ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2025

Final Exam

Instructors: Yuyao Zhang and Xin Liu

Time: Jan 7th 8:00-10:00

INSTRUCTIONS

Please read and follow the following instructions:

- This exam has 8 questions, for a total of 100 points.
- You are not allowed to bring any papers, books, or electronic devices, including regular calculators.
- You are not allowed to discuss or share anything with others during the exam.
- You should write the answer to every problem in the dedicated box **clearly**.
- You should write **your name and your student ID** as indicated on the top of **each page** of the exam sheet.
- If you need more space, write "Continued on Page #" and continue your solution on the referenced scratch page at the end of the exam sheet.

Name	
Student ID	
Exam Classroom Number	
Seat Number	
(please copy this and sign)	All the work on this exam is my own.

THIS PAGE INTENTIONALLY LEFT BLANK.

DO NOT WRITE ANY ANSWER IN THIS PAGE!

1. (15 points) Single Choice

Each question has **exactly one** correct answer. Fill your answers **in the box below**.

Notice: Make sure to fully mark the answer, or we may take it unspecified.

(a)	◯ A	◯ B	◯ C	◯ D
(b)	◯ A	◯ B	◯ C	◯ D
(c)	◯ A	◯ B	◯ C	◯ D
(d)	◯ A	◯ B	◯ C	◯ D
(e)	◯ A	◯ B	◯ C	◯ D

$\boxed{3}$ (a) In a disjoint-set of size n that only uses path compression, what is the maximum change in height after a find operation with path compression?

A. n.

B. $n-1$.

C. $n-2$.

D. $n-3$.

$\boxed{3}$ (b) Consider a set of precedence constraints among tasks labeled as $\{A, B, C, D, E, F, G\}$:

- A must be done before D and E;
- B before D and F;
- C before E and F;
- D and E before G;

Which order of completing tasks is **NOT** possible?

A. A, B, C, D, E, F, G.

B. B, C, A, E, D, F, G.

C. C, B, A, D, E, F, G.

D. A, B, C, E, G, D, F.

$\boxed{3}$ (c) Suppose there is an undirected weighted connected graph with a very large number of vertices $|V|$, and the number of edges $|E|$ satisfies $|E| = \Theta(|V|^{1.5})$. All edge weights are non-negative. The task is to find the shortest paths from a single source vertex to all other vertices. Among the following combinations of algorithms and data structures, which one is optimal in terms of asymptotic time complexity?

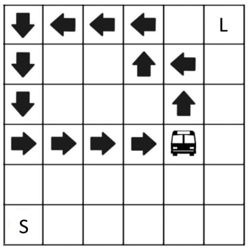
A. Using adjacency matrix storage with Dijkstra's algorithm that scans linearly to find the vertex with the minimum distance each time.

B. Using adjacency list storage with Dijkstra's algorithm with a binary min-heap.

C. Using adjacency list storage with the Bellman-Ford algorithm.

D. Using adjacency matrix storage with the Floyd-Warshall algorithm to compute all-pairs shortest paths.

$\boxed{3}$ (d) Logan needs to go to the school from his company every day. As shown in the figure, his company is located in the top-right corner (M, N) and the school is located in the bottom-left corner $(0, 0)$ of an $M \times N$ grid. He is able to move up, down, left, or right one grid per timestamp on foot. However, there exists a subway route in the grid. He is allowed to take the subway along the route and move along the route at the rate of 3 squares per timestamp.



For A* search, consider the family of heuristics

$$h_k(x, y) \;=\; k \cdot \Big(|x - 0| + |y - 0|\Big),$$

where (x, y) is Logan's current location and the goal is $(0, 0)$.

What is the **largest** constant k such that h_k is guaranteed to be **consistent** for **any possible** subway route placement?

A. $k = 1$

B. $k = \frac{1}{2}$

C. $k = \frac{1}{3}$

D. $k = \frac{1}{4}$

3 (e) We have a graph $G = (V, E)$ where $|V| = n$ and $|E| = m$. How many of the following problems have their **optimal** solution time complexity correctly labeled?

(1) Determine whether there exists a path from a given vertex s to a given vertex t in an **unweighted directed** graph. $\Theta(n)$

(2) Given a **DAG**, compute a topological ordering of all vertices. $\Theta(n + m)$

(3) Find the shortest path distances from a source s to all vertices when edge weights are in $\{0, 1\}$. $\Theta((n + m) \log n)$

(4) Find the shortest path between **every pair** of vertices when $m = \Theta(n)$, edge weights $w \in \mathbb{N}^+$. $\Theta(n^3)$

(5) Determine whether there is a simple cycle in an undirected graph. $\Theta(n)$

A. 1

B. 2

C. 3

D. 4

2. (20 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 2 points if you select a non-empty subset of the correct answers. Fill your answers **in the box below**.

Notice: Make sure to fully mark the answer, or we may take it unspecified.

(a)	◯ A	◯ B	◯ C	◯ D
(b)	◯ A	◯ B	◯ C	◯ D
(c)	◯ A	◯ B	◯ C	◯ D
(d)	◯ A	◯ B	◯ C	◯ D
(e)	◯ A	◯ B	◯ C	◯ D

$\boxed{4}$ (a) In a weighted and connected graph $G = (V, E)$, which of the following statement(s) is/are **TRUE**?

A. If two edges in E have the same weight, the graph must have multiple minimum spanning trees.

B. If we add an edge to any spanning tree in G, there will be exactly one cycle in the new graph.

C. The time complexity of Kruskal's algorithm is $O(|V| \log |E|)$.

D. Prim's algorithm can not work correctly when there are negative edges in the graph.

$\boxed{4}$ (b) Suppose we modify a graph algorithm to terminate early. Which of the following early-termination rules still guarantees the correctness of the algorithm's output?

A. Stop Kruskal's algorithm as soon as $V - 1$ edges have been added to the MST.

B. Assuming all edge weights are nonnegative, stop Bellman–Ford as soon as a full pass completes (all edges are relaxed) without decreasing any `dist[]` value.

C. Stop Floyd–Warshall if one outer-loop k produces no changes to the matrix.

D. Stop Dijkstra's algorithm as soon as every vertex has been inserted into the priority queue.

$\boxed{4}$ (c) Which of the following statement(s) is/are **TRUE**?

A. For an admissible heuristic function, A* tree search may have higher time complexity than A* graph search, but A* graph search may return a sub-optimal solution.

B. In a connected undirected graph, regardless of the graph structure and implementation, Prim's and Kruskal's algorithms have the same time complexity.

C. If the graph is not connected, Floyd's algorithm cannot work correctly.

D. In a directed graph, the Bellman-Ford algorithm can be used to detect negative cycles.

$\boxed{4}$ (d) Which of the following statement(s) is/are **TRUE**?

A. In the *Weighted Interval Scheduling* problem with n intervals, both top-down (memoization) and bottom-up dynamic programming can achieve a time complexity of $O(n \log n)$.

B. In the *Coin Changing* problem with coin set $\{c_i\}_{i=1}^n$, let $OPT(v)$ be the minimum number of coins to make value v. Then $OPT(v) = \begin{cases} 0, & \text{if } v \le 0, \\ \min\limits_{1 \le i \le n} \big(OPT(v - c_i) + 1\big), & \text{if } v > 0. \end{cases}$

C. From a DP perspective, in the graph $G = (V, E)$, the Floyd-Warshall algorithm defines $\Theta(|V|^2)$ sub-problems, and each sub-problem is solved in $\Theta(|V|)$ time, resulting in a time complexity of $\Theta(|V|^3)$.

D. If there are n houses and m colors, the *House Coloring* problem can be solved in $\Theta(nm)$ time complexity.

4 (e) Suppose that Problem A is in P, Problem B is in NP, and Problem C is NP-Complete. Which of the following can you infer?

A. 3-SAT polynomial time reduces to Problem C.

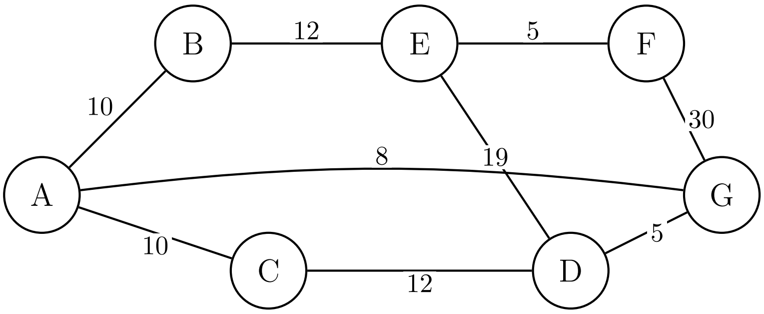
B. If Problem B can be solved in polynomial time, then P=NP.

C. If Problem B cannot be solved in polynomial time, then neither can Problem C.

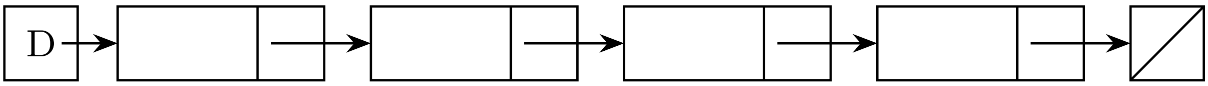
D. If Problem C polynomial time reduces to Problem B, then B is NP-Complete.

3. (10 points) A World of Graph

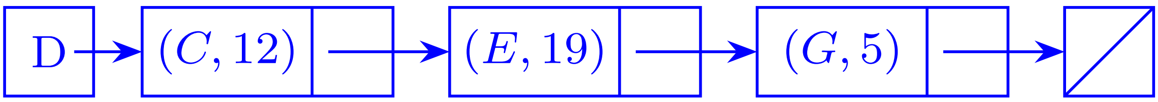
I. Consider a weighted undirected graph $G = (V, E)$:



$\boxed{2}$ (a) For the weighted graph shown above, fill in the adjacency lists for vertex D. For a vertex u, its adjacency list is the set of all pairs (v, w), where v is a neighbor of u and w is the weight of edge (u, v). List the pairs in alphabetical order of v.



Solution:



(b) Fill in the blanks. For these questions, **Ties are broken by choosing the vertex whose label comes first alphabetically.**

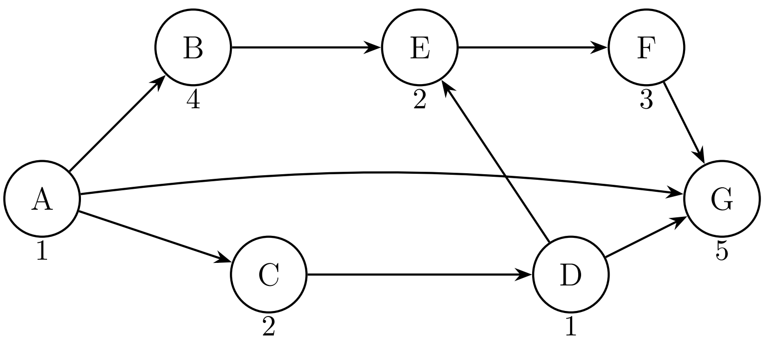
$\boxed{1}$ i. Is $\langle B, E, D, C, A\rangle$ a simple cycle? **No**

$\boxed{1}$ ii. Perform breadth-first search starting from vertex A. List the vertices in the order they are enqueued. $A \to B \to C \to G \to E \to D \to F$

$\boxed{1}$ iii. Perform recursive depth-first search starting from vertex A. List the vertices in the order they are visited. $A \to B \to E \to D \to C \to G \to F$

$\boxed{1}$ iv. Run Kruskal's algorithm on this graph. What are the edges in the MST? **Write their weights** in the order we add them. If the MST is not unique, write a valid possible solution. $5, 5, 8, 10, 10, 12$

II. Now consider the directed graph below. The number below each vertex represents the vertex weight.



(c) Answer the questions (Write down all answers that satisfy the condition):

1 i. Which of the vertices have the same in-degree as out-degree? **B, C, F**

1 ii. Which of the vertices have the minimum out-degree? **G**

(d) Now interpret each vertex as a task that can be executed in parallel, and let the processing time of each task be its vertex weight.

1 i. What's the **critical time of all tasks**? **15**

1 ii. What's the corresponding **all critical path**? **ABEFG**

4. (8 points) Trip

Alice is preparing for a trip. Alice wants to minimize her cost during the trip, which includes transportation and accommodation costs. There are $n+1$ cities along the way. Alice starts at 0^{th} city and the destination is the n^{th} city. The trip takes a few days, so if Alice stays overnight in the $i^{\text{th}}(1 \le i \le n)$ city, she has to pay for the accommodation fee of $a_i(a_i > 0)$. Alice must stay overnight at the destination. You are required to tell Alice the minimum cost for this trip, given the mode of transportation. Let $OPT(i)$ denote the minimum cost to arrive at city i and **stay overnight**.

(a) If Alice decides to ride a bike, it will not incur any transportation costs. Alice can ride her bike through 1 or 2 cities within a day. That is, if Alice stays overnight at city i, Alice can stay overnight at city $i+1$ or $i+2$ the next day.

$\boxed{1}$ i. What is the answer to this question in terms of OPT?

Solution: $OPT(n)$.

$\boxed{2}$ ii. Give your Bellman equation to solve the subproblems.

Solution:

$$OPT(i) = \begin{cases} a_i & \text{if } 1 \le i \le 2 \\ a_i + \min\{OPT(i-1), OPT(i-2)\} & \text{if } 2 < i \le n \end{cases}$$

(b) If Alice gives up riding a bicycle and decides to take a private jet, which can fly to any city but only once a day. Flying to city j from city i will incur $(i-j)^2$ cost.

$\boxed{3}$ i. Give your Bellman equation to solve the subproblems.

Solution: Define $OPT(0) = 0$.
$OPT(i) = \min_{1 \le j \le i}\{OPT(i-j) + j^2\} + a_i$

$\boxed{2}$ ii. What is the time complexity of your algorithm? (answer in $\Theta(\cdot)$ and in the most simplified form)

Solution: There are n sub-problems. For sub-problem i, we need to check all the sub-problems $j < i$. Therefore, the time complexity is $\Theta(n^2)$.

5. (11 points) Project Scheduling

As a student at HaishangTech, you are in RRR week. You have n projects due right now, but you haven't started any of them, so they are all going to be late. Each project requires d_i days to complete, and has a cost penalty of c_i per day. So if project i ends up being finished t days late, then it incurs a penalty of $c_i t$. Assume that once you start working on a project, you must work on it until you finish it, and that you cannot work on multiple projects at the same time.

For example, suppose you have three problem sets: CS100 takes 3 days and has a penalty of 12 points/day, CS101 takes 4 days and has a penalty of 20 points/day, and CS110 takes 2 days and has a penalty of 4 points/day. The best order is then CS101, CS100, CS110 which results in a penalty of $20 \times 4 + 12 \times (4+3) + 4 \times (3+4+2) = 200$ points.

$\boxed{3}$ (a) Describe your greedy algorithm that outputs an ordering of the projects that minimizes the total penalty for all the projects.

Solution: Schedule projects in nondecreasing order of the ratio $\frac{d_i}{c_i}$ (treat $c_i = 0$ as $\frac{d_i}{c_i} = +\infty$ and place them last). Output that ordering.

$\boxed{8}$ (b) Analyze the running time and prove the correctness of your algorithm by Exchange Arguments.

Solution: Compute ratios in $O(n)$ time and sort the n projects by $\frac{d_i}{c_i}$ in $O(n \log n)$ time. Total running time is $O(n \log n)$.

Simply: If unsorted, we can improve by swapping.

$$\frac{d_i}{c_i} > \frac{d_j}{c_j} \;\Rightarrow\; c_j d_i + (c_i d_i + c_j d_j) > c_i d_j + (c_i d_i + c_j d_j)$$
$$\Rightarrow\; c_j (d_i + d_j) + c_i d_i > c_i (d_i + d_j) + c_j d_j$$

Here is the full proof:

1. Definition of a solution. A solution is an ordering (permutation) π of the n projects. If C_i is the completion time (days from now) of project i under π, then since all projects are due now, project i is finished $t = C_i$ days late and its penalty is $c_i C_i$. The total penalty is

$$P(\pi) = \sum_{i=1}^{n} c_i C_i.$$

2. Definition of optimality. An ordering $\pi^\star$ is optimal if it minimizes $P(\pi)$ over all orderings π.

3. How to change any optimal solution to a greedy solution iteratively. Take an optimal ordering $\pi^\star$. If it is not sorted by nondecreasing $\frac{d_i}{c_i}$, then it contains an adjacent inverted pair: two consecutive projects i then j with

$$\frac{d_i}{c_i} > \frac{d_j}{c_j}.$$

Swap this adjacent pair. Repeat this operation while an adjacent inversion exists. When no adjacent inversion remains, the ordering is sorted by nondecreasing $\frac{d_i}{c_i}$, i.e., it matches the greedy rule.

4. Why does such a change not make the solution worse? Consider any adjacent pair i then j in a schedule, and let T be the total processing time of all projects before them (so T is fixed for this comparison).

If we do i then j, the pair contributes

$$P_{ij} = c_i(T + d_i) + c_j(T + d_i + d_j).$$

If we swap to j then i, the pair contributes

$$P_{ji} = c_j(T + d_j) + c_i(T + d_j + d_i).$$

Compute the difference:

$$P_{ij} - P_{ji} = c_j d_i - c_i d_j.$$

So $P_{ji} \leq P_{ij}$ (swapping is not worse) exactly when $c_j d_i \geq c_i d_j$, equivalently

$$\frac{d_i}{c_i} \geq \frac{d_j}{c_j}.$$

Therefore, whenever there is an adjacent inversion $\frac{d_i}{c_i} > \frac{d_j}{c_j}$, swapping that pair does not increase the total penalty (it decreases it). Hence, repeatedly swapping inversions never makes an optimal schedule worse, and the final inversion-free (ratio-sorted) schedule is also optimal. This is exactly the greedy schedule, so the greedy algorithm is correct.

6. (12 points) Dijkstra's Algorithm on Negative-weight Graphs

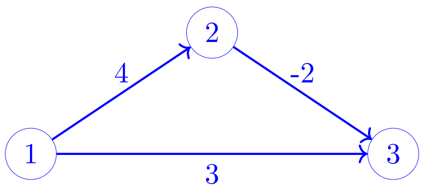
There's a directed graph $G = (V, E)$ without negative cycles. There could be negative-weight edges in E.

2 (a) We know that Dijkstra's algorithm doesn't work correctly on negative-weight graphs. Please give an example of G to show that Dijkstra's algorithm fails on G starting at vertex 1.

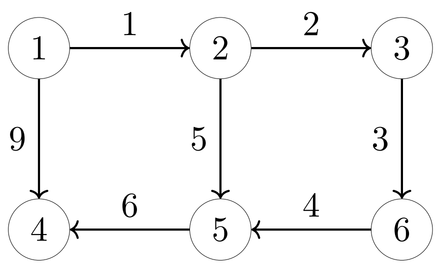
Requirements:

- $V = \{1, 2, 3\}$.
- There are no negative cycles in G.

Solution:



(b) Given a graph G:



2 i. Please find the shortest path from vertex 1 and fill the distance in the table below.

Vertex	1	2	3	4	5	6
Distance	0	______	______	______	______	______

2 ii. If we add a negative edge $(2, 6)$ with weight -8. Please find the shortest path from vertex 1 and fill the distance in the table below.

Vertex	1	2	3	4	5	6
Distance	0	______	______	______	______	______

Solution:

Vertex	1	2	3	4	5	6
Distance (i.)	0	1	3	9	6	6
Distance (ii.)	0	1	3	3	-3	-7

6 (c) If there is exactly **one** edge with negative weight in G, denoted as (p, q). Please design an algorithm that is more efficient than Bellman-Ford (i.e., $o(|V||E|)$ time) to calculate the length of the shortest path from s to other vertices. You may assume $s, p, q \in V$ are distinct, and you can call Dijkstra's algorithm directly.

Hint: you can run multiple instances of Dijkstra's algorithm from different "sources".

Write your answer on the next page....

(c) Answer:

Solution: Algorithm:

1. Construct sub-graph $G' = (V, E')$, where $E' = E \setminus \{(p, q)\}$.
2. Run the Dijkstra's algorithm on G' from s, q, respectively, storing the distances in $d_s[\cdot], d_q[\cdot]$.
3. The shortest path from s to u must be one of the following situations:
   - The path not contains edge (p, q), the distance of the shortest path in G is $d_s[u]$.
   - The path contains edge (p, q) (i.e. $[s, \ldots, p, q, \ldots, u]$), the distance of the shortest path in G is $d_s[p] + w(p, q) + d_q[u]$.

   Therefore, in the original graph G, the length of shortest path from s to u is $\min\{d_s[u], d_s[p] + w(p, q) + d_q[u]\}$.

Time complexity analysis:

- Construct sub-graph: $O(1)$ (or $O(|V| + |E|)$) time.
- Run 2 times of Dijkstra's algorithm: $O(|E| \log |V|)$ (or $O(|V|^2)$) time.
- Calculate answers for every vertex: $O(|V|)$ time.

Therefore, the time complexity is $O(|E| \log |V|)$ (or $O(|V|^2)$).

7. (12 points) Marble Game

Logan and Joel are playing a strategy game on a directed acyclic graph (DAG) $G = (V, E)$ with n vertices and m edges. Each directed edge $(v \to u) \in E$ has an integer weight $w(v, u) \in \{1, 2, \ldots, K\}$, where K is the maximum edge weight.

The game proceeds as follows:

- Logan's marble starts at vertex $s_L \in V$, and Joel's marble starts at $s_J \in V$.
- Logan moves first, and they alternate turns.
- On a player's turn, they move their marble along an outgoing edge $(x \to y) \in E$ such that $w(x, y) \geq \ell$, where ℓ is the weight of the edge used by the opponent in their last move (initialized to 0 before the first move).
- If a player cannot make a valid move, they lose.

Both players play optimally. Your task is to determine the winner of the game (Logan or Joel) for **every possible starting pair** $(s_L, s_J) \in V \times V$.

$\boxed{2}$ (a) How will you define the sub-problems?

Solution: Define $dp(a, b, k)$ as a boolean value representing whether the player whose turn it is can win from vertex a for the current player and vertex b for the opponent, with the last edge weight having code k.

$\boxed{6}$ (b) Describe and justify the Bellman Equation for computing the solution to sub-problems.

Solution: The recurrence for the DP is as follows:

- If there is no valid move from vertex a with a weight $\geq k$, then the current player loses.
- Otherwise, for each outgoing edge $(a \to a')$ with weight $w(a, a') \geq k$, the current player has two options:
  - Move to vertex a' with weight $w(a, a')$, and the opponent now has a turn to play.
  - If the opponent is forced into a losing position (i.e., $dp(b, a', w(a, a')) = \text{false}$), then the current player wins.

The recurrence relation is:

$$dp(a, b, k) = \begin{cases} \text{false} & \text{if there is no edge } (a \to a') \text{ with } w(a, a') \geq k, \\ \text{true} & \text{if there exists an edge } (a \to a') \text{ with } w(a, a') \geq k \\ & \quad \text{and } dp(b, a', w(a, a')) = \text{false}, \\ \text{false} & \text{otherwise.} \end{cases}$$

Or

$$dp(a, b, k) = \bigvee_{\substack{(a \to a') \in E \\ w(a, a') \geq k}} \neg\, dp\big(b, a', w(a, a')\big),$$

This ensures that we always check if there exists a move that forces the opponent to lose, following the optimal play strategy.

Solution:

2 (c) What is the solution in terms of your sub-problems?

Solution: The solution to the original problem, i.e., the winner of the game for all initial positions of the marbles, is given by $dp(s_L, s_J, 0)$, where s_L is the starting position of Logan's marble, and s_J is the starting position of Joel's marble. If $dp(s_L, s_J, 0) = \text{true}$, Logan wins; otherwise, Joel wins.

2 (d) What is the runtime complexity of your algorithm? (answer in $\Theta(\cdot)$ and in the most simplified form, and give proof).

Solution: The number of sub-problems is $O(n^2k)$, and each fixed $dp(*, b, k)$ sub-problem requires $O(m)$ time to compute (since we check all outgoing edges of a vertex). Therefore, the total time complexity is $\Theta(nmk)$.

8. (12 points) Independent Set Partition

There is a **decision problem**, called Independent-Set-Partition ($\mathsf{ISP}_{\mathsf{n,m}}$):

In an undirected graph $G = (V, E)$ and two positive integers n and m, **determine** whether the vertex set V can be partitioned into exactly n subsets (some subsets may be empty) such that:

- Each subset is an *independent set* of G.
- The size of each subset is at most m.

The yes-instances $\mathsf{ISP}_{\mathsf{n,m}}$ is:

$$\mathsf{ISP}_{\mathsf{n,m}} = \left\{ \langle G = (V, E), n, m \rangle \;\middle|\; \begin{array}{l} \text{In a simple undirected } G = (V, E), \exists V_1, \cdots, V_n \subseteq V \text{ s.t.} \\ \quad \text{- For every pair of } (V_i, V_j) : V_i \cap V_j = \emptyset, \cup_{i=1}^{n} V_i = V \\ \quad \text{- For any } u, v \in V_i : (u, v) \notin E \text{ and } |V_i| \leq m. \end{array} \right\}$$

Note: Recall $\mathsf{k\text{-}Coloring}$: Given an undirected graph $G = (V, E)$, **determine** whether there exists a coloring of the vertices with at most k colors such that no two adjacent vertices share the same color. Here is its yes-instance:

$$\mathsf{k\text{-}Coloring} = \left\{ \langle G = (V, E), k \rangle \;\middle|\; \begin{array}{l} G = (V, E) \text{ is an undirected graph, and there exists a coloring} \\ c : V \to 1, 2, \ldots, k \text{ such that for every edge } u, v \in E, c(u) \neq c(v). \end{array} \right\}$$

$\boxed{2}$ (a) Prove that $\mathsf{ISP}_{\mathsf{n,m}}$ is in NP (Show your certificate and certifier with a brief explanation.)

> **Solution:**
> Certificate: A disjoint t-partition $\bar{V} = [V_1, \cdots, V_t]$.
> Certifier: First check whether $\bar{V}$ is a disjoint partition and check each set $|V_i| \leq m$, which can be computed in $O(|V|)$ time. Then check whether for every V_i those vertices in it are independent, which takes $O(t|V|^2)$ time.
> Given an input $s = \langle G = (V, E), n, m \rangle$, if there exists a certificate $\bar{V}$ passes the certifier, then the V can be partitioned into $\bar{V}$, meaning $s \in \mathsf{ISP}_{\mathsf{n,m}}$. Otherwise, V can't be divided in that way, indicating $s \notin \mathsf{ISP}_{\mathsf{n,m}}$.

$\boxed{4}$ (b) If $n = 2$, the problems may be a little different from $n > 2$. For convenience, you can assume **the graph is connected**. From what you learned, show that $\mathsf{ISP}_{\mathsf{2,m}} \in \mathsf{P}$, more exactly, takes $O(|E| + |V|)$ time.

> **Solution:** Use BFS on graphs: From a vertex $v \in V$, note that the two subsets are symmetry. W.L.O.G assume $v \in V_1$. Put v in the queue. Do till the queue is empty or conflict happens: Get the front of the queue v, then for any neighbor u of v:
>
> - If u is visited, and u, v is in the same set. Then G is a no-instance of $\mathsf{ISP}_{\mathsf{2,m}}$.
> - If u is visited, and u, v are in different sets. Then ignore u.
> - Otherwise, put u in the opposite set of the set of v. Push u into the queue.
>
> All vertices will be visited once. Then check of size of the set V_1, V_2 less than m to find whether it's valid. It takes $O(|E| + |V|)$ time.
> The correctness depends on using BFS to solve 2-coloring problems.

(c) Now you are required to prove $\mathsf{ISP_{n,m}}$ is in NP-Complete when $n \geq 3$.

$\boxed{2}$ i. Write your polynomial time reduction f from k-Coloring to $\mathsf{ISP_{n,m}}$.

Solution: Set $t = k$, $k \geq 3$ infers that t-Coloring is NP-Complete. Given an instance $\langle G, k\rangle$ of t-Coloring, we construct an instance of $\mathsf{ISP_{k,|V|}}$: $\langle G, k, |V|\rangle$, using the same graph. This construction clearly takes polynomial time.

$\boxed{2}$ ii. Prove that x is a yes-instance of k-Coloring $\Rightarrow f(x)$ is a yes-instance of $\mathsf{ISP_{n,m}}$.

Solution: Suppose G has a valid t-coloring $c : V \to \{1, \ldots, t\}$. Construct an partition as: $V_i = \{v \in V | c(v) = i\}$. $V_1, \cdots, V_k$ is an partition.
Correctness:
For each subset V_i, $V_i \leq |V|$, and $\forall (u, v) \in V_i$, we have $c(u) = c(v)$, which means $(u, v) \notin E$. $V_i \leq |V|$ and V_i is an independent set of G. Therefore, $\langle G, |V|, k\rangle$ is a yes-instance of $\mathsf{ISP_{n,m}}$.

$\boxed{2}$ iii. Prove: $f(x)$ is a yes-instance of $\mathsf{ISP_{n,m}} \Rightarrow x$ is a yes-instance of k-Coloring.

Solution: Suppose the $\mathsf{ISP_{n,m}}$ instance has a feasible partition $V_1, \cdots, V_k$. Define a coloring $c(v) = \sum_{i=1}^{t} i[v \in V_i]$ i.e. $c(v) = i$ if and only if $v \in V_i$.
Correctness:
For each edge $(u, v) \in E$, assume $c(u) = c(v) = x$. That is, $u, v \in V_x$, which means subset V_x is not an independent set. It's a contradiction. Therefore, $c(u) \neq c(v)$. Hence, c is a valid t-coloring of G.

THIS PAGE INTENTIONALLY LEFT BLANK.

Fill the circle if you need to continue your solution on this page: ◯ Problem: ____________