MA 515: Introduction to Algorithms & MA353 : Design and Analysis of Algorithms [3-0-0-6]

Lecture 37

http://www.iitg.ernet.in/psm/indexing_ma353/y09/index.html

Partha Sarathi Mandal

psm@iitg.ernet.in

Dept. of Mathematics, IIT Guwahati

Mon 10:00-10:55 Tue 11:00-11:55 Fri 9:00-9:55

Class Room : 2101

NP-Completeness Theory

- Theory is based considering decision problems (have YES/NO answers)
- Ex: Does there exist a path from node *u* to node *v* in graph G with at most *k* edges ?
  - instead of:
    - What is the length of the shortest path from *u* to *v* ?
    - Or even: What is the shortest path from *u* to *v* ?

Decision Problems

Why focus on decision problems ?

- Solving the general problem is at least as hard as solving the decision problem version.
- For *many* natural problems, we only need polynomial additional time to solve the general problem if we already have a solution to the decision problem.
- Lets us use "language acceptance" notions.

Languages and Decision Problems

- Language: a set of strings over some alphabet
- Decision problem: a language consisting of exactly those strings that encode YES instances of the problem
- Encode ?

Encodings

- Every abstract problem has to be represented somehow for the computer to work on it - ultimately a binary representation.
- Consider the problem: "Is *x* prime?"
- Each input is a positive integer
- Common way to encode an integer is in binary
- Primes decision problem is {10,11,101,111,…} since 10 encodes 2, 11 encodes 3, 101 encodes 5, 111 encodes 7, etc.

Languages and Decision Problems

- Language (L): a set of strings over some alphabet ($\sum$)
- Decision problem (Q): a language, L consisting of exactly those strings, *x* that encode YES instances of the problem. L = $\{x \in \sum^*: Q(x) = 1\}$
- Encode ??
  - Ex: $\sum$={0,1} L={10,11,101,111,1011,1101,10001,..} set of prime numbers
  - $\sum^*$={ε,0,1,00,01,10,11,000,…}

More Complicated Encodings

- Suggest an encoding for the shortest path decision problem.
- Represent *G*, *u*, *v* and *k* somehow in binary.
- Decision problem is all encodings of a graph *G*, two nodes *u* and *v*, and an integer *k* such that *G* really does have a path from *u* to *v* of length at most *k*.
  - Instance $i = \langle G,u,v,k \rangle$ Parth(*i*) =1(yes) if a shortest path from *u* to *v* has length at most *k*,
  - Parth(*i*)=0 (no) otherwise.

Definition of P

- P is the set of all decision problems that can be computed in time $O(n^k)$, where n is the length of the input string and k is a constant
- "Computed" means there is an algorithm that correctly returns YES or NO whether the input string is in the language.

Example of a Decision Problem in P

- "Given a graph *G*, nodes *u* and *v*, and integer *k*, is there a path in *G* from *u* to *v* with at most *k* edges ?"
- Why is this a decision problem?
  - Has YES/NO answers
- We are glossing over the particular encoding (tedious but straightforward)
- Why is this problem in P ?
  - Do BFS on G in polynomial time.

Definition of NP

- NP = set of all decision problems for which a candidate solution can be verified in polynomial time.
- Does *not* stand for "not polynomial"
  - in fact P is a subset of NP
- NP stands for "nondeterministic polynomial"
- A *nondeterministic* algorithm is one that can "guess" the right answer or solution.
- The solution, then has to verify that it is correct.

Example of a Decision Problem in NP

- Decision problem: Is there a path in G from u to v of length at most k?
- Candidate solution: a sequence of nodes v_0, v_1, …, v_ℓ
- To verify:
  - check if $\ell \leq$ k
  - check if $v_0 = u$ and $v_\ell = v$
  - check if each (v_i, v_{i+1}) is an edge of G

Example of a Decision Problem in NP

- Decision problem: Does G have a Hamiltonian cycle ?
- Candidate solution: a sequence of nodes $v_0, v_1, \ldots, v_\ell$
- To verify:
  - check if ℓ = number of nodes in G
  - check if $v_0 = v_\ell$ and there are no repeats in $v_0, v_1, \ldots, v_{\ell-1}$
  - check if each (v_i , v_{i+1}) is an edge of G

Going From Verifying to Solving

- for each candidate solution do
  - verify if the candidate really works
  - if so then return YES
- return NO

Difficult to use in practice, though, if number of candidate solutions is large

Number of Candidate Solutions

- "Is there a path from *u* to *v* in *G* of length at most *k* ?": more than *n*! candidate solutions where *n* is the number of nodes in *G*
- "Does *G* have a Hamiltonian cycle ?": *n*! candidate solutions.

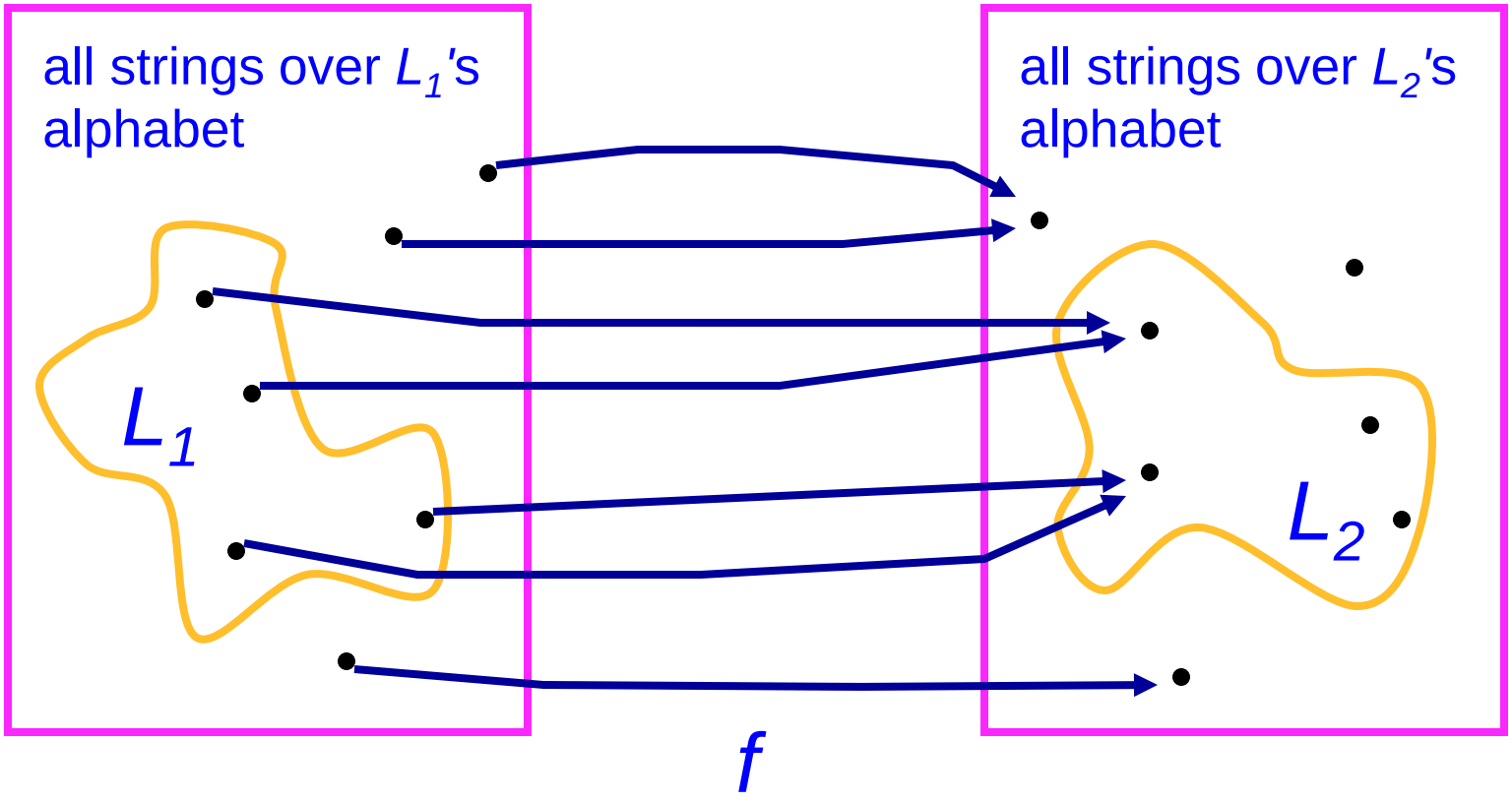
Trying to be Smarter

- For the length-*k* path problem, we can do better than the brute force approach of trying all possible sequences of nodes
  - use BFS
- For the Hamiltonian cycle problem, no one knows a way that is significantly faster than trying all possibilities
  - but no one has been able to prove that there is NOT a faster way

Polynomial Reduction

- A polynomial reduction (or transformation) from language L_1 to language L_2 is a function f from strings over L_1's alphabet to strings over L_2's alphabet such that
  1) f is computable in polynomial time
  2) for all x, x is in L_1 if and only if $f(x)$ is in L_2

Polynomial Reduction



Polynomial Reduction

- YES instances map to YES instances
- NO instances map to NO instances
- computable in polynomial time
- Notation: $L_1 \leq_p L_2$
- Think: L_2 is at least as hard as L_1

Polynomial Reduction Theorem

- Theorem: If $L_1 \leq_p L_2$ and L_2 is in P, then L_1 is in P.
- Proof: Let A_2 be a polynomial time algorithm for L_2. Here is a polynomial time algorithm A_1 for L_1.
- input: x $|x| = n$
- compute $f(x)$ takes $p(n)$ time
- run A_2 on input $f(x)$ takes q(p(n)) time
- return whatever A_2 returns takes O(1) time

Implications

- If there is a polynomial time algorithm for L_2, then there is a polynomial time algorithm for L_1.
- If there is no polynomial time algorithm for L_1, then there is no polynomial time algorithm for L_2.
- Note the asymmetry!

Definition of NP-Complete

- L is NP-complete if
- (1) L is in NP and
- (2) for all L' in NP, $L' \leq_p L$.

In other words, L is at least as hard as every language in NP.

Implication of NP-Completeness

Theorem: Suppose L is NP-complete.

(a) If there is a poly time algorithm for L, then P = NP.

(b) If there is no poly time algorithm for L, then there is no poly time algorithm for any NP-complete language.

Showing NP-Completeness

- How to show that a problem (language) L is NP-complete?
- Direct approach: Show

  (1) L is in NP

  (2) every other language in NP is polynomially reducible to L.
- Better approach: once we know some NP-complete problems, we can use reduction to show other problems are also NP-complete.

Showing NP-Completeness with a Reduction

To show L is NP-complete:

- (1) Show L is in NP.
- (2.a) Choose an appropriate known NP-complete language L'.
- (2.b) Show $L' \leq_p L$.

Why does this work ? By transitivity: Since every language L" in NP is polynomially reducible to L', L" is also polynomially reducible to L.

First NP-Complete Problem

- How do we get started? Need to show via brute force that some problem is NP-complete.
- Logic problem "satisfiability" (or SAT).
- Given a boolean expression (collection of boolean variables connected with ANDs and ORs), is it satisfiable, i.e., is there a way to assign truth values to the variables so that the expression evaluates to TRUE ?