Com S 311 Section B
Introduction to the Design and Analysis of Algorithms

Lecture Two for Week 11

Xiaoqiu (See-ow-chew) Huang

Iowa State University

April 9, 2021

A Formal-Language Framework

A reason for focusing on decision problems is to take advantage of the machinery of formal-language theory.

An **alphabet** Σ is a finite set of symbols.

A **language** L over Σ is any set of strings made up of symbols from Σ.

For example, if $\Sigma = \{0, 1\}$, the set $L = \{01, 011, 0111, 01111, ...\}$ is the language of binary strings that begin with 0 and contain only one or more 1s at each of the remaining positions.

The **empty string** (with no symbol) is denoted by ϵ. The **empty language** (with no string) is denoted by θ.

For example, if $\Sigma = \{0, 1\}$, then
$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$ is the set of all binary strings.
Every language is a subset of Σ^*.

Operations on Languages

Set operations such as **union** and **intersection** apply directly to languages.

The **complement** of L is defined by $\overline{L} = \Sigma^* - L$.

The **concatenation** of two languages L_1 and L_2 is defined by

$L_1 L_2 = \{x_1 x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2\}$.

The **closure** or **Kleene star** of a language L is defined by

$L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup ...,$

where L^k is the language obtained by concatenating L to itself k times.

Kleene Star

Let L be a language over an alphabet Σ and let $x = b_1 b_2 b_3 ... b_n$ be a string over Σ, where $b_i \in \Sigma$ for $i = 1, 2, 3, ..., n$.

If $n = 0$, then x is the empty string and is in L^*.

Assume $n > 0$. Consider a substring of x from position i to position j: $x_{i,j} = b_i, b_{i+1}, ..., b_j$ with $1 \leq i \leq j \leq n$.

How do we check if $x_{i,j}$ is in L^*.

If $x_{i,j}$ is in L, then it is in L^*.

If $x_{i,j}$ is not in L, then $x_{i,j}$ is in L^* if and only if $i < j$ and there exists a k, $i \leq k < j$, such that both $x_{i,k}$ and $x_{k+1,j}$ are in L^*.

Any Decision Problem as a Language

The set of instances for any decision problem Q is the set Σ^*, where $\Sigma = \{0, 1\}$.

Because Q is characterized by those problem instances with a 1 (yes) answer, Q is viewed as a language over $\Sigma = \{0, 1\}$, where

$L = \{x \in \Sigma^* : Q(x) = 1\}$.

For example, the decision problem PATH corresponds to the following language:

PATH $= \{ < G, u, v, k >: G = (V, E)$ is an undirected graph,
$u, v \in V$,
$k \geq 0$ is an integer, and
there exists a path from u to v in G with at most k edges$\}$.

Accepting Languages

An algorithm A **accepts** a string $x \in \{0,1\}^*$ if its output $A(x)$ is 1.

The language **accepted** by an algorithm A is the set of strings accepted by it:
$L = \{x \in \{0,1\}^* : A(x) = 1\}$.

An algorithm A **rejects** a string $x \in \{0,1\}^*$ if its output $A(x)$ is 0.

Note that an algorithm that accepts a language L does not need to terminate on a string $x \notin L$.

A language L is **accepted in polynomial time** by an algorithm A if it is accepted by A and if there exists a constant k such that any length-n string in L is accepted by A in time $O(n^k)$.

Deciding Languages

A language L is **decided** by an algorithm A if every binary string in L is accepted by A and every binary string not in L is rejected by A.

A language L is **decided in polynomial time** by an algorithm A if there exists a constant k such that for any length-n string $x \in \{0,1\}^*$, the algorithms correctly decides whether $x \in L$ in time $O(n^k)$.

Note that an algorithm that decides a language L needs to terminate on every string in $\{0,1\}^*$ and to report either 1 or 0.

Example Languages and Algorithms

Consider an example language over $\Sigma = \{0, 1\}$ consisting of all binary strings of lengths at least 2 that begin with 0 followed only by 1s.

Algorithm accepting this language checks if an input binary string begins with 0 and contains only one or more 1s at the other positions.

If so, the algorithm terminates and reports 1. Otherwise, it runs forever.

An algorithm deciding the language always terminates and reports either 1 or 0, depending on whether the input binary string meets the language membership requirement or not.

An Alternative Definition of the Complexity Class P

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

We show that P is also the class of languages that can be accepted in polynomial time.

Theorem 34.2

$\mathrm{P} = \{\ L \subseteq \{0,1\}^* :$ there exists an algorithm A that accepts L in polynomial time $\}$.

Proof

Note that the class of languages decided by polynomial-time algorithms is a subset of the class of languages accepted by polynomial-time algorithms.

Let L be a language accepted by a polynomial-time algorithm A.

We just need to prove that L is decided by a polynomial-time algorithm.

We use a classic 'simulation' argument to show that there exists another polynomial-time algorithm B that decides L.

By the assumption, there exist a constant k and a constant c such that A accepts L in at most cn^k steps.

For any input string x of length n, algorithm B simulates cn^k steps of A.

Proof

After that, algorithm B inspects the behavior of A.

IF A has accepted x, then B accepts x by reporting an output of 1.

Otherwise, B rejects x by reporting an output of 0.

The overhead of B simulating A does not increase the running time by more than a polynomial factor.

Thus, algorithm B is a polynomial-time algorithm that decides L.

Note that this proof just shows that algorithm B exists; it does not show how to construct algorithm B.

Polynomial-Time Verification

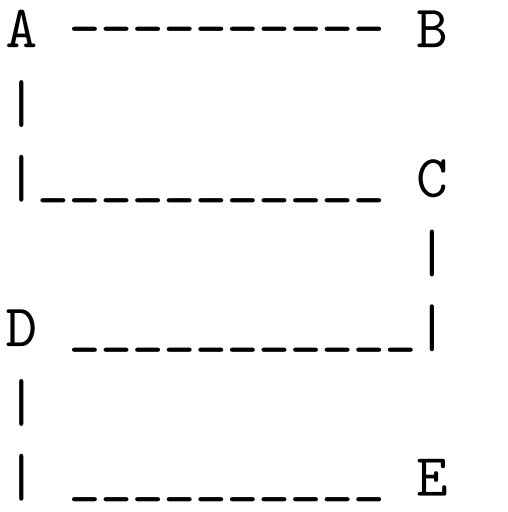
We introduce the concept of polynomial-time verification to define the complexity class NP.

Consider the problem of finding a hamiltonian cycle in an undirected graph.

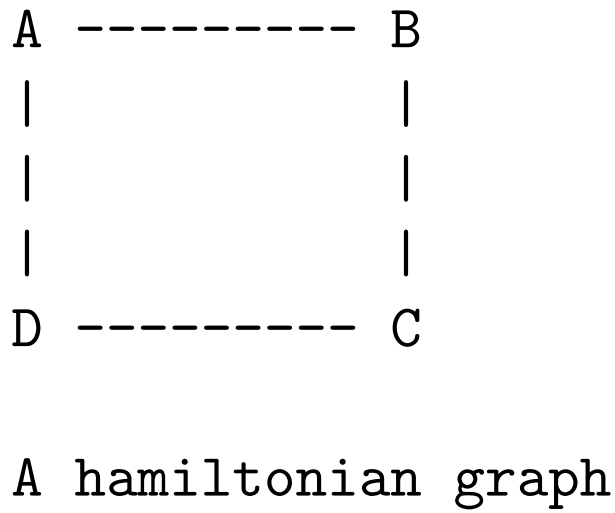
A **hamiltonian cycle** of an undirected graph $G = (V, E)$ is a simple cycle that contains each vertex in V.

A graph that contains a hamiltonian cycle is **hamiltonian**; otherwise, it is **nonhamiltonian**.

We define the **hamiltonian-cycle problem** (Does a graph G contain a hamiltonian cycle?) as a formal language:
HAM-CYCLE $= \{ < G >: G \text{ is a hamiltonian graph } \}$.



A nonhamiltonian graph



A hamiltonian graph

Verification Algorithms

A **verification algorithm** takes two arguments: one argument is an ordinary input string x and the other is a binary string y called a **certificate**.

A two-argument algorithm **verifies** an input string x if there exists a certificate y such that $A(x, y) = 1$.

The **language verified** by a verification algorithm A is
$L = \{ x \in \{0,1\}^* : \text{there exists } y \in \{0,1\}^* \text{ such that } A(x, y) = 1 \}$.

An algorithm A verifies a language L if for any string $x \in L$, there exists a certificate Y that A can use to prove that $x \in L$.

For any string $x \notin L$, there must be no certificate to prove that $x \in L$.

Example

In the hamiltonian-cycle problem, the certificate is the list of vertices in some hamiltonian cycle.

If a graph is hamiltonian, then the hamiltonian cycle itself offers enough information to verify this fact.

If a graph is not hamiltonian, then no list of vertices can fool the verification algorithm, because it carefully checks the proposed cycle to be true.

The Complexity Class NP

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

Formally, a language L belongs to NP if and only if there exists a two-input polynomial-time algorithm A and a constant c such that
$L = \{ x \in \{0,1\}^* :$there exists $y \in \{0,1\}^*$
with $|y| = O(|x|^c)$ such that $A(x,y) = 1 \}$.

We say that algorithm A **verifies** language L **in polynomial time**.

Note that NP stands for 'nondeterministic polynomial time.' The textbook uses the somewhat simpler yet equivalent notion of verification.

The Complexity Class NP

From the previous discussion on the hamiltonian-cycle problem, we conclude that HAM-CYCLE $\in NP$.

If $L \in P$, then $L \in NP$. So $P \subseteq NP$.

This is because a polynomial algorithm deciding L can be converted into a two-argument verification algorithm that simply ignores any certificate and decides exactly those input strings it determines to be in L.

P = NP?

It is unknown whether P = NP.

Most computer scientists believe that P and NP are not the same class.

One major argument for this is that the class P consists of problems that can be solved quickly, but that the class NP consists of problems that can be verified quickly.

We have learned from experience that it is often more difficult to solve a problem from scratch than to verify a clearly presented solution.

More compelling evidence is that there exist languages that are NP-complete.