CS 341: ALGORITHMS

Lecture 22: intractability V – More NPC transformations

Readings: see website

Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

1

LAST TIME

- Polynomial transformations
  - Poly transformation from Clique to Vertex Cover
- NP Completeness
  - SAT is NP complete (NPC)
  - Got part way through showing 3SAT is NPC
    - Did poly transformation from SAT to 3SAT
    - Need to also show 3SAT is in NP

2

LET'S DO A BRIEF REVIEW

of **NPC**, **poly transformations**, and showing a **problem is in NP**

3

COMPLEXITY CLASS **NP-COMPLETE** (NPC)

The complexity class **NPC** denotes the set of all decision problems Π that satisfy the following two properties:

$\Pi \in$ **NP**

For all $\Pi' \in$ **NP**, $\Pi' \leq_P \Pi$.

NPC is an abbreviation for **NP-complete**.

Note that the definition does not imply that NP-complete problems exist!

Mechanics of proving $\Pi \in$ NPC
1. Show Π is in NP
2. Show a poly transformation from some NPC problem to Π

4

MECHANICS OF SHOWING A PROBLEM IS **IN NP**

- How to show $\Pi \in NP$
1. Define a yes-certificate
2. Design a poly-time $verify(I, C)$ algorithm
3. Correctness proof
   - **Case 1:** Let I be any yes-instance;
     Find C such that $verify(I, C) = true$
   - **Case 2:** Let I be any no-instance,
     and C be any certificate;
     Prove $verify(I, C) = false$

5

POLYNOMIAL TRANSFORMATION FOR PROVING $\mathbf{\Pi_2}$ **IS IN NPC**

Known NPC problem

Problem you want show is NPC

- Let Π_1 and Π_2 be decision problems
- $\mathbf{\Pi_1 \leq_P \Pi_2}$ **iff** there exists $f : \mathcal{I}(\Pi_1) \rightarrow \mathcal{I}(\Pi_2)$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(\Pi_1)$
  - If $I \in \mathcal{I}_{yes}(\Pi_1)$ then $f(I) \in \mathcal{I}_{yes}(\Pi_2)$
  - **If $\boldsymbol{f(I) \in \mathcal{I}_{yes}(\Pi_2)}$ then $\boldsymbol{I \in \mathcal{I}_{yes}(\Pi_1)}$**

6

LET'S FINISH SHOWING 3SAT ∈ NPC

- Already poly transformed SAT to 3SAT
- Need to show 3SAT in NP

PROVING 3SAT IS **IN NP**

1. Define desired YES-certificate
2. Design a poly-time $verify(I, C)$ algorithm
3. Correctness proof
   - **Case 1:** Let I be any yes-instance; Find C such that $verify(I, C) = true$
   - **Case 2:** Let I be any no-instance, and C be any certificate; Prove $verify(I, C) = false$
   - **Contrapositive of case 2:** Suppose $verify(I, C) = true$; Prove I is a yes-instance

3SAT input $I = (Clauses[1..m], n)$: a list of m **clauses**, and the number n of variables. Each clause contains literals. Each literal is a pair (var, neg): a variable ∈ $\{1..n\}$ & a negation bit

YES-certificate C = array with one bit per variable in $\{1..n\}$ representing a **satisfying assignment**

```
verify3SAT(I=(Clauses[1..m], n), C)
    if C is not an array of n bits return false

    numSat = 0
    for each c in Clauses
        for each literal (var, neg) in c
            if (C[var] && !neg) or (!C[var] && neg)
                numSat++
                break

    return (numSat == m)
```

This takes $O(|\text{Clauses}|)$ time, which is polynomial in $\text{Size}(I)$

MECHANICS OF SHOWING A PROBLEM IS **IN NP**

1. Define desired YES-certificate
2. Design a poly-time $verify(I, C)$ algorithm
3. Correctness proof
   - **Case 1:** Let I be any yes-instance; Find C such that $verify(I, C) = true$
   - **Case 2:** Let I be any no-instance, and C be any certificate; Prove $verify(I, C) = false$
   - **Contrapositive of case 2:** Suppose $verify(I, C) = true$; Prove I is a yes-instance

Let I be a yes-instance of 3SAT. Then it has a satisfying assignment $\boldsymbol{A_s}$. And, $verify(I, A_s)$ will see that each clause contains a literal satisfied by this assignment, so $verify$ will see $numSat = |Clauses|$ and return true.

Suppose $verify(I, C)$ returns true. Then $numSat = |Clauses|$, so $numSat$ was incremented in each iteration of the loop over clauses, so each clause contains a satisfied literal, so the 3SAT formula in I is satisfied by C, so I is a yes-instance.

It follows that **3SAT is in NP.** Since we have already shown SAT $\leq_P$ 3SAT, we now know that **3SAT is NP-COMPLETE.**

Summary of Polynomial Transformations

SAT ← **Every problem in NP can be poly transformed to**

↓ **can be poly transformed to**

3-SAT — Since SAT is NP-complete, so is 3-SAT!

Today and next time let's start filling out a hierarchy of reductions that prove several problems are NP complete

But first, since you need to know **NP hardness** for your assignment…

NP-HARDNESS

Intuitively: problems that are ***<u>at least as hard</u>*** *as NP-complete (but are not necessarily decision problems)*

NP-hard Problems

TSP-Optimal Value is also NP-hard (and not in NP)

This version returns the **total weight** of an optimal Hamiltonian cycle

A problem Π is **NP-hard** if there exists a problem $\Pi' \in$ **NPC** such that $\Pi' \leq_P^T \Pi$.

Every NP-complete problem is automatically NP-hard, but there exist NP-hard problems that are not NP-complete.

Typical examples of NP-hard problems are optimization problems corresponding to NP-complete decision problems.

Reduction from lecture 19/20

For example, $\boxed{\textbf{TSP-Decision} \leq_P^T \textbf{TSP-Optimization}}$ and **TSP-Decision** $\in$ **NPC**, so **TSP-Optimization** is NP-hard.

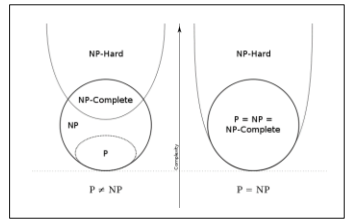
Returns an **optimal Hamiltonian cycle**

COMPARING NPC AND NP HARD

- $\Pi \in \mathrm{NPC}$
  - Must be a decision problem
  - Must poly transform some NPC problem to Π
  - Must show Π in NP
- $\Pi \in \mathrm{NPHard}$
  - Does not need to be a decision problem
  - Can use either poly transform or poly Turing reduction
  - Does not need to be in NP (and can't be if not decision)

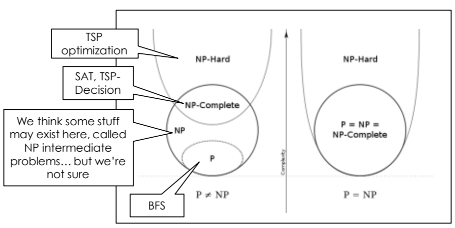
13

TWO POSSIBLE REALITIES…



14

SOME PROBLEMS IN EACH



15

ESTABLISHING ANOTHER NPC PROBLEM

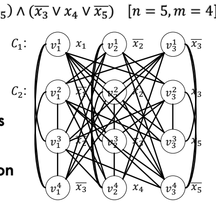
… BY TRANSFORMING **3-SAT** TO **CLIQUE**

(Proving 3-SAT $\leq_P$ Clique)

16

SHOWING 3-SAT $\leq_P$ CLIQUE

- Let I be an instance of 3-SAT with n variables $x_1 \dots x_n$ and m clauses $C_1 \dots C_m$
  - *E.g.,* $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee x_5) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_5})$ $[n = 5, m = 4]$
- We construct **Clique** input $\boldsymbol{f(I) = (G, k)}$:
  - Node v_ℓ^c for each literal $1 \leq \ell \leq 3$ in each clause $1 \leq c \leq m$ (so $|V| = 3m$)
  - Edges between all **non-contradictory** pairs of nodes (no $x_i \wedge \overline{x_i}$) in **different clauses**
  - $k = m$ (can we find an $\boldsymbol{m}$-clique?)
  - Must prove this is a **polynomial transformation**



Reasonable 3-SAT representation: $array[1..m]$ of clauses <l_1, l_2, l_3> of literals <v, neg> where $v \in \{1..n\}$.

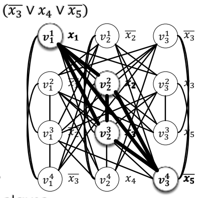
Note $O(m) \subseteq O(Size(I))$,
So runtime $O(m^2) \subseteq O(Size(I)^2)$ → **polytime!**

Runtime: create $3m$ nodes, $O(m^2)$ edges, at $O(1)$ time each

17

SHOWING 3-SAT $\leq_P$ CLIQUE

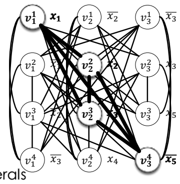
- Let I be an instance of 3-SAT with n variables $x_1 \dots x_n$ and m clauses $C_1 \dots C_m$
  - *E.g.,* $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee x_5) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_5})$
- **<u>Case 1:</u>** Suppose I is a **yes**-instance of 3-SAT, and show $\boldsymbol{f(I)}$ is a **yes**-instance of $\boldsymbol{m}$**-clique**
- Since I is a yes-instance, ∃ a **satisfying** assignment
  - *E.g.,* $x_1 = 1,\ x_2 = 1,\ x_3 = 1,\ x_4 = 0,\ x_5 = 0$
- For each clause $\boldsymbol{C_i}$, let $\boldsymbol{s_i}$ be a **<u>satisfied</u> literal** in $\boldsymbol{C_i}$
  - *E.g.,* $\boldsymbol{s_1} = x_1,\ \boldsymbol{s_2} = x_2,\ \boldsymbol{s_3} = x_3,\ \boldsymbol{s_4} = \overline{x_5}$
- **Claim:** the *corresponding nodes* form an $\boldsymbol{m}$**-clique**
  - There are m of these nodes, each in a different clause
  - None of them represent contradictory truth assignments
  - So, there are edges between all pairs of them → they form an m-clique



18

SHOWING 3-SAT $\leq_P$ CLIQUE

- Let I be an instance of 3-SAT with n variables $x_1 \dots x_n$ and m clauses $C_1 \dots C_m$
  - *E.g.,* $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee x_5) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_5})$
- **Case 2:** Suppose $\boldsymbol{f(I)}$ is a **yes**-instance of m-clique, and show I is a **yes**-instance of **3-SAT**
- Since $f(I)$ is a yes-instance, it contains an m-clique
- Clique contains edges between all pairs of nodes
- There are no edges between nodes in same clause, so clique contains **one node from each clause**
- Set the corresponding **literals** to be **satisfied**
- Clique contains **no edges** between contradictory literals (i.e., no edge connects x_i and $\overline{x_i}$ for any i)
- So, truth assignment is consistent and satisfies each clause (and the formula)



19

LAST STEP: SHOW **CLIQUE IS IN NP**

- YES-certificate: array of k nodes forming a clique
- Verify(I,C):
  - Check certificate is array of length k, containing vertex IDs
  - Check all-to-all edges to verify these vertices form a clique
  - $O(k^2) \subseteq O(|V|^2)$ runtime → polytime
  - **Correctness: exercise!** Need to prove:
    - if I is a yes instance, verify returns yes, and
    - if verify returns yes then I is a yes instance

20

Summary of Polynomial Transformations

SAT ← Every problem in NP can be poly transformed to
↓
3-SAT
↓
Clique

21

Summary of Polynomial Transformations

SAT ← Every problem in NP can be poly transformed to
↓
3-SAT
↓
Clique
↓
Vertex Cover

This additional poly transformation was proved last class (CL to VC)! **We also need to show Vertex Cover is in NP. Exercise. ☺**

22

REDUCING **VERTEX-COVER** TO **SUBSET-SUM**

(Proving Vertex-Cover $\leq_P$ Subset-Sum)

(if we have time)

SUBSET-SUM (SLIGHTLY **DIFFERENT** FROM BEFORE)

Problem 7.18
Subset Sum
Instance: *A list of* **sizes** $S = [s_1, \dots, s_n]$; *and a* **target sum** W. *These are all positive integers.*
Question: *Does there exist a subset* $J \subseteq \{1, \dots, n\}$ *such that* $\sum_{i \in J} s_i = W$?

- Earlier, we defined Subset-Sum with a **target sum of 0**
- Here we add a **target sum T** and take **positive integers** as input

Goal: transform instance I of VC into instance $f(I)$ of SS (in poly time) **such that** I is a yes-instance of VC iff $f(I)$ is a yes-instance of SS	**Idea:** turn **nodes and edges** into a **list of integers** and a **target sum W.** Sum W should be achievable **IFF** there is a k-vertex cover.

Somehow want **the array of integers** to **encode** which edges are covered by various nodes, and **target sum** to **encode** that every edge is covered if W is achieved

Vertex Cover $\le_P$ Subset Sum

Suppose $I = (G, k)$, where $G = (V, E)$, $|V| = n$, $|E| = m$ and $1 \le k \le n$.

Input to Vertex Cover

Suppose $V = \{v_1, \ldots, v_n\}$ and $E = \{e_0, \ldots, e_{m-1}\}$. For $1 \le i \le n$, $0 \le j \le m-1$, let $C = (c_{ij})$, where

$$c_{ij} = \begin{cases} 1 & \text{if } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise.} \end{cases}$$

Sort of like an adjacency matrix, but instead of storing which node-pairs are adjacent, store **which edges are incident to each node**

c_{ij} = is edge j covered by node i?

Vertex Cover $\le_P$ Subset Sum

Suppose $I = (G, k)$, where $G = (V, E)$, $|V| = n$, $|E| = m$ and $1 \le k \le n$.

Suppose $V = \{v_1, \ldots, v_n\}$ and $E = \{e_0, \ldots, e_{m-1}\}$. For $1 \le i \le n$, $0 \le j \le m-1$, let $C = (c_{ij})$, where

$$c_{ij} = \begin{cases} 1 & \text{if } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise.} \end{cases}$$

Define $n + m$ ints and a target sum W as follows:

$$b_j = 10^j \quad (0 \le j \le m-1)$$

Each **edge** becomes a **unique** integer in the array: edge e_j becomes 10^j

E.g.,

$b_0 = 1$
$b_1 = 10$
$b_2 = 100$
$b_3 = 1000$
$b_4 = 10000$

Vertex Cover $\le_P$ Subset Sum

Suppose $I = (G, k)$, where $G = (V, E)$, $|V| = n$, $|E| = m$ and $1 \le k \le n$.

Suppose $V = \{v_1, \ldots, v_n\}$ and $E = \{e_0, \ldots, e_{m-1}\}$. For $1 \le i \le n$, $0 \le j \le m-1$, let $C = (c_{ij})$, where

$$c_{ij} = \begin{cases} 1 & \text{if } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise.} \end{cases}$$

Define $n + m$ ints and a target sum W as follows:

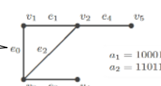
$$a_i = 10^m + \sum_{j=0}^{m-1} c_{ij} 10^j \quad (1 \le i \le n)$$

Each **node** becomes a integer in the array: 10^m + the integers for **all** edges **incident** to the node

$$b_j = 10^j \quad (0 \le j \le m-1)$$

Each **edge** becomes a **unique** integer in the array: edge e_j becomes 10^j

E.g.,



$a_1 = 100011$
$a_2 = 110110$

$b_0 = 1$
$b_1 = 10$
$b_2 = 100$
$b_3 = 1000$
$b_4 = 10000$

$+10^m$ means the integer for a **node** is at least **one digit longer** than the integers for all **edges**

Vertex Cover $\le_P$ Subset Sum

Suppose $I = (G, k)$, where $G = (V, E)$, $|V| = n$, $|E| = m$ and $1 \le k \le n$.

Suppose $V = \{v_1, \ldots, v_n\}$ and $E = \{e_0, \ldots, e_{m-1}\}$. For $1 \le i \le n$, $0 \le j \le m-1$, let $C = (c_{ij})$, where

$$c_{ij} = \begin{cases} 1 & \text{if } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise.} \end{cases}$$

Why twice? If both endpoints of e_j are in the vertex cover, it is counted twice. Otherwise once, and can add b_j.

ints and a target sum W as follows:

$$a_i = 10^m + \sum_{j=0}^{m-1} c_{ij} 10^j \quad (1 \le i \le n)$$

Each **node** becomes a integer in the array: 10^m + the integers for **all** edges **incident** to the node

$$b_j = 10^j \quad (0 \le j \le m-1)$$

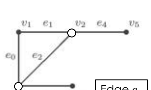
Each **edge** becomes a **unique** integer in the array: edge e_j becomes 10^j

This target weight asks for ***k* nodes** and for **all edges** to be included **twice**

$$W = k \cdot 10^m + \sum_{j=0}^{m-1} 2 \cdot 10^j$$

Then define $f(I) = (a_1, \ldots, a_n, b_0, \ldots, b_{m-1}, W)$.

EXAMPLE



Edge e_4

Edge e_0

Node v_1

Node v_5

$$\begin{array}{cc} & \text{edge} \\ C = \text{node} & \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Sum of edge **sizes** incident to v_1, plus 10^m

Note: no "carrying" can occur even if we sum **everything**

Most significant digit(s) of W accurately capture **# of nodes**

Other digits are in [0,3]. An edge is **definitely covered** by a node if its digit is 2.

$a_1 = 100011$ $b_0 = 1$
$a_2 = 110110$ $b_1 = 10$
$a_3 = 101101$ $b_2 = 100$
$a_4 = 101000$ $b_3 = 1000$
$a_5 = 110000$ $b_4 = 10000$

$$W = 222222 = a_2 + a_3 + b_0 + b_1 + b_3 + b_4$$

Looking for **2** nodes

All 5 edges counted twice

Is there a **2**-VC? Use subset sum to search for $W = \mathbf{2}22222$

Subset sum looks for a subset of $\{a_1, a_2, a_3, a_4, a_5, b_0, b_1, b_2, b_3, b_4\}$ that sums to W

It finds $W = \boldsymbol{a_2} + \boldsymbol{a_3} + b_0 + b_1 + b_3 + b_4$

$\boldsymbol{a_2 + a_3 = 211211}$

Edge e_2 counted twice, other edges once. Sum uses b_0, b_1, b_3, b_4 to get all to be counted twice.

Correctness of the Transformation

<u>Case 1:</u> Suppose I is a **yes**-instance of Vertex-Cover. There is a vertex cover $V' \subseteq V$ such that $|V'| = k$. For $i = 1, 2$, let E^i denote the edges having exactly i vertices in V'. Then $E = E^1 \cup E^2$ because V' is a vertex cover.

one endpoint in V'

both endpoints in V'

Let

$$A' = \{a_i : v_i \in V'\} \quad \text{and} \quad B' = \{b_j : e_j \in E^1\}.$$

Contains **node** ints

Contains **edge** ints

The sum of the ints in A' is

$$k \cdot 10^m + \sum_{\{j : e_j \in E^1\}} 10^j + \sum_{\{j : e_j \in E^2\}} 2 \times 10^j.$$

e_j has **one endpoint** in V', so nodes in V' contribute $\mathbf{1 \times 10^j}$ to W

e_j has **both endpoints** in V', so nodes in V' contribute $\mathbf{2 \times 10^j}$ to W

The sum of the ints in B' is

$$\sum_{\{j : e_j \in E^1\}} 10^j.$$

Add another $\mathbf{1} \times^{\boldsymbol{j}}$ to W for each e_j with **one endpoint** in V'

Therefore the sum of all the chosen ints is

$$k \cdot 10^m + \sum_{\{j : e_j \in E\}} 2 \cdot 10^j = k \cdot 10^m + \sum_{j=1}^{m} 2 \cdot 10^j = W.$$

To get 2×10^j for all e_j, plus 10^m for each node

Case 2: Suppose $f(I)$ is a **yes**-instance of Subset Sum.

- We show I is a **yes**-instance of Vertex-Cover
- Since $f(I)$ is a yes-instance, there exists $A' \cup B'$ that sums to W
  - where A' contains node ints and B' contains edge ints
- Define $V' = \{v_i : a_i \in A'\}$. We claim V' is a vertex cover of size k.
  - We must have $|V'| = k$ for the coefficient of 10^m to be k (no carrying)
  - Suppose (for contra.) V' does **not** cover some edge $e_j = (u, v)$
  - Then the coefficient of 10^j is **zero** for every $a_i \in A'$
  - But the coefficient of 10^j is 2, so a subset of B' must sum to 2×10^j
  - But this is impossible (so e_j is covered, so all edges are covered)

Vertex Cover $\leq_P$ Subset Sum

Complexity of the transformation: Easy! Included for your notes.

Suppose $I = (G, k)$, where $G = (V, E)$, $|V| = n$, $|E| = m$ and $1 \leq k \leq n$.

Suppose $V = \{v_1, \ldots, v_n\}$ and $E = \{e_0, \ldots, e_{m-1}\}$. For $1 \leq i \leq n$, $0 \leq j \leq m-1$, let $C = (c_{ij})$, where

Assume adjacency matrix and unit cost model for simplicity

$$c_{ij} = \begin{cases} 1 & \text{if } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise.} \end{cases}$$

Compute C with trivial algorithm in $O(nm)$ time

Define $n+m$ sizes and a target sum W as follows:

$$a_i = 10^m + \sum_{j=0}^{m-1} c_{ij} 10^j \quad (1 \leq i \leq n)$$

Compute a_i by visiting all incident edges. Trivial algorithm yields $O(m)$ time for each a_i, totaling $O(nm)$ over all i

$$b_j = 10^j \quad (0 \leq j \leq m-1)$$

Trivial to compute all b_j in $O(m)$ time

$$W = k \cdot 10^m + \sum_{j=0}^{m-1} 2 \cdot 10^j$$

Trivial to compute W in $O(m)$ time

Then define $f(I) = (a_1, \ldots, a_n, b_0, \ldots, b_{m-1}, W)$.

Total $O(nm)$ time. This is polynomial in the input graph size!

Summary of Polynomial Transformations

CNF-SAT ← Every problem in NP can be poly transformed to

↓ can be poly transformed to

3-CNF-SAT

↓

Clique

↓

Vertex Cover

↙

Subset Sum

Technically need to also show SubsetSum with target T is in NP (exercise) to know it is in NPC

IS 2-SAT ALSO HARD?

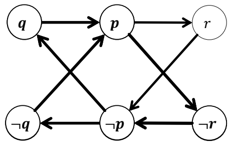
(IF WE HAVE TIME – VERY UNLIKELY)

2-SAT EXAMPLES

- $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg r \vee \neg p)$
  - Satisfiable: $p = 0, q = 1, r \in \{0,1\}$
- $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg r \vee \neg p) \wedge (p \vee \neg q)$

Logical refresher: $p \Rightarrow q$ is **equivalent** to $\neg p \vee q$.

Therefore, $p \vee q$ is **equivalent** to $\neg p \Rightarrow q$ **and equivalent** to $\neg q \Rightarrow p$



Edges (implications of clauses)…

$\neg p \Rightarrow q$	$p \Rightarrow r$	$r \Rightarrow \neg p$	$\neg p \Rightarrow \neg q$
$\neg q \Rightarrow p$	$\neg r \Rightarrow \neg p$	$p \Rightarrow \neg r$	$q \Rightarrow p$

$q \Rightarrow p \Rightarrow \neg r \Rightarrow \neg p \Rightarrow \neg q$ … so q cannot be $true$

$\neg q \Rightarrow p \Rightarrow \neg r \Rightarrow \neg p \Rightarrow q$ … so q cannot be $false$

Therefore the formula **cannot** be satisfied!

(variable names are integers in 1..|X|)

$\boxed{\text{2-SAT}}$ can be solved in polynomial time. Suppose we are given an instance I of $\boxed{\text{2-SAT}}$ on a set of boolean variables $\boxed{X = \{1..|X|\}}$

(1) For every clause $x \vee y$ (where x and y are literals), construct two directed edges $\overline{x}y$ and $\overline{y}x$. We get a directed graph on vertex set $X \cup \overline{X}$.

(2) Determine the strongly connected components of this directed graph.

(3) I is a yes-instance if and only if there is no strongly connected component containing x and $\overline{x}$, for any $x \in X$.

Suppose no variable x is in the same SCC as $\bar{x}$, then to get a satisfying assignment do the following:
For each x, if ∃ path from x to $\bar{x}$, then set $x = false$ else set $x = true$.

BONUS SLIDES

SUMMARY OF COMPLEXITY CLASSES

See this slide's notes

- **P** (Poly-time) — E.g., (**decision** problem variants of:) BFS, Dijkstra's, **some** DP algorithms
  - **Decision** problems that can be solved by algorithms with runtime poly(input size)
- **NP** (Non-deterministic poly-time) — **All of P**, and e.g.,, vertex cover, clique, SAT, subset sum
  - **Decision** problems for which **certificates** can be **verified** in time poly(input size)
  - Equivalently: decision problems that can be solved in poly-time if you have access to a non-deterministic oracle that returns a yes-certificate if one exists
- **NPC** (NP-complete) — E.g., vertex cover, clique, SAT, subset sum, TSP-decision
  - **Decision** problems $\Pi \in \boldsymbol{NP}$ s.t. every $\Pi' \in NP$ can be **transformed** to Π in poly-time
- **NP-hard** (at least as hard as NPC) — **All of NPC**, and e.g., TSP-optimization, TSP-optimal value
  - problems Π s.t. every $\Pi' \in NP$ can be **reduced** to Π in poly-time

- Note: P, NP and NPC problems are **decidable**

Found this neat image online