# CS222 Homework 7

Algorithm Analysis & Deadline: 2020-11-29 Wednesday 23:59

Exercises for Algorithm Design and Analysis by Li Jiang, 2020 Autumn Semester

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.**

   1) All the subroutines called are polynomial-time, and hence have a upper bound $T(subroutines) \leq n^d$, where $d$ is a fixed constant positive integer. Then the input size of $i-th$ call is bounded by $Input_i \leq n^{d^{i-1}}$. Assume we calls $k$ subroutines, where $k$ is a fixed constant positive integer. The total time is $T(n) \leq \sum_{i=1}^{k} n^{d^i} + T_{additional}(n) \leq n^{d^k+1} + T_{additional}(n)$, which is obviously a polynomial time.
   2) Assume there is a subroutine which take in $s$ numbers and output $2s$ numbers. Obviously this subroutine is polynomial-time. And assume our input is $n$ numbers, and we call the subroutines for $n$ times (a polynomial number of calls) in the way that the last call's output is the current call's input. Then we finally get a output with size $n \cdot 2^n$, which infers that the algorithm is exponential-time.

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.**

   1) **NP:** Obviously, this **0-1 integer programming problem** has a poly-time certificate algorithm with poly-size certifier.
   2) **NP-complete:** Just to show **3-SAT**$\leq_P$**0-1 integer programming problem**: For every variable in **3-SAT** we construct two variables $V(a) = a_1, V(\bar{a}) = a_0$, and construct two constraints: $V(a)+V(\bar{a}) \leq 1$ and $-V(a)-V(\bar{a}) \leq -1$. For every clause $(x\vee y\vee z)$ in $\Phi$, we construct a constraint: $-V(x)-V(y)-V(z) \leq -1$. In this way we get an **0-1 integer programming problem** and it has a solution if and only if the original **3-SAT** problem has a solution.

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.**

   1) **NP:** This problem can be checked in a poly-time certificate algorithm ($O(|Opinions||S|K)$) with a poly-size ($O(K)$) certifier.
   2) **NP-complete:** First we show that: **Set Cover problem** $\leq_p$ **ELECTION problem**:

- For the given set $U$, we construct a opinion $op_e = \emptyset$ for every element $e$ in $U$.
- For every subset $S_i$ we construct a student $i$. For every element $e$ in $S_i$, we add the student $i$ into the corresponding opinion $op_e$, i.e. $op_e \leftarrow op_e \cup \{i\}$.
- **Set Cover** has a solution $|\{S_{j_1}, S_{j_2}, \cdots, S_{j_k}\}| = k \leq K \Rightarrow$ **ELECTION problem** has a solution union $S' = \{j_1, j_2, \cdots, j_k\}, |S'| \leq K$. Because for every $e \in U$, $\exists S_j \in \{S_{j_1}, \cdots, S_{j_k}\}, e \in S_j \Rightarrow$ for every opinion $op_e$, exists $j \in$ union $S', j \in op_e$, i.e. $j$ is from opinion $op_e$.
- **ELECTION problem** has a solution union $S' = \{j_1, j_2, \cdots, j_k\}, |S'| \leq K \Rightarrow$ the corresponding **Set Cover** has a solution $\{S_{j_1}, S_{j_2}, \cdots, S_{j_k}\}$ with size no more than $K$. The reason is almost the same as the one showed above.

Then we have **3-SAT** $\leq_P$ **INDEPENDENT-SET** $\equiv_P$ **VERTEX-COVER** $\leq_P$ **Set Cover problem** $\leq_P$ **ELECTION problem**.

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.**
**Lemma** If $x$ is a solution for a NAE-$k$-SAT problem $P$, and we invert the assignment of variables in $x$ to get a dual solution $\bar{x}$, then $\bar{x}$ is also a solution for this NAE-$k$-SAT problem $P$.
**Proof.** Obviously by definition of NAE-$k$-SAT problem.

1) **NP: NAE-3-SAT** can be checked by a poly-time certificate algorithm with a poly-size certifier.
2) To show **3-SAT**$\leq_P$**NAE-4-SAT**:
   - We construct an additional $\{0, 1\}$ variable $v$.
   - For every clause $(a_i \vee b_i \vee c_i)$ in **3-SAT**, we construct a corresponding **NAE-4-SAT** clause $R_4(a_i, b_i, c_i, v)$.
   - The **3-SAT** problem has a solution $\Rightarrow$ the **NAE-4-SAT** problem has a solution. Reason: just set $v = 0$.
   - The **NAE-4-SAT** problem has a solution $\Rightarrow$ the **3-SAT** problem has a solution. Reason: If $v = 0$ in this solution, then the solution satisfies the constraints in **3-SAT**. If $v = 1$ in this solution, then by the **Lemma** above, we just invert the assignment of variables in this solution.
3) To show **NAE-4-SAT**$\leq_P$**NAE-3-SAT**:
   - For every clause $R_4(a, b, c, d)$ in **NAE-4-SAT** problem, we construct a group of corresponding **NAE-3-SAT** clauses $R_3(a, b, x) \wedge R_3(b, c, y) \wedge R_3(c, d, z) \wedge R_3(x, y, z)$, where $x, y, z$ are three new variables. (For every clause we create 3 different new variables.)
   - $\exists a, b, c, d, R_4(a, b, c, d) = 1 \Leftrightarrow \exists a, b, c, d, x, y, z, R_3(a, b, x) \wedge R_3(b, c, y) \wedge R_3(c, d, z) \wedge R_3(x, y, z) = 1$. Reason: (1) $\Rightarrow$: $a, b, c, d$ are not all equal, then there are at least a group of neighbors which are different. Without loss of generality, we say $a \neq b$. Then let $y = \bar{b}, z = \bar{c}, x = \bar{y}$, and get a solution. (2) $\Leftarrow$: There are at least a 1 and a 0 in $x, y, z$. A 1 means that there are at least a 0 in $a, b, c, d$, and a 0 means that there are at least a 1 in $a, b, c, d$, so the $R_4(a, b, c, d) = 1$.

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.

   (a) 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.

   (b) 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.

   (c) 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.

   (d) 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.**

(a) I assume that $x, y$ are all positive rational numbers (it seems not reasonble that the denomination of coins to be irrational numbers). It can be solve with a poly-time algorithm:
Denote there are $A$ coins with denomination $x$, $B$ coins with denomination $y$. We need to find a non-negative integer solution $(a, b)$ s.t. $ax + by = (A - a)x + (B - b)y$, i.e. $\frac{A - 2a}{2b - B} = \frac{y}{x}$. We transform $\frac{y}{x}$ to the irreducible fraction $\frac{y_0}{x_0}$, and this process can be done within polynomial time (finding gcd is $O(\log x)$, linear to the input length). Now we need to solve $\frac{A - 2a}{2b - B} = \frac{y_0}{x_0}$. It has a solution if and only if:

| $y_0$ | $x_0$ | $A$ | $B$ | criterion |
|---|---|---|---|---|
| odd | odd | odd | odd | $A \geq y_0 \land B \geq x_0$ |
| odd | odd | even | odd | False |
| odd | odd | odd | even | False |
| odd | odd | even | even | $A \geq 2y_0 \land B \geq 2x_0$ |
| odd | even | odd | odd | False |
| odd | even | odd | even | $A \geq y_0 \land B \geq x_0$ |
| odd | even | even | odd | False |
| odd | even | even | even | $A \geq 2y_0 \land B \geq 2x_0$ |
| even | odd | odd | odd | False |
| even | odd | odd | even | False |
| even | odd | even | odd | $A \geq y_0 \land B \geq x_0$ |
| even | odd | even | even | $A \geq 2y_0 \land B \geq 2x_0$ |

(b) Can be solved within polynomial time.
Denote the the set of all the coins as $U$. $S \subseteq U$ is a subset of all the coins. $Sum(S)$ represents the total amount of the coins in $S$. $X$ is the goal amount we want to reach, and a coin $c^*$ has the maximum denomination $2^m$ less than or equal to the goal amount $X$. We assert that:
**Lemma :** $\exists S_1 \subseteq U$ s.t. $Sum(S_1) = X \Leftrightarrow \exists S_2 \subseteq U$ s.t. $c^* \in S_2 \land Sum(S_2) = X$.
Proof:

- $\Leftarrow$: Obviously.
- $\Rightarrow$: Consider two cases:
  * $c^* \in S_1$, go straight.
  * $c^* \notin S_1$. We have $Sum(S_1) = X = 2^m + l$, where $l \geq 0$. We find a coin $c_{m_1} \in S_1$ which has the minimum denomination $2^a$ in $S_1$, $2^a \leq 2^m$. We have $0 \equiv Sum(S_1) \equiv 2^m + l \equiv l \mod 2^a$. If $l > 0$, then $l \geq 2^a$, and then we have $Sum(S_1 \setminus c_{m_1}) = 2^m + l'$, where $l' = l - 2^a \geq 0$. If $l' > 0$, repeat the process above (select and remove the minimum coin $c_{m_2}$ in $S_1 \setminus c_{m_1}$). Finally we will get $Sum(S_1 \setminus c_{m_1} \setminus c_{m_2} \setminus \cdots \setminus c_{m_k}) = 2^m + 0$. We construct a solution $S_2 = \{c_{m_1}, c_{m_2}, \cdots, c_{m_k}, c^*\}$.

This lemma confirms the correctness of the greedy algorithm below.

1. To check is there a selection (a set of coins) to make the coins' total amount equal to $X$.
2. Sort the coins in descending order of denomination.
3. Discuss each coin in the order. If the coin $c$ has a denomination $2^k \leq X$, then select this coin and $X \leftarrow X - 2^k$.
4. Finally, we check whether $X = 0$. If $X = 0$, there is a solution, otherwise no solution.

(c) NP-complete. Proof: **Subset Sum** $\leq_P$ **(c)**.
Reason: For a **Subset Sum** problem, we have a set of numbers $n_1, n_2, \cdots, n_m$ with total sum $S$, and want to find a subset whose numbers have a sum $S_1$. We construct a problem (c) with checks' amounts $n_1, n_2, \cdots, n_m, |S - 2S_1|$, and then the two problem have a solution concurrently.

(d) NP-complete. Proof: **(c)** $\leq_P$ **(d)**.
Reason: For a problem (c) with check amounts $c_1, c_2, \cdots, c_n$, we construct a problem (d) with check amounts $200c_1, 200c_2, \cdots, 200c_n$. They have a solution concurrently.

**Remark**: You need to upload your .pdf file and write the pseudocode.