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

COMPLEXITY CLASS NP-COMPLETE (NPC)
The complexity class NPC denotes the set of all decision problems II that satisfy the following two properties:
1. \( \Pi \in \text{NP} \)
2. For all \( \Pi' \in \text{NP} \), \( \Pi' \leq_p \Pi \).

NPC is an abbreviation for NP-complete.
Note that the definition does not imply that NP-complete problems exist!

MECHANICS OF SHOWING A PROBLEM IS IN NP
1. Define a yes-certificate
2. Design a poly-time \( \text{verify}(I, C) \) algorithm
3. Correctness proof
   - **Case 1**: Let \( I \) be any yes-instance; Find \( C \) such that \( \text{verify}(I, C) = \text{true} \)
   - **Case 2**: Let \( I \) be any no-instance, and \( C \) be any certificate; Prove \( \text{verify}(I, C) = \text{false} \)

POLYNOMIAL TRANSFORMATION FOR PROVING \( \Pi_2 \) IS IN NPC

- Let \( \Pi_1 \) and \( \Pi_2 \) be decision problems
- \( \Pi_1 \leq_p \Pi_2 \) if there exists \( f : \text{domain}(\Pi_1) \to \text{domain}(\Pi_2) \) such that:
  - \( f(I) \) is computable in poly-time, for all \( I \in \text{domain}(\Pi_1) \)
  - If \( I \in \text{Yes}(\Pi_1) \) then \( f(I) \in \text{Yes}(\Pi_2) \)
  - If \( f(I) \in \text{Yes}(\Pi_2) \) then \( I \in \text{Yes}(\Pi_1) \)
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

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

NP-HARDNESS

Intuitively: problems that are at least as hard as NP-complete (but are not necessarily decision problems)
COMPARING NPC AND NP HARD

- \(\Pi \in \text{NPC}\)
  - Must be a decision problem
  - Must poly transform some NPC problem to \(\Pi\)
  - Must show \(\Pi\) in NP
- \(\Pi \in \text{NP-Hard}\)
  - 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)

SOME PROBLEMS IN EACH

ESTABLISHING ANOTHER NPC PROBLEM

SHOWING 3-SAT \(\leq_P\) CLIQUE

- Let \(I\) be an instance of 3-SAT with \(n\) variables \(x_1 \ldots x_n\) and \(m\) clauses \(C_1 \ldots C_m\)
  - E.g., \((x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_3)\) \(\chi = 5, m = 4\)
- We construct CLIQUE input \(f(I) = (G, k)\):
  - Node \(v_i\) for each literal \(1 \leq i \leq 3\) in each clause \(1 \leq c \leq m\) (so \(|V| = 3m\))
  - Edges between all non-contradictory pairs of nodes (no \(x_i \lor \neg x_i\) in different clauses)
  - \(k = m\) (can we find an \(m\)-clique?)
- Must prove this is a polynomial transformation

Showing 3-SAT \(\leq_P\) CLIQUE

- Let \(I\) be an instance of 3-SAT with \(n\) variables \(x_1 \ldots x_n\) and \(m\) clauses \(C_1 \ldots C_m\)
  - E.g., \((x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_3)\)
  - Case 1: Suppose \(I\) is a yes-instance of 3-SAT, and show \(f(I)\) is a yes-instance of \(m\)-clique
    - Since \(I\) is a yes-instance, \(\exists\) a satisfying assignment
  - E.g., \(x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 0\)
  - For each clause \(C_i\), let \(x_i\) be a satisfied literal in \(C_i\)
  - E.g., \(x_1 = x_1, x_2 = x_2, x_3 = x_3, x_4 = x_5\)
  - Claim: the corresponding nodes form an \(m\)-clique
    - There are \(m\) of these nodes, each in a different clause
    - None of them represent contradictory true assignments
    - So, there are edges between all pairs of them \(\rightarrow\) they form an \(m\)-clique
**SHOWING 3-SAT \( \leq_P \) CLIQUE**

Let \( I \) be an instance of 3-SAT with \( n \) variables \( x_1 \ldots x_n \) and \( m \) clauses \( C_1 \ldots C_m \).

E.g., \((x_1 \lor x_2 \lor x_3) \land (x_4 \lor x_5 \lor x_6) \land (x_7 \lor x_8 \lor x_9) \land (x_{10} \lor x_{11} \lor x_{12})\)

**Case 2:** Suppose \( f(I) \) 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 \( \neg x_i \) for any \( i \)).

So, truth assignment is consistent and satisfies each clause (and the formula).

**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 \( \Rightarrow \) 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.

---

**Summary of Polynomial Transformations**

<table>
<thead>
<tr>
<th>SAT</th>
<th>3-SAT</th>
<th>CLIQUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can be poly transformed to</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Subset-Sum (Slightly Different From Before)**

**Problem 7.18**

**Subset Sum**

**Instance:** A list of sizes \( S = [s_1, \ldots, s_n] \); and a target sum \( W \).

These are all positive integers.

**Question:** Does there exist a subset \( J \subseteq \{1, \ldots, n\} \) such that \( \sum_{j \in J} s_j = W \)?

- 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 \( J \) of VC into instance \( f(J) \) of SS (in poly time) such that \( f(J) \) is a yes-instance of VC if \( J \) 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. If 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.
Input to Vertex Cover
Sort of like an adjacency matrix, but instead of storing which node pairs are adjacent, store which edges are incident to each node

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

Each edge becomes a unique integer in the array:

- \( e_j \) becomes \( 10^j \)

Each node becomes a unique integer in the array:

- Contains node int
- contains edge int
- Add another \( 10^m \) for all \( e_j \) with one endpoint in \( V' \)

This target weight asks for \( k \) nodes and all edges to be included twice.

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

EXAMPLE

Suppose \( I \) is a yes instance of Vertex-Cover. There is a vertex cover \( V' \subset V \) such that \( |V'| = k \). For \( i = 1, 2 \), let \( E' \) denote the edges having exactly \( i \) vertices in \( V' \). Then \( E' = E'_{1} \cup E'_{2} \) because \( V' \) is a vertex cover.

Let \( x \) be an int with one endpoint in \( V' \) and \( y \) be an int with both endpoints in \( V' \).

The sum of the int in \( V' \): \( \sum_{(x,y) \in E'_{1}} \) for nodes in \( V' \) contribute

Add another \( 10^m \) for \( e_j \) with one endpoint in \( V' \)

\[ W = k \cdot 10^m + \sum_{(x,y) \in E'_{1}} 2 \cdot 10^m \]

\[ W = k \cdot 10^m + \sum_{(x,y) \in E'_{1}} 2 \cdot 10^m \]

Therefore, the sum of the all chosen ints is

To get \( 2 \cdot 10^m \) for all \( e_j \), plus \( 10^m \) for each node.
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 \( \sum \) where \( A' \) contains node ints and \( B' \) contains edge ints

Define \( V' = \{v_i \mid a_i \in A' \} \). 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 \( e_j \).

Then the coefficient of \( 10^j \) is 0 for every \( a_i \in A' \).

But the coefficient of \( 10^j \) is 2, so a subset of \( B' \) must sum to 2 \( \times 10^j \).

But this is impossible (so \( e_j \) is covered, so all edges are covered)

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

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

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

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

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

Every problem in NP can be poly transformed to can be poly transformed to

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

\[ (p \lor q) \land (\neg p \lor r) \land (\neg r \lor \neg p) \]

Satisfiable: \( p = 0, q = 1, r \in \{0, 1\} \)

Edges (implications of clauses)...

Logical refresher: \( p \lor q \) is equivalent to \( \neg \neg (p \land q) \) and equivalent to \( \neg (\neg p \land \neg q) \)

2-SAT can be solved in polynomial time. Suppose we are given an instance \( I \) on a set of boolean variables \( \{x \mid x \in \{1, \ldots, n\} \} \)

(1) For every clause \( x \lor y \) (where \( x \) and \( y \) are literals), construct two directed edges \( xy \) and \( yx \). 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 \( \overline{x} \), then to get a satisfying assignment do the following:

For each \( x \), if \( \exists \) path from \( x \) to \( \overline{x} \), then set \( x = \text{false} \) else set \( x = \text{true} \).
SUMMARY OF COMPLEXITY CLASSES

**P** (Poly-time)
- **Decision** problems that can be solved by algorithms with runtime poly(input size)
- Example: subset sum, some DP algorithms

**NP** (Non-deterministic poly-time)
- **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
- Example: vertex cover, clique, SAT, subset sum

**NPC** (NP-complete)
- Decision problems \( \Pi \in NP \) s.t. every \( \Pi' \in NP \) can be transformed to \( \Pi \) in poly-time
- Example: vertex cover, clique, SAT, subset sum, TSP-decision

**NP-hard** (at least as hard as NPC)
- Problems \( \Pi \) s.t. every \( \Pi' \in NP \) can be reduced to \( \Pi \) in poly-time
- Example: TSP-optimization, TSP-optimal value

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

Found this neat image online.