Today:
– NP-Completeness
Reading Assignments

• Today’s class:
  – Chapter 34
NP-Completeness

• So far we’ve seen a lot of good news!
  – Such-and-such a problem can be solved quickly
    (i.e., in close to linear time, or at least a time that
    is some small polynomial function of the input
    size)

• NP-completeness is a form of bad news!
  – Evidence that many important problems can not
    be solved quickly.

• NP-complete problems really come up all the time!
Why should we care?

• Knowing that they are hard lets you stop beating your head against a wall trying to solve them...
  – Use a heuristic: come up with a method for solving a reasonable fraction of the common cases.
  – Solve approximately: come up with a solution that you can prove that is close to right.
  – Use an exponential time solution: if you really have to solve the problem exactly and stop worrying about finding a better solution.
Optimization & Decision Problems

• Decision problems
  – Given an input and a question regarding a problem, determine if the answer is yes or no

• Optimization problems
  – Find a solution with the “best” value

• Optimization problems can be cast as decision problems that are easier to study
  – *E.g.*: Shortest path: $G =$ unweighted directed graph
    • Find a path between $u$ and $v$ that uses the fewest edges
    • Does a path exist from $u$ to $v$ consisting of at most $k$ edges?
Algorithmic vs Problem Complexity

• The *algorithmic complexity* of a computation is some measure of how *difficult* is to perform the computation (i.e., specific to an algorithm).

• The *complexity of a computational problem* or task is the complexity of the algorithm with the *lowest* order of growth of complexity for solving that problem or performing that task.
  – *e.g.*, the problem of searching an ordered list has *at most* $\log n$ time complexity.

• **Computational Complexity**: deals with classifying problems by how hard they are.
Class of “P” Problems

- **Class P** consists of (decision) problems that are solvable in polynomial time.

- Polynomial-time algorithms
  - Worst-case running time is $O(n^k)$, for some constant $k$.

- Examples of polynomial time:
  - $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$

- Examples of non-polynomial time:
  - $O(2^n)$, $O(n^n)$, $O(n!)$
Tractable/Intractable Problems

• Problems in P are also called **tractable**

• Problems **not** in P are **intractable** or **unsolvable**
  – Can be solved in reasonable time only for small inputs
  – Or, can not be solved at all

• Are non-polynomial algorithms always worst than polynomial algorithms?
  - $n^{1,000,000}$ is *technically* tractable, but really impossible
  - $n^{\log \log \log n}$ is *technically* intractable, but easy
Example of Unsolvable Problem

• Turing discovered in the 1930’s that there are problems **unsolvable** by *any* algorithm.

• The most famous of them is the **halting problem**
  – Given an arbitrary algorithm and its input, will that algorithm eventually halt, or will it continue forever in an “**infinite loop**?”
Examples of Intractable Problems

Hamiltonian Paths

*Optimization Problem*: Given a graph, find a path that passes through every vertex exactly once

*Decision Problem*: Does a given graph have a Hamiltonian Path?

Traveling Salesman

*Optimization Problem*: Find a minimum weight Hamiltonian Path

*Decision Problem*: Given a graph and an integer $k$, is there a Hamiltonian Path with a total weight at most $k$?
Intractable Problems

• Can be classified in various categories based on their degree of difficulty, e.g.,
  – NP
  – NP-complete
  – NP-hard

• Let’s define NP algorithms and NP problems ...
Nondeterministic and NP Algorithms

Nondeterministic algorithm = two stage procedure:

1) Nondeterministic ("guessing") stage:
   generate randomly an arbitrary string that can be thought of as a candidate solution ("certificate")

2) Deterministic ("verification") stage:
   take the certificate and the instance to the problem and returns YES if the certificate represents a solution

NP algorithms (Nondeterministic polynomial)
verification stage is polynomial
Class of “NP” Problems

- **Class NP** consists of problems that could be solved by NP algorithms
  - i.e., verifiable in polynomial time
- If we were given a “certificate” of a solution, we could verify that the certificate is correct in time polynomial to the size of the input
- **Warning:** NP does **not** mean “non-polynomial”
**E.g.:** Hamiltonian Cycle

- **Given:** a directed graph $G = (V, E)$, determine a simple cycle that contains each vertex in $V$
  - Each vertex can only be visited once

- **Certificate:**
  - Sequence: $\langle v_1, v_2, v_3, \ldots, v_{|V|} \rangle$

![Diagram of a Hamiltonian Cycle](image1)

![Diagram of a non-Hamiltonian Cycle](image2)
Is $P = NP$?

• Any problem in $P$ is also in $NP$:

$$P \subseteq NP$$

• The big (and open question) is whether $NP \subseteq P$ or $P = NP$
  – i.e., if it is always easy to check a solution, should it also be easy to find a solution?

• Most computer scientists believe that this is false but we do not have a proof ...
NP-Completeness (informally)

- **NP-complete** problems are defined as the hardest problems in NP
- Most practical problems turn out to be either P or NP-complete.
- Study NP-complete problems ...
Reductions

• Reduction is a way of saying that one problem is “easier” than another.

• We say that problem A is easier than problem B, (i.e., we write “$A \leq B$”) if we can solve A using the algorithm that solves B.

• Idea: transform the inputs of A to inputs of B
Polynomial Reductions

- Given two problems A, B, we say that A is polynomially \textit{reducible} to B (A \leq_p B) if:

1. There exists a function $f$ that converts the input of A to inputs of B in polynomial time

2. $A(i) = \text{YES} \iff B(f(i)) = \text{YES}$
NP-Completeness (formally)

• A problem B is **NP-complete** if:
  
  1. $B \in \text{NP}$
  
  2. $A \leq_p B$ for all $A \in \text{NP}$

• If B satisfies only property (2) we say that B is **NP-hard**

• No polynomial time algorithm has been discovered for an **NP-Complete** problem

• No one has ever proven that no polynomial time algorithm can exist for any **NP-Complete** problem
Implications of Reduction

- If $A \leq_p B$ and $B \in P$, then $A \in P$
- if $A \leq_p B$ and $A \notin P$, then $B \notin P$
Proving Polynomial Time

1. Use a **polynomial time** reduction algorithm to transform A into B
2. Run a known **polynomial time** algorithm for B
3. Use the answer for B as the answer for A
Proving NP-Completeness In Practice

• Prove that the problem B is in NP
  – A randomly generated string can be checked in polynomial time to determine if it represents a solution

• Show that one known NP-Complete problem can be transformed to B in polynomial time
  – No need to check that all NP-Complete problems are reducible to B
Revisit “Is P = NP?”

Theorem: If any NP-Complete problem can be solved in polynomial time \( \Rightarrow \) then \( P = NP \).
P & NP-Complete Problems

• **Shortest simple path**
  – Given a graph $G = (V, E)$ find a **shortest** path from a source to all other vertices
  – **Polynomial solution:** $O(VE)$

• **Longest simple path**
  – Given a graph $G = (V, E)$ find a **longest** path from a source to all other vertices
  – **NP-complete**
P & NP-Complete Problems

• **Euler tour**
  
  – \( G = (V, E) \) a connected, directed graph find a cycle that traverses each edge of \( G \) exactly once (may visit a vertex multiple times)
  
  – **Polynomial solution** \( O(E) \)

• **Hamiltonian cycle**
  
  – \( G = (V, E) \) a connected, directed graph find a cycle that visits each vertex of \( G \) exactly once
  
  – **NP-complete**
Once one problem (SAT) shown to be NP-Complete, can show many others...

Example reductions (From CLRS, Ch. 34):

- CIRCUIT-SAT
- SAT
- 3-CNF-SAT
- CLIQUE
- SUBSET-SUM
- VERTEX-COVER
- HAM-CYCLE
- TSP
NP-Complete Problem: Circuit-SAT

• *Circuit-SAT problem*: Given a boolean combinational circuit, determine if there is a satisfying assignment to inputs such that the circuit’s output is 1.
Example Circuit-SAT

• Is this circuit satisfiable?
Reductions in CLRS...
NP-Complete Problem: Satisfiability (SAT)

- **Satisfiability problem**: Given a logical expression $\Phi$, find an assignment of values (F, T) to variables $x_i$ that causes $\Phi$ to evaluate to T:

  $$\Phi = x_1 \lor \neg x_2 \land x_3 \lor \neg x_4$$

- SAT was the historically first problem shown to be NP-complete

- Proof required showing property 2 of the NP-completeness definition:

  A problem $B$ is **NP-complete** if:

  1. $B \in \text{NP}$
  2. $A \leq_p B$ for all $A \in \text{NP}$

- Required creativity!
Reductions in CLRS...
NP-Complete Problem: CNF Satisfiability

• CNF is a special case of SAT
• \( \Phi \) is in “Conjunctive Normal Form” (CNF)
  – “AND” of expressions (i.e., clauses)
  – Each clause contains only “OR”s of the variables and their complements

\[ \text{E.g.: } \Phi = (x_1 \lor x_2) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor \neg x_2) \]

clauses
NP-Complete Problem:  
**3-CNF Satisfiability**

A subcase of CNF problem:
- Contains three clauses

- **E.g.:**
  \[
  \Phi = (x_1 \lor \neg x_1 \lor \neg x_2) \land (x_3 \lor x_2 \lor x_4) \land \\
  (\neg x_1 \lor \neg x_3 \lor \neg x_4)
  \]

- **3-CNF** is NP-Complete

- Interestingly enough, **2-CNF** is in P!
Reductions in CLRS...
NP-Complete Problem: Clique

Clique Problem:
- Undirected graph $G = (V, E)$
- **Clique**: a subset of vertices in $V$ all connected to each other by edges in $E$ (i.e., forming a complete graph)
- **Size of a clique**: number of vertices it contains

Optimization problem:
- Find a clique of maximum size

Decision problem:
- Does $G$ have a clique of size $k$?

Examples:
- $\text{Clique}(G, 2) = \text{YES}$
- $\text{Clique}(G, 3) = \text{NO}$
- $\text{Clique}(G, 3) = \text{YES}$
- $\text{Clique}(G, 4) = \text{NO}$
Clique Verifier

• **Given**: an undirected graph $G = (V, E)$

• **Problem**: Does $G$ have a clique of size $k$?

• **Certificate**: 
  – A set of $k$ nodes

• **Verifier**: 
  – Verify that for all pairs of vertices in this set there exists an edge in $E$
Clique example

• What is the maximum clique here?

![Clique graph]

The maximum clique in this graph includes nodes 1, 2, 3, 4, 5, 6, and 7.
Reductions in CLRS...
Vertex Cover Example

• What is the minimum vertex cover here?

[Diagram of a vertex cover example with vertices 1 to 6 and edges between them]
NP-Complete Problem: Vertex Cover

Vertex Cover Problem:
- Undirected graph $G = (V, E)$
- **Vertex cover**: a subset $V' \subseteq V$ such that each edge in the graph is covered by some vertex in $V'$ (i.e., if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both.)
- **Size of a VC**: number of vertices it contains

Optimization problem:
- Find a VC of maximum size

Decision problem:
- Does $G$ have a VC of size $k$?

$VC(G, 2) = YES$
$VC(G, 1) = NO$
Reductions in CLRS...
NP-Complete Problem: 
Subset Sum

Subset Sum Problem:
— Given finite set $S$ of positive integers and integer target $t > 0$.
— Is there a subset $S' \subseteq S$ such that $t = \sum_{s \in S'} s$

Example:
$S = \{1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993\}$

$T = 138457$

Answer is yes, for $S' = \{1, 2, 7, 98, 343, 686, 2409, 17206, 117705\}$
Reading Assignments

• Next class:
  – Chapter 34.3
    • Looking more deeply at reductions