NP-Completeness

Almost all algorithms considered so far run in worst-case polynomial time. That is,

$$T(n) = O(n^k) \text{ for some constant k}$$
$$n = \text{input size}$$

	Size n					
Complexity	10	20	30	40	50	
n	.00001 s	.00002 s	.00003 s	.00004 s	.00005 s	
n^2	.0001 s	.0004 s	.0009 s	.0016 s	.0025 s	
n^3	.001 s	.008 s	.027 s	.064 s	.125 s	
n^5	.1 s	3.2 s	24.3 s	1.7 min	5.2 min	
2^n	.001 s	1.0 s	17.9 min	12.7 days	35.7 years	
3^n	.059 s	58 min	6.5 years	3855 centuries	$2x10^8$ centuries	1.3

P

The class of algorithms that run in polynomial time is called **P**.

Algorithms that require more (exponential) time are "intractable"

Some *problems* seem to inherently require more time

One class of such problems is Nondeterministically Polynomial (NP), also called polynomial-time verifiable

Obviously, $P \subseteq NP$, but $P \subset NP$ (or $P = NP$) is an open question

An NP-Complete problem is in NP and is as hard as any problem in NP. Such a problem not necessarily in NP is called NP-Hard.

If $P = NP$, then a large class of NP-Complete problems would have a polynomial-time solution.
Thus, most researchers advocate $P \subset NP$ ($P \neq NP$)

We would like to know the class to which a problem belongs.

Problems

A __________ Q is a binary relation on a set I of ________________________ and a set S of ____________.

Example: Shortest-Path Problem

Instance: graph G
vertices u and v

Solution: sequences of vertices (shortest path)

Decision Problems

A ________________________ is a problem whose solution set S = {no, yes} or {0, 1}.

Example: Path decision problem

Instance: graph G
vertices u and v
non-negative integer k

Solution: 1, if path u $\rightsquigarrow$ v with length at most k
0, otherwise

Encoding Problems

An ___________ of a problem is a mapping from problem instances to symbol strings over some alphabet Σ, where $|\Sigma| >= 2$.
Typically, $\Sigma = \{0, 1\}$.

Problems represented as binary strings are called ___________ problems.
An algorithm ________ a concrete problem in time O(T(n)) if, when provided any problem instance i of length n = |i|, the algorithm can produce the solution in at most O(T(n)) time.
A concrete problem is ________________________________ if there exists an algorithm to solve it in time O(n^k) for some constant k.

The ________________________ is the set of concrete decision problems solvable in polynomial time.

Formal Languages

These provide a convenient framework for analyzing decision problems.

An __________ Σ is a finite set of symbols.

A __________ L over Σ is any set of strings made up of symbols in Σ.

Denote **empty string** ϵ and **empty language** $\emptyset$.

The language of all strings over Σ is Σ^*.
E.g., if $\Sigma = \{0, 1\}$, $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, ..\}$

Example: PATH decision problem language

PATH $= \{\langle G, u, v, k\rangle \mid G = (V, E)$ is a directed graph, $u, v \in V$, $k \geq 0$ is an integer, and there exists a path from u to v in G whose length is at most k$\}$

Note that the problem $\langle G, u, v, k\rangle$ is encoded as a binary string.

Decision Problems and Algorithms

An algorithm A __________ a string $x \in \{0,1\}^*$ if, given input x, the algorithm outputs $A(x) = 1$

The language __________ by an algorithm A is the set $L = \{x \in \{0,1\}^* \mid A(x) = 1\}$

An algorithm A __________ a string x if $A(x) = 0$

A language L is __________ by an algorithm A if every binary string is either accepted or rejected by the algorithm.

Example

The language PATH is decided by the following algorithm in polynomial time:

```
    Use Bellman-Ford to find shortest path from u to v in G
If length(path) $\leq$ k
then output 1
else output 0
```

Decision Problems and Algorithms

A ______________________ is a set of languages, membership in which is determined by a _________________________ (e.g., running time) on an algorithm that determines whether a given string belongs to a language.

Example

$P = \{L \subseteq \{0, 1\}^* \mid$ there exists an algorithm A that decides L in polynomial time$\}$

Theorem 36.2

$P = \{L \mid L$ is accepted by a polynomial time algorithm$\}$

Proof:

There exists an algorithm A' that runs algorithm A for a polynomial amount of time and rejects if A has not yet accepted the string; otherwise accepts.

Polynomial-Time Verification

Given a problem instance and a solution (**certificate**), verify that the solution solves the problem.

Example: PATH problem

Given: ⟨G, u, v, k⟩, path p

Verify: length(p) $\leq$ k

In some cases, having a certificate does not help much since verification is no faster than generating a solution from scratch (e.g., PATH).

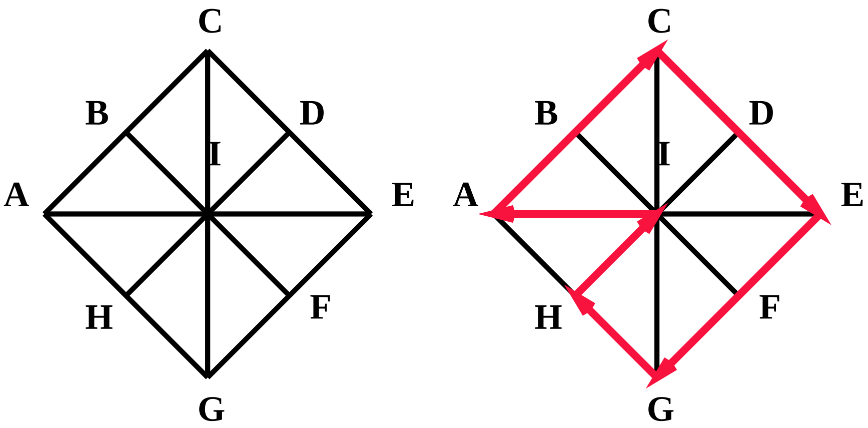
However, this is not true of all problems...

Hamiltonian Cycles

A **Hamiltonian Cycle** of an undirected graph G = (V, E) is a simple cycle that contains each vertex in V.

Hamiltonian Cycle Decision Problem: Does a graph G have a Hamiltonian Cycle?

Language: HAM-CYCLE = {⟨G⟩ | G contains a Hamiltonian Cycle}



Naive Solution: Try all possible cycles.

If encode graph as an adjacency matrix and n = $|\ \langle G \rangle\ |$, then the number of vertices m in G is $\Omega(\sqrt{n})$. There are m! permutations of vertices (possible cycles); thus, running time is $\Omega(m!) = \Omega(\sqrt{n}!) = \Omega(2^{\sqrt{n}})$, which is $\neq$ O(n^k) for any constant k.

In fact, HAM-CYCLE is NP-Complete.

Verification Algorithms

Consider a corresponding verification problem for HAM-CYCLE:

Given a cycle and a graph G, verify if cycle is a Hamiltonian cycle in G.

Running time: O(n^2)

- A **verification algorithm** is a two-argument algorithm A, where one argument is an ordinary input string x, and the other argument is a binary string y called a **certificate**. Algorithm A **verifies** x if there exists a y such that A(x,y) = 1.

- The **language verified** by a verification algorithm A is
  L = {x ∈ {0,1}* | there exists y ∈ {0,1}* such that A(x,y) = 1}

NP

The **complexity class NP** is the class of languages that can be verified by a polynomial-time algorithm.

L ∈ NP if algorithm A verifies language L in polynomial time.

Example: HAM-CYCLE ∈ NP

Reducibility

A problem Q can be **reduced** to another problem Q' if any instance of Q can be "easily rephrased" as an instance of Q', whose solution provides a solution to the instance of Q.

Example: Solving $ax+b=0$ reduces to solving $0x^2 + ax + b = 0$.

A language L_1 is **poly-time reducible** to language L_2, written $L_1 \leq_P L_2$, if there exists a poly-time computable function $f : \{0,1\}* \rightarrow \{0,1\}*$ such that for all x ∈ {0,1}*:

$$x \in L_1 \text{ iff } f(x) \in L_2$$

where f is the **reduction function**.

This is a one-way function. Q' will not always reduce to Q.

Examples

The following example illustrates the concept of reducibility. Consider three problems, A, B, and C:

- A=Prime(n): The problem of determining whether or not n is a prime number.
- B=Numberfactor(n): The problem of counting the number of distinct primes that divide n.

- C=Smallestfactor(n): The problem of finding the smallest integer $x \geq 2$ such that x divides n.

In this example A $\leq_P$ C, and B $\leq_P$ C. Why?

Thus the solution of Smallestfactor(n) tells us that n is not a prime.

To see how B $\leq_P$ C we need a simple algorithm that counts the number of distinct divisors of n using C.

Lemma 36.3

If L_1, $L_2 \subseteq$ {0,1}*, and $L_1 \leq_P L_2$, then $L_2 \in$ P implies $L_1 \in$ P.

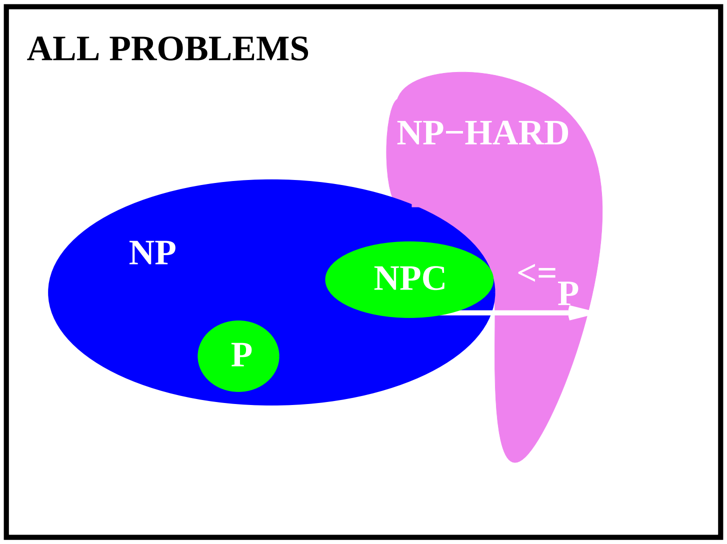
For any instance of L_1
map to L_2 (poly time)
solve L_2 (poly time)
Thus if we can solve L_2 in poly time we can solve L_1 in poly time.

NP-Completeness

NP-Complete problems are the hardest problems (no problem is harder) in NP, i.e., every problem in NP reduces to an NP-Complete problem.

- A language L $\subseteq$ {0,1}* is **NP-Complete** if L $\in$ NP, and $L' \leq_P L$ for every L' $\in$ NP.
- The class of NP-Complete languages is called **NPC**.
- A language L $\subseteq$ {0,1}* is **NP-Hard** if $L' \leq_P L$ for every L' $\in$ NP.

- A language that is NP-Hard is not necessarily in NP.
  E.g., Kth Largest Subset is NP-Hard, but not NPC.
  KLS: are there at least K distinct subsets A' of set A such that $\Sigma_{a \in A'} a \leq B$?



If we can solve one NPC problem in polynomial time, we can solve every problem in NP in polynomial time. For this reason, many assume P $\neq$ NP.

Theorem 36-4

If any NP-Complete problem is poly-time solvable, then P = NP.

If any problem in NP is provably not poly-time solvable, then all NP-Complete problems are not poly-time solvable.

If we can prove one problem is NP-Complete, then we can prove others more easily by showing an NP-Complete problem reduces to them.