CS 170 Algorithms
Spring 2009 David Wagner

Final

PRINT your name: _____, _____
(last) (first)

SIGN your name: _____

PRINT your Unix account login: _____

Your TA's name: _____

Name of the person sitting to your left: _____

Name of the person sitting to your right: _____

You may consult any books, notes, or other paper-based inanimate objects available to you. Calculators and computers are not permitted. Please write your answers in the spaces provided in the test. We will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there. Unless specified otherwise, you do not need to justify your answers on this exam, and we will not grade any justification you may provide. When asked for answers using $O(\cdot)$ notation, provide the most specific answer possible (for example: do not write $O(n^3)$ if $O(n^2)$ is also correct).

You have 180 minutes (3 hours). There are 8 questions, of varying credit (100 points total). The questions are of varying difficulty, so avoid spending too long on any one question.

Do not turn this page until your instructor tells you to do so.

Problem 1	/ 8
Problem 2	/ 12
Problem 3	/ 21
Problem 4	/ 10
Problem 5	/ 15

Problem 6	/ 7
Problem 7	/ 13
Problem 8	/ 14
Total	

Problem 1. [True or false] (8 points)

Circle TRUE or FALSE.

Give a brief justification for your answer to each part. A sentence or a picture is plenty.

(a) TRUE or FALSE: Representing a graph in adjacency matrix representation takes $\Theta(|V|^2)$ space.

Justification:

True. The matrix has a value for each edge, there may be up to V*V edges.
So the dimensions of the dimensions of the matrix is V^2.

(b) TRUE or FALSE: The minimum spanning tree in a weighted graph G is always unique, no matter what the graph G or the weights are.

Justification:

False. If G contains a cycle with multiple edges of maximum weight, then the spanning tree is not unique.
Only if the edge weights are unique is the MST unique as well.

(c) TRUE or FALSE: Suppose G is a graph with n vertices and $n^{1.5}$ edges, represented in adjacency list representation. Then depth-first search in G runs in $O(n^{1.5})$ time.

Justification:

True. DFS runs in O(E+V) time which is O(n+n^1.5) = O(n^1.5)
Note: Not O(EV)!!!!

Problem 2. [Recurrence relations] (12 points)

Consider the following code for driving a robot in a spiral-like pattern:

SPIRAL(n):
1. If $n \leq 1$, return (without doing any driving).
2. Drive north n miles.
3. Drive west n miles.
4. Drive south n miles.
5. Drive east $n-1$ miles.
6. Drive north 1 mile.
7. Call SPIRAL($n-2$).

Let $M(n)$ denote the number of miles that the robot drives (in total) if we call SPIRAL(n).

(a) $M(1) =$ 0.

(b) $M(2) =$ 2+2+2+1+1 = 8.

(c) Write a recurrence relation for $M(n)$:

$M(n) =$ 4n+M(n-2).

(d) Solve for $M(n)$, as a function of n. Write your answer using $O(\cdot)$ notation.

$M(n) =$ O(n^2).

Using Muster Theorem, a = 1 and d = 1. So O(n^2)

Master Theorem (for divide and conquer recurrences):
Let $T(n)$ be a function defined on positive n, and having the property

$$T(n) \leq \begin{cases} c, & \text{if } n \leq 1, \\ aT(n/b) + f(n), & n > 1, \end{cases}$$

for some constants $c, a > 0, b > 1, d \geq 0$, and function $f(n)$. If $f(n)$ is in $O(n^d)$, then

$$T(n) \text{ is in } \begin{cases} O(n^d), & \text{if } d > \log_b(a), \\ O(n^d \log(n)), & \text{if } d = \log_b(a), \\ O(n^{\log_b(a)}), & \text{if } d < \log_b(a). \end{cases}$$

We will discuss many applications of the Master theorem. First for the sake of comparison, here is a theorem with similar structure, useful for the analysis of some algorithms. Jokingly, call it the

Muster Theorem (for subtract and conquer recurrences):
Let $T(n)$ be a function defined on positive n, and having the property

$$T(n) \leq \begin{cases} c, & \text{if } n \leq 1, \\ aT(n-b) + f(n), & n > 1, \end{cases},$$

for some constants $c, a > 0, b > 0, d \geq 0$, and function $f(n)$. If $f(n)$ is in $O(n^d)$, then

$$T(n) \text{ is in } \begin{cases} O(n^d), & \text{if } a < 1, \\ O(n^{d+1}), & \text{if } a = 1, \\ O(n^d a^{n/b}), & \text{if } a > 1. \end{cases}$$

□

Proof: Expanding the recursion by one step, we have

$$\begin{aligned} T(n) &\leq a(\ T(n-2b) + f(n-b)\ ) + f(n) \\ &= a^2 T(n-2b) + af(n-b) + f(n). \end{aligned}$$

Taking another step, we get

$$T(n) \leq a^3 T(n-3b) + a^2 f(n-2b) + af(n-b) + f(n),$$

etc. This leads to $T(n) \leq \sum_{i=0}^{n/b} a^i f(n-ib)$ plus a constant. Since $f(n-ib)$ is in $\mathrm{O}(n-ib)$ which is in $\mathrm{O}(n)$, we have that $T(n)$ is in $O(n^d \sum_{i=0}^{n/b} a^i)$. The job is finished by this property of the sum:

$$\sum_{i=0}^{n/b} a^i \text{ is in } \begin{cases} O(1), & \text{if } a < 1, \\ O(n), & \text{if } a = 1, \\ O(a^{n/b}), & \text{if } a > 1. \end{cases}$$

Problem 3. [Best guess] (21 points)

Circle TRUE, SUSPECTED TRUE, SUSPECTED FALSE, or FALSE.

A statement is SUSPECTED TRUE if it would be surprising to most computer scientists if the statement turned out to be false—but we have no proof that the statement is true. For example, $1+1=2$ is TRUE (not SUSPECTED TRUE!), while $\mathbf{P} \neq \mathbf{NP}$ is SUSPECTED TRUE.

Give a brief justification for your answer to each part. A sentence or two is plenty.

(a) TRUE, SUSPECTED TRUE, SUSPECTED FALSE, or FALSE: 3SAT is in **NP**.

Justification: 3SAT is NP. TRUE

(b) TRUE, SUSPECTED TRUE, SUSPECTED FALSE, or FALSE: 3SAT is NP-complete.

Justification:

3SAT is NP-Complete. TRUE

(c) TRUE, SUSPECTED TRUE, SUSPECTED FALSE, or FALSE: There is a polynomial-time algorithm for 3SAT.

Justification:

(d) TRUE, SUSPECTED TRUE, SUSPECTED FALSE, or FALSE: The search problem "Given a graph G, vertices s,t, and a threshold h, find a flow from s to t whose value exceeds h, or report that none exists" is NP-complete.

Justification:

(e) TRUE, SUSPECTED TRUE, SUSPECTED FALSE, or FALSE: Given a graph where all edge lengths are integers (possibly negative), there is a polynomial-time algorithm to test whether the graph contains a negative cycle (i.e., a cycle where the sum of the lengths of the edges in the cycle is negative).

Justification:

(f) TRUE, SUSPECTED TRUE, SUSPECTED FALSE, or FALSE: The search problem "Given a graph G with n vertices, find a way to color the graph with $n-1$ colors, or report that no such coloring exists" is in **P**.

Justification:

(g) TRUE, SUSPECTED TRUE, SUSPECTED FALSE, or FALSE: Linear programming is in **P**.

Justification:

Problem 4. [Updating distances] (10 points)

We have a directed graph $G = (V, E)$, where each edge (u, v) has a length $\ell(u, v)$ that is a positive integer. Let n denote the number of vertices in G. In other words, $n = |V|$. Suppose we have previously computed a $n \times n$ matrix $d[\cdot, \cdot]$, where for each pair of vertices $u, v \in V$, $d[u, v]$ stores the length of the shortest path from u to v in G. The $d[\cdot, \cdot]$ matrix is provided to you.

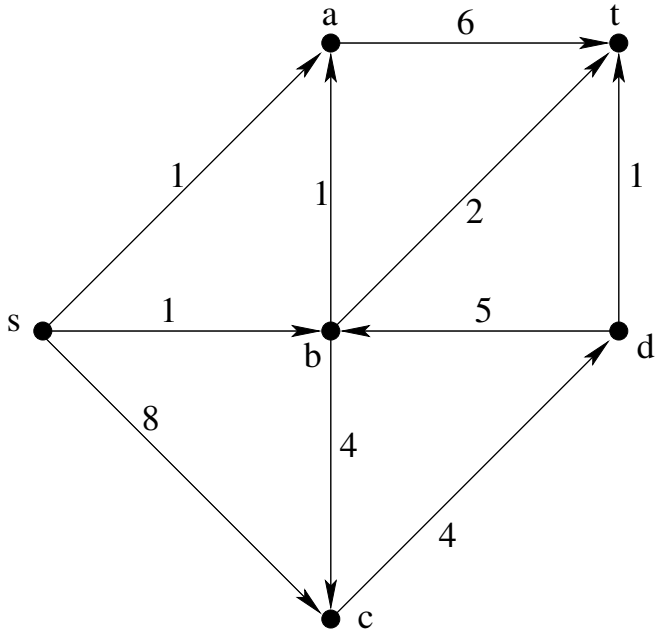
Now we add a single edge (a, b) to get the graph $G' = (V, E')$, where $E' = E \cup \{(a, b)\}$. Let $\ell(a, b)$ denote the the length of the new edge. Your job is to compute a new distance matrix $d'[\cdot, \cdot]$, which should be filled in so that $d'[u, v]$ holds the length of the shortest path from u to v in G', for each $u, v \in V$.

(a) Write a concise and efficient algorithm to fill in the $d'[\cdot, \cdot]$ matrix. You should not need more than about 3 lines of pseudocode.

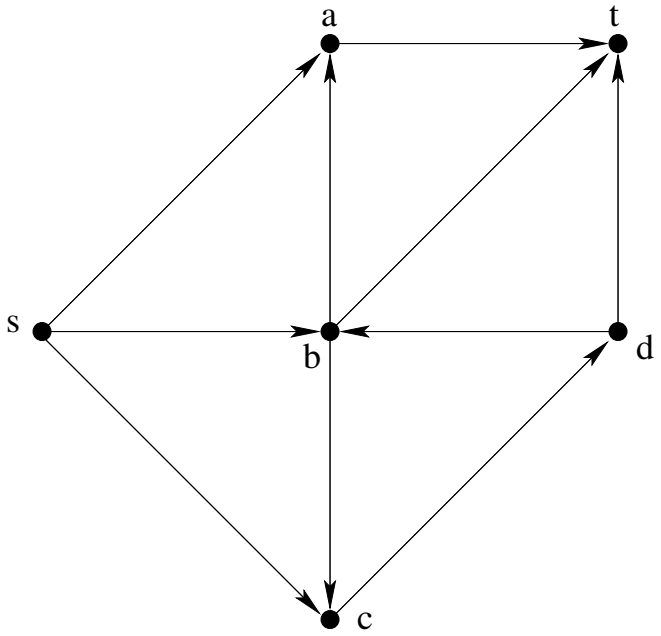
(b) What is the asymptotic worst-case running time of your algorithm? Express your answer in $O(\cdot)$ notation, as a function of n and $|E|$.

Problem 5. [Network flow] (15 points)

Consider the following directed graph. Each edge is labelled with the capacity of that edge. For instance, the edge (s, c) has capacity 8.

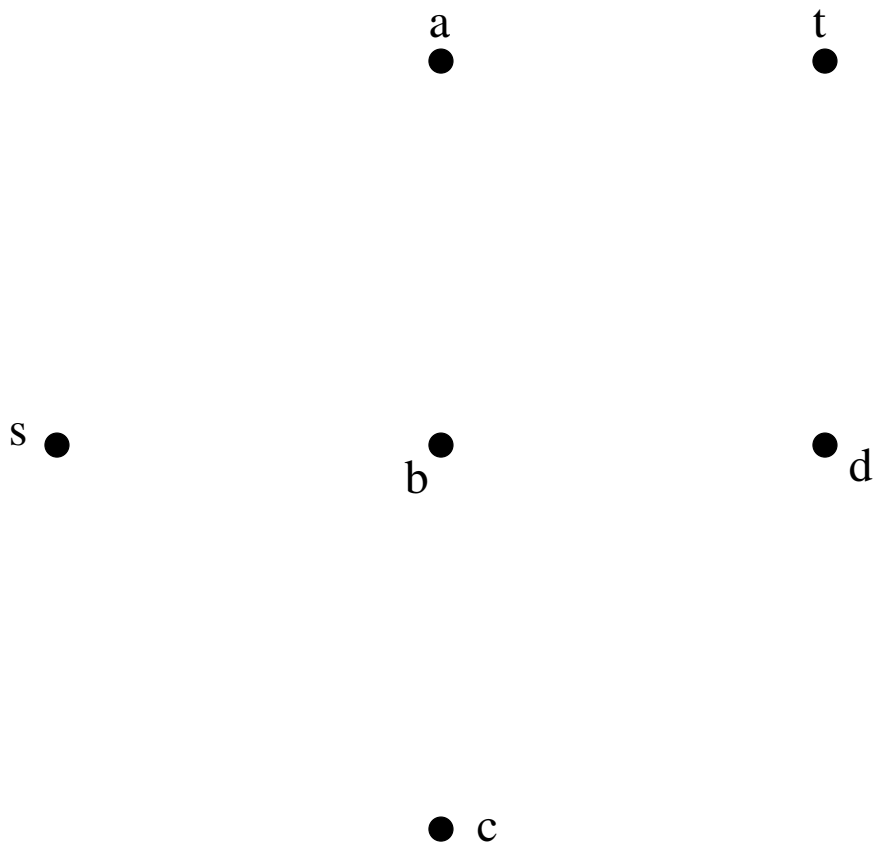


(a) Find the maximum flow from s to t in this graph. Fill in the graph below with your flow: label each edge with the amount of flow you are sending along that edge.



(b) What is the value of your flow?

(c) Draw the residual graph corresponding to your flow from part (a). I've provided the verticies; you fill in the edges of the residual graph. Label each edge in your residual graph with its capacity.



(d) Find the minimum-capacity cut (L,R) between s and t in this graph. Show your answer by drawing a circle around the vertices of L, in the picture above.

(e) What is the capacity of the cut you identified in part (d)?

(f) How could you check you didn't make any mistakes in part (a)? In other words, how could you use your answers to these questions to convince someone else that the flow you drew in part (a) truly is a maximum flow in the graph (without asking them to find the maximum flow themselves from scratch)? Explain, in one or two sentences.

Problem 6. [A reduction] (7 points)

GRAPH 3-EQUICOLORING is the following search problem:

> **Given:** An undirected graph G with $3n$ vertices, for some integer n
> **Find:** A 3-equicoloring of G, or report that none exists

Recall that a 3-coloring is a way to color each vertex of G with one of 3 colors, say Blue, Gold, and Tan, so that no pair of adjacent vertices share the same color. A *3-equicoloring* is a 3-coloring where each color is used on the same number of vertices (so in a graph with $3n$ vertices there are exactly n Blue vertices, n Gold vertices, and n Tan vertices). You may assume that GRAPH 3-EQUICOLORING is NP-complete.

CLIQUE is the following search problem:

> **Given:** An undirected graph G and a positive integer k
> **Find:** A clique of size k in G, or report that none exists

A *clique* of size k is a set of k vertices such that every pair of vertices in the clique are connected by an edge. In other words, vertices $v_1, \ldots, v_k$ form a clique of size k if G includes the edge (v_i, v_j) for every i, j.

Professor Lilac proposes to prove that CLIQUE is NP-complete, by reducing GRAPH 3-EQUICOLORING to CLIQUE. She suggests the following argument:

> Suppose we have an algorithm B that solves the CLIQUE problem. I will show how to build an algorithm A to solve GRAPH 3-EQUICOLORING, using B as a subroutine. The algorithm A is given an undirected graph $G = (V, E)$ with $3n$ vertices. A will construct a new undirected graph $G^* = (V^*, E^*)$, as follows. Each vertex of G^* corresponds to a set of n vertices from G that have no edges between them. In other words,
>
> $$V^* = \{S : S \subseteq V, |S| = n, \text{ and } \forall u, v \in S \,.\, (u, v) \notin E\}.$$
>
> The graph G^* will have an edge (S, T) between the sets S, T if these two sets are disjoint. In other words,
>
> $$E^* = \{(S, T) : S, T \in V^* \text{ and } S \cap T = \emptyset\}.$$
>
> We run algorithm B on input G^* and with $k = 3$, to determine whether there exists a clique of size 3 in G^*. If B finds a clique of size 3 in G^*, then this corresponds to a 3-equicoloring of G (each vertex $S \in V^*$ in the 3-clique corresponds to a set of vertices in V that all receive the same color). Moreover, if there is no clique of size 3 in G^*, then there is no 3-equicoloring of G. This proves that CLIQUE is at least as hard as GRAPH 3-EQUICOLORING. Since we know that GRAPH 3-EQUICOLORING is NP-complete, and since CLIQUE is in NP, it follows that CLIQUE is NP-complete, too.

Is Professor Lilac's reasoning correct? Circle YES or NO below. If you circle NO, briefly explain why in the space below (one or two sentences is plenty).

YES NO

Problem 7. [A game] (13 points)

Consider the following two-player game, which is *very fun*. We start with a $n \times n$ matrix M filled with positive integers. When it is her turn, a player can delete the last row or last column of the matrix, but only if the sum of the numbers in that row/column is even. If the sum of numbers in the last row is odd, and the sum of numbers in the last column is odd, then the player has no legal move and loses the game.

For instance, suppose we initially have a 3×3 matrix

$$\begin{pmatrix} 2 & 2 & 3 \\ 7 & 2 & 1 \\ 2 & 2 & 4 \end{pmatrix}.$$

If Alice moves first, she can delete either the bottom row or the rightmost column (they both sum to 8, which is even). Suppose she deletes the rightmost column, leaving

$$\begin{pmatrix} 2 & 2 \\ 7 & 2 \\ 2 & 2 \end{pmatrix}.$$

Now it is Bob's turn. He has the option of deleting either the bottom row or rightmost column (their sums are both even). Suppose he chooses to delete the bottom row, leaving Alice with

$$\begin{pmatrix} 2 & 2 \\ 7 & 2 \end{pmatrix}.$$

Alice has no choice but to delete the right column, leaving Bob with

$$\begin{pmatrix} 2 \\ 7 \end{pmatrix}.$$

Now Bob loses, since there is no legal move he can make. Well played, Alice!

Your task: given the $n \times n$ matrix $M[\cdot,\cdot]$, design a polynomial-time algorithm to classify the game as either W (first player wins) or L (first player loses). W means that the first player has a strategy to ensure she will win, no matter how the second player plays the game; L means that no matter what the first player does, the second player has a strategy that will ensure him the ultimate victory. (For instance, the 3×3 matrix given above would be classified as W, since there is a strategy Alice can use to ensure she will win.)

In other words, your algorithm should accept $M[1..n, 1..n]$ as input and return either W or L. The running time of your algorithm should be polynomial in n.

(a) In 5 words or less, what would be the best hint you could give to another CS170 student, that would give away how to approach this problem?

(continued on next page)

(b) Show the main idea and pseudocode for your algorithm.

Main idea:

Pseudocode:

Problem 8. [Running time analysis] (14 points)

Professor Wagner has n topics he'd like to lecture on. As he is preparing his lectures, from time to time he notices a dependency: topic i will need to come before topic j. Let's build a data structure to record these dependencies. The data structure has two operations:

- BEFORE(i, j) records that topic i must come before topic j.
- QUERY(i, j) returns true if the previous calls to BEFORE imply that i must come before topic j, or false otherwise.

Note that "comes before" is a transitive relation: if we know that i must come before j, and we know that j must come before k, then it follows that i will have to come before k.

Example. Suppose we execute the following sequence of code:

BEFORE$(1,2)$.
$x :=$ QUERY$(1,3)$.
BEFORE$(2,3)$.
$y :=$ QUERY$(1,3)$.
$z :=$ QUERY$(1,4)$.

Then x and z should be false, and y should be true.

Alice's Algorithm. Alice proposes the following approach. She will maintain a directed graph $G = (V, E)$ with vertices $V = \{1, 2, \ldots, n\}$ corresponding to the n topics, represented in memory using adjacency list representation. At any point in time, the graph G will have an edge (i, j) if and only if Professor Wagner has previously called BEFORE(i, j). Alice suggests to implement QUERY using depth-first search in this graph. In pseudocode:

BEFORE(i, j):
1. Add the edge (i, j) to the graph, if it is not already present.

QUERY(i, j):
1. Perform a depth-first search, starting from vertex i (without restarts).
2. If vertex j was reached, return true; otherwise, return false.

Answer the following questions.

(a) Suppose Professor Wagner performs m calls to BEFORE, followed by a single call to QUERY, using Alice's algorithm. What is the worst-case running time of the last call to QUERY? Express your answer as a function of n and m, using $O(\cdot)$ notation. You may assume $0 \leq m \leq n^2$.

(b) Suppose Professor Wagner performs a sequence of $n^{1.5}$ calls to BEFORE and $n^{1.8}$ calls to QUERY, interleaved in some order. As in part (a), he uses Alice's algorithm. What is the total running time of all of these calls, in the worst case? Express your answer as a function of n, using $O(\cdot)$ notation.

Joe's algorithm. Joe has a different idea. He likes the idea of maintaining a directed graph $G = (V, E)$, but he prefers to store his graph in an adjacency matrix representation. Also, Joe's graph is a bit different: at any point in time, his graph will have an edge (i, j) if and only if QUERY(i, j) would return true at that point in time. Joe scribbles quickly on the back of an envelope, and you look over his shoulder and see the following pseudocode:

BEFORE(i, j):
1. If the edge (i, j) is already present in the graph, return.
2. Add the edge (i, j) to the graph.
3. For each k such that (j, k) is an edge in the graph, call BEFORE(i, k).
4. For each h such that (h, i) is an edge in the graph, call BEFORE(h, i).

QUERY(i, j):
1. If (i, j) is an edge in the graph, return true; otherwise, return false.

(c) Joe's pseudocode has a small bug in it: sometimes it gives a wrong answer. Circle the error in the pseudocode above. Then, show a simple way to fix the bug using a very small change to the pseudocode.

(d) Suppose Professor Wagner performs m calls to BEFORE, followed by a single call to QUERY, using Joe's algorithm (with your fix). What is the worst-case running time of the last call to QUERY? Express your answer as a function of n and m, using $O(\cdot)$ notation. You may assume $0 \leq m \leq n$.

(e) Suppose Professor Wagner performs m calls to BEFORE, followed by one more call to BEFORE, using Joe's algorithm (and the fix). How many edges could Professor Wagner's last call to BEFORE have added to the graph, at most (in the worst case)? In other words, if k_{before} denotes the number of edges in the graph just before Professor Wagner's last call to BEFORE, and k_{after} denotes the number of edges in the graph after this call to BEFORE returns, how large could $k_{\text{after}} - k_{\text{before}}$ be? Express your answer as a function of n and m, using $O(\cdot)$ notation. You may assume $0 \leq m \leq n$.

(Continued on next page)

(f) Suppose Professor Wagner performs m calls to BEFORE, followed by one more call to BEFORE, again using Joe's algorithm (and the fix). What is the worst-case running time of Professor Wagner's last call to BEFORE? Express your answer as a function of n and m, using $O(\cdot)$ notation. You may assume $0 \leq m \leq n$.

(g) Suppose Professor Wagner performs a sequence of $n^{1.5}$ calls to BEFORE and $n^{1.8}$ calls to QUERY, interleaved in some order. He continues to use Joe's algorithm (and your fix). What is the total running time of all of these calls, in the worst case? Express your answer as a function of n, using $O(\cdot)$ notation.

Ungraded. [Optional feedback] (0 points)

This question is ungraded, purely optional, and not worth any points. Solve all other problems before looking at this one. It will not affect your grade in any way.

Do you have any advice for future CS170 students? Is there any advice you wish you had gotten at the start of the semester?