CS222 Algorithm Design and Analysis Homework 7

Zhou Litao 518030910407 F1803016

November 26, 2020, Fall Semester

Exercise 1 Show that if an algorithm makes at most a constant number of calls to polynomial-time subroutines and performs an additional amount of work that also takes polynomial time, then it runs in polynomial time. Also show that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.

Solution.

Let k be the number of subroutines. The algorithm starts out with an input data of size n. Each subroutine takes (some of) the available data as an input, performs some steps, then returns some amount of data as an output. Since each subroutine runs in polynomial time, the output must also have a size polynomial in the size of the input. Therefore, we can assume the output data size has a upper bound of $p(n) = n^d$ where d is a constant.

An upper bound of the total data involved in the computation can be expressed as follows. Let $n_0 = n$ be the initial input size and $n_i = n_{i-1} + p(n_{i-1})$ for all $1 \leq i \leq k$. This can be proved through induction. Assume for $i-1$, the true data accumulating size n' is valid ($\leq n_{i-1}$), then for the next call $n_i = n_{i-1} + p(n_{i-1}) \leq n' + p(n') \leq n_{i-1} + p(n_{i-1})$ is still valid due to the monotonicity of $p()$

Then we can use to show that each n_i is polynomial in n., which follows directly since a composition of two polynomials is surely to be polynomial. Now we have n_k as a polynomial to n as the total upper bound of the input size of the subroutines. Since the calls of subroutines are constant, the running time of the subroutines will also be polynomial to n.

For the second part, note that if we have a subroutine whose output is always twice the size of its input, and we call this subroutine n times, starting with input of size 1 and always feeding the previous output back into the subroutine, the final output will have size 2^n. To generate such output, the algorithm is surely to take exponential time.

□

Exercise 2 Given an integer $m \times n$ matrix A and an integer m-vector b, the **0-1 integer programming problem** asks whether there exists an integer n-vector x with elements in the set $\{0, 1\}$ such that $Ax \leq b$. Prove that 0-1 integer programming is NP-complete. (*Hint:* Reduce from 3-CNF-SAT.)

Solution. Given a certificate solution, we can just replace the variables by the solution and check whether each inequality holds, which can be done in polynomial time. The problem is in NP.

For any 3-CNF-SAT problem instance (which is NP-complete) with literals $x_1, \ldots, x_n$, we can reduce it to an integer programming model as follows:

1. The solution will be a n-vector $\mathbf{x} = [x_1, \ldots, x_n]$ representing the assignment of each literal to be true (=1) or false (=0).
2. For each CNF-clause x_i, x_j, x_k, $x_i + x_j + x_k \geq 1$
3. The above inequalities will be solved through 0-1 integer programming.

The integer programming will have a soluiton if and only if the 3-SAT has a solution. Since SAT problem is NP-complete, 0-1 integer programming problem is also NP-complete. □

Exercise 3 Algorithm class is a democratic class. Denote class as a finite set S containing every students. Now students decided to raise a student union $S' \subseteq S$ with $|S'| \leq K$.

As for the members of the union, there are many different opinions. An opinion is a set $S_o \subseteq S$. Note that number of opinions has nothing to do with number of students.

The question is whether there exists such student union $S' \subseteq S$ with $|S'| \leq K$, that S' contains at least one element from each opinion. We call this problem *ELECTION* problem, prove that it is NP-complete.

Solution. First we show the election problem is NP, given a certificate S', we can check for every opinion in S_o whether the intersection of S' and opinion is not empty. The certificate can be verified in polynomial time.

For NP-complete, We can reduce the 3-CNF-SAT to election problem as follows

1. Each literal correspond to an element in S, the student elected will be in the set S', indicating the assignment of the elements in S' will be true.
2. Each clause will correspond to an element in the opinion set.
3. Solve the election problem.

Since all the opinions should have at least one student to be elected, all clauses will have at least one literal to be true. Thus the 3-SAT problem has a solution if and only if the election problem has a solution. The election problem is NP-complete. □

Exercise 4 Not-All-Equal Satisfiability (NAE-SAT) is an extension of SAT where every clause has at least one true literal and at least one false one. NAE-3-SAT is the special case where each clause has exactly 3 literals. Prove that NAE-3-SAT is NP-complete. (*Hint:* reduce 3-SAT to NAE-k-SAT for some $k > 3$ at first)

Solution. Given a NAE-3-SAT certificate, we can simply put the solution into the CNF and check whether it is satisfiable. Thus NAE-3-SAT is in NP.

First we show that 3-SAT can be reduced to NAE-4-SAT. Every (x_i, x_j, x_k) clasue will correspond to a 4-element (x'_i, x'_j, x'_k, y) clasue in the NAE-4-SAT, where x_i will be true if and only if $x'_i \neq y$ and y is a "global" variable for the whole CNF. It can be shown that such transformed NAE-4-SAT is satisfiable if and only if 3-SAT is satisfiable.

1. Assume for any clause in NAE-4-SAT, (x'_i, x'_j, x'_k, y) is satisfiable. Then at least one element is true and at least one literal is false. If $y = true$, we can always find an element in x'_i, x'_j, x'_k to be false, and the corresponding literal in 3-SAT will be true. If $y = false$, we can always find an element in x'_i, x'_j, x'_k to be true, and the corresponding literal in 3-SAT will be true.
2. Conversely, if 3-SAT is satisfiable, we simply let $y = false$, then the corresponding NAE-4-SAT will be satisfiable. Since at least one of the x is true, the NAE property will also be satisfied.

Then we transform the NAE-4-SAT to NAE-3-SAT. For every (x'_i, x'_j, x'_k, y), let the 3-SAT be $(x'_i, x'_j, w) \wedge (x'_k, y, \neg w)$, where w can be selected to make the clause satisfiable. The two caluses are equivalent since both requires that x'_i, x'_j, x'_k, y can't be same and at least one element should be true.

It follows that 3-SAT can be reduced to NAE-3-SAT. Thus NAE-3-SAT is NP-complete.

□

Exercise 5 Amy and Jack had just robbed a bank. They grabbed a bag of money and planned to divide the money. For each of the following scenarios, give a polynomial time algorithm, or prove that the problem is NP-complete. The input in each case is a list of n items in the bag and the value of each item.

1. There are n coins in the bag, but there are only two different denominations: some denominations x dollars, and some denominations y dollars. Amy and Jack want to divide the money evenly.

2. The bag contains n coins, with an arbitrary number of different denominations, but each denomination is a non-negative integer power of 2, i.e., the possible denominations are 1 dollar, 2 dollars, 4 dollars, etc. Amy and Jack wish to divide the money exactly evenly.

3. The bag contains n checks, which are, in an amazing coincidence, made out to Amy and Jack. They wish to divide the checks so that they each get the exact same amount of money.

4. The bag contains n checks as in part (c), but this time Amy and Jack are willing to accept a split in which the difference is no larger than 100 dollars.

Solution.

1. The problem has a polynomial time solution w.r.t. n.
   (a) Assume there are a dollars with denomination x and b dollars with denomination y.
   (b) Check for every (i, j) where $i \leq a$ and $j \leq b$ if $xi + bj = (ax + by)/2$
   (c) We need $a \times b < n \times n \in O(n^2)$ computations

2. The problem has a polynomial time solution.
   (a) Sort the coins from large to small
   (b) Start from the largest denomination, if there are even number of coins remained, assign them evenly to Amy and Jack.
   (c) If there are odd number of coins $(2k+1)$, give Amy k coins and Jack $k+1$ coins. Then give Amy smaller denomination coins until they tie again.
   (d) If there are remaining coins, go back to step 2.
   (e) If there are no more coins, check whether Amy and Jack have equal denomination of coins, which is the solution.

   The algorithm only involves a sorting and traversing, so it is easy to check its polynomial runtime. As for the correctness, consider the case when there are odd number of coins with same denomination. Since the denominations are strictly decreasing in half, we can always make sure that our selection from the largest to small coins will either fail to bridge the gap or exactly fill the gap and go back to step 2. In other words, there exist no possibility that we must skip some large coins in order to bridge the gap with smaller coins. Thus, an invariant that our running assignment will be valid if and only if the problem is satisfiable. The algorithm is correct.

3. The problem is NP because for every certificate, we can check whether the total sum of Amy's checks and Jack's checks are equal within one computation. We can show its NP-completeness by reducing the subset sum problem to this problem. For every subset sum problem with numbers $w_1, \ldots, w_n$ and goal W, we let the check price to be $w_1, \ldots, w_n, 2\sum_i w_i - W, \sum_i w_i + W$. Note the last two numbers can't be assigned to the same person, otherwise the remaining number can't be greater or equal to the sum of the two. Therefore, one person will have the $2\sum_i w_i - W$ check and the other with $\sum_i w_i + W$ check. The person with the $2\sum_i w_i - W$ check will have all the other checks as the solution to subset sum problem. Thus subset sum can be reduced to this check partition problem. Therefore, the problem is NP-complete.

4. The problem is NP because for every certificate, we can check whether the total sum of Amy's checks and Jack's checks are less than 100 in difference within one computation. We can also reduce subset sum problem into this problem. For every subset sum problem with numbers $w_1, \ldots, w_n$ and goal W, we first scale all the numbers by 101. Then we can make sure that the difference between any of the two elements are more than 100. By making the same transformations as 3, it follows that the reduced check partition problem will have a solution only if the scaled values add up to the same value. Therefore, the partial sum problem has a solution if and only if the reduced check partition problem has a solution. The problem is NP-complete.

□