

THE THEORY OF NP-COMPLETENESS

P, NP, co-NP, NPC and NP-hard

Chapter 17-17.2.1 of GT

DISCLAIMER: *Report an error / typo to the author so that he resolves it.*
These notes are NOT to be distributed outside of NJIT without the expressly written permission of A. Gerbessiotis

Definition 1 (Decision Problem). A decision problem is a problem with a YES or NO answer.

Definition 2 (Optimization Problem). An optimization problem is a problem whose output finds an answer to a question specified in the input.

Definition 3 (Deterministic Algorithm). An algorithm A is deterministic if all of its decisions are deterministic.

Definition 4 (Randomized Algorithm). An algorithm A is a randomized algorithm if some of its decisions are not deterministic but are decided through access and usage of a random number generator.

Decision and Optimization Problems

Introduction

Some example of decision problems that we encountered earlier are listed below.

Example 1. For a sequence A of n keys and a given key k , does $k \in A$? (Answer with a YES or NO.)

Example 2. For an input graph $G = (V, E)$ represented by its adjacency list representation is G topologically sorted? (Answer with a YES or NO.)

Example 3. For an input graph $G = (V, E)$ represented by its adjacency list representation and two vertices $u \in V$ and $v \in V$ does there exist a path from u to v ? (Answer with a YES or NO.)

Decision and Optimization Problems

Introduction

We rephrase the three problems defined earlier as optimization problems below.

Example 4. For a sequence A of n keys and a given key k , find j such that $A[j] == k$ or else return -1 .

Example 5. For an input graph $G = (V, E)$ represented by its adjacency list representation find the topological sorting of the vertices of G if such exists and say NO otherwise.

Example 6. For an input graph $G = (V, E)$ represented by its adjacency list representation and two vertices $u \in V$ and $v \in V$ find the path from u to v if such a path exists and say NO otherwise.

Polynomial time Overview

Definition 5 (Polynomial time algorithm). An algorithm A is a poly-time algorithm (polynomial time algorithm) if its running time is $O(n^k)$, where k is a constant greater than zero, and n is the problem size (loosely input size).

Definition 6 (Polynomial time problem). A polynomial time (poly-time) problem is a computational problem that admits an algorithm for its solution that is polynomial time.

All the algorithms that we have examined since the beginning of the semester are algorithms that can be characterized by the following properties.

- They are primarily **polynomial time** (deterministic) algorithms, and
- they are primarily optimization algorithms.

In the remainder of this section we will be discussing decision problems rather than optimization problems. Moreover most if not all of such problems we suspect that they do not have polynomial time algorithms for their solution.

Polynomial time Class P

Definition 7 (Class P). The class P contains all decision problems that are poly-time. Therefore a problem $A \in P$ if and only if the following is true

$$A \in P \iff \exists k > 0 : T_A(n) = O(n^k), \mathbf{k \text{ constant.}}$$

$T_A(n)$ is the running time of an efficient algorithm for solving problem A , and n is problem size.

Note that for class P we only deal with decision problems. Optimization problems are irrelevant.

Problem 1.

Problem AeBmC.

Input: Three (unsigned) integers $A, B, C \geq 0$, where $0 \leq B \leq A$ and $0 \leq C \leq A$.

Output: Output YES if $A == B * C$, and NO otherwise.

We read AeBmC as 'A is equal to B multiplied by C'.

Proposition 1. AeBmC is in P .

Proof. The problem size for that problem is the number of digit (or bit) of the largest integer which is A . In other words $n = |A| = \lceil \lg(A + 1) \rceil$. All A, B, C require at most n bit. Then the elementary integer multiplication algorithm followed by a subtraction to realize $A - B * C$ resolves this problem in $O(n^2)$ time.

AeBmC(A,B,C)

1. $x = A - B * C$;
2. if $x == 0$ print('YES');
3. else print('NO');

Thus $T_{AeBmC} = O(n^2)$, where n is problem size or in other words, $T_{AeBmC} = O((\lg A)^2)$. In the latter we express running time in terms of the largest integer's number of bit. ■

An interesting class
Class NP

Definition 8 (Class NP). The class NP contains all decision problems whose solution or a certificate of a solution can be verified in poly-time.

Definition 9 (Composite (integer) number). An unsigned integer n is composite if there exists a p such that $p \neq 1$ and $p \neq n$ such that $n = p * q$ for some integer q .

The p, q are called the **divisors** of n . Since they are neither one nor n they are also known as non-trivial divisors of n .

Definition 10 (Prime number). An unsigned integer n is a prime number if it is not a composite number.

An interesting class
Class NP

A more appropriate definition is the following one.

Definition 11 (Prime number). An unsigned integer n is a prime number if the only unsigned integer p, q such that $n = p * q$ are 1 and n .

Problem 2.

Problem Composite.

Input: Unsigned integer A .

Output: Output YES if A is composite, and NO if A is a prime number (i.e. not composite).

Proposition 2. Composite is in NP.

Proof. A certificate of the compositeness of A are integer B, C such that $B \neq 1$ and $B \neq A$ such that $A = B * C$. The latter problem is AeBmC and we know that AeBmC is in P thus the pair (B, C) , a certificate of the solution or compositeness of A can be verified in polynomial time by multiplying B, C into $B * C$, and then subtracting from $A - B * C$ to certify the result is 0 and thus $A = B * C$. ■

Problem 3.

Problem C (Cycle).

Input: Graph $G = (V, E)$ represented by its adjacency list representation.

Output: Output YES if G has a cycle, and NO if G does not have a cycle.

Theorem 1. $C \in P$. (*Cycle is in P.*)

Proof. Run DFS on G . If there is a back edge say YES, else say NO. Running time is $O(n + m)$ linear to n and m . Problem size is $N = n + m$ thus running time $O(n + m) = O(N)$ and in the definition of P , $n = N$ and $k = 1 > 0$ and constant. ■

Theorem 2. $C \in NP$. (*Cycle is in NP.*)

Proof. A certificate of the solution of C is a cycle i.e. a sequence of vertices $T = \langle u_1, u_2, \dots, u_k \rangle$ such that $(u_i, u_{i+1}) \in E$ and moreover $(u_k, u_1) \in E$ as well. A cycle has at most n vertices and its edges are thus at most n . An edge can be verified whether it exists or not in $O(n)$ time. (For edge (u_i, u_{i+1}) we check in $O(n)$ time whether u_{i+1} is in the adjacency list of u_i .) Thus in time $O(n^2)$, which is polynomial time, we can verify that T is a certificate of the solution for C . ■

Finally, we conclude that

Theorem 3. $P \subset NP$.

A more interesting class
Class co-NP

Definition 12 (Class co-NP). The class co-NP contains all decision problems whose complement are in NP.

$$X \in co - NP \iff \overline{X} \in NP.$$

Problem 4.

Problem Prime.

Input: Unsigned integer A .

Output: Output YES if A is prime, and NO if A is a composite number (i.e. not prime).

Corollary 1. The Composite problem and the Prime problem are complementary. Therefore

$$Prime = \overline{Composite}, \quad Composite = \overline{Prime}.$$

Theorem 4. *Prime is in co-NP.*

$$Prime \in co - NP.$$

Proof. To show that Prime is in co-NP we need to show that $\overline{Prime}$ is in NP. But $\overline{Prime}$ is Composite and we know that Composite is indeed in NP. Proof completed. ■

A more interesting class
Class co-NP

For Prime we know the following results.

Theorem 5 (Pratt(1970)). *Prime is in NP.*

$$\text{Prime} \in NP.$$

Theorem 6 (Agarwal, Kayal, Saxena(2004)). *Prime is in P.*

$$\text{Prime} \in P.$$

Theorem 7. *We have that $P \subset NP \cap co - NP$.*

Definition 13 (Graph isomorphism). A graph $G_1 = (V_1, E_1)$ is isomorphic to graph $G_2 = (V_2, E_2)$, if there is a permutation p of the vertices i.e. $p : V_1 \rightarrow V_2$ that preserves the edges i.e. for $(u, v) \in E_1$ we have that $(p(u), p(v)) \in E_2$.

Problem 5.

Problem ISO (Graph Isomorphism).

Input: Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

Output: Output YES if G_1 is isomorphic to G_2 and NO otherwise.

Theorem 8. *ISO is in NP.*

Proof. The certificate of a solution is the permutation p . A solution can be verified in $O(n + m)$ time. ■

Structure of Class NP

Hamiltonian cycle

Problem 6.

Problem HC (Hamiltonian Cycle).

Input: Graph $G = (V, E)$ and its adjacency list.

Output: Output YES if G has a Hamiltonian cycle, and NO otherwise. A Hamiltonian cycle is a cycle of length n (that traverses each vertex exactly once).

Theorem 9. *HC is in NP.*

Proof. The certificate of a solution is the cycle. A solution can be verified in $O(n^2)$ time. ■

Structure of Class NP

Problems complete in NP

Definition 14 (Class NPC). The class NPC contains all decision problems that are NP-complete, i.e. they are complete for the class NP.

$$X \in NPC \iff X \text{ is NP-complete.}$$

Here is a characterization of the class NP.

The class NP contains

- some easy problems, all those problems in P ,
- some challenging problems such as ISO, and
- a collection of some difficult problems which is the class NPC.

We need to provide a definition for NP-complete or what it means for a set of problems to be complete in NP.

Definition 15 (Complete in NP). A set (class) of decision problems are complete for the class NP if resolving the difficulty of one such problem resolved the difficulty of all the problems in the set (class). In other words,

- if one problem is easy (poly-time) all problems are easy (poly-time), or
- if one problem is NOT easy, then all problems are not easy. (Not easy means not poly-time.)

Definition 16 (Poly-time reduction). Suppose we have two problems B and A . B is poly-time reducible to A if we can find a poly-time reduction/function $f()$ of B into A such that x is a solution of B if and only if $f(x)$ is a solution of A .

NP-complete problems

Class NPC

Definition 17 (Class NPC). $A \in NPC$ if and only if

$$A \in NP,$$

and

$$\forall B \in NP : B \Rightarrow A.$$

The latter says that B is poly-time reducible to A . We also say that A is NP-complete.

Definition 18 (NP-complete). A is NP-complete if and only if $A \in NPC$.

Definition 19 (NP-hard). A is NP-hard if and only

$$\forall B \in NP : B \Rightarrow A.$$

The latter says that B is poly-time reducible to A .

Alternative definitions

P and NP

More formally, by describing the decision problem as a language with all strings (representation of the input) in the language generating a YES decision, and all strings not in the language generating a NO decision, we can more formally define P as follows.

Definition 20 (Alternative for P). P consists of all languages L that have a polynomial time algorithm A such that for any input x ,

1. for any $x \in L$ then, $A(x)$ accepts (i.e outputs YES), and
2. for any $x \notin L$ then, $A(x)$ rejects (i.e outputs NO).

Definition 21 (Alternative for NP). NP consists of all languages L that have a polynomial time algorithm A such that for any input x ,

1. for any $x \in L$ then, there exists a y (a.k.a. certificate) such that $A(x, y)$ accepts (i.e outputs YES), and the size $|y|$ of y is bounded by a polynomial in the size of $|x|$, and
2. for any $x \notin L$ then, FOR ALL y , $A(x, y)$ rejects (i.e outputs NO).

Alternative definitions

P and NP

The y in the definition serves the role of the certificate. For the Hamiltonian cycle problem the graph G is the input x , and if there is a solution then y serves the role of the certificate, which is a solution. In other words, in order to prove that a decision problem or equivalently a language is in NP we work as follows: (a) "guess the solution (one always guesses right) of polynomial size in the input" and (b) then "verify that it is indeed a solution in polynomial time".

Definition 22 (Poly-time reduction). A polynomial reduction from a language L_1 to a language L_2 is a function f such that

1. there is a polynomial time algorithm that computes f , and
2. for all x , $x \in L_1$ if and only if $f(x) \in L_2$.

Definition 23 (Alternative for NP-hard). A language L is NP-hard if for all $L' \in NP$ there is a polynomial reduction from L' to L .

Definition 24 (Alternative for NP-complete). A language L is NP-complete if L is in NP and L is also NP-hard.

Theorem 10. If $L \in NPC$ and $L \in P$ then $P = NP$.

Definition 25 (RP). The class RP (Randomized Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input x ,

1. for any $x \in L$ then, the probability that $A(x)$ accepts is at least $1/2$, and
2. for any $x \notin L$ then, the probability that $A(x)$ accepts is 0 (i.e. $A(x)$ rejects).

The choice of the bound for the error probability $1/2$ is arbitrary.

An RP algorithm is a **Monte Carlo** algorithm that can err only for an $x \in L$. This is what we call an one-sided error. The class co-RP errs only for $x \notin L$.

A problem belonging to both RP and co-RP can be solved by a randomized algorithm with zero-sided error i.e. a **Las-Vegas** algorithm.

Definition 26 (ZPP). The class ZPP (Zero-error Probabilistic Polynomial time) is the class of languages that have **Las Vegas** algorithms running in expected polynomial time.

A Las Vegas algorithm is a **Monte Carlo** algorithm for $x \in L$ and a Monte Carlo algorithm for $x \notin L$. It never lies but its running time is probabilistic.

Definition 27 (PP). The class PP (Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input x ,

1. for any $x \in L$ then, the probability that $A(x)$ accepts is greater than $1/2$, and
2. for any $x \notin L$ then, the probability that $A(x)$ accepts is less than $1/2$.

The definition of PP is weak as we have no bound on how far from $1/2$ the two probabilities are and therefore we cannot be sure whether we can reduce the error in a polynomial number of repeated trials. A more useful complexity class is BPP .

Definition 28 (BPP). The class BPP (Bounded-error Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input x ,

1. for any $x \in L$ then, the probability that $A(x)$ accepts is greater than $3/4$, and
2. for any $x \notin L$ then, the probability that $A(x)$ accepts is less than $1/4$.

The $3/4$ and $1/4$ bounds are arbitrary and we can have $1/2 + \epsilon$ and $1/2 - \epsilon$ as long as ϵ is bounded below by the inverse of a polynomial in input size $|x|$.

We sometimes call a BPP algorithm an **Atlantic-city** algorithm.