- All problems are worth 15 points.
- Please write down your name and student ID on every page.

Problem 1

Given n positive integers $\{h_i\}_{1\le i\le n}$, you are tasked to find $\max_{1\le l\le r\le n}[(r-l+1)\times\min\{h_l, h_{l+1}, \ldots, h_r\}]$ in $O(n)$ time complexity.

(a) (2 points) If $\{h_i\}_{i=1,\ldots,6}$ is given by $\{2, 1, 4, 7, 5, 3\}$, what is the maximum value?

$12 = 3\times\min\{4, 7, 5\} = 4\times\min\{4, 7, 5, 3\}$.

(b) (3 points) Let $[l_i, r_i]$ be the most wide range such that $h_i \le h_k$ for all $l_i \le k \le r_i$. Prove $\max_{1\le l\le r\le n}[(r-l+1)\times\min\{h_l, h_{l+1}, \ldots, h_r\}] = \max_{1\le i\le n}[(r_i - l_i + 1)\times h_i]$.

For arbitrary l and r, if $h_i = \min\{h_l, h_{l+1}, \ldots, h_r\}$, then $r-l+1 \le r_i - l_i + 1$.
Thus, $(r-l+1)\times\min\{h_l, h_{l+1}, \ldots, h_r\} = (r-l+1)\times h_i \le (r_i - l_i + 1)\times h_i$.
Also, $(r_i - l_i + 1)\times h_i \in \{(r-l+1)\times\min\{h_l, h_{l+1}, \ldots, h_r\} | 1 \le l \le r \le n\}$.

(c) (10 points) Design an algorithm $O(n)$ and justify the time complexity. Write the pseudocode.

Algorithm 1 Problem 1-(c)

```
function MaxVal(n, {h_i}_{1≤i≤n})
    answer ← 0
    stack                                          ▷ Define an empty stack
    for i = 1, ..., n do
        while stack is not empty and h_i < h_{stack.top()} do
            j ← stack.pop()
            l_j ← 1 if stack is empty else stack.top() + 1
            r_j ← i − 1
            answer ← max(answer, (r_j − l_j + 1) × h_j)
        stack.push(i)
    while stack is not empty do
        j ← stack.pop()
        l_j ← 1 if stack is empty else stack.top() + 1
        r_j ← n
        answer ← max(answer, (r_j − l_j + 1) × h_j)
    return answer
```

It pushes the element n times, and one elment is popped out for each while loop. Therefore, it is $O(n)$ time complexity.

Grading Criteria:

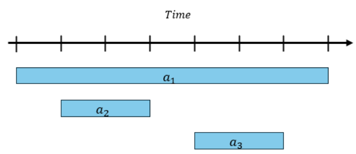
- (+2 points) Correct answer for (a).
- (+3 points) Correct answer for (b).
- (+4 points) Correct calculation for l_j and r_j.
- (+3 points) $O(n)$ with plausible l_j and r_j.
- (+3 points) No minor error for (c).
- (-8 points) Incorrect time complexity in (c).

Problem 2

The **scheduling problem** is the problem of finding a maximal conflict-free schedule with several activities given the start and finish times. Prove/disprove the correctness of three different greedy approaches. If the approach is correct, then justify it **using inductive exchange argument**. If not, provide a **counter-example**.

If you solve the problem to find the maximum time, you only get a maximum of 2 points for each subproblem.

(a) (5 points) (1) Sort activities by **start times** (2) Scan activities in the sorted order and select activities with **the earliest start time** that does not conflict with selected activities so far.



Output : $\{a_1\}$
Optimal Output : $\{a_2, a_3\}$

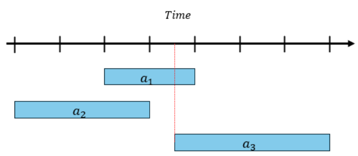
(+5 points) Correct example.

(b) (5 points) (1) Sort activities by **start times** (2) Scan activities in the sorted order and select activities with **the latest start time** that does not conflict with selected activities so far.

Correct. Let f be the last activity to start. Suppose we have a maximal conflict-free schedule X that does not include f (+2 points). Let g be the last activity in X to start. Since f starts after g does, f cannot conflict with any class in the set $X - \{g\}$. Thus, the schedule $X' = X \cup \{f\} - \{g\}$ is also conflict-free. Since X' has the same size as X, it is also maximal. Therefore, the best schedule that contains f is an optimal schedule (+2 points). The best schedule that includes f must contain an optimal schedule for L which is the subset of activities that finish before f starts. By the inductive hypothesis, computes an optimal schedule of L (+1 points).

Grading Criteria:
(+1 points) If you proved the equivalent of a greedy approach to select activities with the earliest finish time.

(c) (5 points) (1) Sort activities by **duration** (**finish** $-$ **start**) (2) Scan activities in the sorted order and select activities with **the shortest duration** that does not conflict with selected activities so far.



Output : $\{a_1\}$
Optimal Output : $\{a_2, a_3\}$

(+5 points) Correct example.

Problem 3

You are a clerk at a mom-and-pop shop, and your task is to provide the correct change using coins. When given a target amount $C \in \mathbb{N}^+$ and coin values $\{v_1, v_2, \dots, v_n \mid v_i \geq v_j \text{ if } i < j\}$, your goal is to determine the minimum number of coins required to make that amount. More formally, the problem can be expressed as the following optimization:

$$\min_{\{a_1, a_2, \dots, a_n\}} \sum_{i=1}^{n} a_i \quad \text{s.t.} \quad \sum_{i=1}^{n} a_i v_i = C, \quad a_i \geq 0 \; \forall i \in \{1, 2, \dots, n\}.$$

Consider coin values $\{v_1 = 50, v_2 = 10, v_3 = 5, v_4 = 1\}$.

(a) (2 points) Say C=78. What is the minimum number of coins to reach C, and what is the coin combinations $\{a_1, a_2, a_3, a_4\}$ for such a value?

The minimum number of coins is 7, which is when $\{a_1, a_2, a_3, a_4\} = \{1, 2, 1, 3\}$. **Grading Criteria:**

- (+1 points) Got the minimum number of coins correct.
- (+1 points) A correct coin combination to reach C is given.

(b) (3 points) Create an algorithm with a **greedy approach** that will correctly return the **minimum number of coins** needed for the above case in (a). You may use pseudocode if needed.

Algorithm:

(a) Sort the coin denominations in descending order.

(b) Initialize coins = 0 and remaining = C.

(c) For each coin value v_i in the coin list sorted by decreasing value:
   - Take as many coins as possible: count = $\lfloor \text{remaining}/v_i \rfloor$.
   - Update remaining: remaining = remaining − (count × v_i).
   - Increment coins by count.

(d) Repeat until remaining = 0.

(e) Return coins.

Pseudocode:

```
function greedyCoinChange(C, coins={50,10,5,1}):
    total_coins = 0
    remaining = C
    for coin in coins:
        count = remaining // coin
        remaining -= count * coin
        total_coins += count

    return total_coins
```

Grading Criteria:

- (+1 points) Algorithm correctly outputs the number of coins for C=78, and coin values $\{v_1 = 50, v_2 = 10, v_3 = 5, v_4 = 1\}$.

- (+2 points) Correct greedy policy.

(c) (7 points) Does the algorithm in (b) return the correct value for every target amount $C \in \mathbb{N}^+$ under the given coin values $\{v_1 = 50, v_2 = 10, v_3 = 5, v_4 = 1\}$? Either prove its correctness, or present a counterexample.

Since $v_4 = 1$, it is obvious that the greedy approach stops to return a valid solution. Now, we use **induction** to prove the greedy algorithm's correctness. Suppose the greedy algorithm returns the optimal solution for $C < k$, and consider $k + 1$.

Let $v_j := max\{v_i | v_i \leq k+1\}$. Recall that the greedy policy will pick the v_j coin in this case.

Consider the optimal solution that has the coin combinations $\{a_1, a_2, a_3, a_4\}$. Let's assume the optimal solution does not contain v_j. Here, $a_i \leq \frac{v_{i+1}}{v_i} - 1$ for $i < j$, as if $a_i = \frac{v_{i+1}}{v_i}$, we can just substitute the $\frac{v_{i+1}}{v_i}$ coins with v_i value with the one with v_{i+1} value.

*(However, this is **not enough** to prove the optimality of the greedy solution, as we need to disprove the possiblity of there being more optimal solution that doesn't pick v_j, but still abides by the above rule.)*

Thus, the total value of the optimal solution is $\sum_{i=1}^{n} a_i v_i = a_1 * v_1 + \ldots a_{j-1} * v_{j-1} \leq (\frac{v_2}{v_1} - 1) * v_1 + \cdots + (\frac{v_j}{v_{j-1}}) * v_{j-1} = v_j - v_1 = v_j - 1 < v_j < k+1$. This is a contradiction as $\sum_{i=1}^{n} a_i v_i = C$ by definition. In other words, a solution that chooses not to select v_j cannot reach C, unless $a_i > \frac{v_{i+1}}{v_i} - 1$ for some $i < j$, which is suboptimal.

Thus, the optimal solution must choose v_j coin, identical to the greedy policy. Since the remaining target amount is $k + 1 - v_j < k$, by induction, the greedy solution is optimal for k+1, and evidently for any $C \in \mathbb{N}^+$.

Grading Criteria:

- (+1 points) Correct answer.
- (+3 points) Show $a_i \leq \frac{v_{i+1}}{v_i} - 1$ must be satisfied using the exchange argument.
- (−2 points) Flawed exchange argument.
- (+3 points) Show that the optimal solution for C that abides the above rule must follow the greedy solution (i.e. unique) or else you cannot reach C.
- (−2 points) Did not generalize or sufficiently prove above statement.

(d) (3 points) Now let us consider a different coin system. Does the above algorithm in (b) return the correct value for every target amount $C \in \mathbb{N}^+$ under the given coin values of coin values $\{v_1 = 500, v_2 = 200, v_3 = 50, v_4 = 10, v_5 = 1\}$.? Either prove its correctness, or present a counterexample.

Yes. We follow a similar logic with (c), but since we've already shown that a coin system with $v2, v3, v4, v5$ outputs an optimal value as v_i is divisible by v_{i+1}, we only need to show that for every optimal solution for $C > 500$ that doesn't pick the 500 coin can be exchanged with one that does while not increasing the coin count.

Consider the optimal solution for a given target $C > 500$ has $\{0, a_2, a_3, a_4, a_5\}$.
If $a_2 >= 3$, three 200 coins can be exchanged with one 500 coin and two 50 coins.
If $a_2 = 2$, then to achieve $C > 500$, $a_3 > 2$. At which point, we can exchange two 200 coins and two 50 coins with one 500 coin.
If $a_2 < 2$, then this contradicts $C > 500$, as the maximimum value for this optimal solution is $C < 200 * 1 + 50 * 3 + 10 * 4 + 1 * 9 = 399$.

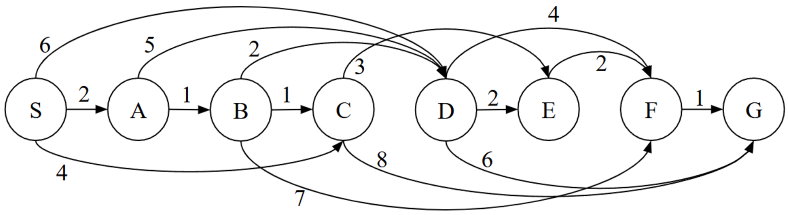
Thus, we've shown that there exists an optimal solution that contains a 500 coin given $C > 500$, and by induction (as done in (c)), the greedy algorithm produces the optimal value for every target.

Grading Criteria:

- (+1 points) Correct answer
- (+2 points) Correct proof on the algorithm's correctness.
- (-1 points) Misses handling some cases.

Problem 4

Consider the linearized graph G.



Algorithm 2 Minimum Cost

Input: $G = (V, E)$; $v \in V$, $(m, n) \in E$
Initialize:
$H[v] = \infty$; $v \in V$ ▷ C, H is a hashtable
$H[S] = 0$
$C[(m, n)] = cost$; $(m, n) \in E$
Run:
for each $n \in V$ except S, in linearized order **do**
    $H[n] =$ ____________
return $H[G]$

(a) (3pts) Fill in the blank to find the minimum cost from S to G.

$$H[n] = \underline{min_{(m,n)\in E}(H[m] + C[(m, n)])}$$

(+3 pts) Correct answer.
(−0.5 pts) Missing iteration of edges.

(b) (2pts) Apply the algorithm and fill in the chart to get a minimum cost from S to $v \in V$.

S	A	B	C	D	E	F	G
0	2	3	4	5	7	9	10

(+0.25 pts) Correct answer per blank.

(c) (4pts) Write down the asymptotic complexity of the 'Minimum Cost' algorithm (no need to include the cost for initialization) and briefly explain it in one sentence.

$$O(V + E)$$

The algorithm iterates through all vertices $n \in V$ and all $(m, n) \in E$ without duplicate, with $n \in V$ as the destination node for each vertex during iteration.
(+1 pts) Correct complexity ($O(|V| + |E|)$, $O(V^2)$, $O(n^2)$).
(+3 pts) Correct explanation. Partial(1 to 2) points (major to minor) error in explanation.

(d) (2pts) Which condition of the graph makes the above algorithm work? Write in one word.

acyclic or linearity

(+2 pts) Correct answer. I gave full credit if the meaning is right.

(e) (4pts) Convert a few lines of pseudocode to change the 'Minimum Cost' algorithm to the 'Maximum Cost' algorithm. The 'Maximum Cost' algorithm should compute the maximum cost of a graph.

line 3: $H[v] = 0;\ v \in V$
line 8: $H[n] = max_{(m,n)\in E}(H[m] + C[(m,n)])$
(+2 pts) Each line.
Other allowed answer for L8: converting into negative weight and return $-H[G]$.

Problem 5

Solve the following problems using dynamic programming.

(a) (5 points) Assume there is a 1D array A of integers with a length L. For any s, e satisfying $0 \leq s \leq e \leq L-1$, the goal is to compute the sum $A[s] + A[s+1] + \cdots + A[e]$. There are K such arbitrary pairs of (s, e). Explain a method to compute all these sums with a time complexity of $O(L + K)$, assuming that both L and K are very large integers.

Let's define an array P of length $L + 1$ as follows:

$$P[0] = 0$$
$$P[i] = P[i-1] + A[i-1] \quad (1 \leq i \leq L)$$

Using this definition, the array P can be constructed in time complexity of $O(L)$.

The value $P[k]$ represents the cumulative sum $A[0] + A[1] + \cdots + A[k-1]$. For any arbitrary pair (s, e), the sum of the elements $A[s] + A[s+1] + \cdots + A[e]$ can be expressed as:

$$A[s] + A[s+1] + \cdots + A[e] = P[e+1] - P[s]$$

Once the array P is pre-computed, the sum for any (s, e) can be calculated in $O(1)$ time. Given K such pairs (s, e), the total time complexity for computing all sums is $O(K)$.

Thus, the overall time complexity is:

$$O(L + K)$$

Grading Criteria:

- (+5 points) Correct answer.
- (−1 points) Do not verify the time complexity when calculating the prefix sum.
- (−1 points) Wrong index in the formula for prefix sum.
- (−1 points) Wrong notation.
- (−4 points) Wrong time complexity.
- (−5 points) Do not solve using dynamic programming.
- (−5 points) Provide incomplete/unclear explanation or contain fatal error.

(b) (10 points) Assume there is a 2D array B of integers with dimensions $M \times N$. For any indices (s_1, e_1) and (s_2, e_2) such that $0 \leq s_1 \leq e_1 \leq M-1$ and $0 \leq s_2 \leq e_2 \leq N-1$, the goal is to compute the sum of all elements in the submatrix defined by these indices:

$$\text{Sum} = \sum_{i=s_1}^{e_1} \sum_{j=s_2}^{e_2} B[i][j].$$

Explain a method to compute all the sums with a time complexity of $O(M \cdot N) + O(K)$, where K is the number of submatrix queries. Assume that M, N, and K are very large integers.

Let's define an array Q of size $(M+1) \times (N+1)$ as follows:

$$\begin{aligned}
&Q[i][0] = 0 \quad (0 \leq i \leq M) \\
&Q[0][j] = 0 \quad (0 \leq j \leq N) \\
&Q[i][j] = B[i-1][j-1] + Q[i-1][j] + Q[i][j-1] - Q[i-1][j-1] \quad (1 \leq i \leq M,\ 1 \leq j \leq N)
\end{aligned}$$

Using this definition, the array Q can be constructed in time complexity of $O(M \cdot N)$.

For any arbitrary pair (s_1, e_1, s_2, e_2), the sum of the submatrix can be expressed as:

$$\text{Sum} = Q[e_1+1][e_2+1] - Q[s_1][e_2+1] - Q[e_1+1][s_2] + Q[s_1][s_2]$$

Thus, once Q is constructed, the sum for any (s_1, e_1, s_2, e_2) can be computed in $O(1)$. Given that there are K such pairs, the total time complexity for computing all sums is $O(K)$.

Therefore, the overall time complexity is:

$$O(M \cdot N + K)$$

Grading Criteria:

- (+10 points) Correct answer.
- (−1 points) Wrong notation.
- (−2 points) Wrong index in the formula for prefix sum.
- (−3 points) Contain error in the formula for the array Q / Do not provide enough explanation about the array Q.
- (−3 points) Contain error in the formula for prefix sum.
- (−4 points) Do not verify the time complexity when calculating the prefix sum.
- (−5 points) Include intermediate steps but do not calculate the prefix sum.
- (−8 points) Wrong time complexity.
- (−10 points) Do not solve using dynamic programming.
- (−10 points) Provide incomplete/unclear explanation or contain fatal error.

Problem 6

Given a rectangular sheet of paper with integer side lengths n and m, you must cut it into **square pieces**, each with *integer* side length. You are allowed **only straight horizontal or vertical** cuts that **completely divide** the current piece into two distinct rectangles. (After all cuts are made, every resulting piece should be a perfect square.)

(a) (8 points) Design a dynamic programming approach to determine the minimum number of square pieces. Let $num[n, m]$ denote the minimum number of square pieces obtained from a sheet with side lengths n and m. Determine the base cases and formulate the recurrence relation.

$$num[n, m] = 1 \text{ if } n = m \text{ (Base case)}$$

$$\begin{aligned} num[n, m] = \min\{ &\min_{1 \leq k \leq \lfloor n/2 \rfloor} (num[k, m] + num[n - k, m]), \\ &\min_{1 \leq k \leq \lfloor m/2 \rfloor} (num[n, k] + num[n, m - k])\} \text{ (Recurrence relation)} \end{aligned}$$

Grading Criteria:

- (+3 points) Correct base case.
- (+5 points) Correct recurrence relation.

(b) (7 points) Develop a dynamic programming approach to determine the minimum total cutting length, where the cutting length is defined as the sum of all cut lengths made. Let $len[n, m]$ denote the minimum cutting length obtained from a sheet with side lengths n and m. Determine the base cases and formulate the recurrence relation.

$$len[n, m] = 0 \text{ if } n = m \text{ (Base case)}$$

$$\begin{aligned} len[n, m] = \min\{ &\min_{1 \leq k \leq \lfloor n/2 \rfloor} (len[k, m] + len[n - k, m] + m), \\ &\min_{1 \leq k \leq \lfloor m/2 \rfloor} (len[n, k] + len[n, m - k] + n)\} \text{ (Recurrence relation)} \end{aligned}$$

Grading Criteria:

- (+3 points) Correct base case.
- (+4 points) Correct recurrence relation.

Problem 7

The decision version of the MAX 2-SAT problem is defined as follows:

Given a 2-CNF formula F (a conjunction of clauses, each with at most 2 literals) and an integer k, decide whether there exists an assignment of truth values to the variables of F such that at least k clauses of F are satisfied.

(a) (3 points) Prove that the decision version of MAX 2-SAT is in NP.

Given an assignment of variables which is known to make k clauses of F satisfied, plug the assignment to the CNF and check if there are k clauses which are satisfied.

Grading Criteria:

- (+3 points) Correct explanation.
- (0 points) Incorrect or missing explanation.

(b) (2 + 3 points) Given a clause $c = (l_1 \lor l_2 \lor l_3)$, consider the following 2-CNF formula with a new variable x.

$$F_c = (l_1) \land (l_2) \land (l_3) \land (x) \land (\neg l_1 \lor \neg l_2) \land (\neg l_2 \lor \neg l_3) \land (\neg l_3 \lor \neg l_1) \land (l_1 \lor \neg x) \land (l_2 \lor \neg x) \land (l_3 \lor \neg x)$$

1. (2 points) Prove that if c is not satisfied, then at most 6 clauses of F_c can be satisfied.

   The only case when c is not satisfied is when all l_i's are false. If x is true, then there are 4 clauses which are satisfied. If x is false, then there are 6 clauses which are satisfied.

   Grading Criteria:

   - (+2 points) Correct explanation.
   - (0 points) Incorrect or missing explanation.

   Therefore, if c is false, then there are at most 6 clauses which are satisfied.

2. (3 points) Prove that if c is satisfied, then there is some assignment of variables so that exactly 7 clauses of F_c are satisfied. Also prove that no more than 7 clauses can be satisfied.

   Consider the case when all l_i's are true. Then by choosing x to be true, 7 clauses out of 10 clauses of F can be satisfied. If x is false, there are only 6 clauses of F which are satisfied.

   Now, without loss of generality, let l_1 be true and l_3 be false. If l_2 is true, regardless of x, there are 7 clauses of F which are true. If l_2 is false, there are 7 clauses of F which are satisfied when x is false, and 6 if x is true.

   Therefore, we proved that there is an assignment which 7 out of 10 clauses of F_c becomes true when c is satisfied, and additionally no more clauses can be satisfied.

   Grading Criteria:

   - (+3 points) Correct explanation.
   - (1 point) Showed that 7 clauses can be satisfied in any case, but no explanation that 7 clauses are the maximum number of clauses that can be satisfied.
   - (0 points) No explanation that 7 clauses can be satisfied in any case.
   - (0 points) Incorrect or missing explanation.
   - (-1 point) per missing case.

(c) (7 points) Using the fact that 3-SAT is NP-COMPLETE, prove that the decision version of MAX 2-SAT is also NP-COMPLETE.

Given a 3-CNF formula F, convert into a corresponding 2-CNF formula F' as stated in (b), so that F' contains m more variables than F and contains $10m$ clauses, where m is the number of clauses of F. Then there is some assignment of variables which $7m$ clauses of F'

can be satisfied, if and only if F can be satisfied, by (b). Therefore there is a polynomial time reduction from 3-SAT to the decision version of Max 2-SAT, which implies that the decision version of Max 2-SAT is NP-Hard. As the decision version of Max 2-SAT is also in NP by (a), we may conclude that the decision version of Max 2-SAT is NP-Complete.

Grading Criteria:

- (+1 point) Noted that Max 2-SAT is in NP.
- (+4 points) Correctly reduced 3-SAT to Max 2-SAT, with correct k.
- (+2 points) Along with the above, concluded that Max 2-SAT is in NP-Hard.
- (0 points) Incorrect or missing explanation.
- (-1 point) per mistake.

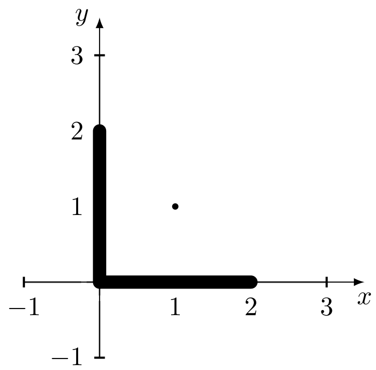
Problem 8

In the MINIMUM STEINER TREE problem, you are given a finite complete graph G, with $T \subseteq V(G)$ which is a set of terminal nodes, with a distance function $d : V(G) \times V(G) \to \mathbb{R}_{\geq 0}$ which satisfies the triangle inequality. (In other words, it satisfies three rules stated below.) Also, let the weight of the subgraph H of G be the sum of distances of edges of H. The goal is to find a tree which is a subgraph of G with minimum edge weight (the sum of distances of edges) which passes through all vertices in T. This tree may or may not include vertices of $V(G)$.

1. For every $u, v \in V(G)$, $d(u, v) = 0$ if and only if $u = v$.
2. For every $u, v \in V(G)$, $d(u, v) = d(v, u)$.
3. For every $u, v, w \in V(G)$, $d(u, w) \leq d(u, v) + d(v, w)$.

For simplicity, let $w(H)$ denote a weight of H, a subgraph of G.

(a) (2 points) Let G be a graph which $V(G) = \{(0,0),(0,2),(1,1),(2,0)\}$, $T = \{(0,0),(0,2),(2,0)\}$, and $d((x_1, y_1),(x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Draw a minimum Steiner tree of G on the xy-plane, and compute its weight. You don't need an explanation for this problem.



The weight of the minimum Steiner tree of G is 4.

Grading Criteria:

- (+2 points) Both the figure and the value are correct.
- (0 points) Incorrect.

(b) (4 points) Prove that, given an optimal Steiner tree G' of G and some terminal nodes $u, v \in T$, the weight of any path from u to v on G' is not smaller than the weight of the edge uv.

Recall that for any two distinct nodes u, v of a tree, there is a unique path from u to v on the tree.

The case when $u = v$ is trivial, so we only consider the case when $u \neq v$. Let the path from u to v in the Steiner tree G' of G be $ut_1 \dots t_n$ with $t_n = v$, and let $u, v \in T$. We prove that for any two distinct vertices $u, v \in V(G')$, the weight of the path $ut_1 \dots t_n$ with $t_n = v$ of G' from u to v is not less than the weight of the edge uv in G, by induction on n. If $n = 1$, then the weight of the path $ut_1 = uv$ is equal to the weight of the edge uv, so the statement holds. If $n > 1$, then by the induction hypothesis, the weight $w(ut_1 \dots t_{n-1}$ of the path from u to t_{n-1} is not less than the weight of the edge ut_{n-1}, which is $w(ut_{n-1}) = d(u, t_{n-1})$. By the triangle inequality,

$$\begin{aligned} w(ut_1 \dots t_{n-1}t_n) &= w(ut_1 \dots t_{n-1}) + d(t_{n-1}, t_n) \\ &\geq d(u, t_{n-1}) + d(t_{n-1}, t_n) \geq d(u, t_n) = d(u, v) \end{aligned}$$

Therefore, for any path from u to v of G', its weight is not less than the weight of the edge uv. Choose u and v to be in T, and conclude that the problem statement holds.

Grading Criteria:

- (+4 points) Both the figure and the value are correct.
- (0 points) Incorrect.
- (-1 point) per mistake.

(c) (7 points) Prove that the weight of a minimum spanning tree of T on G does not exceed twice the weight of the minimum Steiner tree of G.

Choose some vertex $v \in T$ of G', the minimum Steiner tree of G. Consider a DFS walk $s_0 \dots s_n$ on G' starting from $v = s_0$ and ending at $v = s_n$. Then each edge of G' is counted twice on the walk s, so $w(s_0 \dots s_n) = 2w(G')$. If we let $t_i \in T$ for $0 \le i < |T|$ so that all t_i are distinct and $t_0 \dots t_{|T|-1}$ is a subsequence of $s_0 \dots s_n$,

$$w(S) \le w(t_0 \dots t_{|T|-1}) \le w(s_0 \dots s_n) = 2w(G')$$

where S is a minimum spanning tree of T with respect to G. The second inequality holds by (b) and the fact that the walk from u to v on G' passes through all edges of a path from u to v.

Grading Criteria:

- (7 points) Reasonable proof of the problem statement.
- (0 points) Incorrect.
- (-1 point) per mistake.

(d) (2 points) Briefly explain a 2-approximation algorithm for the Minimum Steiner tree problem.

Find a minimum spanning tree of T with respect to G. By (c), this is a 2-approximation of the minimum Steiner tree of G.