CSC373

Weeks 9 & 10: Complexity

Introduction to Complexity

- You have a problem at hand
- You try every technique known to humankind for finding a polynomial time algorithm but fail.
- You try every technique known to humankind for proving that there cannot exist a polynomial time algorithm for your problem but fail.
- What do you do?
  - ➢ Prove that it is NP-complete, of course!

Turing Machines

- "Which problems can a computer (not) solve in a certain amount of time?"
  - How do we mathematically define what a computer is?

- Alan Turing ("Father of Computer Science"), 1936
  - Introduced a mathematical model
  - "Turing machine"
  - All present-day computers can be simulated by a Turing machine
  - Fun fact: TMs can simulate quantum computers too, just inefficiently

Turing Machines

- We won’t formally introduce…but at a high level…
- Turing machine
  - Tape
    - Input is given on tape
    - Intermediate computations can be written there
    - Output must be written there
  - Head pointer
    - Initially pointing at start of input on tape
  - Maintains an internal “state”
  - A transition function describes how to change state, move head pointer, and read/write symbols on tape

Computability

- Church-Turing Thesis
  - “Everything that is computable can be computed by a Turing machine”
  - Widely accepted, cannot be “proven”
  - There are problems which a Turing machine cannot solve, regardless of the amount of time available
    - E.g., the halting problem

- What about the problems we *can* solve? How do we define the time required?
  - Need to define an encoding of the input and output

Encoding

- What can we write on the tape?
  - S = a set of finite symbols
  - S^* = $\cup_{n\geq 0} S^n$ = set of all finite strings using symbols from S
- Input: $w \in S^*$
  - Length of input = $|w|$ = length of w on tape
- Output: $f(w) \in S^*$
  - Length of output = $|f(w)|$
  - Decision problems: output = "YES" or "NO"
    - E.g., "does there exist a flow of value at least 7 in this network?"

Encoding

- Example:
  - "Given $a_1, a_2, \ldots, a_n$, compute $\sum_{i=1}^{n} a_i$"
    - Suppose we are told that $a_i \leq C$ for all i
  - What $|S|$ should we use?
    - $S = \{0,1\}$ ($|S| = 2$, binary representation)
      - Length of input = $O(\log_2 a_1 + \cdots + \log_2 a_n) = O(n \log_2 C)$
    - What about 3-ary ($|S| = 3$) or 18-ary ($|S| = 18$)?
      - Only changes the length by a constant factor, still $O(n \log C)$
    - What about unary (conceptually, $|S| = 1$)?
      - Length blows up exponentially to $O(nC)$
    - Binary is already good enough, but unary isn't

Efficient Computability

- Polynomial-time computability
  - A TM solves a problem in polynomial time if there is a polynomial p such that on every instance of n-bit input and m-bit output, the TM halts in at most $p(n, m)$ steps
  - Polynomial: $n, n^2, 5n^{100} + 1000n^3, n \log^{100} n = o(n^{1.001})$
  - Non-polynomial: $2^n, 2^{\sqrt{n}}, 2^{\log^2 n}$

- Extended Church-Turing Hypothesis
  - "Everything that is efficiently computable is computable by a TM in polynomial time"
  - Much less widely accepted, especially today
  - But in this course, efficient = polynomial-time

P

- P (polynomial time)
  - The class of all decision problems computable by a TM in polynomial time
- Examples
  - Addition, multiplication, square root
  - Shortest paths
  - Network flow
  - Fast Fourier transform
  - Checking if a given number is a prime [Agrawal-Kayal-Saxena 2002]
  - …

NP

- NP (nondeterministic polynomial time) intuition
  - Subset sum problem:

    Given an array $\{-7, -3, -2, 5, 8\}$, is there a zero-sum subset?
  - Enumerating all subsets is exponential
  - But…given {-3, -2, 5}, we can verify in polynomial time that it is indeed a valid subset and has zero sum
  - A nondeterministic Turing machine could “guess” the solution and then test if it has zero sum in polynomial time

NP

- NP (nondeterministic polynomial time)
  - The class of all decision problems for which a YES answer can be verified by a TM in polynomial time given polynomial length “advice” or “witness”.

  - There is a polynomial-time verifier TM V and another polynomial p such that
    - For all YES inputs x, there exists advice y with $|y| = p(|x|)$ on which $V(x, y)$ returns ACCEPT
    - For all NO inputs x, $V(x, y)$ returns REJECT for every possible y

  - Informally: “Whenever the answer is YES, there’s a short proof of it.”
    - When the answer is NO, there may not be any short proof for it.

co-NP

- co-NP
  - Same as NP, except whenever the answer is NO, there is a short proof of it
- Open questions
  - NP = co-NP?
  - P = NP ∩ co-NP?
  - And…drum roll please…

$$P = NP?$$

P versus NP

- Lance Fortnow in his article on P and NP in Communications of the ACM, Sept 2009

"The P versus NP problem has gone from an interesting problem related to logic to perhaps the most fundamental and important mathematical question of our time, whose importance only grows as computers become more powerful and widespread."

Millenium Problems

- Award of \$1M for each problem by the Clay Math institute

1. Birch and Swinnerton-Dyer Conjecture
2. Hodge Conjecture
3. Navier-Stokes Equations
4. P = NP?
5. Poincare Conjecture (Solved)[1]
6. Riemann Hypothesis
7. Yang-Mills Theory

Claim: Worth >> \$1M

[1]Solved by Grigori Perelman (2003): Prize unclaimed

Cook's Conjecture

- Cook's conjecture
  - (And every sane person's belief…)
  - P is likely not equal to NP

- Why do we believe this?
  - There is a large class of problems (NP-complete)
  - By now, contains thousands and thousands of problems
  - Each problem is the "hardest problem in NP"
  - If you can efficiently solve *any one of them,* you can efficiently solve *every problem in NP*
    - Despite decades of effort, no polynomial time solution has been found for *any of them*

Reductions

- Problem A is p-reducible to problem B (denoted $A \leq_p B$) if an "oracle" (subroutine) for B can be used to efficiently solve A
  - You can solve A by making polynomially many calls to the oracle for B and doing additional polynomial-time computation

- Question: If A is p-reducible to B, then which of the following is true?
  a) If A cannot be solved efficiently, then neither can B.
  b) If B cannot be solved efficiently, then neither can A.
  c) Both.
  d) None.

Reductions

- Problem A is p-reducible to problem B (denoted $A \leq_p B$) if an "oracle" (subroutine) for B can be used to efficiently solve A
  - You can solve A by making polynomially many calls to the oracle and doing additional polynomial computation

- Question: If I want to prove that my problem X is "hard", I should:
  a) Reduce my problem X to a known hard problem.
  b) Reduce a known hard problem to my problem X.
  c) Both.
  d) None.

NP-completeness

- NP-completeness
  - A problem B is NP-complete if it is in NP and every problem A in NP is p-reducible to B
  - Hardest problems in NP
  - If one of them can be solved efficiently, every problem in NP can be solved efficiently, implying P=NP

- Observation:
  - If A is in NP, and some NP-complete problem B is p-reducible to A, then A is NP-complete too
    - Every problem in NP $\leq_p B \leq_p A$

NP-completeness

- But this uses an already known NP-complete problem to prove another problem is NP-complete

- How do we find the *first* NP-complete problem?
  - How do we know there are *any* NP-complete problems at all?
  - Key result by Cook
  - First NP-complete problem: SAT
    - By a direct reduction from every problem in NP to SAT
    - "From first principles"

CNF Formulas

- Conjunctive normal form (CNF)
  - Boolean variables $x_1, x_2, \dots, x_n$
  - Their negations $\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n$
  - Literal ℓ: a variable or its negation
  - Clause $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_r$ is a disjunction of literals
  - CNF formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ is a conjunction of clauses
    - kCNF: Each clause has at most k literals
    - Exact kCNF: Each clause has exactly k literals

  - Example of (Exact) 3CNF

$$\varphi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee x_1)$$

SAT and Exact 3SAT

- Example of (Exact) 3CNF

  $\varphi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee x_1)$

- "SAT" (Satisfiability) Problem:
  - A CNF formula φ is satisfiable if there is an assignment of truth values (TRUE/FALSE) to variables under which the formula evaluates to TRUE
    - That means, in each clause, at least one literal is TRUE
  - SAT: "Given a CNF formula φ, is it satisfiable?"
  - Exact 3SAT: "Given an exact 3CNF formula φ, is it satisfiable?"

SAT and Exact 3SAT

- Cook-Levin Theorem
  - SAT (and even Exact 3SAT) is NP-complete

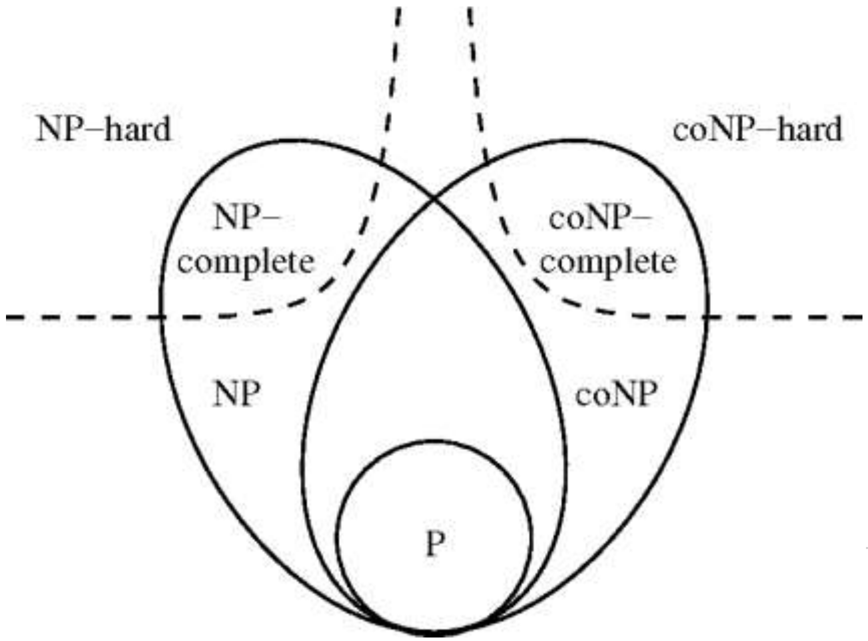
- Doesn't use any known NP-complete problem
  - Directly reduces any given NP problem to SAT
  - Reduction is a bit complex, so we'll defer it until later
  - But for now, let's assume SAT and Exact 3SAT are NP-complete and reduce them to other problems (and then those problems to other problems…)

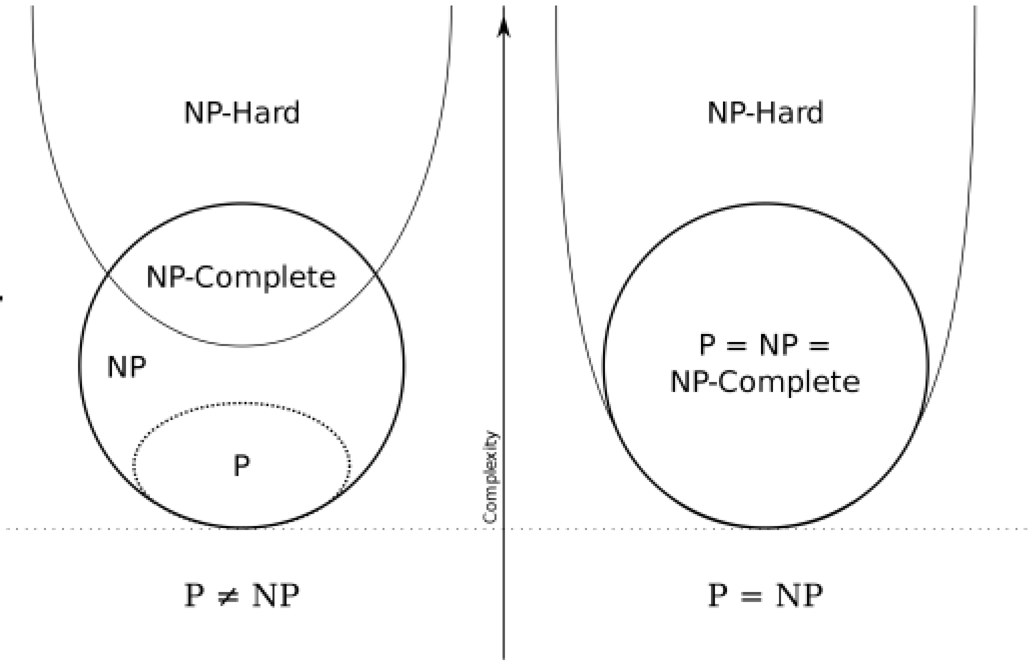
NP-Complete Examples

- NP-complete problems
  - SAT = first NP complete problem
  - Decision TSP: Is there a route visiting all n cities with total distance at most k?
  - 3-Colorabitility: Can the vertices of a graph be colored with at most 3 colors such that no two adjacent vertices have the same color?
  - Karp's 21 NP-complete problems

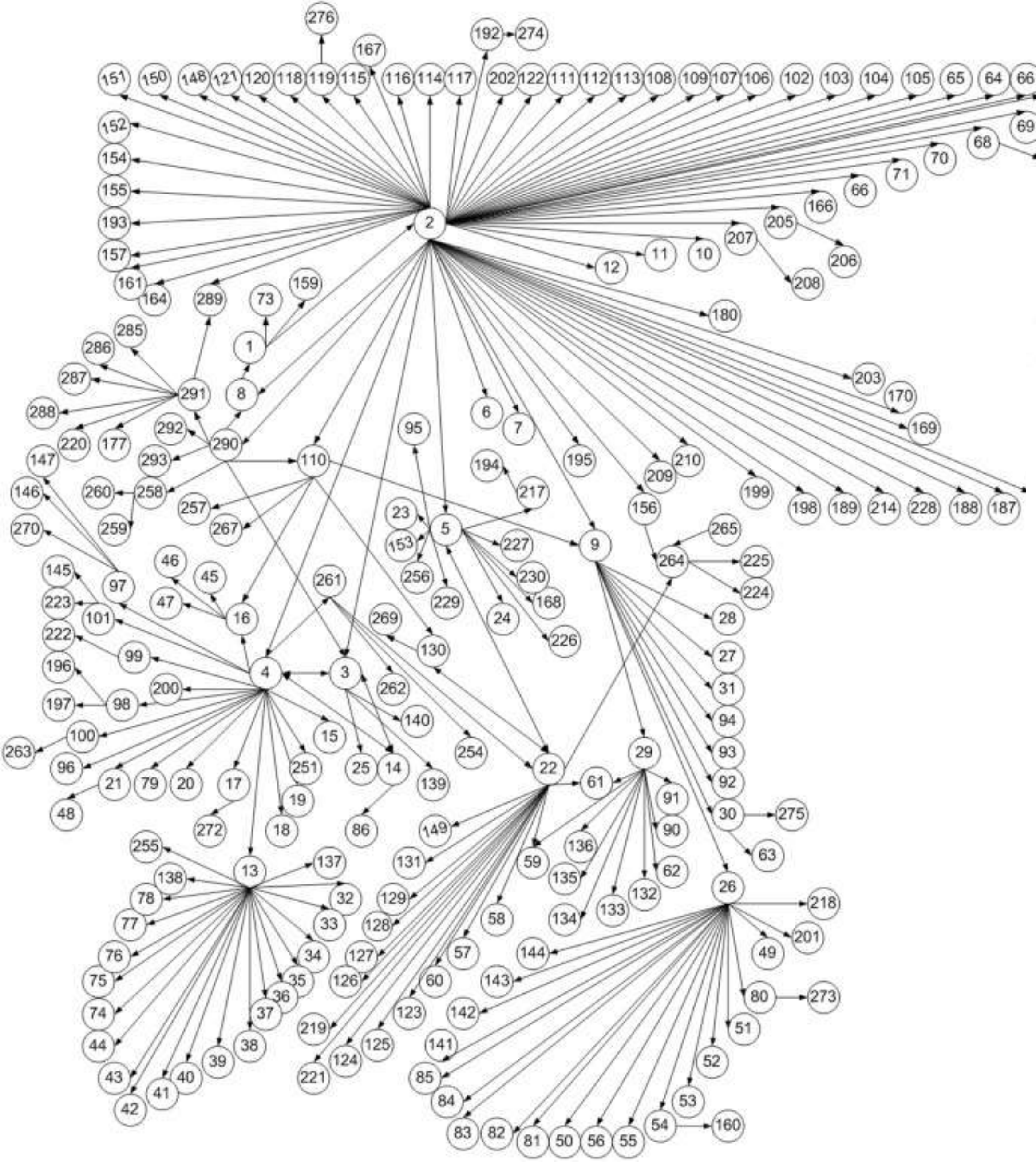
- co-NP-complete
  - Tautology problem ("negation" of SAT):
    - "Given a CNF formula φ, does it always evaluate to TRUE regardless of variable assignments?"

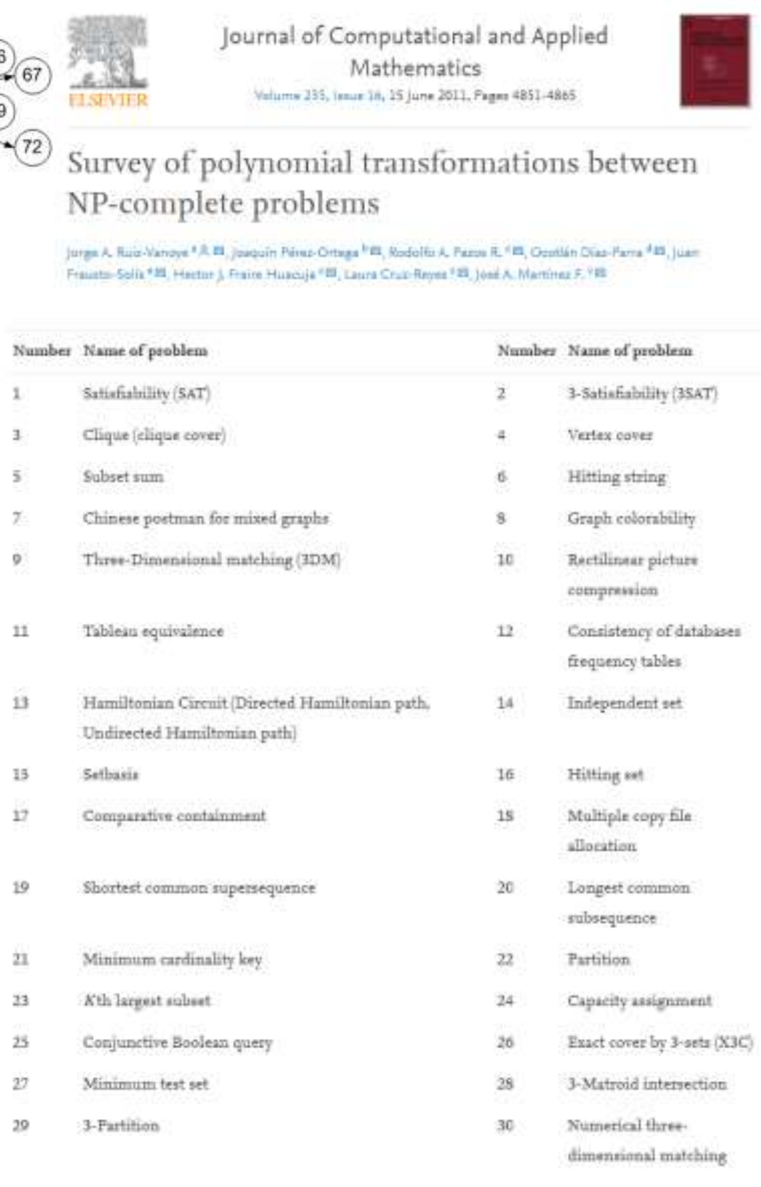
Complexity





By Behnam Esfahbod, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=3532181





Journal of Computational and Applied Mathematics

Volume 235, Issue 16, 15 June 2011, Pages 4851-4865

Survey of polynomial transformations between NP-complete problems

Jorge A. Ruiz-Vanoye [a], Joaquín Pérez-Ortega [b], Rodolfo A. Pazos R. [c], Ocotlán Díaz-Parra [d], Juan Frausto-Solís [e], Hector J. Fraire Huacuja [f], Laura Cruz-Reyes [f], José A. Martínez F. [f]

Number	Name of problem	Number	Name of problem
1	Satisfiability (SAT)	2	3-Satisfiability (3SAT)
3	Clique (clique cover)	4	Vertex cover
5	Subset sum	6	Hitting string
7	Chinese postman for mixed graphs	8	Graph colorability
9	Three-Dimensional matching (3DM)	10	Rectilinear picture compression
11	Tableau equivalence	12	Consistency of databases frequency tables
13	Hamiltonian Circuit (Directed Hamiltonian path, Undirected Hamiltonian path)	14	Independent set
15	Setbasis	16	Hitting set
17	Comparative containment	18	Multiple copy file allocation
19	Shortest common supersequence	20	Longest common subsequence
21	Minimum cardinality key	22	Partition
23	Kth largest subset	24	Capacity assignment
25	Conjunctive Boolean query	26	Exact cover by 3-sets (X3C)
27	Minimum test set	28	3-Matroid intersection
29	3-Partition	30	Numerical three-dimensional matching

⋮ ⋮

Polynomial-Time Reductions

constraint satisfaction

$$\begin{tikzcd}
 & \boxed{\text{3-SAT}} \arrow[dl, "\text{3-SAT reduces to INDEPENDENT SET}"'] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\boxed{\text{INDEPENDENT SET}} \arrow[d] & \boxed{\text{DIR-HAM-CYCLE}} \arrow[d] & \boxed{\text{GRAPH 3-COLOR}} \arrow[d] & \boxed{\text{SUBSET-SUM}} \arrow[d] \\
\boxed{\text{VERTEX COVER}} \arrow[d] & \boxed{\text{HAM-CYCLE}} \arrow[d] & \boxed{\text{PLANAR 3-COLOR}} & \boxed{\text{SCHEDULING}} \\
\boxed{\text{SET COVER}} & \boxed{\text{TSP}} & &
\end{tikzcd}$$

packing and covering | sequencing | partitioning | numerical

Dick Karp (1972)
1985 Turing Award

Just A Tad Bit of History

- [Cook 1971]
  - Proved Exact 3SAT is NP-complete in seminal paper

- [Karp 1972]
  - Showed that 20 other problems are also NP-complete
  - “Karp's 21 NP-complete problems”
  - Renewed interest in this idea

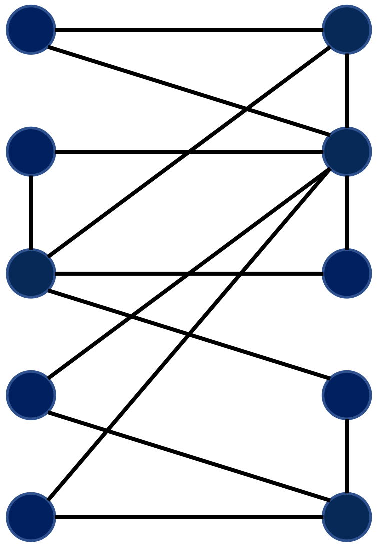
- 1982: Cook won the Turing award

Independent Set

- Problem
  - Input: Undirected graph $G = (V, E)$, integer k
  - Question: Does there exist a subset of vertices $S \subseteq V$ with $|S| = k$ such that no two vertices in S are adjacent?

Example:

- Does this graph have an independent set of size 6?
  - Yes!
- Does this graph have an independent set of size 7?
  - No!



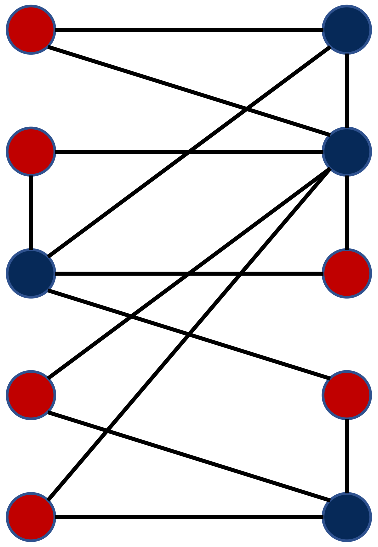
● = independent set

Independent Set

- Problem
  - Input: Undirected graph $G = (V, E)$, integer k
  - Question: Does there exist a subset of vertices $S \subseteq V$ with $|S| = k$ such that no two vertices in S are adjacent?

Example:

- Does this graph have an independent set of size 6?
  - Yes!
- Does this graph have an independent set of size 7?
  - No!



● = independent set

Independent Set

- Claim: Independent Set is in NP

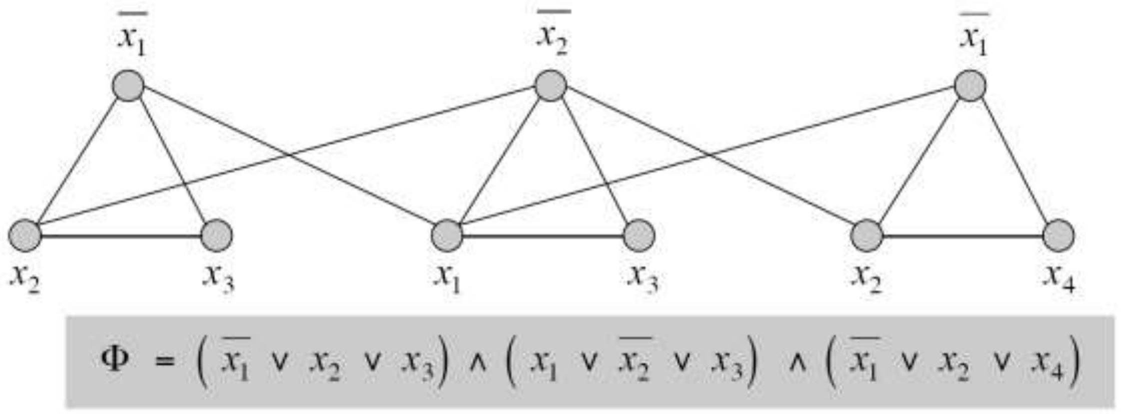
- Recall: We need to show that there is a polynomial-time algorithm which
  - Can accept every YES instance with the right polynomial-size advice
  - Will not accept a NO instance with any advice

- Advice: the actual independent set S
- Algorithm: check if S is an independent set and if $|S| = k$
- Simple!

Independent Set

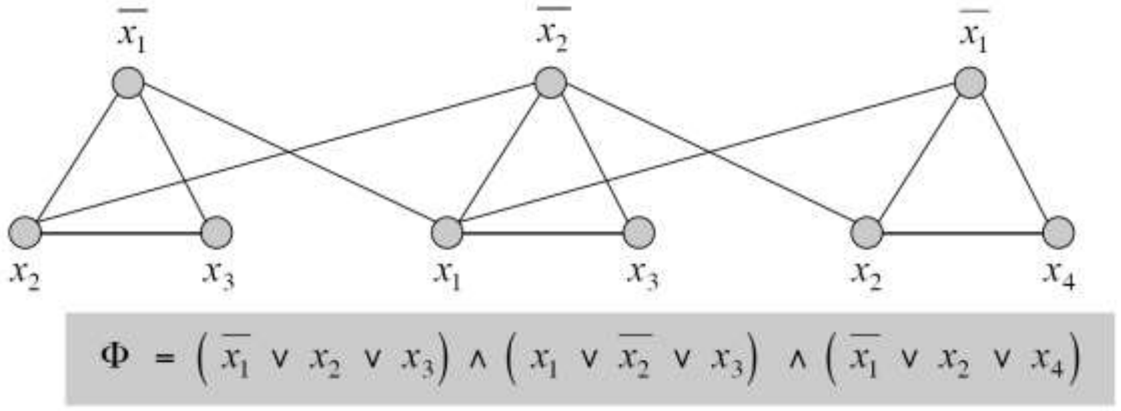
- Claim: Exact 3SAT $\leq_p$ Independent Set

➢ Given a formula φ of Exact 3SAT with k clauses, construct an instance (G, k) of Independent Set as follows
  - Create 3 vertices for each clause (one for each literal)
  - Connect them in a triangle
  - Connect the vertex of each literal to each of its negations


$\overline{x_1}$
$\overline{x_2}$
$\overline{x_1}$
x_2
x_3
x_1
x_3
x_2
x_4
$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

Independent Set

- Why does this work?
  - Exact 3SAT = YES ⇒ Independent Set = YES
    - From each clause, take any literal that is TRUE in the assignment
  - Independent Set = YES ⇒ Exact 3SAT = YES
    - Independent set S must contain one vertex from each triangle
    - No literal and its negation are both in S
    - Set literals in S to TRUE, their negations to FALSE, and the rest to arbitrary values



$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Different Types of Reductions

- $A \leq B$
  - Karp reductions
    - Take an arbitrary instance of A, and in polynomial time, construct a single instance of B with the same answer
    - Very restricted type of reduction
    - The reduction we just constructed was a Karp reduction
  - Turing/Cook reductions
    - Take an arbitrary instance of A, and solve it by making polynomially many calls to an oracle for solving B and some polynomial-time extra computation
    - Very general reduction
    - In this course, we'll allow Turing/Cook reductions, but whenever possible, see if you can construct a Karp reduction

Subset Sum

- Problem
  - Input: Set of integers $S = \{w_1, \dots, w_n\}$, integer W
  - Question: Is there $S' \subseteq S$ that adds up to exactly W?

- Example
  - $S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}, W = 3754?$
  - Yes!
    - $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754$

Subset Sum

- Claim: Subset Sum is in NP

- Recall: We need to show that there is a polynomial-time algorithm which
  - Can accept every YES instance with the right polynomial-size advice
  - Will not accept a NO instance with any advice

- Advice: the actual subset S'
- Algorithm: check that S' is indeed a subset of S and sums to W
- Simple!

Subset Sum

- Claim: Exact 3SAT $\leq_p$ Subset Sum

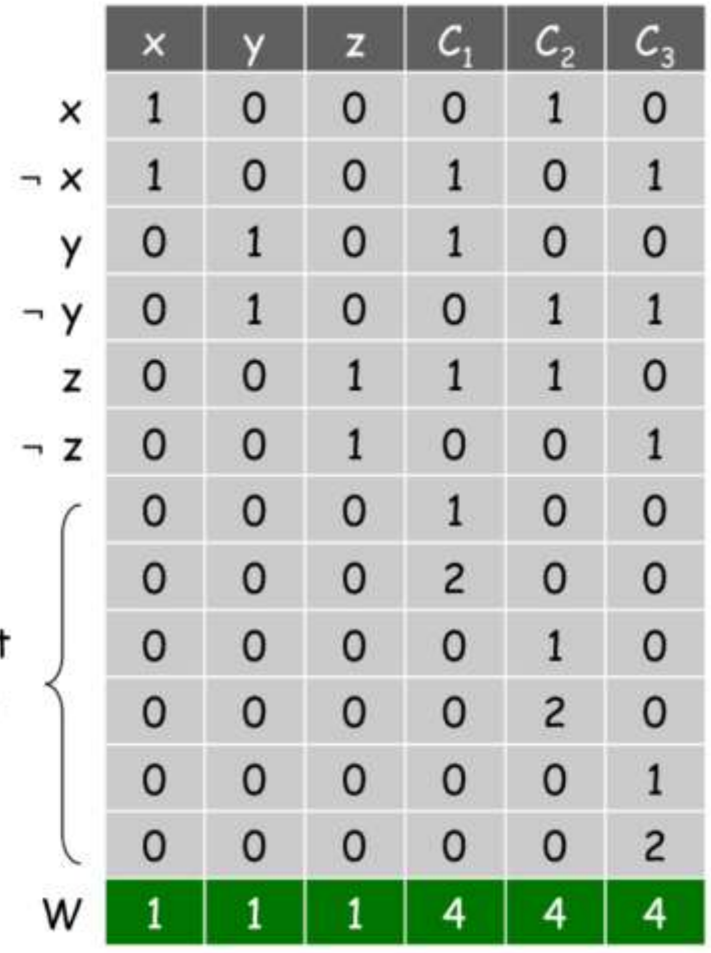
- Given a formula φ of Exact 3SAT, we want to construct (S, W) of Subset Sum with the same answer
- In the table in the following slide:
  - Columns are for variables and clauses
  - Each row is a number in S, represented in decimal
  - Number for literal ℓ : has 1 in its variable column and in the column of every clause where that literal appears
    - Number selected = literal set to TRUE
  - "Dummy" rows: can help make the sum in a clause column 4 if and only if at least one literal is set to TRUE

Subset Sum

- Claim: Exact 3SAT $\leq_p$ Subset Sum

$$C_1 = \bar{x} \vee y \vee z$$
$$C_2 = x \vee \bar{y} \vee z$$
$$C_3 = \bar{x} \vee \bar{y} \vee \bar{z}$$

Decimal representation



	x	y	z	C_1	C_2	C_3
x	1	0	0	0	1	0
¬ x	1	0	0	1	0	1
y	0	1	0	1	0	0
¬ y	0	1	0	0	1	1
z	0	0	1	1	1	0
¬ z	0	0	1	0	0	1
	0	0	0	1	0	0
	0	0	0	2	0	0
	0	0	0	0	1	0
	0	0	0	0	2	0
	0	0	0	0	0	1
	0	0	0	0	0	2
W	1	1	1	4	4	4

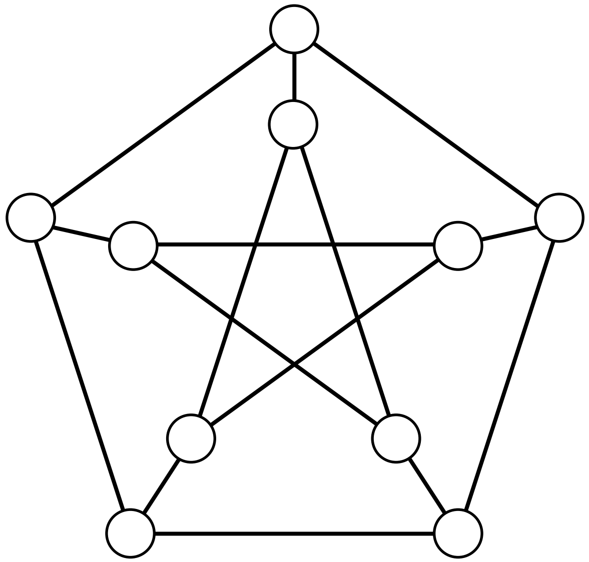
dummies to get clause columns to sum to 4

Subset Sum

- Note
  - The Subset Sum instance we constructed has "large" numbers
    - Their values are exponentially large (~$10^{\#variables+\#clauses}$)
    - But the number of bits required to write them is polynomial
  - Can we hope to construct Subset Sum instance with numbers whose values are only $poly(\#variables, \#clasuses)$ large?
    - Unlikely, as that would prove $P = NP$!
    - Like Knapsack, Subset Sum can be solved in pseudo-polynomial time
      - That is, in polynomial time if the numbers are only polynomially large in value

3-Coloring

- Problem
  - Input: Undirected graph $G = (V, E)$
  - Question: Can we color each vertex of G using at most three colors such that no two adjacent vertices have the same color?



3-Coloring

- Claim: 3-coloring is in NP

- Recall: We need to show that there is a polynomial-time algorithm which
  - Can accept every YES instance with the right polynomial-size advice
  - Will not accept a NO instance with any advice

- Advice: colors of the nodes in a valid 3-coloring
- Algorithm: check that this is a valid 3-coloring
- Simple!

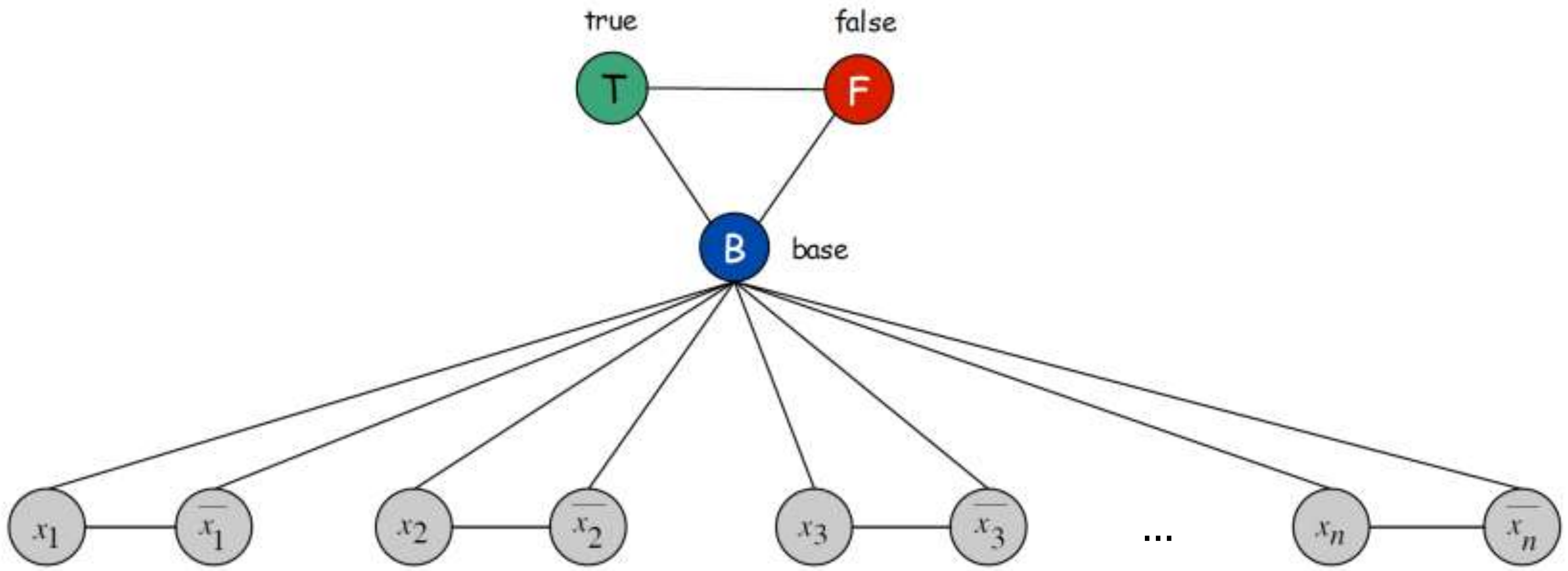
3-Coloring

- Claim: Exact 3SAT $\leq_p$ 3-Coloring

  - Given an Exact 3SAT formula φ, we want to construct a graph G such that G is 3-colorable if and only if φ has a satisfying assignment
  - G will have the following nodes:
    - Type 1: true, false, base, one for each x_i, one for each $\overline{x_i}$
    - Type 2: additional nodes for each clause C_j
  - 1-1 correspondence between valid 3-colorings of type 1 nodes and valid truth assignments:
    - All literals with the same color as "true" node are set to true
    - All literals with the same color as "false" node are set to false
  - Claim: Fix any colors of type 1 nodes. There exists a valid 3-coloring of G giving these colors to type 1 nodes if and only if the corresponding truth assignment is satisfying for φ.

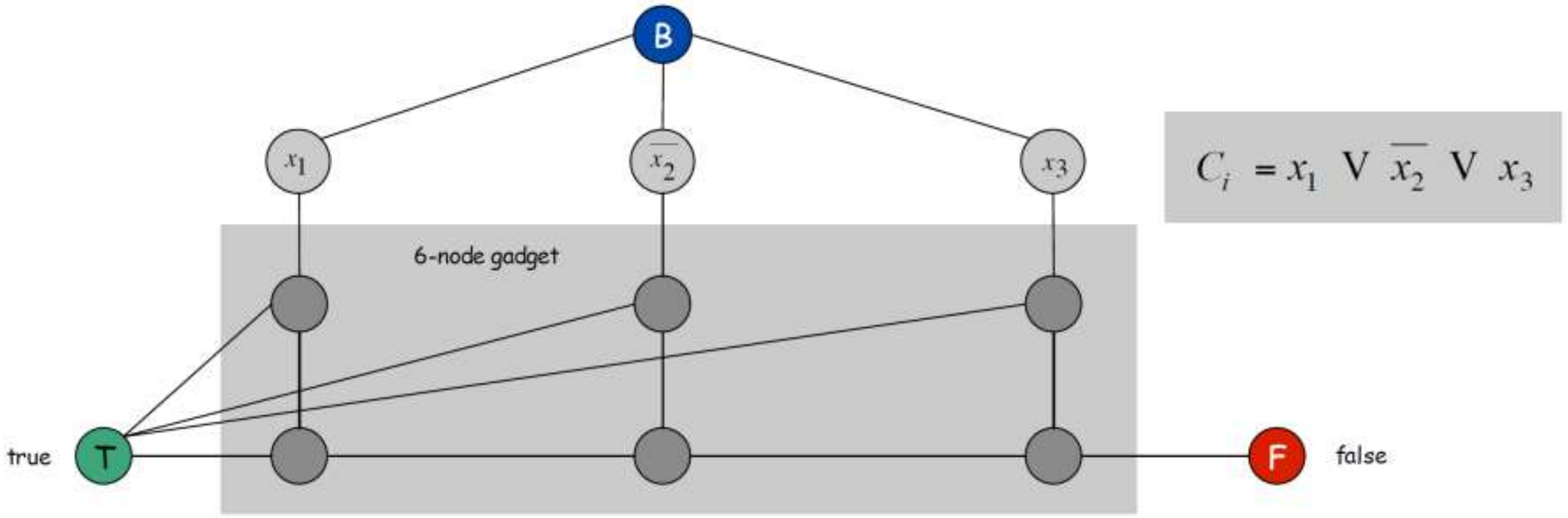
3-Coloring

- Create 3 new nodes T, F, and B, and connect them in a triangle
- Create a node for each literal, connect it to its negation and to B
- T-F-B must have different colors, and so must B-x_i-$\bar{x}_i$
  - Each literal has the color of T or F; its negation has the other color
  - Valid 3-coloring ⇔ valid truth assignment (set all with color T to true)



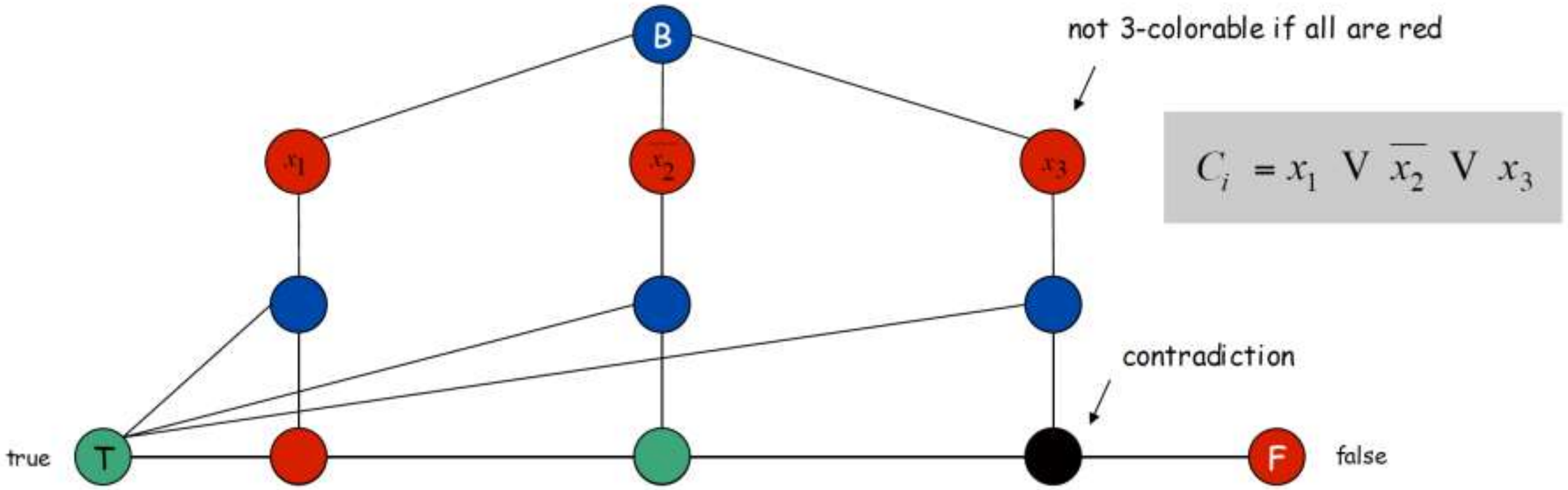
3-Coloring

- We also need valid 3-coloring ⇔ *satisfying* truth assignment
  - For each clause, add the following gadget with 6 nodes and 13 edges
  - Claim: Clause gadget is 3-colorable ⇔ at least one of the nodes corresponding to the literals in the clause is assigned color of T



3-Coloring

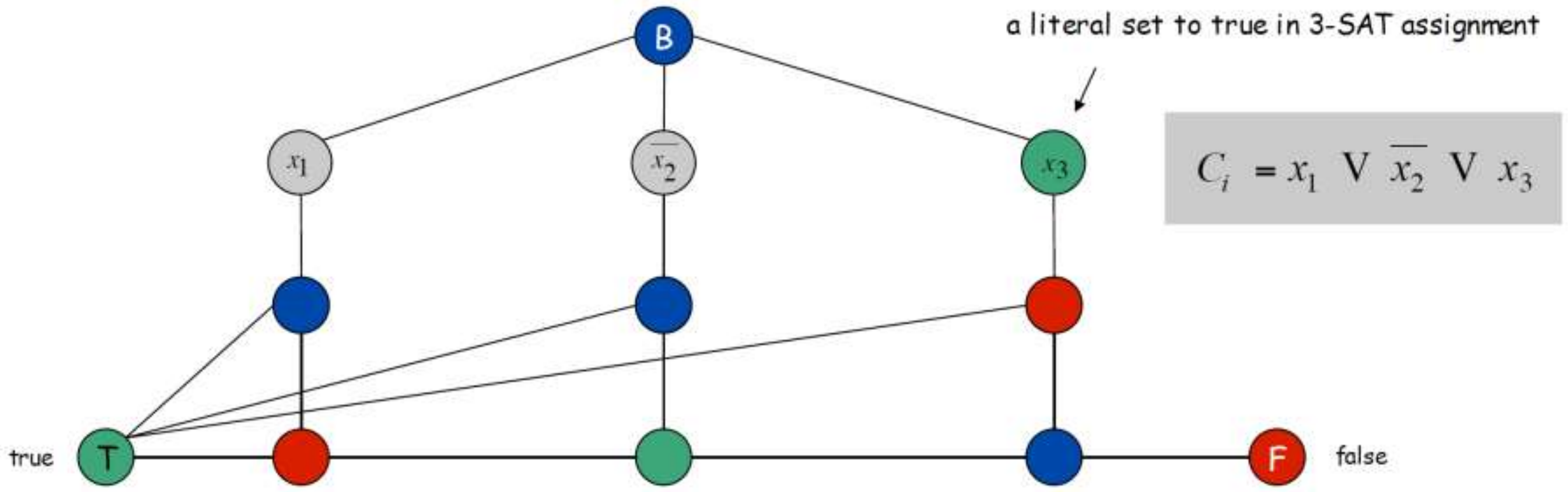
- Claim: Valid 3-coloring ⇒ truth assignment satisfies φ
  - Suppose a clause C_i is not satisfied, so all its three literals must be F
  - Then the 3 nodes in top layer must be B
  - Then the first two nodes in bottom layer must be F and T
  - No color left for the remaining node ⇒ contradiction!



B
not 3-colorable if all are red
x_1
$\overline{x_2}$
x_3
$C_i = x_1 \vee \overline{x_2} \vee x_3$
contradiction
true
T
F
false

3-Coloring

- We just proved: valid 3-coloring ⇒ satisfying assignment
- Claim: satisfying assignment ⇒ valid 3-coloring
  - Each clause has at least one literal with color T
  - Exercise: Regardless of which literal has color T and which color (T/F) the other literals have, the clause widget can always be 3-colored


a literal set to true in 3-SAT assignment
$C_i = x_1 \vee \overline{x_2} \vee x_3$
B
x_1
$\overline{x_2}$
x_3
true
T
F
false

Review of Reductions

- If you want to show that problem B is NP-complete
- Step 1: Show that B is in NP
  - Some polynomial-size advice should be sufficient to verify a YES instance in polynomial time
  - No advice should work for a NO instance

  - Usually, the solution of the "search version" of the problem works
    - But sometimes, the advice can be non-trivial
    - For example, to check LP optimality, one possible advice is the values of both primal and dual variables, as we saw in the last lecture

Review of Reductions

- If you want to show that problem B is NP-complete
- Step 2: Find a known NP-complete problem A and reduce it to B (i.e., show A $\leq_p$ B)
  - This means taking an arbitrary instance of A, and solving it in polynomial time using an oracle for B
    - Caution 1: Remember the direction. You are "reducing known NP-complete problem to your current problem".
    - Caution 2: The size of the B-instances you construct should be polynomial in the size of the original A-instance
  - This would show that if B can be solved in polynomial time, then A can be as well
  - Some reductions are trivial, some are notoriously tricky…