Reductions

Linear time reductions
Polynomial time reductions
NP-completeness

Reduction
Problem X reduces to problem Y if given a subroutine for Y, can solve X.

- Cost of solving X = cost of solving Y + cost of reduction.
- May call subroutine for Y more than once.

Ex: X = baseball elimination, Y = max flow.

Consequences:
- Establish relative difficulty between two problems. (classify problems)
- Given algorithm for Y, can also solve X. (design algorithms)
- If X is hard, then so is Y. (establish intractability)

Linear Time Reductions

Problem X linear reduces to problem Y if X can be solved with:
- Linear number of standard computational steps.
- One call to subroutine for Y.
- Notation: \( X \leq_L Y \).

Examples we’ve already seen in the course.
- Removing duplicates reduces to sorting.
- Voronoi diagram reduces to Delaunay triangulation.
- Arbitrage reduces to negative cycle detection.
- Bipartite matching reduces to max flow.
- Brewer’s problem reduces to linear programming.

Other most common type of reduction.
- X polynomial reduces to Y.
- Stay tuned for NP-completeness.

PRIME: Given (the decimal representation of) an integer x, is x prime?

COMPOSITE: Given an integer x, does x have a nontrivial factor?

FACTOR: Given two integers x and y, does x have a nontrivial factor less than y?

Claim. COMPOSITE \( \leq_L \) PRIME.

```
composite (x)
if (prime(x)) return false;
else return true;
```
Linear Time Reduction: Examples

**PRIME**: Given (the decimal representation of) an integer $x$, is $x$ prime?

**COMPOSITE**: Given an integer $x$, does $x$ have a nontrivial factor?

**FACTOR**: Given two integers $x$ and $y$, does $x$ have a nontrivial factor less than $y$?

other than 1 and $x$

Claim. $\text{PRIME} \leq_L \text{COMPOSITE}$.

```plaintext
prime (x)
if (composite(x)) return false;
else         return true;
```

**Problem Equivalence**

Tool for classifying problems.

- Equivalence: If $X \leq_L Y$ and $Y \leq_L X$ then we write $X \equiv_L Y$.
  - given any algorithm for $X$, can solve $Y$ in same running time, and vice versa
- Transitivity: if $X \leq_L Y$ and $Y \leq_L Z$ then $X \leq_L Z$.

Equivalence: $\text{PRIME} \leq_L \text{COMPOSITE}$ and $\text{COMPOSITE} \leq_L \text{PRIME}$.

Transitivity: $\text{PRIME} \leq_L \text{COMPOSITE} \leq_L \text{FACTOR}$.

**Primality Testing and Factoring**

We established: $\text{PRIME} \leq_L \text{FACTOR}$.

Natural question: Does $\text{FACTOR} \leq_L \text{PRIME}$?

- Consensus opinion = no.

State-of-the-art.

- $\text{PRIME}$ is in $P$.
- $\text{FACTOR}$ not believed to be in $P$.

RSA cryptosystem.

- Based on dichotomy between two problems.
- To use RSA, must generate large primes efficiently.
- Can break RSA with efficient factoring algorithm.
Reduction Gone Wrong

Caveat.
- System designer specs the interfaces for project.
- One programmer might implement `isComposite` using `isPrime`.
- Another programmer might implement `isPrime` using `isComposite`.
- Be careful to avoid infinite reduction loops in practice.

```java
public static boolean isComposite(int x) {
    if (isPrime(x)) return false;
    else return true;
}
```

```java
public static boolean isPrime(int x) {
    if (isComposite(x)) return false;
    else return true;
}
```

Undirected Shortest Path Reduces to Directed Shortest Path

Undirected shortest path (with nonnegative weights) linearly reduces to directed shortest path.
- Replace each directed arc by two undirected arcs.
- Shortest directed path will use each edge at most once.

Shortest Path with Negative Costs

Caveat: Reduction invalid in networks with negative cost arcs, even if no negative cycles.

```
5 7 3 9
5
```

Remark: can still solve shortest path problem in undirected graphs if no negative cycles, but need more sophisticated techniques.
- Reduce to weighted non-bipartite matching. (!)

Reduction: Min Cut Reduces to Max Flow

Max-flow min-cut theorem says value of max flow = capacity of min cut.

Min cut linear reduces to max flow.
- Given a max flow, find all vertices reachable from source in residual graph to get min cut.

Does max flow linear reduce to min cut?
- Apparently no easy way to determine max flow from min cut.
- But no better way known to compute a min cut than via max flow.
### Integer Arithmetic

**Integer multiplication:** given two N-digit integer s and t, compute $s \times t$.

**Integer division:** given two integers s and t of at most N digits each, compute the quotient $q = \lfloor s / t \rfloor$ and remainder $r = s \mod t$.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Grade School</th>
<th>Best Known Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>$O(N)$</td>
<td>$\Omega(N)$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$O(N^2)$</td>
<td>$\Omega(N)$</td>
</tr>
<tr>
<td>Division</td>
<td>$O(N^2)$</td>
<td>$\Omega(N)$</td>
</tr>
</tbody>
</table>

**Fundamental questions.**
- Is multiplication easier than division?
- Is addition easier than multiplication?
- Is division easier than multiplication?

### Sorting and Convex Hull

**Sorting.** Given N distinct integers, rearrange in increasing order.

**Convex hull.** Given N points in the plane, find their convex hull in counter-clockwise order.

**Lower bounds.**
- Recall, under comparison-based model of computation, sorting N items requires $\Omega(N \log N)$ comparisons.
- We show sorting linearly reduces to convex hull.
- Hence, finding convex hull of N points requires $\Omega(N \log N)$ "comparisons" where comparison means ccw.
### Sorting Reduces to Convex Hull

**Sorting instance (integers):**
\[ x_1, x_2, \ldots, x_N \]

**Convex hull instance:**
\[ (x_1, x_1^2), (x_2, x_2^2), \ldots, (x_N, x_N^2) \]

**Key observation.**
- Region \( \{ x : x^2 \geq x \} \) is convex \( \Rightarrow \) all points are on hull.
- Starting at point with most negative \( x \), counter-clockwise order of convex hull yields items in sorted order.

### 3-SUM Reduces to 3-COLLINEAR

**Claim.** If \( a, b, \) and \( c \) are distinct then \( a + b + c = 0 \) if and only if \( (a, a^3) \), \( (b, b^3) \), and \( (c, c^3) \) are collinear.

**Proof.** Necessary and sufficient conditions for two line segments to be equal.
\[
\frac{a^3-b^3}{a-b} = \frac{b^3-c^3}{b-c} \iff \frac{(a-b)(a^2+ab+b^2)}{a-b} = \frac{(b-c)(b^2+bc+c^2)}{b-c} \\
\iff \frac{c^2+bc-a^2-ab}{b-c} = \frac{(c-a)(c+a+b)}{b-c} \\
\iff c = a \quad \text{or} \quad a + b + c = 0
\]

### 3-SUM Reduces to 3-COLLINEAR

**3-SUM:** Given \( N \) distinct integers \( x_1, x_2, \ldots, x_N \), are there 3 distinct integers \( x_i, x_j, x_k \) such that \( x_i + x_j + x_k = 0 \)?

**3-COLLINEAR:** Given \( N \) distinct points \( (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N) \), are there 3 points that all lie on the same line?

**Conjecture:** Any algorithm for 3-SUM requires \( \Omega(N^2) \) time.

**Claim.** \( 3\text{-SUM} \leq_p 3\text{-COLLINEAR} \).

**Corollary.** Unless you can solve 3-SUM is sub-quadratic time, any algorithm for 3-COLLINEAR requires \( \Omega(N^2) \) time.

**Reduction.** To determine if there is a solution to 3-SUM instance \( x_1, x_2, \ldots, x_N \), determine if there is a solution to 3-COLLINEAR instance with \( (x_1, x_1^3), (x_2, x_2^3), \ldots, (x_N, x_N^3) \).

### Polynomial-Time Reduction

**X polynomial reduces to Y if X can be solved using:**
- Polynomial number of standard computational steps.
- Polynomial number of calls to subroutine for Y.
- Notation: \( X \leq_p Y \).

**Alternate viewpoint.** Can solve \( X \) in polynomial time given special piece of hardware that solves instances of \( Y \) in a single step.

**Ex:** Baseball elimination reduces to max flow.
- Solve \( N \) max flow problems on a graph with \( N^2 \) vertices.

**Remark 1:** If \( X \leq_L Y \) then \( X \leq_p Y \).

**Remark 2:** If \( X \) can be solved in polynomial time, then \( X \leq_p Y \) for any \( Y \).
Polynomial-Time Reduction

Goal: classify and separate problems according to relative difficulty.
- Those that can be solved in polynomial time.
- Those that (probably) require exponential time.

Establish tractability. If $X \leq_P Y$ and $Y$ can be solved in polynomial-time, then $X$ can be solved in polynomial time.

Establish intractability. If $X \leq_P Y$ and $X$ cannot be solved in polynomial-time, then $Y$ cannot be solved in polynomial time.

Hamilton Path

HAMILTON-PATH. Given an undirected graph, is there a path that visits every vertex exactly once?

EULER-PATH. Given an undirected graph, is there a path that visits every edge exactly once?

Hamilton Path Reduces to Shortest Path

Claim. HAMILTON-PATH $\leq_P$ SHORTEST-PATH.

For each undirected edge, make two directed edges of weight $-1$.
For all pairs of vertices $v$ and $w$, find shortest path from $v$ to $w$.
If shortest path has length $-(V-1)$ then this is a Hamilton path.

Hamilton Path Reduces to Shortest Path

Claim. HAMILTON-PATH $\leq_P$ SHORTEST-PATH.

Conjecture. No polynomial algorithm exists for HAMILTON-PATH.

Corollary. Polynomial algorithm for SHORTEST-PATH is unlikely.

- This explains why we needed the "no negative cycles" assumption for shortest path algorithms.

<table>
<thead>
<tr>
<th>Shortest Path</th>
<th>Algorithm</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonnegative weights</td>
<td>Dijkstra</td>
<td>$E \log V$</td>
</tr>
<tr>
<td>No negative cycles</td>
<td>Bellman-Ford</td>
<td>$E V$</td>
</tr>
<tr>
<td>Arbitrary weights</td>
<td>Brute force</td>
<td>$2^V$</td>
</tr>
</tbody>
</table>
Subset Sum Reduces To Integer Programming

SUBSET-SUM. Given $N$ integers $a_1, a_2, \ldots, a_N$, and another integer $b$, is there a subset of integers that sums to exactly $b$?

Integer programming. Given integers $b_i, a_{ij}$ find 0/1 variables $x_j$ that satisfy a linear system of equations.

$$\sum_{j=1}^{N} a_{ij} x_j = b_i \quad 1 \leq i \leq M$$
$$x_j \in \{0, 1\} \quad 1 \leq j \leq N$$

SUBSET-SUM polynomial reduces to IP. Solve integer program below and select subset of indices with $x_j = 1$.

$$\sum_{j=1}^{N} a_{ij} x_j = b \quad 1 \leq j \leq N$$

NP-Completeness

$P$. Set of all decision problems solvable in polynomial time on a deterministic Turing machine.

$NP$. Set of all decision problems solvable in polynomial time on a nondeterministic Turing machine.

NP-complete. Decision problem $X$ is NP-complete if every problem in NP polynomial reduces to $X$.

Cook’s theorem. CNF-SAT is NP-complete.

Corollary. If $P \neq NP$, then no polynomial algorithm for CNF-SAT.

Practical consequence. If $P \neq NP$, then can’t hope to design polynomial algorithm for any problem on the previous slide.
Summary

Reductions are important in theory to:
- Classify problems according to their computational requirements.
- Establish intractability.
- Establish tractability.

Reductions are important in practice to:
- Design algorithms.
- Design reusable software modules.
  - sorting, priority queue, symbol table, graph, shortest path, max flow, regular expressions, linear programming
- Determine difficulty of your problem and choose the right tool.
  - use exact algorithm for tractable problems
  - use heuristics for NP-hard problems (e.g., bin packing)