Algorithm Design

- Formulate the problem precisely
- Design an algorithm
- Prove correctness
- Analyze running time

Sometimes you can’t find an efficient algorithm.

Example: Network Design

- Input: undirected graph $G = (V, E)$ with edge costs
- Minimum spanning tree problem: find min-cost subset of edges so there is a path between any $u, v \in V$.
  > $O(m \log n)$ greedy algorithm
- Minimum Steiner tree problem: find min-cost subset of edges so there is a path between any $u, v \in W$ for specified terminal set $W$.
  > No polynomial-time algorithm is known.

Example: Knapsack Problem

- Input: $n$ items with costs and weights, capacity $W$
- Goal: select items to maximize total cost without exceeding $W$
- Fractional knapsack: select fraction in $[0, 1]$ of each item
  > $O(n \log n)$ greedy algorithm
- 0-1 Knapsack: select all or none of each item
  > $O(nW)$ pseudo-polynomial time algorithm
  > No polynomial time algorithm known!

Tractability

- Working definition of efficient: polynomial time
  > $O(n^d)$ for some $d$.
- Huge class of natural and interesting problems for which
  > We don’t know any polynomial time algorithm
  > We can’t prove that none exists
- Goal: develop mathematical tools to say when a problem is hard or “intractable”

Preview of Landscape: Classes of Problems

- $P$: solvable in polynomial time
- $NP$: includes most problems we don’t know about
- $EXP$: solvable in exponential time
NP-Completeness

- **NP-complete**: problems that are “as hard as” every other problem in NP.
- A polynomial time algorithm for any NP-complete problem implies one for every problem in NP.

---

P ≠ NP?

Two possibilities:

- We don’t know which is true, but think P ≠ NP
- $1M prize if you can find out (Clay Institute Millenium Problems)

---

Outline

- **Goal**: develop technical tools to make this precise
- **Polynomial-time reductions**: what it means for one problem to be “as hard as” another
- **Define NP**: characterize mystery problems
- **NP-completeness**: some problems in NP are “as hard as” all others

---

Polynomial-Time Reduction

- Problem Y is **polynomial-time reducible** to Problem X
  - solveY(yInput)
  - Construct xInput // poly-time
  - foo = solveX(xInput) // poly # of calls
  - return yes/no based on foo // poly-time

- ...if any instance of Problem Y can be solved using
  1. A polynomial number of standard computational steps
  2. A polynomial number of calls to a black box that solves problem X

- **Notation** Y ≤P X

---

First Reduction: Independent Set and Vertex Cover

Given a graph G = (V, E),

1. If Y ≤P X and X ∈ P, then Y ∈ P
2. If Y ≤P X and Y ∉ P then X ∉ P

- 1: design algorithms, 2: prove hardness

---

First Reduction: Independent Set and Vertex Cover

- S ⊆ V is an **independent set** if no nodes in S share an edge.
  - Examples: \{3, 4, 5\}, \{1, 4, 5, 6\}
- S ⊆ V is a **vertex cover** if every edge has at least one endpoint in S.
  - Examples: \{1, 2, 6, 7\}, \{2, 3, 7\}

**IND. SET.** Does G have independent set of size at least k?

**VERTEX COVER.** Does G have a vertex cover of size at most k?
## Independent Set and Vertex Cover

- **Claim:** \( S \) is independent set if and only if \( V - S \) is a vertex cover.

1. \( S \) independent set \( \Rightarrow \) \( V - S \) vertex cover
   - Consider any edge \((u, v)\)
   - \( S \) independent \( \Rightarrow \) either \( u \notin S \) or \( v \notin S \)
   - I.e., either \( u \in V - S \) or \( v \in V - S \)
   - \( \Rightarrow V - S \) is a vertex cover
2. \( V - S \) vertex cover \( \Rightarrow \) \( S \) independent set
   - Similar.

## Vertex Cover \( \leq P \) Independent Set

- **Claim:** **Vertecx Cover** \( \leq P \) **Independent Set**
- **Reduction:**
  - On **Independent Set** instance \( \langle G, k \rangle \)
  - Construct **Independent Set** instance \( \langle G, n - k \rangle \)
  - Return Yes if \( \text{solveIS}(\langle G, n - k \rangle) = \text{Yes} \)

- **Correctness** for Yes output:
  - Suppose \( G \) has independent set \( S \) with \( \geq k \) nodes
  - Then \( T = V - S \) is a vertex cover with \( \leq n - k \) nodes
  - The algorithm correctly outputs Yes

- **Correctness** for No output:
  - Suppose \( G \) has no independent set \( S \) with \( \geq k \) nodes
  - Then there is no vertex cover with \( T \) with \( \leq n - k \) nodes,
    otherwise \( S = V - T \) is an independent set with \( \geq k \) nodes.
  - The algorithm correctly outputs No

## Aside: Decision versus Optimization

- For intractability and reductions we will focus on decision problems (Yes/No answers)
- Algorithms have typically been for optimization (find biggest/smallest)
- Can reduce optimization to decision and vice versa. Discuss.

## Reduction Strategies

- Reduction by equivalence (Vertex Cover and Independent Set)
- Reduction to a more general case
- Reduction by “gadgets”

## Reduction to General Case: Set Cover

**Problem.** Given a set \( U \) of \( n \) elements, subsets \( S_1, \ldots, S_m \subset U \), and a number \( k \), does there exist a collection of at most \( k \) subsets \( S_i \) whose union is \( U \)?

- **Example:** \( U = \{ A, B, C, D, E \} \) is the set of all skills, there are five people with skill sets:
  \[
  S_1 = \{ A, C \}, \quad S_2 = \{ B, E \}, \quad S_3 = \{ A, C, E \}
  \]
  \[
  S_4 = \{ D \}, \quad S_5 = \{ B, C, E \}
  \]
  - Find a small team that has all skills, \( S_1, S_4, S_5 \)

**Theorem.** **Vertex Cover** \( \leq P \) **Set Cover**
Reduction of Vertex Cover to Set Cover

Reduction.
- Given Vertex Cover instance \((G, k)\)
- Construct Set Cover instance \((U, S_1, \ldots, S_m, k)\) with \(U = E\) and \(S_v = \) the set of edges incident to \(v\)
- Return \text{Yes} iff \(\text{solveSC}((U, S_1, \ldots, S_m, k)) = \text{Yes}\)

Proof
- Straightforward to see that \(S_v, \ldots, S_v\ell\) is a set cover of size \(\ell\) if and only if \(v_1, \ldots, v_\ell\) is a vertex cover of size \(\ell\)
- This implies the algorithm correctly outputs \text{Yes} if \(G\) has a vertex cover of size \(\leq k\) and \text{No} otherwise
- Polynomial \# of steps outside of \text{solveSC}
- Only one call to \text{solveSC}

A Bad Reduction

Reduction.
- Given Vertex Cover instance \((G, k)\)
- Construct Set Cover instance \((U, S_0, S_1, \ldots, S_m, k)\) as before but with additional set \(S_0 = U\)
- Return \text{Yes} iff \(\text{solveSC}((U, S_0, S_1, \ldots, S_m, k)) = \text{Yes}\)

Analysis
- \textbf{“Yes” instance:} \(G\) has a vertex cover of size \(\leq k\)
  - \(U\) has a set cover of size \(\leq k\)
  - Output is \text{Yes}—correct
- \textbf{“No” instance:} \(G\) does not have a vertex cover of size \(\leq k\)
  - \(U\) does have a set cover of size \(\leq k\) for \(k \geq 1\)
  - Output is \text{Yes}—incorrect