Reminders

NP : Measuring Complexity

Recap

Last time

- Problem reduction (Cook): $X \leq_p Y$ if $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of $Y$ in constant time. ($X$ and $Y$ are problems not algorithms!)
  - Running time of algorithm for $X$ includes time required to compute input for calls to algorithm solving $Y$.
- Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$
- Examples: $3SAT \leq_p INDSET \leq_p VERTEXCOVER \leq_p SETCOVER$ (and $VC \leq_p INDSET$)
- Strategies for reductions
  - Direct equivalence ($INDSET \equiv_p VERTEXCOVER$)
  - Special Case: ($VERTEXCOVER \leq_p SETCOVER$)
  - Gadget Building: ($3SAT \leq_p INDSET$)
- Problem Characteristics
  - Decision Problems: Output is YES/NO
  - Certifiability: If answer is YES, there’s corroborating evidence (a ”proof”) of answer

Goal: Formalize concepts

- Decision Problem
- Polynomial Time Algorithm
- Certifiability
- Polynomial Time Certifiable
- A First Problem Hierarchy

Decision Problems

- Problem: Given set $X$ of strings, is string $s \in X$?
- Problem Size: $|s|$ - length of string $s$
- Algorithm $A$ solves $X$ if $A(s) = YES$ iff $s \in S$
- Examples
  - Does integer $a$ divide integer $b$?
  - Does bipartite graph $G$ contain a perfect matching?
– Does flow network $G$ have a flow of value at least $k$?
– Does $G$ contain an independent set of size at least $k$?
– Is CNF proposition $\Phi$ satisfiable?

**Polynomial Time Algorithm**

- $A$ solves $X$ in polynomial time if $A(s)$ executes at most $O(p(|s|))$ operations, for some polynomial $p()$
- Examples
  - Does integer $a$ divide integer $b$?
  - Does bipartite graph $G$ contain a perfect matching?
  - Does flow network $G$ have a flow of value at least $k$?

**Certifiability**

Given an instance of problem "Does $a$ divide $b"$, if answer is "yes", the integer $c = b/a$ can be used as a "certificate" of the correctness of the answer "yes": just check that $b = ac$

Note: Checking the certificate requires its own algorithm!

**Definition 1 (Certifier).** An algorithm $C(s, t)$, where $s$ and $t$ are strings, is a certifier for decision problem $X$ if for every $s$

$s \in X$ if and only if there is some string $t_s$ such that $C(s, t_s)$ returns "yes".

The string $t_s$ is called a certificate for $s$.

Examples
- For Perfect Matching Problem, $t_s$ is a perfect matching (if there is one)
- For Network Flow Problem, $t_s$ is a flow $f$ having $v(f) = k$
- For INDSET, $t_s$ is an independent set

**NP : Polynomial Time Certifiability**

**Definition 2.** A certifier $C(s, t)$ for decision problem $X$ is a polynomial-time certifier if

- $|t_s| \leq p(|s|)$ for some polynomial $p()$ (the certifier $t_s$ is not too big!)
- $C(s, t_s)$ runs in time $q(s)$ for some polynomial $q()$

**Definition 3.** $NP = \{ X : \text{There is a polynomial-time certifier } C(s, t) \text{ for } X \}$

NP stands for Non-deterministic Polynomial

Another way to think about $NP$: I can take polynomially much time to guess a $t_s$ then polynomially much time to certify $s$ using $t_s$.

All of 3SAT, INDSET, VERTEX COVER, SETCOVER are in NP
A First Problem Hierarchy

**Definition 4.** $P = \{ X : \text{There is a polynomial-time algorithm } A() \text{ that decides } X \}$

**Theorem 1.** $P \subseteq NP$

Proof. Let $A()$ be a polynomial time algorithm to decide $X \in P$.

- Let $C(s, t) = A(s)$ and $t_s = \emptyset$
- Then $C(s, t_s)$ runs in time polynomial in $|s|$
- So $X$ is in $NP$

**Definition 5.** $EXP = \{ X : \text{There is an exponential-time algorithm } A() \text{ that decides } X \}$

*Exponential time* means time $f(n) \in \Theta(2^{p(n)})$ for some polynomial $p()$

**Theorem 2.** $NP \subseteq EXP$

Proof. Let $X \in NP$. Create an algorithm $A()$ in EXP to decide $X$

- Let $C(s, t)$ be a poly-time certifier for $X$.
- There is a polynomial $p()$ such that $|t_s| \leq p(|s|)$.
- There is a polynomial $q()$ such that $C(s, t_s)$ executes at most $q(s)$ steps. Here’s $A()$
- Forall $t$ (by size of $t$)
  - Run up to $q(s)$ steps of $C(s, t)$. If $C(s, t)$ returns "yes" then return "yes"
  - If no $t$ returns "yes", return "no"
- Claim: $A() \in EXP$, since it takes time $O(c^{p(|s|)})$ times the running time of $C(s, t)$, where $c$ is the size of the alphabet used to describe $s$ and $t$.

**Theorem 3.** $P \subset EXP$

Consequence of *Time Hierarchy Theorem*

The Big Question

Is $P = NP$?
- Consensus view is "no"
- Most fundamental problem in computer science
- Win $1M$: Clay Foundation offers $1,000,000$ prize for solution

But what about optimization problems? Why focus on decision problems?
- Easy to reason about.
- Can often reduce optimization problem to decision problem
• Example: Vertex Cover: Find size of largest vertex cover in graph $G$
  – Let $X$ be the decision version: Is there a vertex cover of size $k$?
  – Let $A()$ solve $X$
  – Run $A()$ multiple times, ”binary-searching” for smallest $k' \leq n$ for which answer is ”yes”
  – For each vertex $v$ in $G$, run $A()$ on $G - v, k' - 1$ until $v$ is found for which answer is ”yes”
  – Then $v$, together with any vertex cover of $G - v$ of size $k' - 1$ is a vertex cover of $G$ of size $k$.
  – This algorithm runs in polynomial time assuming calls to $A()$ take constant time

• Thus the OPTIMALVERTEXCOVER $\leq_p$ VERTEXCOVER