CS101 Algorithms and Data Structures
Fall 2024
Homework 12

Due date: 23:59, January 1st, 2025

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: What your polynomial-size certificate is and 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 B to instances of problem A.
   (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 B to *yes*-instance of problem A and map *no*-instance of problem B to *no*-instance of problem A) by showing:
   (a) x is a *yes*-instance of problem $B \Rightarrow f(x)$ is a *yes*-instance of problem A.
   (b) x is a *yes*-instance of problem $B \Leftarrow f(x)$ is a *yes*-instance of problem A.

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:

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

To prove that the decision version of Set-Cover is in NP-Complete, we follow these steps:

1. **Membership in NP**:
   (a) A set of indices $\{i_1, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$, whose size is polynomial of input size .
   (b) Check whether $S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_k} = U$, whose run-time is polynomial of input size.
2. **Reduction from Vertex-Cover**:
   (a) We choose the decision version of Vertex-Cover as the problem to reduce from.

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

   (b) Given an undirected graph $G = (V, E)$ and a positive integer $k' \in \mathbb{Z}^+$, we construct the reduction $f(\langle G, k' \rangle) = \langle U, S_1, \ldots, S_n, k \rangle$ as follows: $U = E$. $n = |V|$ and $k = k'$. For each $i \in \{1, \ldots, n\}$, $S_i = \{e \in E \mid e = (v_i, u) \text{ for some } u \in V \setminus \{v_i\}\}$.
   (c) We prove the correctness of the reduction:
   "$\Rightarrow$": If $\langle G, k' \rangle$ is a *yes*-instance of Vertex-Cover, then there exists a set $V^* = \{v_{i_1}, \ldots, v_{i_{k'}}\}$ of vertices that covers all edges. By construction, $\{S_{i_1}, \ldots, S_{i_{k'}}\}$ covers all edges in U, so $\langle U, S_1, \ldots, S_n, k \rangle$ is a *yes*-instance of Set-Cover.
   "$\Leftarrow$": Conversely, if $\langle U, S_1, \ldots, S_n, k \rangle$ is a *yes*-instance of Set-Cover, then there exists a set $\{S_{i_1}, \ldots, S_{i_k}\}$ of sets that covers all edges in U. By construction, the corresponding set of vertices $\{v_{i_1}, \ldots, v_{i_k}\}$ covers all edges in G, so $\langle G, k' \rangle$ is a *yes*-instance of Vertex-Cover.

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

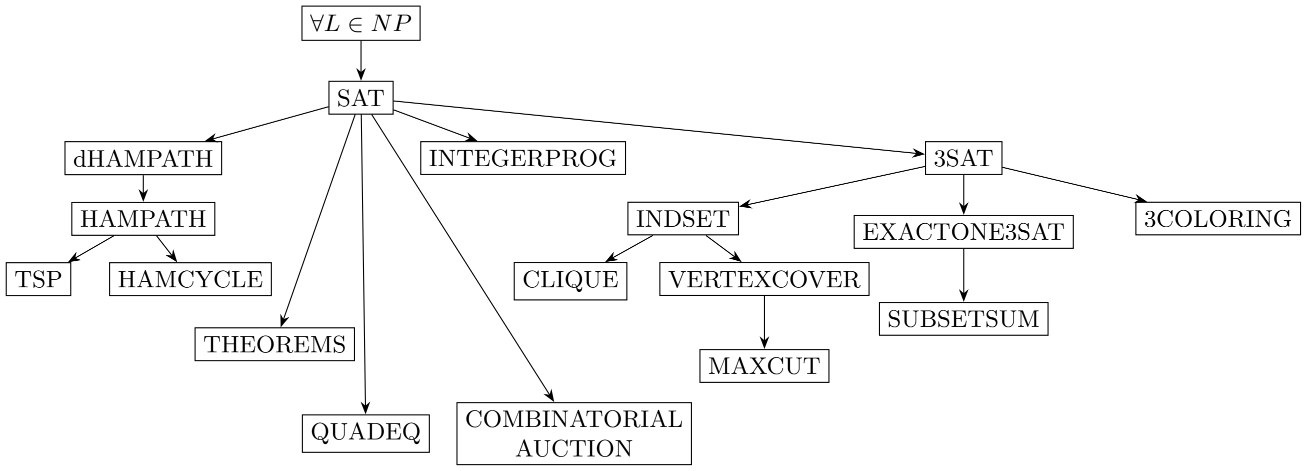
In fact, any two problems in P can reduce to each other in polynomial time, indicating they share the same "hardness." To illustrate, consider decision problems A and B in $\mathcal{P}$. Given an instance x of A:

1. Prepare polynomial-time copies of known *yes*- and *no*-instances of B.
2. Determine if x is a *yes*-instance of A in polynomial time.
3. Return the corresponding *yes*- or *no*-instance of B.

For example, let A be the minimum spanning tree cost problem and B be the shortest path cost problem in undirected weighted graphs. Given an instance G of A:

1. Prepare graphs G_1 and G_2 with known shortest path costs relative to c'.
2. Find G's minimum spanning tree cost using Kruskal's algorithm and compare it with c.
3. Return G_1 if the cost is no more than c, otherwise return G_2.

A brief map of common NP-Complete problems:



(cite: Computational Complexity: A Modern Approach by Sanjeev Arora and Boaz Barak.)

2. (21 points) Multiple Choice(s)

For each multiple-choice, 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. **Write your answers in the following table; otherwise, we may take your answers as unspecified.**
Notice: "must be true" means the proposition holds no matter whether P = NP or not. If you feel confused about some of the definitions, we encourage you to search the friendly web. But remember not to cheat or just use GenAI to fill in the blanks.

2(a)	2(b)	2(c)	2(d)	2(e)	2(f)	2(g)
CD	C	DE	ACD	ABD	BCD	CD

(a) (3') Choose those problems which must be in in NP-Complete:

A. CYCLE: Given a graph $G = (V, E)$, check whether it has a cycle.

B. GI: Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, check whether G_1 is isomorphic to G_2. ($\exists f : V_1 \to V_2$ is bijective and $(v_1, u_1) \in E_1$ if and only if $(f(v_1), f(u_1)) \in E_2$).

C. K-COLORING: Given a graph $G = (V, E)$, check whether it has a k-coloring: $f : V \to [k]$ that $\forall e = (u, v) \in E, f(u) \neq f(v)$. ($k \geq 3$)

D. KNAPSACK: Given n items and a capacity W where the weight and value of each item is w_i and v_i respectively. Check whether there is a subset of items such that the total weights of items are below W and the total value are above V. ($n, w_i, W \in \mathbb{Z}^+, v_i, V \in \mathbb{R}^+$.)

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

A. $A \in \mathsf{P} \implies B \in \mathsf{P}$

B. $A \in$ NP-Complete $\implies B \in$ NP-Complete.

C. $B \in \mathsf{P} \implies A \in \mathsf{P}$.

D. $B \in$ NP-Complete $\implies A \in$ NP-Complete.

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

A. Given a problem A, if there exists a polynomial-time certifier f so that for any no-instance X of A, there exists a polynomial-size certificate u, then A is also in NP.

B. If a problem X can be solved in polynomial time in the range of the output result, then $X \in \mathsf{P}$.

C. If a problem X can be solved in polynomial space in the length of the input result, then $X \in \mathsf{P}$.

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

E. $\mathsf{P} = \mathsf{NP}$ if and only if NP = NP-Complete.

(d) (3') Which of the following statement(s) must be true?

A. Every problem in NP can be solved in exponential time.

B. Finding a polynomial-time algorithm for problems in NP can prove P = NP.

C. There exists a polynomial-time many-one reduction from 2-SAT to every problem in NP.

D. There exists a polynomial-time many-one reduction from every problem in NP to 3-SAT.

Given a problem set L, we call a problem X in L-Complete if and only if every problem $Y \in L$ can be reduced to X in polynomial time. Recall P and NP are defined on decision problems i.e. problems giving out YES or NO. From this finish the following 2 problems:

(e) (3') Which of the following statement(s) of NP and NP-Complete is/are true?

A. If $\exists L \in$ NP-Complete $\cap$ P, then P = NP.

B. If $L_1 \leq_p L_2, L_2 \leq_p L_3$, then $L_1 \leq_p L_3$. Here $A \leq_p B$ means there exist a polynomial reduction from A to B.

C. The N in NP means "No polynomial solutions" since P $\neq$ NP is mostly believed.

D. The P in P means "Polynomial" since $X \in$ P indicates X can be solved in polynomial time.

(f) (3') Given a decision problem A and its corresponding certifier C, the counting version of A, denoted as $\#A$, is defined such that for every instance x, the output of $\#A(x)$ is the number of certificates y for which $C(x, y) = 1$. Specifically, $\#A(x) = |\{y|C(x, y) = 1\}|$. Furthermore, $\#P$ is defined as $\#P = \{\#A|A \in \mathsf{NP}\}$. Which of the following statements must be true?

A. $\forall A \in$ NP, $A \in$ NP-Complete if and only if $\#A \in \#$P-Complete.

B. If #SAT can be solved in polynomial time, then the decision problem P = NP.

C. $\forall A \in$ NP-Complete, $\#A \in \#$P-Complete.

D. If $\exists \#A \in \#$P can be solved in polynomial time, then $A \in$ P.

(g) (3') Which of the following statement(s) is/are true?

A. "There exists an NP problem that is not an NP-Complete problem." is false regardless of whether P equals to NP or not.

B. "Shortest-Path is not in NP-Complete." is false regardless of whether P equals to NP or not.

C. "There exists an NP problem that is not an NP-Complete problem." is true if and only if P $\neq$ NP.

D. "Shortest-Path is not in NP-Complete." is true if and only if P $\neq$ NP.

3. (15 points) Dichotomy

3-SAT is a well-known NP-Complete problem and it has many variants. There exists a kind of dichotomy theorems due to the complexity problem defined by S is either in P or is NP-Complete. Schaefer's dichotomy theorem shows that a large set of SAT-like problems are NP-Complete, while only 6 kinds of problems can be solved in polynomial time.

First of all we should claim some notations of SAT(the Boolean satisfiability problem). Given a variable set $U = \{u_1, u_2, \cdots, u_n\}$, a **boolean formula** is defined as the combination of unary/binary operators $\vee$(OR), $\wedge$(AND), $\neg$(NOT) and the variables in the variable set U. Given a boolean formula ϕ, if there exists a variable combination that makes ϕ true, then ϕ is satisfiable, otherwise ϕ is unsatisfiable.

A **SAT formula** is a logical formula in conjunctive normal form (CNF) Specifically, a **3-SAT formula** ϕ is a conjunction (AND) of clauses $C_1, C_2, \ldots, C_m$, and each clause C_i is a disjunction (OR) of three literals (variables or their negations) from $X \cup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}$. We will give out the initial NP-Complete problem: 3-SAT.

3-SAT: Given a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ and a **3-SAT formula** ϕ on X, determine whether there exists at least one truth assignment τ that makes the formula ϕ evaluate to true. The yes-instance of 3-SAT is:

$$\text{3-SAT} = \left\{ \langle\phi\rangle \middle| \begin{array}{l} \phi \text{ is a 3-SAT formula with variables } X = x_1, x_2, \ldots, x_n \\ \text{and clauses } C_1, C_2, \ldots, C_m \text{ such that there exists a truth} \\ \text{assignment } \tau : X \to \{\text{true}, \text{false}\} \text{ such that } \tau(\phi) = \text{true}. \end{array} \right\}$$

In this problem, you are supposed to give out a reduction from the following problem to those problems in NP-Complete if it is in NP-Complete, otherwise give out a polynomial time algorithm to solve it. Moreover, you are required to give out the yes-instances of those problems if they are in NP-Complete.

(a) (5') Horn-SAT: Given a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ and a **Horn-formula** ϕ on X, determine whether there exists at least one truth assignment τ that makes the formula ϕ evaluate to true.

A **Horn-fomula** is either a single literal (a positive or negative variable) or a disjunction of at most one positive literal and one or more negative literals.

Solution: Horn-SAT: This problem is in P. Horn clauses have a special structure that allows for an efficient algorithm. Specifically, we can use a greedy algorithm that iteratively assigns truth values to the variables in such a way that it satisfies all the clauses. Starting with all variables unassigned, we repeatedly choose an unassigned variable that appears in a unit clause (a clause with only one literal) and assign it the value that satisfies the clause. If no unit clause is available, we choose an unassigned variable that appears in a clause with only one positive literal and assign it the value that makes the positive literal true, thus satisfying the clause. This process continues until all clauses are satisfied or we determine that the formula is unsatisfiable. The yes-instances of Horn-SAT are those for which there exists a truth assignment that makes the formula evaluate to true.

(b) (5') 4-SAT: Given a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ and a **4-SAT formula**(a conjunction of clauses where each clause C_i is a disjunction of 4 literals) ϕ on X, determine whether there exists at least one truth assignment τ that makes the formula ϕ evaluate to true.

Solution:

$$\textsf{4-SAT} = \left\{ \langle\phi\rangle \middle| \begin{array}{l} \phi \text{ is a 4-SAT formula with variables } X = x_1, x_2, \ldots, x_n \\ \text{and clauses } C_1, C_2, \ldots, C_m \text{ such that there exists a truth} \\ \text{assignment } \tau : X \to \{\text{true}, \text{false}\} \text{ such that } \tau(\phi) = \text{true}. \end{array} \right\}$$

4-SAT is in NP-Complete. Given a 3-SAT formula, introduce a new variable x for every clause $C = a \vee b \vee c$, then we obtain

$$(a \vee b \vee c) \;\Leftrightarrow (a \vee b \vee c \vee x) \wedge (a \vee b \vee c \vee \neg x).$$

The reduction is for any 3-SAT yes-instance $C_1 \wedge C_2 \wedge \ldots \wedge C_n$, it can transformed into a 4-SAT yes-instance $D_1 \wedge E_1 \wedge \ldots \wedge D_n \wedge E_n$, where $D_i = (C_i \vee x), E_i = (C_i \vee \neg x)$.

(c) (5') (3,3)-SAT: Given a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ and a **3-SAT formula** ϕ on X, where **each variable appears at most three times**, determine whether there exists at least one truth assignment τ that makes the formula ϕ evaluate to true.
Hint: Consider how $a \Leftrightarrow b$ can be transformed into CNF.

Solution:

$$\textsf{(3,3)-SAT} = \left\{ \langle\phi\rangle \middle| \begin{array}{l} \phi \text{ is a 3-SAT formula with variables } X = x_1, x_2, \ldots, x_n \text{ where} \\ \text{each variable appears at most three times such that there exists} \\ \text{a truth assignment } \tau : X \to \{\text{true}, \text{false}\} \text{ such that } \tau(\phi) = \text{true}. \end{array} \right\}$$

(3,3)-SAT: This problem is in NP-Complete.
Given a 3-SAT formula, for every variable x appears n times ($n > 3$), use $x_1, x_2, \ldots, x_n$ to replace them in different clauses. Then introduce $4(n-1)$ clauses and a new variable: $x_i \vee \neg x_{i+1} \vee u$, $\neg x_i \vee x_{i+1} \vee u$, $x_i \vee \neg x_{i+1} \vee \neg u$, $\neg x_i \vee x_{i+1} \vee \neg u$ $(1 \le i < n)$. Since $(x_i \vee \neg x_{i+1}) \vee u \wedge (x_i \vee \neg x_{i+1} \vee \neg u)$ can be simplified into $x_i \Rightarrow x_{i+1}$, the conjunction of 4 clauses is equivalent to $x_i \Leftrightarrow x_{i+1}$, which indicates all x_i is equal. After transforming all the variables, a 3-SAT instance with n clauses and m variables become a $O(n+m)$ clauses and $n+m$ variables 3,3-SAT instance.
The reduction is for any 3-SAT yes-instance, it can be transformed into a 3,3-SAT yes-instance by the methods above.

4. (10 points) Carpenter's Rule

In this problem, you need to prove is Carpenter in NP-Complete.

Carpenter: Given an array $L = [l_1, l_2, \cdots, l_n]$ of non-negative integers, determine whether there exists a sequence $D = [d_1, d_2, \cdots, d_n]$ where $d_i \in \{\pm 1\}$ such that $\max_{j=0}^{n}\{\sum_{i=1}^{j} d_i l_i\} - \min_{j=0}^{n}\{\sum_{i=1}^{j} d_i l_i\} \leq k$.

Intuitively speaking, give a sequence of rigid rods of various integral lengths connected end-to-end by hinges, can it be folded so that its overall length is at most k? Correspondingly, l_i indicates the length of i-th rigid rod and d_i indicates the direction of i-th rigid rod (fold left or right). The max and min indicate the leftmost and rightmost positions.

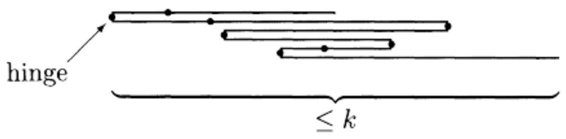


Figure 1: Example of Hinge

The *yes*-instances of Carpenter is:

$$\text{Carpenter} = \left\{ \langle l_1, \ldots, l_n, k \rangle \;\middle|\; \begin{array}{l} n \in \mathbb{Z}^+, l_1, \ldots, l_n, k \in \mathbb{N}, \exists D = [d_1, d_2, \cdots, d_n], d_i \in \{\pm 1\} \\ \text{i.e. } D \in \{1, -1\}^n \text{ s.t. } \max\limits_{j=0}^{n}\{\sum\limits_{i=1}^{j} d_i l_i\} - \min\limits_{j=0}^{n}\{\sum\limits_{i=1}^{j} d_i l_i\} \leq k. \end{array} \right\}$$

We choose Equivalent-Partition to reduce from. Here is the *yes*-instance of it:

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

Recall the definition of Equivalent-Partition:

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).

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

Solution:
Our certificate and certifier for Carpenter goes as follows:

- Certificate: A sequence $D = [d_1, d_2, \cdots, d_n]$ where $d_i \in \{\pm 1\}$.
- Certifier: Define $c_j = \sum_{i=1}^{j} d_i l_i (c_0 = 0)$, check whether $\max_{i=0}^{n} c_i - \min_{i=0}^{n} c_i \leq k$, which can be done in linear time.

(b) Consider how to construct your polynomial-time many-one reduction f that maps instances of Equivalent-Partition to instances of Carpenter. Unfortunately, the following reductions are not correct. Show that they are wrong by counterexamples.

i. (1') HaraoClesc gives out a reduction as follows:
Let $n = m$ and $L = [l_1, l_2, \ldots, l_n]$ be the sorted version in ascending order of $B = [b_1, \ldots, b_m]$ i.e. l_1 is the minimum one in B, a_n is the maximum of in B and l_i is the i-th minimum one in B and $k = \frac{1}{2}\sum_{i=1}^{n} b_i$. Then $\langle l_1, \ldots, l_n, k\rangle$ is a *yes*-instance of Carpenter if and only if $\langle b_1, b_2, \ldots, b_m\rangle$ is a *yes*-instance of Equivalent-Partition.

Solution:
$B = [3, 2, 3]$, easy to show that $\langle B\rangle$ is a no-instance of Equivalent-Partition. However, from reduction, $L = [2, 3, 3]$, $\langle L, 4\rangle$ is a yes-instance of Carpenter. ($D = [1, -1, -1]$ is one certificate.)

ii. (1') GeniusIdaiyo gives out another reduction as follows: Let $n = m + 2$, then

$$\langle l_1, \ldots, l_n, k\rangle = f(\langle b_1, \ldots, b_m\rangle) \triangleq \langle \max\{B\}, b_1, \ldots, b_m, \max\{B\}, \max\{B\}\rangle$$

He deduces that $\langle l_1, \ldots, l_n, k\rangle$ is a *yes*-instance of Carpenter if and only if $\langle b_1, b_2, \ldots, b_m\rangle$ is a *yes*-instance of Equivalent-Partition.

Solution:
$B = [2, 2, 2]$, easy to show that $\langle B\rangle$ is a no-instance of Equivalent-Partition. However, from reduction, $L = [2, 2, 2, 2, 2]$, $\langle L, 2\rangle$ is a yes-instance of Carpenter. ($D = [1, -1, 1, -1, 1]$ is one certificate.)

(c) From those wrong reductions above, FHKQ obtains a correct polynomial-time many-one reduction f that maps instances of Equivalent-Partition to instances of Carpenter. Show

i. (2') Your reduction in the format of (b).ii. (Show both n and L.)
Hint: Try to modify the above reductions into the correct one.

Solution: $n = m+4$, $S = \sum_{i=1}^{m} b_i$. (Any $S \geq \sum_{i=1}^{m} b_i$ suffices. If $S \not\equiv 0 (\mod 2)$, then the instances of Equivalent-Partition must be wrong-instances.)

$$\langle l_1, \ldots, l_n, k\rangle = f(\langle b_1, \ldots, b_m\rangle) \triangleq \langle S, \frac{S}{2}, b_1, \ldots, b_m, \frac{S}{2}, S, S\rangle$$

ii. (2') x is a yes-instance of Equivalent-Partition $\Rightarrow$ $f(x)$ is a yes-instance of Carpenter.

Solution: If $x = \langle b_1, \ldots, b_m\rangle$ is a yes-instance of Equivalent-Partition, and $T \in [m]$ is the subset such that $\sum_{i\in T} b_i = \sum_{j\in[m]\setminus T} b_j$ i.e. $\sum_{i=1}^{m}(-1)^{[i\in T]}b_i = 0$. Let $d_{i+2} = (-1)^{[i\in T]}$ and $d_1 = d_{m+4} = 1, d_2 = d_{m+3} = -1$. Then

$$\forall j \in [m], \sum_{i=1}^{j+2} d_i l_i = \frac{S}{2} + \sum_{i=1}^{j}(-1)^{[i\in T]}b_i, \max_j(\sum_{i=1}^{j}(-1)^{[i\in T]}b_i) \leq \sum_{i\in T}^{j} b_i = \frac{S}{2},$$

which indicates $\max_j\{\sum_{i=1}^{j} d_i l_i\} = \max(S, \frac{S}{2}, \frac{S}{2}) = S$. Similarly, $\min\{\sum_{i=1}^{j} d_i l_i\} = \max(0, \frac{S}{2}, 0) = 0$. Therefore, $\max_j\{\sum_{i=1}^{j} d_i l_i\} - \min_j\{\sum_{i=1}^{j} d_i l_i\} \leq k$, which indicates that $f(\langle b_1, \ldots, b_m\rangle) = \langle S, \frac{S}{2}, b_1, \ldots, b_m, \frac{S}{2}, S, S\rangle$ is a yes-instance of Carpenter.

iii. (2') $f(x)$ is a yes-instance of Carpenter $\Rightarrow$ x is a yes-instance of Equivalent-Partition.

Solution: Let $\langle l_1, \ldots, l_n, k\rangle = f(\langle b_1, \ldots, b_m\rangle) = \langle S, \frac{S}{2}, b_1, \ldots, b_m, \frac{S}{2}, S, S\rangle$ be a yes-instance of Carpenter and let $D = [d_1, \ldots, d_n]$ be the certificate.
Denote $c_j = \sum_{i=1}^{j} d_i l_i$. Since $[d_1, \ldots, d_n]$ and $[-d_1, \ldots, -d_n]$ both can be the valid certificate, we assume $d_1 = 1$. Then $d_2 = -1$ since $c_2 - c_0 = S + d_2\frac{S}{2} - 0 \in [0, S]$. Moreover, $\forall i \in [n], c_i \in [0, S]$. And since $|c_n - c_{n-1}| = S$, $c_{n-2} = \frac{S}{2}$ no matter $c_n = S$ or $c_n = 0$, which indicates $c_{n-2} - c_2 = 0$ i.e. $\sum_{i=1}^{m} d_{i+2} b_i = 0$. Denote $T = \{i | d_i = 1\}$ ($[m]\setminus T = \{i | d_i = -1\}$. Then

$$\sum_{i\in T} b_i - \sum_{j\in[m]\setminus T} b_j = 0 \Rightarrow \sum_{i\in T} b_i = \sum_{j\in[m]\setminus T} b_j,$$

which shows that $\langle b_1, \ldots, b_m\rangle$ is a yes-instance of Equivalent-Partition.

5. (9 points) Hamiltonian-Cycle Problem

Show that the Hamiltonian-Cycle problem on the **undirected graph** is NP-complete. The Hamiltonian-Cycle problem is determining whether the graph G contains a cycle that visits every vertex in G exactly once.

Hint: You can reduce this problem from the D-Hamiltonian-Cycle problem which determines whether there exists the Hamiltonian cycle on **the directed graph**.

Solution: First, note that Hamiltonian-Cycle on undirected graphs is in NP, since a nondeterministic machine could guess an ordering of our vertices and then check in polynomial time that it forms a cycle.

Next, to demonstrate that the problem is NP-hard, we reduce from the D-Hamiltonian-Cycle problem for directed graphs.

Assume we possess a directed graph G comprising n vertices, namely $v_1, v_2, \ldots, v_n$. We then construct an undirected graph G' with $3n$ vertices, labeled as $x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n$. The vertices are interconnected as detailed below.

First, it is worth noting that for each vertex in G, we have created three corresponding vertices in G'. We introduce edges (x_i, y_i) and (y_i, z_i) for all i.

Subsequently, we assign all outgoing edges from v_i in G to x_i and all incoming edges to v_i to z_i. Specifically, if (v_i, v_j) is an edge in G, then (x_i, z_j) is an edge in G'.

We now prove that a cycle exists in G' if and only if a cycle exists in G. Initially, suppose there is a cycle in G, given by $v_{i_1}, v_{i_2}, v_{i_3}, \ldots, v_{i_n}, v_{i_1}$. The corresponding edges in G' form the set $(x_{i_j}, z_{i_{j+1}})$ (for $j = 1, 2, \ldots, n-1$) and (x_{i_n}, z_{i_1}). Consequently, the following cycle emerges in G':

$$x_{i_1}, z_{i_2}, y_{i_2}, x_{i_2}, z_{i_3}, \ldots, x_{i_n}, z_{i_1}, y_{i_1}, x_{i_1}$$

Note that each y_i is included in the cycle by virtue of the edges (x_i, y_i) and (y_i, z_i).

Conversely, assume there is a cycle in G'. The sole manner in which a cycle can incorporate y_i is through the edges (x_i, y_i) and (y_i, z_i), which must necessarily be part of the cycle. Therefore, each x_i must be connected to a unique z_j by an edge. Consequently, we can identify a cycle in the original graph G by tracing the edge from v_i to v_j whenever x_i is connected to z_j.