CS 170: Efficient Algorithms and Intractable Problems

Spring 2017

•

Homework 9

Due on Tuesday, April 18th, 2017 at 11:59am

•

Solutions by

Ninh DO

25949105

In collaboration with

NONE

Problem 1: Basic Complexity Concepts

★ **Level**

a. Suppose we reduce a problem A to another problem B. This means that if we have an algorithm that solves ___, we immediately have an algorithm that solves ___.

Solution
Suppose we reduce a problem A to another problem B. This means that if we have an algorithm that solves B, we immediately have an algorithm that solves A.

b. Again, suppose we show a (polynomial-time) reduction from A to B. This implies that up to polynomial factors, ___ cannot be any easier than ___.

Solution
Again, suppose we show a (polynomial-time) reduction from A to B. This implies that up to polynomial factors, B cannot be any easier than A.

c. Define the class **NP** in one sentence. *There are several correct characterizations. One of them is the most intuitive, and the definition we use in this class. Give this definition.*

Solution
The search problems whose suggested solutions can be checked in polynomial time.

d. Define the class **NP**-hard in one sentence.

Solution
The problems to which all NP problems reduce to. They cannot be solved in polynomial time but can be solved in exponential time, can or cannot be checked in polynomial time.

e. Define the class **NP**-complete in one sentence.

Solution
The problems which are NP and NP-hard

f. Show that for any problem Π in **NP**, there is an algorithm which solves Π in time $\mathcal{O}(2^{p(n)})$, where n is the size of the input instance and $p(n)$ is a polynomial (which may depend on Π).

Solution
Since all problems in NP reduce to NP-complete problem in polynomial time. For example, KNAPSACK problem can be solved in exponential time $O(2^{f(n)})$. Reduction of Π to KNAPSACK takes $g(n)$ time, so the total runtime is $O(g(n)*2^{f(n)})$ which is bounded by $O(2^{g(n)+f(n)}) = O(2^{p(n)})$, for $p(n) = g(n)+f(n)$.

Problem 2: Proving NP-completeness by generalization

★★ Level

Proving **NP**-completeness by generalization. For each of the problems below, prove that it is **NP**-complete by showing that it is a generalization of some **NP**-complete problem we have seen in the lecture or in the book. You only need to show the reduction part in this problem (there is no need to show that the problem is in **NP**).

a. SUBGRAPH ISOMORPHISM: Given as input two undirected graphs G and H, determine whether G is a subgraph of H (that is, whether by deleting certain vertices and edges of H we obtain a graph that is, up to renaming of vertices, identical to G), and if so, return the corresponding mapping of $V(G)$ into $V(H)$.

Solution
This is a generalization of the CLIQUE problem: Given a graph H and a positive integer k, a full graph G of k vertices is a subgraph of H iff it is a k-vertex clique.

b. LONGEST PATH: Given a graph G and an integer g, find in G a simple path of length g.

Solution
This is a generalization of the RUDRATA CYCLE/HAMILTON PATH problem: Given a graph G of all edges 1 and $g = |V| - 1$, there exists a simple path of length g iff there is a Hamilton path.

c. MAX SAT: Given a CNF formula and an integer g, find a truth assignment that satisfies at least g clauses.

Solution
This is a genalization of the 3SAT problem: Given a CNF formula of m clauses and an integer $g = m$, there exists a truth assignment that satisfies at least $g = m$ clauses iff there is a solution to 3SAT.

d. DENSE SUBGRAPH: Given a graph and two integers a and b, find a set of a vertices of G such that there are at least b edges between them.

Solution
This is a generalization of the CLIQUE problem: Given a graph G and two integers a and $b = a(a-1)/2$, there exists a set of a vertices $V' \subset V$ with $b = a(a-1)/2$ edges between them iff there is a a-vertex clique.

e. SPARSE SUBGRAPH: Given a graph and two integers a and b, find a set of vertices of G such that there are at most b edges between them.

Solution
This is a generalization of the INDEPENDENT SET problem: Given a graph G and two integers a and $b = 0$, there exists a set of a vertices $V' \subset V$ with $b = 0$ edge between them iff there is a a-vertex independent set.

Problem 3: Reduction, Reduction, Reduction

★★★ **Level**

Instruction: In this problem, you will prove three problems are **NP**-Complete. Please be aware that to prove a problem is **NP**-Complete, you need to argue it is in **NP** and then briefly justify a reduction that shows it is **NP**-hard.

1. Alice and Bob go out on a date at a nice restaurant. At the end of the meal, Alice has eaten c_A dollars worth of food, and has in her wallet a set of bills $A = \{a_1, a_2, \ldots, a_n\}$. Similarly, Bob owes the restaurant c_B dollars and has bills $B = \{b_1, b_2, \ldots, b_m\}$. Now, Alice and Bob are very calculating people, so they agree that each of them should pay their fair share (c_A and c_B, respectively).

   One thing they don't mind doing, however, is fairly trading bills. That is, Alice can exchange a subset of $A' \subseteq A$ of her bills for a subset $B' \subseteq B$ of Bob's bills, so long as $\sum_{a \in A'} a = \sum_{b \in B'} b$. Under the above conditions, Alice and Bob wish to find, after trading as many times as desired, subsets of their bills A^*, B^* such that $\sum_{a \in A^*} a = c_A$ and $\sum_{b \in B^*} b = c_B$.

   Show that Fair Date is **NP**-complete.

   Hint: You may assume SUBSET SUM is **NP**-complete.

   Solution
   NP: Given a solution, two sets of bills from A and B, we can verify if their sums make c_A and c_B, respectively, in linear time. Then we can check back if the trading was fair by identifying these bills that are not in the original A and B sets. These bills are the result of trading and their sums should be equal. This verification can be done in polynomial time. Thus, the problem is NP.
   NP-hard: The SUBSET SUM problem reduces to this problem in the way that if we set the set B empty and we let B eat nothing, then Alice has to take a subset of bills, from her set A, whose amount exactly equal to c_A. Similar to Bob if Alice eats nothing and has no bills in her set A. Thus the problem is NP-hard.
   The problem is both NP and NP-hard, so it is NP-complete.

2. Consider the following problem Public Funds:

   > You are looking to build a new fence for your mansion, to keep out pesky people protesting profligate purchases. You have m bank account at your disposal to use to pay for your fence; each account i has a balance of b_i. You must choose one of n options for your fence; each fence j costs c_j dollars. You would like to withdraw from at most k of the bank accounts to build the fence, and due to peculiar UC accounting rules, if you use a particular bank account, you must use the whole balance (all b_m dollars.)
   >
   > Determine whether it is possible to exactly pay for some fence j; that is, whether there is a j between 1 and n such that you can withdraw exactly c_j dollars given the bank account balances $b_1, \ldots, b_m$, the fence costs $c_1, \ldots, c_n$ and k, and if so return the corresponding choice of fence and set of of bank accounts that you withdraw from.

   Show that Public Funds is **NP**-Complete.

   Solution **NP:** Given a solution, we can easily verify that: 1. it forms a subset of m balances (quadratic time), 2. their sum is equal to some c_j (linear time), 3. the number of balance is at most k (constant or linear time). In other words, the solution can be verified in polynomial time. Thus, the problem is NP.
   NP-hard: the SUBSET SUM problem reduces to this problem in the way that if we narrow down the set of fence into one fence c and we are allow to withdraw at most $k = m$ of the bank accounts from total m accounts, then the problem is equivalent to the SUBSET SUM problem: find a subset of the set of m bank accounts whose sum is equal c, the value of the fence. Thus, there exists a solution for this SUBSET SUM problem iff there is a solution for PUBLIC FUND problem with $k = m$ and the set of fences has 1 element. This reduces the SUBSET SUM problem to PUBLIC FUND problem, and since the former is NP-complete, the latter is either NP-complete or NP-hard.
   The PUBLIC SUM problem is both NP and NP-hard, so it is NP-complete.

3. Consider the search problem MAX-ACYCLIC-INDUCED-SUBGRAPH:

   INPUT: A *directed* graph $G = (V, E)$, and a positive integer k.

   OUTPUT: A subset $S \subseteq V$ of size k such that the graph G_S obtained from G by keeping exactly those edges both whose endpoints are in S is a DAG.

   Show that MAX-ACYCLIC-INDUCED-SUBGRAPH is **NP**-complete.

Solution **NP:** Given a solution, we can verify if it forms a k-vertex subset of V in polynomial time (either linear or quadratic or any in between), and if the graph G_S is a DAG in linear time. Thus, the solution can be verified in polynomial time. It is NP.

NP-hard: The INDEPENDENT SET problem reduces to this problem in the way that given an undirected graph G, we can make it directed by replacing every edge by two oppositely directed edges, called graph G'. Then we will see G has k-vertex independent set iff G' has DAG. Since, if there is an edge between two vertices in G', there is a directed cycle between them and they are also not in an independent set. Thus, there must not be any edge between any two vertices in the independent set that forms a DAG. The MAX-ACYCLIC-INDUCED-SUBGRAPH is NP-hard.

The MAX-ACYCLIC-INDUCED-SUBGRAPH problem is both NP and NP-hard, so it is NP-complete.

Problem 4: Graphs and Matrix Multiplication.

★★★★ Level

Consider the MATRIX MULTIPLICATION problem:

Input: Two $n \times n$ matrices M_1 and M_2.
Output: The matrix product $M_1 M_2$.

Suppose there is an algorithm which solves MATRIX MULTIPLICATION in time $t(n)$.

(a) Show that given M and an integer $p > 0$, you can computer M^p in time $\mathcal{O}(t(n) \log p)$. You only need to prove a main idea and show how to satisfy the time complexity requirement.

Solution
Main idea:

$$M^p = \begin{cases} \left(M^{\frac{p}{2}}\right)^2 & \text{if } p \text{ even} \\ \left(M^{\lfloor \frac{p}{2} \rfloor}\right)^2 \cdot M & \text{if } p \text{ odd} \end{cases}$$

Use devide-and-conquer: recursively divide the power by half, we can build a tree of height $\log p$. In best case $p = 2^k$ we perform $\log p$ matrix multiplications, each with $t(n)$ time. In worst case $p = 2^k - 1$ we perform $2 \log p$ matrix multiplications, each with $t(n)$ time. Therefore, the total runtime is $O(t(n) \log p)$.
Proof of correctness:
It is obvious, we use recursion without magic, based on the equation above. It is correct.
Run time:
$O(t(n) \log p)$ as above.

(b) Given a directed graph G in adjacency matrix representation and two nodes s and t, you want to know whether there is a path from s to t. This is callsed the GRAPH REACHABILITY problem. Show that you can solve GRAPH REACHABILITY in time $\mathcal{O}(t(n) \log n)$, where n is the number of nodes in the graph. You need to first give the main idea, and justify the time complexity requirement. Also you need to prove that your algorithm is correct.
Hint 1: It may help to use part (a).
Hint 2: In matrix multiplication, what does $(A^k)_{ij}$ mean?

Solution
Main idea:
We call s, t start and terminal, as well as the matrix indices of s row and t column, respectively.
We perform matrix multiplication in binary search style, i.e. A^2, A^4, A^8, etc. until either $\left(A^k\right)_{st} \neq 0$ or $\left(A^k\right)_{s:} = 0's$ (s row of A^k has all zeros). If it is the former case, the algorithm terminates, there is a path from s to t. If it is the latter case, we know that we might have passed the order where $\left(A^k\right)_{st} \neq 0$, we have to back-track in binary search style to find that order (or maybe there is no such an order). The way we back-track is we jump to the middle of the last interval. For example, if any $\left(\left(A^{16}\right)_{s:}\right) \neq 0$ and $\left(A^{32}\right)_{s:} = 0's$, we check $\left(A^{24}\right)_{s:}$. If any $\left(\left(A^{24}\right)_{s:}\right) \neq 0$, we check $\left(A^{28}\right)_{s:}$; otherwise, we check $\left(A^{20}\right)_{s:}$, and so on. When we find $\left(A^k\right)_{st} \neq 0$ or there is no such k, we terminate our algorithm.
Again, if there exists any k that $\left(A^k\right)_{st} \neq 0$, there is a path from s to t. Otherwise, there is not.
Proof of correctness:
We observe that $\left(A^2\right)_{ij}$ indicates if there is a path of two edges from i to j, $\left(A^3\right)_{ij}$ indicates if there is a path of three edges from i to j, ..., $\left(A^k\right)_{ij}$ indicates if there is a path of k edges from i to j.
Back to our algorithm, instead of sweeping from 1 to n, we jump in binary search way to reduce the runtime. If we jump over past the value k, we jump back in binary search way again. This guarantees that we will not miss the interesting value k while still reduce the runtime from a factor of n to a factor of $\log n$ **Run time:**
Since we use binary search strategy, and there are maximum n vertices, the number of matrix multipli-

Solution (cont.)

cation is $\log n$. Each matrix multiplication takes $t(n)$ time, so the total runtime is $t(n) \log n$

(c) Now, suppose you know that graph reachability cannot be solved in time $\mathcal{O}(T(n) \log n)$. Show that MATRIX MULTIPLICATION cannot be solved in time $\mathcal{O}(T(n))$. This should be a very simple proof.

Solution

We use contradiction: assume that matrix multiplication can be solved in time $\mathcal{O}(T(n))$, then the graph reachability can be solved in $\mathcal{O}(T(n) \log n)$ according to part (b). This contradicts the fact that graph reachability cannot be solved in time $\mathcal{O}(T(n) \log n)$. Thus, matrix multiplication can be solved in time $\mathcal{O}(T(n))$.

Problem 5: Finding Zero(s)

★★★★★ **Level**

Consider the problem INTEGER-ZEROS.

INPUT: A multivariate polynomial $P(x_1, x_2, x_3, \ldots, x_n)$ with integer coefficients, specified as a sum of monomials.
OUTPUT: Integers $a_1, a_2, \ldots, a_n$ such that $P(a_1, a_2, a_3, \ldots, a_n) = 0$.

Show that 3-SAT reduces in polynomial time to INTEGER-ZEROS. (You do not need to show that INTEGER-ZEROS is in **NP**: in fact, it is know *not* to be in **NP**).
Hint 1: Given a 3-SAT formula ϕ in the variables $x_1, x_2, \ldots, x_n$, your reduction f will produce a polynomial P in the same variables such that satisfying assignments correspond to 0, 1 valued zeros of P.
Hint 2: If your polynomial constructed above has exponential amount of terms, it is not optimal! You need to reduce it to polynomial amount. Think about the equality $a^2 + b^2 = 0$ if and only if $a = b = 0$ when a, b are real to achieve it. [1]

Solution
Consider a special case of the INTEGER-ZEROS problem that all variable $x_1, x_2, \cdots, x_n$ take value either 0 or 1. If we specify that 0 indicates true and 1 indicates false, we will see that each monomial is 0 if at least one factor is 0. Thus, each monomial is equivalent to a clause of disjunction of a number of variables.
Each monomial has non-negative value, so the sum of all monomials, or the value of polynomial, is 0 (true) iff all monomials is 0 (true, or satisfied). Thus, the polinomial is equivalent to the Boolean formula.
In conclusion, given a multivariable polynomial P of $x_1, x_2, \cdots, x_n$ taking either 0 or 1, there exists a set $a_1, a_2, \cdots, a_n$ such that $P(a_1, a_2, \cdots, a_n) = 0$ iff there is a satisfying truth assisgnment for the equivalent 3SAT problem.

[1] Fun fact: This problem INTEGER-ZEROS is *really* hard: the decision version of the problem is *undecidable*. This means that there is provably no algorithm which, given a multivariate polynomial, will decide correctly whether or not it has integer zeros. However, if we relax the problem to ask if there are *real* numbers at which the given polynomial vanishes, then the problem surprisingly becomes decidable! For Blue and Gold jingoists: the above decidability result was proven by Alfred Tarski, a Berkeley professor, in 1949. The undecidability of INTEGER-ZEROS was proven by Yuri Matiyasevich (at the ripe old age of 23!) in 1970, building upon previous work of another Berkeley professor, Julia Robinson.