Algorithms 12

CS201

Kaustuv Nag

Decision Problems vs. Optimization Problems

- **Optimization Problems:** Each feasible (i.e., “legal”) solution has an associated value, and we wish to find a feasible solution with the best value.
  - SHORTEST-PATH: We are given an undirected graph G and vertices u and v, and we wish to find a path from u to v that uses the fewest edges.
- **Decision Problems:** The answer is simply “yes” or “no” (or, “1” or “0”).
  - Given a number n, is it prime?
- We usually can cast a given optimization problem as a related decision problem by imposing a bound on the value to be optimized.
  - A decision problem related to SHORTEST-PATH is PATH: given a directed graph G, vertices u and v, and an integer k, does a path exist from u to v consisting of at most k edges?
  - We can solve PATH by solving SHORTEST-PATH and then comparing the number of edges in the shortest path found to the value of the decision-problem parameter k.
- If an optimization problem is *easy*, its related decision problem is *easy* as well.
- We restrict attention to decision problems.

Abstract Problems and the Complexity Class P

- We define an abstract problem Q to be a binary relation on a set I of problem instances and a set S of problem solutions.
  - If $i = \langle G, u, v, k \rangle$ is an instance of the decision problem PATH, then $\text{PATH}(i) = 1$ (yes) if a shortest path from u to v has at most k edges, and $\text{PATH}(i) = 0$ (no) otherwise.
- An *encoding* of a set S of abstract objects is a mapping e from S to the set of binary strings.
- We call a problem whose instance set is the set of binary strings a concrete problem.
- We say that an algorithm solves a concrete problem in time $O(T(n))$ if, when it is provided a problem instance i of length $n = |i|$, the algorithm can produce the solution in $O(T(n))$ time.
- A concrete problem is polynomial-time solvable, if there exists an algorithm to solve it in time $O(n^k)$ for some constant k.
- The complexity class P is the set of concrete decision problems that are polynomial-time solvable.

Language

- 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.
  - If $\Sigma = \{0,1\}$, the set $L = \{10, 11, 101, 111, 1011, 10001, \ldots\}$ is the language of binary representations of prime numbers.
- We denote the empty string by ε, the empty language by $\emptyset$, and the language of all strings over Σ by Σ^*.
- We can perform a variety of operations on languages.
  - Set-theoretic operations, such as union and intersection, follow directly from the set-theoretic definitions.
  - We define the complement of L by $\bar{L} = \Sigma^* - L$.
  - The concatenation L_1L_2 of two languages L_1 and L_2 is the language $L = \{x_1x_2 : x_1 \in L_1, x_2 \in L_2\}$.
  - The closure or Kleene star of a language L is the language $L = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots$ where L^k is the language obtained by concatenating L to itself k times.

Language

- We say that an algorithm A accepts a string $x \in \{0,1\}^*$ if, given input x, the algorithm's output $A(x)$ is 1.
- The language accepted by an algorithm A is the set of strings $L = \{x \in \{0,1\}^* : A(x) = 1\}$, that is, the set of strings that the algorithm accepts.
- An algorithm A rejects a string x if $A(x) = 0$.
- 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 accepted in polynomial time by an algorithm A if it is accepted by A and if in addition there exists a constant k such that for any length-n string $x \in L$, algorithm A accepts x in time $O(n^k)$.
- 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 algorithm correctly decides whether $x \in L$ in time $O(n^k)$.

P Class, Certificate, NP Class

P Class

- $P = \{L \subseteq \{0,1\}^* :$ there exists an algorithm A that decides L in polynomial time$\}$.
- $P = \{L : L$ is accepted by a polynomial-time algorithmg$\}$.

Certificate

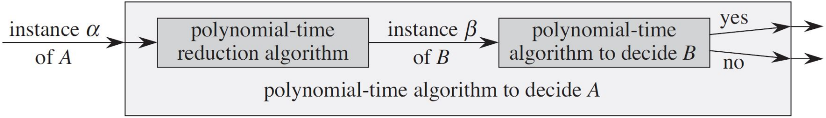
Certificate is a piece of evidence that allows us to verify in polynomial time that a string is in a given language.

NP Class

NP is defined to be the class of all languages that can be verified in polynomial time. Clearly, $P \subseteq NP$. It is widely believed that $P \neq NP$.

Reduction

- Let us consider a decision problem A, which we like to solve in polynomial time.
- Suppose that we already know how to solve a different decision problem B in polynomial time.
- Suppose that we have a procedure that transforms any instance α of A into some instance β of B with the following (We call such a procedure a polynomial-time reduction algorithm) characteristics:
  - The transformation takes polynomial time.
  - The answers are the same. That is, the answer for α is "yes" if and only if the answer for β is also "yes."
- Then, it provides us a way to solve problem A in polynomial time.

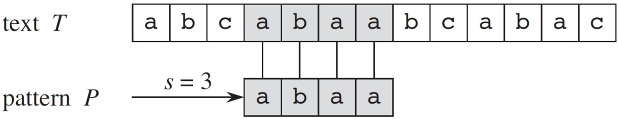


String Matching

- We assume that the text is an array $T[1..n]$ of length n and that the pattern is an array $P[1..m]$ of length $m \leq n$.
- We further assume that the elements of P and T are characters drawn from a finite alphabet Σ.
- The character arrays P and T are often called strings of characters.

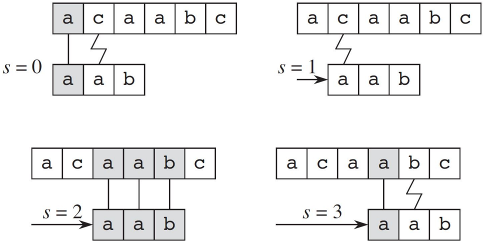
String Matching

- we say that pattern P occurs with shift s in text T (or, equivalently, that pattern P occurs beginning at position $s+1$ in text T) if $0 \le s \le n-m$ and $T[s+1..s+m]$ (that is, if $T[s+j] = P[j]$, for $1 \le j \le m$).
- If P occurs with shift s in T, then we call s a valid shift; otherwise, we call s an invalid shift.
- The string-matching problem is the problem of finding all valid shifts with which a given pattern P occurs in a given text T.



String Matching

```
NAIVE-STRING-MATCHER(T, P)
n = T.length
m = P.length
for s = 0 to n − m
    if P[1..m] == T[s + 1..s + m]
        print "Pattern occurs with shift" s
```



String Matching

- It takes $O((n-m+1)m)$ runtime.
- The naive string-matcher is inefficient because it entirely ignores information gained about the text for one value of s when it considers other values of s.
- For example, if $P = \texttt{aaab}$ and we find that $s = 0$ is valid, then none of the shifts 1, 2, or 3 are valid, since $T[4] = \texttt{b}$.

White Board

White Board