CS 6515/4540 (Fall 2025)

Final Exam

Name: _____

GTID: _____

GT Username: _____

INSTRUCTIONS

1. When the exam starts, write your name (or GT ID or GT Username) on each page.
2. Bringing six pages of notes is permitted.
3. No books, calculators, or other electronic devices permitted.
4. Make your answers clear and concise.
5. ONLY write on the **FRONT** of each sheet. Backs will not be scanned.
6. You may refer to standard algorithms from class without giving full pseudocode.

I am aware of and in accordance with the Academic Honor Code of Georgia Tech and the Georgia Tech Code of Conduct. I will use no person's help on this test. Also, I have read all the instructions on this page.

Signature:

---

1 Short Answer Questions (36 = 9 × 4 points)

1. Consider the following proof by induction that all horses have the same color.

> **k-th Induction statement**: Any k horses have the same color.
>
> **Induction hypothesis**: Assume the induction statement holds for $\leq k$.
>
> **Induction step**: Now, consider any group of $k+1$ horses. By inductive hypothesis, the first k horses have the same color. By inductive hypothesis, the last k horses also have the same color. Since both groups share a common horse (e.g., the second horse in the line), all $k+1$ horses have the same color.

Is this proof correct? If not, where is the mistake?

No. We are missing the base case. Another bug is that the first k and last k have an intersection only if $k \geq 2$, not for $k = 1$.

2. When we update a flow f using an augmenting path P, the flow value $f(e)$ strictly increases for every edge e that is part of the path P. Explain why this is true or false.

False. The edges that are pointing backwards along this path in the residual graph will have a decreased flow.

3. In class we saw that max-independent set on trees can be solved in polynomial time using dynamic programming. Since max-independent set is NP Hard, does this mean some NP Hard problems can be solved in polytime? Explain why this is true or false.

False. Max-independent set is NP Hard on general graphs, not on trees.

4. Consider a Primal Linear Program and its Dual. If the Primal has a finite optimal solution, what can we say about the Dual? No justification required.

   ◯ The Dual also has a finite optimal solution, and but their objective values can be different.

   ◯ The Dual also has a finite optimal solution, and their objective values are equal.

   ◯ The Dual is infeasible.

   ◯ The Dual is unbounded.

   > The Dual also has a finite optimal solution, and their objective values are equal.

5. If there exists an algorithm for the Orthogonal Vectors problem that runs in time $O(n^{1.9}d)$, then the Strong Exponential Time Hypothesis (SETH) is false. Explain why this is true or false (you may assume anything from class).

   > True. We saw in class that SETH reduces to OV.

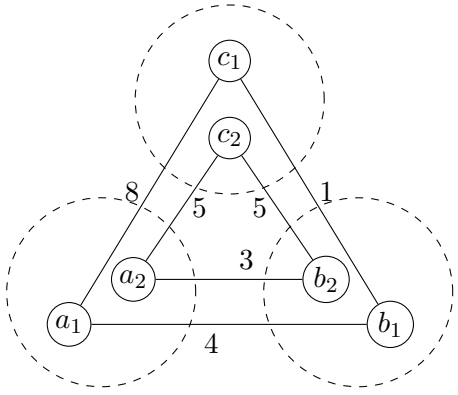
6. Suppose we have an edge-weighted graph G, whose $3n$ vertices are partitioned into three clusters (see figure):

   $$A = \{a_1, \ldots, a_n\}, \quad B = \{b_1, \ldots, b_n\}, \quad C = \{c_1, \ldots, c_n\}.$$

   We have no information about the edges *within* each cluster. The only inter-cluster edges (i.e., edges between different clusters) are the following 6, along with their weights:

   $$w(a_1, b_1) = 4, \quad w(a_2, b_2) = 3, \quad w(b_1, c_1) = 1, \quad w(b_2, c_2) = 5, \quad w(c_1, a_1) = 8, \quad w(c_2, a_2) = 5.$$

   Which of the 6 inter-cluster edges must appear in **every** MST of G? Briefly justify your answer.



Edges (b_1, c_1) and (a_2, b_2) since there are cuts with these edges being the min weight cross edges.

7. In the context of Online Learning for loss minimization, what does it mean for an algorithm to have "no-regret" (or vanishing regret)? No justification required.

   ◯ The algorithm's cumulative (total) loss is $o(T)$ after T rounds.

   ◯ The algorithm's average regret $\frac{R_T}{T}$ goes to zero as $T \to \infty$.

   ◯ The algorithm eventually suffers no loss in any round as $T \to \infty$.

   ◯ The algorithm's cumulative (total) regret R_T is $O(1)$.

   The algorithm's average regret $\frac{R_T}{T}$ goes to zero as $T \to \infty$.

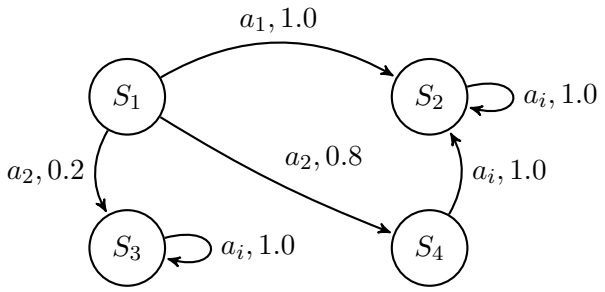
8. Recall the knapsack problem: we have n jobs of sizes s_i and values v_i for $i \in \{1, \ldots, n\}$, and the goal is to select a subset A of jobs with $\sum_{i \in A} s_i \leq B$ while maximizing $\sum_{i \in A} v_i$.

   Consider the greedy algorithm that first renames the jobs such that $v_1/s_1 \geq v_2/s_2 \geq v_3/s_3 \geq \ldots \geq v_n/s_n$, and now selects jobs in this order, picking the next job if it can still be accommodated within budget B (otherwise skipping it). Give an example where this greedy algorithm returns a suboptimal solution.

   Let $n = 2$ where $v_1 = 0.2$ and $s_1 = 0.1$ and $v_2 = 1$ and $s_2 = 1$. For $B = 1$, the optimal solution is to take the 2nd job but greedy takes 1st job.

9. Consider the MDP with 4 states $\{S_1, S_2, S_3, S_4\}$ and 2 actions $\{a_1, a_2\}$ as given in the figure. Here, the tuple a, p on an edge means that if we perform action $a \in \{a_1, a_2\}$ at the start state then we end at the destination state with probability p. (When we write a_i in the figure that means this is true for both a_1 and a_2.)

   Moreover, we receive a reward 0 on pulling any action in S_1; we receive a reward 10 on pulling any action in S_2; we recieve a reward 50 on pulling any action in S_3; we receive a reward 5 on pulling any action in S_4. Starting at S_1, what is the optimal policy for horizon $T = 2$ (i.e., total 2 pulls)? Justify briefly.



Perform a_2 followed by a_i.
If we perform a_1 first and then a_i, our reward is 10. But performing a_2 first ensures we get $0.2 \cdot 50 + 0.8 \cdot 5 = 14$.

2 Divide & Conquer and DP (22 points)

1. **Majority Element (10 points).** An array A of size n has a *majority element* if there is an element x that appears strictly more than $n/2$ times. We want to find this element (or determine if none exists). You can only check if two elements are equal (e.g., $A[i] == A[j]$); elements are not be comparable via $<$ or $>$, so you cannot sort. Our goal is to design a Divide & Conquer algorithm to solve this problem.

   (a) (4 points) Design a procedure `CheckCandidates`(A, m_L, m_R) that takes:

   - an array A, and
   - two candidates m_L and m_R, which are the (possible) majority elements of the left and right halves of A (each may be an element of A or `None`),

   and returns $m \in \{m_L, m_R\}$ if it is the majority element of A or `None` otherwise. You may only use $O(|A|)$ equality checks between elements of A.

   1. Initialize $count_L = 0$ and $count_R = 0$.
   2. If $m_L \neq$ `None`:
      Iterate through A: if $A[i] == m_L$, increment $count_L$.
      If $count_L > |A|/2$, return m_L.
   3. If $m_R \neq$ `None` and $m_R \neq m_L$:
      Iterate through A: if $A[i] == m_R$, increment $count_R$.
      If $count_R > |A|/2$, return m_R.
   4. Return `None`.

   Runtime: We scan A at most twice, so time is $O(|A|)$.

   (b) (3 points) Using `CheckCandidates`, design a divide-and-conquer algorithm to find the majority element of A (or report that none exists). Briefly explain its correctness.

   Procedure: Solve(A):

   1. If $|A| == 0$, return `None`. If $|A| == 1$, return $A[0]$.
   2. Split A into left half L and right half R: $m_L = \text{Solve}(L)$ and $m_R = \text{Solve}(R)$
   3. Return `CheckCandidates`(A, m_L, m_R).

   Correctness: If A has a majority element x (appearing $> n/2$ times), then x **must** be a majority element in at least one of the subarrays L or R. Thus, the recursive calls will identify x, and `CheckCandidates` verifies it against the full array.

   (c) (3 points) Derive the recurrence for the running time of your algorithm and solve it to obtain the asymptotic runtime. (You may use Master's theorem.)

   The recurrence is: $T(n) = 2T(n/2) + O(n)$ This gives $T(n) = O(n \log n)$.

2. **Maximum Matching on a Tree (12 points)**

   You are given a **tree** $G = (V, E)$ with n vertices and **rooted** at $r \in V$. We wish to compute a maximum-size matching using dynamic programming.

   For each vertex v, let $T(v)$ denote the size of maximum matching in the subtree rooted at v.

   (a) (6 points) State a recurrence for $T(v)$ in terms of the DP values of its children (and possibly grandchildren). Briefly justify why your recurrence correctly captures the maximum matching in the subtree rooted at v.

   Let $C(v)$ denote the set of children of v. The recurrence for $T(v)$ is:

   $$T(v) = \max \left(\underbrace{\sum_{u \in C(v)} T(u)}_{\text{Case 1: } v \text{ is unmatched}} \quad , \quad \max_{u \in C(v)} \left\{ 1 + \sum_{g \in C(u)} T(g) + \sum_{w \in C(v), w \neq u} T(w) \right\} \right)$$

   Justification: Consider two cases:

   1. v **is unmatched:** The max matching is simply the sum of the max matchings of all subtrees rooted at its children ($u \in C(v)$).
   2. v **is matched to child** u**:** We gain 1 edge (v, u).
      - Since u is matched to v, u cannot participate in a matching with its own children (the grandchildren of v). Thus, for the subtree at u, we take the sum of matchings of the grandchildren: $\sum_{g \in C(u)} T(g)$.
      - For all other children w of v (where $w \neq u$), their subtrees are unaffected, so we take $\sum T(w)$.

      We maximize over all possible choices of child u.

   (b) (6 points) Analyze the running time of your dynamic programming algorithm.

   Analysis: The algorithm runs in $\boldsymbol{O(n)}$ time. This is because each vertex is considered once in its parent's calculation and once in its grandparent's calculation, so at most twice per vertex.

3 LP + Randomized Algorithms (22 points)

1. **Linear Programming Formulation (12 points).** Consider the **Shortest Path** problem. You are given a *directed* graph $G = (V, E)$ with edge weights $w_{uv} \geq 0$, a source $s \in V$, and a destination $t \in V$ where $s \neq t$. In this problem, we want to show that the optimal value of the following LP, where we have a variable d_v for every vertex $v \in V$, gives the length of shortest path between s and t:

$$\max \qquad d_t - d_s$$

Subject to:

$$d_v - d_u \leq w_{uv} \quad \forall (u, v) \in E$$

$$d_v \geq 0 \quad \forall v \in V$$

(a) (6 points) Consider a solution to the LP where for every vertex $v \in V$ we set d_v to be the shortest path between s and v.
Prove that this is a **feasible** LP solution. What is the objective value of this solution?

Feasibility: We set $d_v = \text{dist}(s, v)$ for all v, which are all non-negative. The shortest path distances satisfy the **triangle inequality**: for any edge (u, v), the shortest path to v is at most the shortest path to u plus the edge weight w_{uv}.

$$\text{dist}(s, v) \leq \text{dist}(s, u) + w_{uv}$$

Substituting our variables:

$$d_v \leq d_u + w_{uv} \implies d_v - d_u \leq w_{uv}$$

Objective Value: Since d_s is the distance from s to itself, $d_s = 0$. The objective value is:

$$d_t - d_s = \text{dist}(s, t) - 0 = \text{dist}(s, t)$$

This equals the length of the shortest path.

(b) (6 points) Prove any feasible LP solution has objective at most the shortest $s - t$ path.
Hint: Take a subset of the constraints and add them up to show that the objective is always at most the shortest path.

Let P be the actual shortest path from s to t. Let the sequence of vertices in this path be $s = v_0, v_1, v_2, \ldots, v_k = t$. Since the LP solution is feasible, the constraint $d_{v_{i+1}} - d_{v_i} \leq w_{v_i, v_{i+1}}$ holds. We sum these inequalities for all edges in path P:

$$\sum_{i=0}^{k-1} (d_{v_{i+1}} - d_{v_i}) \leq \sum_{i=0}^{k-1} w_{v_i, v_{i+1}}$$

The left-hand side is a **telescoping sum** and equals $d_t - d_s$. The right-hand side is exactly the total weight of the shortest path.

2. **Randomized 3-coloring (10 points)**. Consider a simple graph $G = (V, E)$. Our goal is to find a 3-coloring of the vertices V (i.e., give each vertex one of 3 colors $\{blue, red, green\}$) that maximizes the number of "good" edges, i.e., edges with both endpoints of different color. Since this problem is known to be NP Complete, we will now design an approximation algorithm.

   (a) (5 points) Show that if we assign each vertex a random color chosen uniformly and independently from the set $\{blue, red, green\}$ (i.e., each color with probability 1/3), then this randomized algorithm gives us a 2/3 approximation in expectation.

   Each edge has pb 2/3 of being "good", so by linearity of expectation we have in expectation $2|E|/3 \geq 2OPT/3$ good edges.

   (b) (5 points) Show that the probability our random coloring is less than a $\frac{1}{2}$ approximation is at most 2/3. **Hint:** Apply Markov's inequality on the random variable $|E| - X$, where X is the number of "good" edges

   $Pr(|E| - X \geq \frac{1}{2}|E|) \leq \frac{\mathbb{E}[|E|-X]}{\frac{1}{2}|E|} = \frac{\frac{1}{3}|E|}{\frac{1}{2}|E|} = \frac{2}{3}$

4 Convex Optimization (20 points)

Consider the function $f : \mathbb{R} \to \mathbb{R}$ given by:

$$f(x) = 3x^2 + 6x + 10$$

1. (3 points) Is this function α-strongly convex? If so, what is the largest valid α?

Yes. Strong convexity requires $f''(x) \geq \alpha$. Since $f''(x) = 6$, we can choose $\alpha = 6$.

2. (3 points) Is this function β-smooth? If so, what is the smallest valid β?

Yes. Smoothness requires $f''(x) \leq \beta$. Since $f''(x) = 6$, we can choose $\beta = 6$.

3. (6 points) Suppose we run Gradient Descent with step size $\eta = 1/6$, starting from $x_0 = 100$. (a) Calculate x_1? (b) Calculate x^* that minimizes $f(x)$?

Update rule: $x_1 = x_0 - \frac{1}{6} f'(x_0)$.

$$f'(100) = 6(100) + 6 = 606$$

$$x_1 = 100 - \frac{1}{6}(606) = 100 - 101 = -1$$

Global min: $f'(x^*) = 6x^* + 6 = 0 \implies x^* = -1$.

4. (8 pts) Suppose there is an *unknown* 1-dimensional convex function $f : [0, 1] \to \mathbb{R}$. Given an error parameter $0 < \epsilon < 1$ and access to a gradient oracle (blackbox) that evaluates $\nabla f(x) = f'(x)$ at any point $x \in [0, 1]$, provide a **binary search** algorithm that makes $O(\log(\frac{1}{\epsilon}))$ calls to the gradient oracle to find an $x \in [0, 1]$ such that $|x - x^*| \leq \epsilon$, where $x^* \in [0, 1]$ minimizes f.

   (a) (4 points) Describe your algorithm.

Algorithm: We maintain a search interval $[L, R]$ which guarantees that the optimal solution x^* lies within it. We start with $L = 0$ and $R = 1$.

1. While $(R - L) > 2\epsilon$:
   - Compute midpoint $m = \frac{L+R}{2}$ and query the oracle to get $g = f'(m)$.
   - If $g = 0$: Return m (we found the exact minimum).
   - If $g > 0$: The function is increasing at m. Since f is convex, it must be increasing everywhere to the right of m. Set $R = m$.
   - If $g < 0$: The function is decreasing at m. Since f is convex, it must be decreasing everywhere to the left of m. Set $L = m$.
2. Return $\frac{L+R}{2}$.

   (b) (4 points) Prove correctness and that the number of oracle calls is $O(\log(\frac{1}{\epsilon}))$.

Correctness: A 1-dimensional convex function satisfies $f'(x)$ is non-decreasing.

- If $f'(m) > 0$: Since f' is non-decreasing, for all $x > m$, $f'(x) \geq f'(m) > 0$. This means f is strictly increasing on $(m, 1]$, so the minimum cannot lie to the right of m. We correctly discard $[m, R]$.
- If $f'(m) < 0$: Similarly, for all $x < m$, $f'(x) \leq f'(m) < 0$. This means f is strictly decreasing on $[0, m)$, so the minimum cannot lie to the left of m. We correctly discard $[L, m]$.

Therefore, the invariant that $x^* \in [L, R]$ is maintained throughout the loop. When the loop terminates, the interval length is at most 2ϵ. By picking the midpoint, our distance to x^* is at most $\frac{2\epsilon}{2} = \epsilon$.

Complexity: In each iteration, the size of the search interval $[L, R]$ is halved. Since initial width is 1 and final is ϵ, total iterations (oracle calls) is $O(\log(1/\epsilon))$.

END OF EXAM