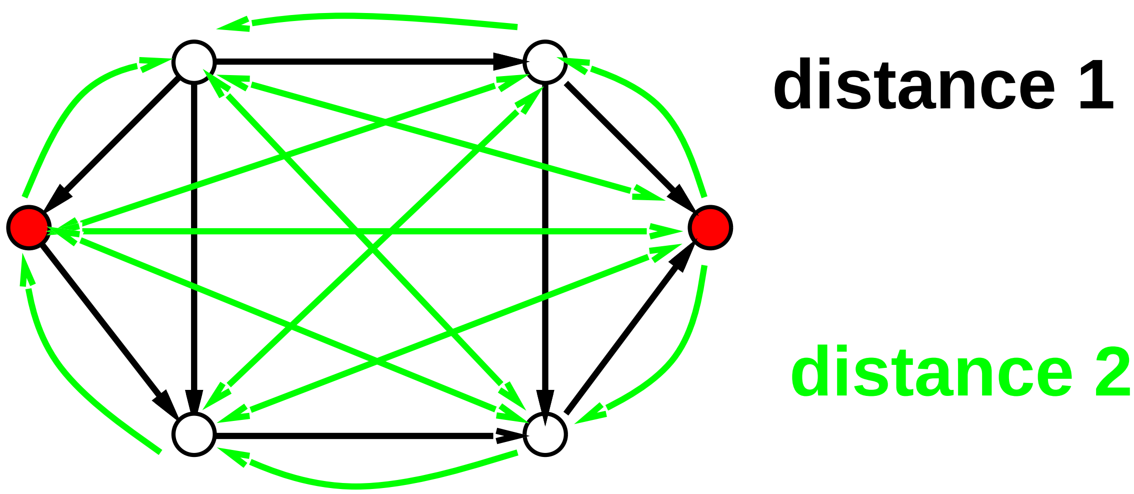
Traveling Salesman vs. Hamiltonian Circuit

Theorem:

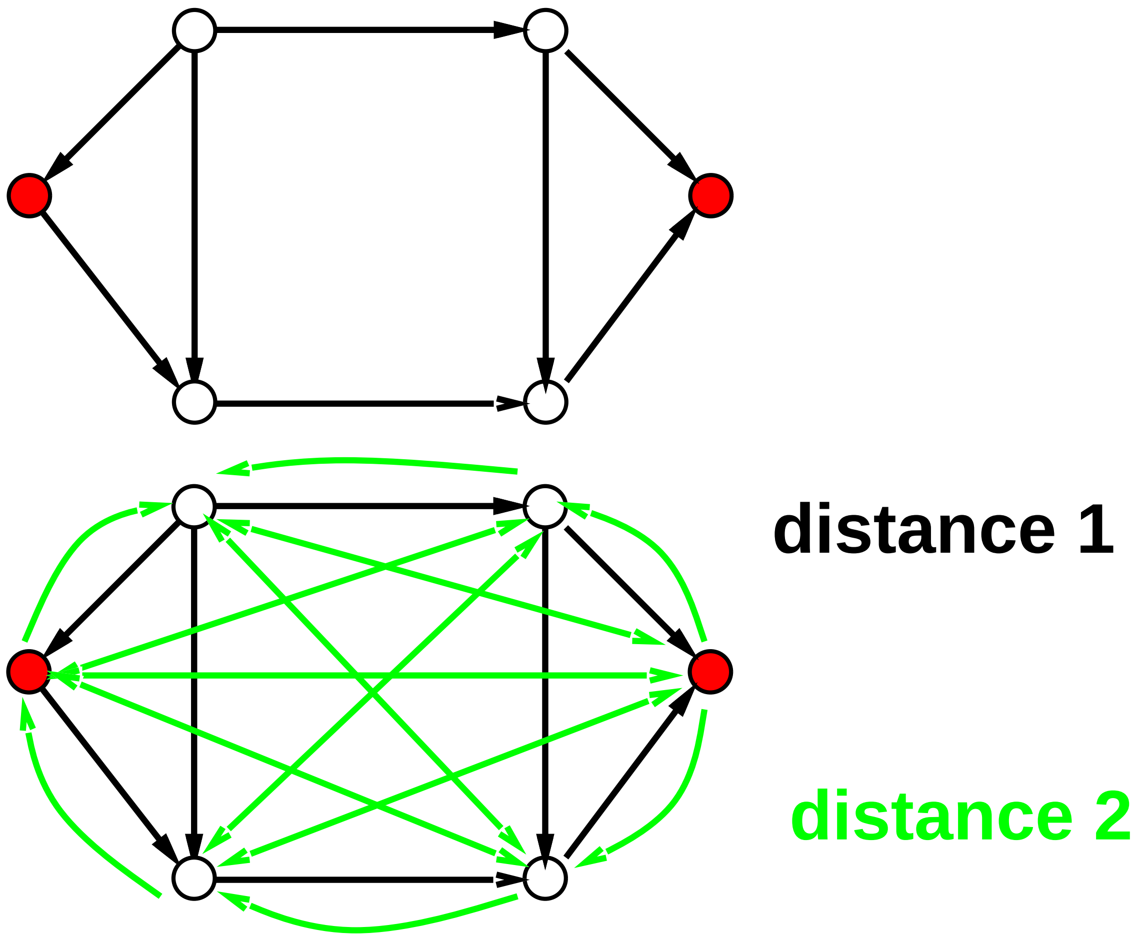
HAMCIRCUIT $\leq_P$ TRAVELING-SALESMAN

Traveling Salesman vs. Hamiltonian Circuit





Traveling Salesman vs. Hamiltonian Circuit



Claim: The is a hamiltonian circuit if and only if there is a tour of length $\leq m$.

Definition

A language B is **NP-complete** if it satisfies

- $B \in NP$, and
- every A in NP is polynomial time reducible to B

Theorem

Theorem: If B is NP-complete and $B \in P$, then $P = NP$.

To show $P = NP$, just find a polynomial-time algorithm for any NP problem.

Theorem

Theorem: If B is NP-complete and $B \leq_P C$, for $C \in NP$, then C is NP-complete.

- We know that $C \in NP$,
- must show that every A in NP is poly-time reducible to C.
- Because B is NP-complete,
- every language in NP is poly-time reducible to B.
- B is poly-time reducible to C
- poly-time reductions compose Why?, so
- A is poly-time reducible to C.

Strategy

- Once we have one NP-complete problem, we can generate more by poly-time reduction.
- Getting the first one requires some work.

Cook-Levin Theorem

We will show that 3SAT is NP complete.

We must show that every problem in NP reduces to 3SAT

We will need some more definitions . . .

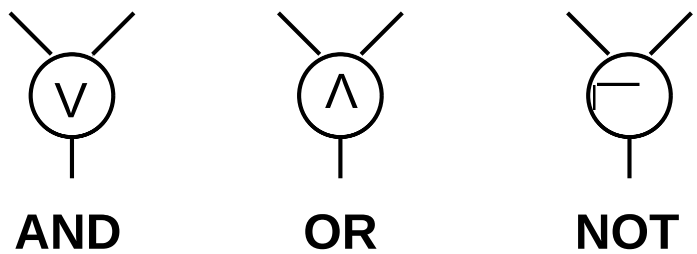
Boolean Circuits

Here is a theoretical analogue of logic circuits.

A **Boolean circuit** is a collection of

- gates
- inputs
- wires
- no cycles

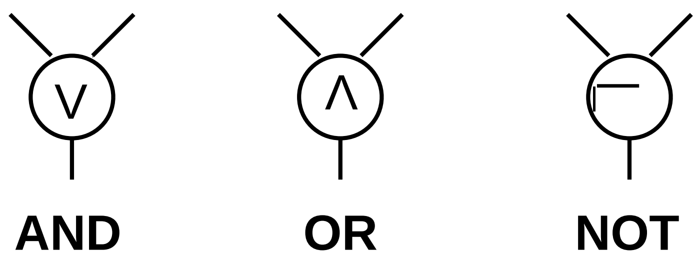
Boolean Circuits



Gates come in three forms

- AND-gates
- OR-gates
- NOT-gates

Boolean Circuits



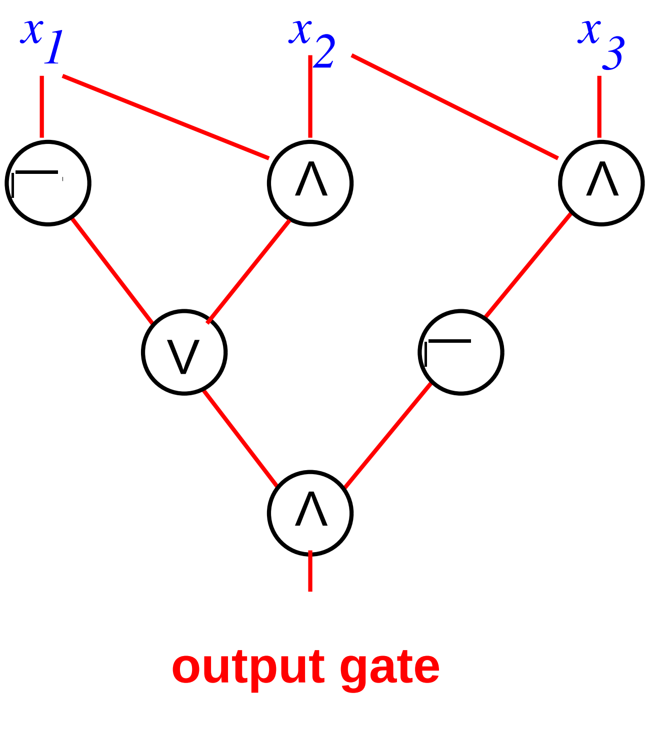
Wires carry 0 or 1.

Gates compute Boolean function

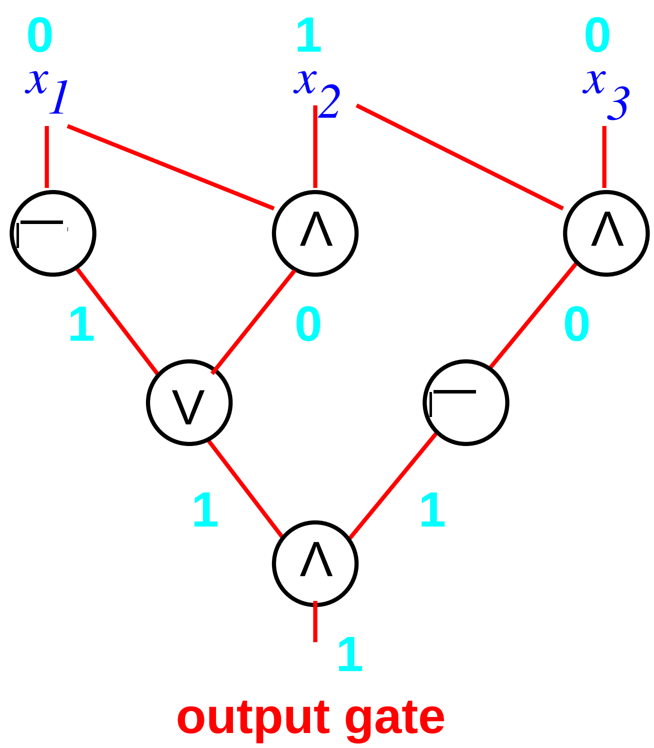
- AND-gates: 1 iff all input wires 1
- OR-gates: 1 iff any input wire 1
- NOT-gates opposite if its input

There is a distinguished *output gate*.

Boolean Circuits



Boolean Circuits



Boolean Circuits

A circuit is *satisfiable* if there is some input setting that produces a 1.

The circuit-satisfiability problem tests whether a circuit is satisfiable.

CIRCUIT-SAT $= \{\langle C \rangle | C$ is a satisfiable Boolean circuit $\}$

Boolean Circuits

Claim: CIRCUIT-SAT is in NP

Pop Quiz: Why?

We will show that circuits can simulate Turing Machines!

Boolean Circuits