CS101 Algorithms and Data Structures
Fall 2023
Homework 12

Due date: 23:59, January 16th, 2024

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. When submitting, match your solutions to the problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.
8. You are recommended to finish this homework with LaTeX.

1. **(0 points) Tutorial on how to prove that a particular problem is in NP-Complete**
   To prove problem A is in NP-Complete, your answer should include:

   1. Prove that problem A is in NP by showing:
      (a) What your polynomial-size certificate is.
      (b) What your polynomial-time certifier is.
   2. Choose a problem B in NP-Complete to reduce from.
   3. Construct your polynomial-time many-one reduction f that maps instances of problem A to instances of problem B.
      (polynomial-time many-one reduction = polynomial transformation = Karp reduction, see presenter notes of page 7 & 61 in lecture slides (.pptx file) for more details.)
   4. Prove the correctness of your reduction (i.e. Prove that your reduction f do map yes-instance of problem A to yes-instance of problem B and map no-instance of problem A to no-instance of problem B) by showing:
      (a) x is a yes-instance of problem $A \Rightarrow f(x)$ is a yes-instance of problem B.
      (b) x is a yes-instance of problem $A \Leftarrow f(x)$ is a yes-instance of problem B.
      (The statement above is the contrapositive of the statement "x is a no-instance of problem $A \Rightarrow f(x)$ is a no-instance of problem B.". These two statements are logically equivalent, but the one listed above would be much eaiser to prove.)

Proof Example

Prove that the decision version of Set-Cover is in NP-Complete. Recall that the yes-instances of the decision version of Set-Cover is:

$$\text{Set-Cover} = \left\{ \langle U, S_1, \dots, S_n, k \rangle \;\middle|\; \begin{array}{l} n \in \mathbb{Z}^+, S_1, \dots, S_n \subseteq U \text{ and there exist } k \text{ sets } S_{i_1}, \dots, \\ S_{i_k} \text{that cover all of } U, \text{ i.e., } S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_k} = U \end{array} \right\}$$

1. Our certificate and certifier for Set-Cover goes as follows:
   (a) A set of indices $\{i_1, \dots, i_k\} \subseteq \{1, 2, \dots, n\}$, whose size is polynomial of input size .
   (b) Check whether $S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_k} = U$, whose run-time is polynomial of input size.
2. We choose the decision version of Vertex-Cover to reduce from. Recall that the yes-instances of the decision version of Vertex-Cover is:

$$\text{Vertex-Cover} = \left\{ \langle G, k' \rangle \;\middle|\; \begin{array}{l} G \text{ is an undirected graph and there exists a set of} \\ k' \text{ vertices that touches all edges in } G. \end{array} \right\}$$

   (We use k' here because k has already appeared before.)
3. Given an undirected graph $G = (V, E)$ and an postive integer $k' \in \mathbb{Z}^+$, we construct $f(\langle G, k' \rangle) = \langle U, S_1, \dots, S_n, k \rangle$ as follows:
   (a) $U = E$, which represents the edges from the graph.
   (b) Define $m = |V|$ and let $n = m$, which means the number of sets equals the number of vertices in G.

(c) Label elements in V as $V = \{v_1, v_2, \ldots, v_m\}$. For each $i \in \{1, \ldots, m\}$, the set S_i is defined as $S_i = \{e \in E \mid e = (v_i, u) \text{ for some } u \in V \setminus \{v\}\}$. In other word, S_i is the set of edges incident to v_i.

(d) $k = k'$.

Our reduction takes polynomial time because:

(a) Generating each S_i just takes polynomial time since each edge is visited twice (once for each endpoint).

(b) Generating U and k trivially takes polynomial time because they are copied directly from the input.

4. Then we prove the correctness of our reduction as follows:

(a) "$\Rightarrow$": Let $\langle G, k' \rangle$ be a yes-instance of Vertex-Cover and let $V^* = \left\{v_{i_1}, \ldots, v_{i_{k'}}\right\}$ be the choice of k' vertices that form the vertex cover. Then for each $e \in E$, there is some v_{i_j} ($j \in [k']$) that is an endpoint of e, which directly translates to for each $e \in U$, there is some S_{i_j} ($j \in [k']$) containing e by our construction of f. Hence, we claim that the sets $S_{i_1}, \ldots, S_{i_{k'}}$ form a set cover of size $k = k'$ for U.

(b) "$\Leftarrow$": Let $\langle U, S_1, \ldots, S_m, k \rangle$ be a yes-instance of Set-Cover and let $\{S_{i_1}, \ldots, S_{i_k}\}$ be the choice of k sets that form the set cover. Then for each $e \in U$, there is some S_{i_j} ($j \in [k]$) that contains e, which directly translates to for each $e \in E$, there is some v_{i_j} ($j \in [k]$) that is an endpoint of e. Hence, we claim that the sets $\{v_{i_1}, \ldots, v_{i_k}\}$ form a vertex cover of size $k' = k$ for G.

Hence, the decision version of Set-Cover is in NP-Complete.

2. (18 points) Multiple / Single Choice(s)

This part consists of multiple choices questions ((a)-(d)) and single choice questions ((e)-(g)).

For each multiple choices question ((a)-(d)), there may be **one or more** correct choice(s). Select all the correct answer(s). For each such question, you will get 0 points if you select **any** wrong choice, but you will get 1 point if you select a non-empty subset of the correct choices.

For each single choice question ((e)-(g)), there is **exactly one** correct choice.

Write your answers in the following table.

2(a)	2(b)	2(c)	2(d)	2(e)	2(f)	2(g)
BD	C	AB	CD	D	A	B

(a) (3') A problem in NP is NP-Complete if:

A. It can be reduced to any other NP problem in polynomial time.

B. Any other NP problem can be reduced to it in polynomial time.

C. It can be reduced to another NP-Complete problem in polynomial time.

D. There exists another NP-Complete problem which can be reduced to it in polynomial time.

Solution:

B. Definition of NP-Complete in slide page 69.

D. Proposition regarding NP-Complete in slide page 76.

(b) (3') Given two decision problems A and B such that there exists a polynomial-time many-one reduction from A to B. Which of the following statements must be true?

A. $A \in P \implies B \in P$

B. $A \in \text{NP-Complete} \implies B \in \text{NP-Complete}$.

C. $B \in P \implies A \in P$**.**

D. $B \in \text{NP-Complete} \implies A \in \text{NP-Complete}$.

Solution:

B. If a problem is to be in NP-Complete, it must first be in NP. However, in this case, since B is not necessarily in NP, you can't conclude that $B \in \text{NP-Complete}$.

C. Slides page 11 (intractability: quiz 1).

(c) (3') Which of the following statements are true?

A. The "N" in NP stands for "nondeterministic", instead of "not".

B. According to Cook-Levin Theorem, any problem in NP can be reduced to Circuit-SAT in polynomial time.

C. k-SAT $\in$ NP-Complete for any positive integer $k \geq 2$.

D. Consider the optimization version of Knapsack problem with $n \in \mathbb{Z}^+$ items and $W \in \mathbb{Z}^+$ where the weight and value of each item is $w_i \in \mathbb{Z}^+$ and $v_i \in \mathbb{R}^+$

respectively. Since there is a dynamic programming algorithm for this problem that runs in $O(nW)$ time, we may deduce that we can solve this problem in polynomial time.

Solution:

A. Slides page 56 (intractability: quiz 6).

B. Slides page 83 (implications of Cook-Levin).

C. $\mathsf{2\text{-}SAT} \in \mathsf{P}$ (slides page 5).

D. In this case, W is not necessarily polynomial of n (i.e. W is not necessarily in $\mathsf{poly}(n)$). Hence, $O(nW)$ is not necessarily a subset of $\mathsf{poly}(n)$.

(By the way, this algorithm actually runs in **pseudo-polynomial time**.)

(d) (3') Which of the following statements are true?

A. If you find a polynomial time algorithm for a problem in NP, then you have proved $\mathsf{P} = \mathsf{NP}$.

B. If you prove that $\mathsf{4\text{-}Color}$ can be reduced to $\mathsf{3\text{-}Color}$ in polynomial time, then you've proved $\mathsf{P} = \mathsf{NP}$.

C. $\mathsf{P} \neq \mathsf{NP}$ **if and only if** $\mathsf{P} \cap \mathsf{NP\text{-}Complete} = \emptyset$**.**

D. $\mathsf{P} = \mathsf{NP}$ **if and only if** $\mathsf{NP} = \mathsf{NP\text{-}Complete}$**.**

Hint: In fact, any two problems in P reduces to each other in polynomial time. Intuitively, you may interpret this as the fact that all the problems in P share the same "hardness". A more formal explanation goes as follows:

For simplicity, we only consider those **decision** problem in P here. Let A and B be any two decision problems in P and we want to show that A can be reduced to B in polynomial time. Given an instance x of A, our reduction goes as follows:

1. Prepare two copies of data containing a yes-instance of B and a no-instance of B respectively, which can be done in polynomial time since $B \in \mathsf{P}$.
2. Determine whether x is a yes-instance of A or not, which can be done in polynomial time since $A \in \mathsf{P}$.
3. If the answer is yes, then we return the copy containing a yes-instance of B. Otherwise, we return the copy containing a no-instance of B.

More specifically, let's see how this idea works in the case where:

1. A is "Given an undirected weighted graph $G = (V, E, \langle w_e \mid e \in E \rangle)$ and $c \in \mathbb{R}$, does the minimum spanning tree of it have cost no more than c?"
2. B is "Given an undirected weighted graph $G' = (V', E', \langle w'_e \mid e \in E' \rangle)$ with no negative-cost cycles and $c' \in \mathbb{R}$, does the shortest path between any pairs of vertices of G' have cost no more than c'?"

Given an undirected weighted graph G (an instance of A):

1. Prepare two graphs G_1 and G_2 where G_1 satisfies the shortest path between any pairs of vertices of G_1 has cost no more than c' and G_2 satisfies there exists a pair of vertices of G_2 such that the shortest path between them has cost more than c'.

2. Find the minimum spanning tree of G with Kruskal's algorithm and compare the answer with c.
3. If the answer is no more than c, then we return G_1. Otherwise, we return G_2.

Solution:

A. First, we definitely have a polynomial-time algorithm for the problem Minimum-Spanning-Tree, which is in P. Second, by slides page 58 (P, NP, and EXP), $\mathsf{P} \subseteq \mathsf{NP}$, which indicates that Minimum-Spanning-Tree is in NP. Hence, we have already found a polynomial time algorithm for the problem Minimum-Spanning-Tree, which is in NP. However, this doesn't imply $\mathsf{P} = \mathsf{NP}$.

   By the way, this statement would be true if you modify it as follows:

   (a) If you can find a polynomial time algorithm for **any** problem in NP, then you have proved $\mathsf{P} = \mathsf{NP}$.

   (b) If you find a polynomial time algorithm for a problem in NP-Complete, then you have proved $\mathsf{P} = \mathsf{NP}$.

B. Theorem: Any two problems in NP-Complete reduces to each other in poly-time.

   Proof:

   Given any two problems A and B that are in NP-Complete, we want to show that not only A reduces to B in poly-time, but also B reduces to A in poly-time:

   First, by Cook-Levin Theorem, we claim that A reduces to Circuit-SAT in poly-time. Second, by implication of Karp's work (slides page 82), we claim that Circuit-SAT reduces to B in poly-time. By transitivity of reduction (slides page 31), we deduce that A reduces to B in poly-time. Similarly, we may also conclude that B reduces to A in poly-time.

   By question 3(c), we claim that 4-Color is in NP-Complete. Hence, we have already proved that 4-Color can be reduced to 3-Color in polynomial time. However, this doesn't imply $\mathsf{P} = \mathsf{NP}$.

   By the way, this statement would be true if you modify it as follows:

   (a) If you prove that 3-Color can be reduced to 2-Color in polynomial time, then you've proved $\mathsf{P} = \mathsf{NP}$.

   This statement is true because:

   (a) First, 2-Color $\in \mathsf{P}$ (it is equivalent to determine whether the graph is a bipartite one).

   (b) Proving that 3-Color can be reduced to 2-Color indicates that you can solve 3-Color within polynomial number of standard computational steps as well as calling the subroutine that solves 2-Color, which is also of polynomial time because 2-Color $\in \mathsf{P}$. In this way, you have found a polynomial time algorithm for 3-Color, an NP-Complete problem, which implies that $\mathsf{P} = \mathsf{NP}$.

C. This statement is equivalent to "$\mathsf{P} = \mathsf{NP}$ if and only if $\mathsf{P} \cap \mathsf{NP\text{-}Complete} \neq \emptyset$", which would be much easier to prove. So we prove the equivalent statement instead as follows:

(a) "$\Rightarrow$": If $\mathsf{P} = \mathsf{NP}$, since $\mathsf{NP\text{-}Complete} \subseteq \mathsf{NP}$ naturally, we may deduce that $\mathsf{NP\text{-}Complete} \subseteq \mathsf{P}$. Hence, $\mathsf{P} \cap \mathsf{NP\text{-}Complete} = \mathsf{NP\text{-}Complete} \neq \emptyset$.

(b) "$\Leftarrow$": If $\mathsf{P} \cap \mathsf{NP\text{-}Complete} \neq \emptyset$, then we would be able to solve some NP-Complete problem in polynomial time. In this way, by definition of NP-Completeness, we would be able to solve all problems in NP in polynomial time, which implies that $\mathsf{P} = \mathsf{NP}$.

D. (a) "$\Rightarrow$": If $\mathsf{P} = \mathsf{NP}$, then any two problems in NP reduces to each other in polynomial time because any two problems in P reduces to each other in polynomial time, which implies that every problem in NP must be in NP-Complete. In this way, we may deduce that $\mathsf{NP} \subseteq \mathsf{NP\text{-}Complete}$. Since $\mathsf{NP\text{-}Complete} \subseteq \mathsf{NP}$ naturally, we conclude that $\mathsf{NP} = \mathsf{NP\text{-}Complete}$.

(b) "$\Leftarrow$": If $\mathsf{NP} = \mathsf{NP\text{-}Complete}$, since $\mathsf{P} \subseteq \mathsf{NP}$ naturally, we may deduce that $\mathsf{P} \subseteq \mathsf{NP\text{-}Complete}$. Hence, $\mathsf{P} \cap \mathsf{NP\text{-}Complete} = \mathsf{P} \neq \emptyset$. By the equivalent statement of choice C, we may conclude that $\mathsf{P} = \mathsf{NP}$.

(e) (2') The statement "Minimum-Spanning-Tree is not in NP-Complete." is ____________?

A. True regardless of whether P equals to NP or not.
B. False regardless of whether P equals to NP or not.
C. True if and only if $\mathsf{P} = \mathsf{NP}$.
D. True if and only if $\mathsf{P} \neq \mathsf{NP}$.

Solution:
First, clearly, $\mathsf{Minimum\text{-}Spanning\text{-}Tree} \in \mathsf{P}$. Then:

1. If $\mathsf{P} = \mathsf{NP}$, then by question 2(d) choice D, $\mathsf{NP\text{-}Complete} = \mathsf{NP} = \mathsf{P}$. Hence, in this case, $\mathsf{Minimum\text{-}Spanning\text{-}Tree} \in \mathsf{NP\text{-}Complete}$.
2. If $\mathsf{P} \neq \mathsf{NP}$, then by question 2(d) choice C, $\mathsf{P} \cap \mathsf{NP\text{-}Complete} = \emptyset$. Hence, in this case $\mathsf{Minimum\text{-}Spanning\text{-}Tree} \notin \mathsf{NP\text{-}Complete}$.

(f) (2') The statement "If problem B is in NP, then for any problem $A \in \mathsf{P}$, A can be reduced to B in polynomial time." is ____________?

A. True regardless of whether P equals to NP or not.
B. False regardless of whether P equals to NP or not.
C. True if and only if $\mathsf{P} = \mathsf{NP}$.
D. True if and only if $\mathsf{P} \neq \mathsf{NP}$.

Solution:
According to the hint in question (d), any two problems in P can be reduced to each

other in polynomial time and by slides page 58 (P, NP, and EXP), $\mathsf{P} \subseteq \mathsf{NP}$ regardless of whether P equals to NP or not.

(g) (2') The statement "There are problems in NP that cannot be solved in exponential time." is ____________?

A. True regardless of whether P equals to NP or not.

B. False regardless of whether P equals to NP or not.

C. True if and only if $\mathsf{P} = \mathsf{NP}$.

D. True if and only if $\mathsf{P} \neq \mathsf{NP}$.

Solution:
By slides page 58 (P, NP, and EXP), $\mathsf{NP} \subseteq \mathsf{EXP}$ regardless of whether P equals to NP or not.

3. (9 points) Reductions

In this question, you are required to construct 3 **correct** direct polynomial-time many-one reduction that respectively:

1. maps instances of Independent-Set to instances of Clique.
2. maps instances of Subset-Sum to instances of Knapsack (decision version).
3. maps instances of 3-color to instances of k-color (k is a **given** positive integer and $k \geq 4$).

In this question, you are **not** required to prove the correctness of your reductions and it suffices to write your **reduction only**. However, make sure your reduction is correct to receive points.

Reminder: Don't forget state that your reduction takes polynomial time!

(a) (3') Consider the following problems:

1.Independet-Set: Given an undirected graph $G = (V, E)$ and a positive integer k, determine whether there exists a subset k (or more) vertices of V such that no two of them are adjacent (i.e. does G contains an independent set of size at least k).

The yes-instances of Independet-Set is:

$$\text{Independet-Set} = \left\{ \langle G, k \rangle \;\middle|\; \begin{array}{l} G = (V, E) \text{ is an undirected graph that contains} \\ k \text{ vertices with no edges between them.} \end{array} \right\}$$

2. Clique: Given an undirected graph $G' = (V', E')$ and a positive integer k', determine whether there exists a subset k' (or more) vertices of V' such that **any** two of them are adjacent (i.e. does G' contains a clique of size at least k').

The yes-instances of Clique is:

$$\text{Clique} = \left\{ \langle G', k' \rangle \;\middle|\; \begin{array}{l} G' = (V', E') \text{ is an undirected graph that contains } k' \\ \text{vertices such that they are connected to each other.} \end{array} \right\}$$

Construct a **correct** direct polynomial-time many-one reduction f_1 that maps instances of Independent-Set to instances of Clique.

Solution:

Given an undirected graph $G = (V, E)$ and an positive integer $k \in \mathbb{Z}^+$, we construct $f_1(\langle G, k \rangle) = \langle G', k' \rangle$ as follows:

1. $G' = \overline{G} \triangleq (V, \overline{E})$ where $\overline{E} \triangleq \{\{u, v\} \mid u, v \in V, \{u, v\} \notin V\}$. ($\overline{G}$ is called the complement graph of G.)
2. $k' = k$.

(b) (3') Consider the following problems:

3.Subset-Sum: Given an array $A = [a_1, a_2, ..., a_m]$ of positive integers and a positive integer k such that $k \leq \sum_{i\in[m]} a_i$, determine whether there exists a subset $S \subseteq [m]$ such that $\sum_{i\in S} a_i = k$ (i.e. determine whether there exists a subset of A such that the sum of its elements is k).

The yes-instances of Subset-Sum is:

$$\text{Subset-Sum} = \left\{ \langle a_1, a_2, \dots, a_m, k \rangle \;\middle|\; \begin{array}{l} m \in \mathbb{Z}^+, a_1, \dots, a_m, k \in \mathbb{Z}^+ \text{ and there} \\ \text{exists a subset of the } a_i\text{'s that sum up} \\ \text{to } k, \text{ i.e. } \exists\, S \subseteq [m] : \sum_{i\in S} a_i = k. \end{array} \right\}$$

4.Knapsack: Given $n \in \mathbb{Z}^+$ items where the weight and value of each item is $w_i \in \mathbb{Z}^+$ and $v_i \in \mathbb{R}^+$ respectively as well as fixed $W \in \mathbb{Z}^+$ and $V \in \mathbb{R}^+$, determine whether there exists a subset $P \subseteq [n]$ such that $\sum_{i\in P} w_i \leq W$ and $\sum_{i\in P} v_i \geq V$ (i.e. determine whether there exists a subset of items such that the sum of their weights is no more than W while the sum of their values is no less than V).

The yes-instances of Knapsack is:

$$\text{Knapsack} = \left\{ \langle w_1, \dots, w_n, v_1, \dots, v_n, W, V \rangle \;\middle|\; \begin{array}{l} n, w_1, \dots, w_n, v_1, \dots, v_n, V, W \in \mathbb{Z}^+ \\ \text{and there exists a subset } P \subseteq [n] \\ \text{such that } \sum_{i\in P} w_i \leq W \text{ and } \sum_{i\in P} v_i \geq V. \end{array} \right\}$$

Construct a **correct** direct polynomial-time many-one reduction f_2 that maps instances of Subset-Sum to instances of Knapsack.

Solution:

Given an array $A = [a_1, a_2, ..., a_m]$ of positive integers and a positive integer k such that $k \leq \sum_{i\in[m]} a_i$, we construct $f_2(\langle a_1, a_2, \dots, a_m, k\rangle) = \langle w_1, \dots, w_n, v_1, \dots, v_n, W, V\rangle$ as follows:

1. $n = m$, which means the number of items equals the number elements in A.
2. $w_i = a_i$ and $v_i = a_i$, $\forall\, i \in [m]$, which means both the value and the weight of the i-th item equal a_i.
3. $W = k$ and $v = k$, which means that both the weight limit and the target value equal k.

(c) (3') Consider the following problems:

5. 3-Color: Given an undirected graph $G = (V, E)$, determine whether its vertices can be colored **within** 3 different colors such that no adjacent nodes have the same color?

The yes-instances of 3-Color is:

$$\text{3-Color} = \left\{ \langle G \rangle \;\middle|\; \begin{array}{l} G = (V, E) \text{ is an undirected graph such that its} \\ \text{vertices can be colored } \textbf{within } 3 \text{ different colors} \\ \text{such that no adjacent nodes have the same color.} \end{array} \right\}$$

6. k-Color: For fixed positive integer $k \geq 4$, given an undirected graph $G' = (V', E')$, determine whether its vertices can be colored **within** k different colors such that no adjacent nodes have the same color?

The yes-instances of k-Color is:

$$\text{k-Color} = \left\{ \langle G' \rangle \;\middle|\; \begin{array}{l} G' = (V', E') \text{ is an undirected graph such that its} \\ \text{vertices can be colored } \textbf{within } k \text{ different colors} \\ \text{such that no adjacent nodes have the same color.} \end{array} \right\}$$

Construct a **correct** direct polynomial-time many-one reduction f_3 that maps instances of 3-Color to instances of k-Color.

Reminder: You may **not** apply mathematical induction for k here. A direct reduction is required.

Solution:

Given an undirected graph $G = (V, E)$, we construct $f_3(\langle G \rangle) = \langle G' \rangle = (V', E')$ as follows:

1. Add $k - 3$ new vertices and define V^* as the set of these newly added vertices.
2. Let $V' = V \cup V^*$.
3. Define $E^* = \{\{u, v\} \mid u \in V, v \in V^*\}$.
4. Define $E^{**} = \{\{u, v\} \mid u, v \in V^*\}$.
5. Let $E' = E \cup E^* \cup E^{**}$

4. (8 points) Equivalent-Partition **is in** NP-Complete

In this question, we will prove that Equivalent-Partition is in NP-Complete.

Equivalent-Partition: Given an array $B = [b_1, b_2, ..., b_n]$ of non-negative integers, determine whether there exists a subset $T \subseteq [n]$ such that $\sum_{i\in T} b_i = \sum_{j\in[n]\setminus T} b_j$ (i.e. determine whether there is a way to partition B into two disjoint subsets such that the sum of the elements in each subset is equivalent).

The *yes*-instances of Equivalent-Partition is:

$$\text{Equivalent-Partition} = \left\{ \langle b_1, \ldots, b_n \rangle \;\middle|\; \begin{array}{l} n \in \mathbb{Z}^+, b_1, \ldots, b_n \in \mathbb{N} \text{ and there exists a} \\ \text{partition of the } b_i\text{'s to two parts whose sums} \\ \text{are equivalent, i.e. } \exists\, T \subseteq [n] : \sum\limits_{i\in T} b_i = \sum\limits_{j\in[n]\setminus T} b_j \end{array} \right\}$$

Based on the tutorial on page 2 and 3, our proof goes as follows:

(a) (2') Prove that Equivalent-Partition is in NP. (Show your certificate and certifier.)

> **Solution:** Our certificate and certifier for Equivalent-Partition goes as follows:
>
> 1. Certificate: A subset of indices $T \subseteq [n]$, whose size is polynomial of input size.
> 2. Certifier: Check whether $\sum_{i\in T} b_i$ equals $\sum_{j\in[n]\setminus T} b_j$, whose run-time is polynomial of input size.

(b) (0') We choose Subset-Sum to reduce from. Recall that the *yes*-instance of Subset-Sum is:

$$\text{Subset-Sum} = \left\{ \langle a_1, a_2, \ldots, a_m, k \rangle \;\middle|\; \begin{array}{l} m \in \mathbb{Z}^+, a_1, \ldots, a_m, k \in \mathbb{Z}^+ \text{ and there} \\ \text{exists a subset of the } a_i\text{'s that sum up} \\ \text{to } k\text{, i.e. } \exists\, S \subseteq [m] : \sum\limits_{i\in S} a_i = k. \end{array} \right\}$$

(c) Construct your polynomial-time many-one reduction f that maps instances of Subset-Sum to instances of Equivalent-Partition.

i. (0') Vixbob proposed a reduction as follows:
Let $n = m$ and $b_i = a_i$ for $\forall\, i \in [m]$. Finally set $k = \frac{1}{2}\sum_{i\in[m]} a_i$. In this way, $\langle a_1, a_2, \ldots, a_m, k\rangle$ is a *yes*-instance of Subset-Sum if and only if $\langle b_1, \ldots, b_n\rangle = \langle a_1, a_2, \ldots, a_m\rangle$ is a *yes*-instance of Equivalent-Partition.
However, this reduction is **wrong**. Why?

> **Solution:**
> Here k is given (fixed) since it's part of the instance of the problem that we want to reduce from (i.e. it's part of the input of your reduction). Thus, you **can't** arbitrarily modify the value of k.

ii. (0') GKxx proposed another reduction as follows:
Define $X = \sum_{i\in[m]} a_i$ and let $n = m + 2$. Then we define our reduction as:

$$\langle b_1, \ldots, b_n\rangle = f(\langle a_1, a_2, \ldots, a_m, k\rangle) \triangleq \langle a_1, a_2, \ldots, a_m, k, X - k\rangle$$

In this way, we may deduce that $\langle a_1, a_2, \ldots, a_m, k\rangle$ is a yes-instance of Subset-Sum if and only if $\langle b_1, \ldots, b_n\rangle = \langle a_1, a_2, \ldots, a_m, k, X - k\rangle$ is a yes-instance of Equivalent-Partition because a subset with sum k can be paired with $X - k$ and the remaining subset with sum $X - k$ can be paired with k, resulting in an equivalent partition.
However, this reduction is **wrong** again. Why?

Solution:
This reduction is not a valid one since the sequence $\langle a_1, a_2, \ldots, a_m, k, X - k\rangle$ **always** has a trivial equivalent partition $\langle a_1, a_2, \ldots, a_m\rangle$ versus $\langle k, X - k\rangle$, which indicates that $f(\langle a_1, a_2, \ldots, a_m, k\rangle)$ is **always** a yes-instance of Equivalent-Partition regardless of whether $\langle a_1, a_2, \ldots, a_m, k\rangle$ is a yes-instance of Subset-Sum or not.

iii. (3') What's your **correct** polynomial-time many-one reduction f that maps instances of Subset-Sum to instances of Equivalent-Partition?
Hint: GKxx's reduction is really close to a correct one. Maybe you can modify it a little bit to make it correct?

Solution:
First, still define $X = \sum_{i\in[m]} a_i$ and let $n = m + 2$. To avoid having k and $X - k$ in the same side of the partition, we add 1 (or any positive number) to both of them. Hence, we define our reduction as

$$\langle b_1, \ldots, b_n\rangle = f(\langle a_1, a_2, \ldots, a_m, k\rangle) \triangleq \langle a_1, a_2, \ldots, a_m, k + 1, X - k + 1\rangle$$

Our reduction takes polynomial time because all computation including computing the sum of all elements, adding 1 and subtracting k can be done in polynomial time.

(d) Prove the correctness of your reduction by showing:

i. (1') x is a yes-instance of Subset-Sum $\Rightarrow$ $f(x)$ is a yes-instance of Equivalent-Partition.

Solution:
Let $\langle a_1, a_2, \ldots, a_m, k\rangle$ be a yes-instance of Subset-Sum and let $S \subseteq [m]$ be the set of indices such that $\sum_{i\in S} a_i = k$. Then we may deduce that $\sum_{i\in[n]\setminus S} a_i = X - k$. Hence:

$$X - k + 1 + \sum_{i\in S} a_i = (X - k + 1) + k = (k + 1) + (X - k) = k + \sum_{i\in[n]\setminus S} a_i$$

which indicates that $f(\langle a_1, a_2, \ldots, a_m, k\rangle) = \langle a_1, a_2, \ldots, a_m, k + 1, X - k + 1\rangle$ is a yes-instance of Equivalent-Partition.

ii. (2') x is a yes-instance of Subset-Sum $\Leftarrow f(x)$ is a yes-instance of Equivalent-Partition.

Solution:
Let $\langle b_1, \ldots, b_n \rangle = f(\langle a_1, a_2, \ldots, a_m, k \rangle) = \langle a_1, a_2, \ldots, a_m, k+1, X-k+1 \rangle$ be a yes-instance of Equivalent-Partition and let $T \subseteq [n]$ be the set of indices such that $\sum_{i \in T} b_i = \sum_{j \in [n] \setminus T} b_j$. Note that $\sum_{i \in [n]} b_i = (k+1) + (X-k+1) + \sum_{i \in [m]} a_i = (k+1)+(X-k+1)+X = 2X+2$, we may deduce that $\sum_{i \in T} b_i = \frac{1}{2}(2X+2) = X+1$. Since all $b_i \geq 0, \forall\, i \in [n]$ while $(k+1)+(X-k+1) > X+1$, we may deduce that $k+1$ and $X-k+1$ cannot be on the same side of the partition. Thus, we may conclude that $X-k+1$ is paired with a subset of element from $a_1, a_2, \ldots, a_m$ such that their overall sum is $X+1$, which indicates that $\exists\, S \subseteq [m]$ such that $\sum_{i \in S} a_i = (X+1)-(X-k+1) = k$. Hence, we claim that $\langle a_1, a_2, \ldots, a_m, k \rangle$ is a yes-instance of Subset-Sum.