CMSC 451: Design and Analysis of Computer Algorithms

Fall 2003

http://www.cs.umd.edu/~mount/451/

Instructor: Dave Mount. Office: AVW 3373. Email: mount@cs.umd.edu. Office phone: (301) 405–2704. Office hours: Mon 2:30-3:30, Wed 3:30-4:30. I am also available immediately after class for questions. If the question is short (a minute or so) drop by my office any time. Please send me email if you cannot make these times. (Don't be shy about doing this. I always set aside at least one hour each week for "unscheduled" office hours.)

Class Time: Tue, Thur 3:30-4:45, CSI 3117.

Teaching Assistant: Pooja Nath. Office: AVW 1112. Email: pooja@cs.umd.edu. Office hours: Tue,Wed 9:00-10:00. If you cannot make these times, please feel free to contact Pooja to set up another time.

Course Overview: This course presents the fundamental techniques for designing efficient computer algorithms, proving their correctness, and analyzing their running times. After a brief review of material from 351 (asymptotics, recurrences, sorting), we will discuss efficient algorithms for basic graph problems (minimum spanning trees, shortest paths, connectivity problem, network flows), solving optimization problems through greedy algorithms and dynamic programming, computational geometry, proofs of intractability and NP-completeness, and approximation algorithms.

Text: (Required) T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms* (2nd Edition), McGraw Hill, 2002. (This is an excellent reference book on algorithm design. I strongly recommend purchasing it, and keeping it for future reference.)

Prerequisites: CMSC 351. Each student is expected to have basic programming skills (programming with loops, pointers, structures, recursion), discrete mathematics (proof by induction, sets, permutations, and combinations, probability), understanding of basic data structures (lists, stacks, queues, trees, graphs, and heaps), knowledge of sorting algorithms (MergeSort, QuickSort, HeapSort) and basic graph algorithms (DFS and BFS), basic calculus (manipulation of logarithms, differentiation, integration). If there is any material that seems unfamiliar, please see me or the teaching assistant as soon as possible to head off any problems.

Course Work: Course work will consist of (around 6) homework assignments (about one every week and a half) and 2 exams (a midterm and a comprehensive final). Tentative weights: Homeworks 25%, midterm 30%, final exam 45%. (Note that these weights are subject to change.) The midterm is tentatively scheduled for Tue, Oct 28 and the final exam will be Sat, Dec 20, 10:30am-12:30pm.

Homeworks are to be turned by the start of class on the due date. No late homeworks will be accepted, but the lowest homework grade will be dropped. (In other words, turn in what you have by the start of class. If your schedule is too busy, just drop the homework. But, try your best to save the dropped homework for the latter half of the semester, when you will really need it.) In exceptional circumstances (illness, university business, religious observances) extensions may be granted. However, all extensions must be approved by me BEFORE the due date.

As a courtesy to the grader, homeworks should be written neatly. Poorly written work will not be graded. When writing algorithms be sure not only that your solution is correct, but also that it is easy for the grader to understand why your solution is correct. Part of your grade will be based not only on correctness, but also on the clarity, simplicity, and elegance of your solution.

Some homeworks and projects will have a special challenge problem. Points from the challenge problems are *extra credit*. This means that I do not consider these points until *after* the final course cutoffs have been set. Each semester extra credit points usually account for at least few students getting one higher letter grade (e.g. from a B+ to A-).

Group Homeworks: Some people learn better in a group setting and others learn better individually. This semester we will experiment with a group homeworks. For each homework assignment, you are allowed to work either individually or in groups of two or three people. You pick your own group members, and may use different groups for different homeworks. Each group shall hand in one homework with the names of all the people of the group, and all members of the group will share the same grade. Grading standards will be the same, irrespective of whether the assignment is done individually or in a group, but I will expect group homeworks to be neatly written and well organized (because there is one document to be prepared for the entire group).

It may seem at first that groups will have an obvious advantage over people working individually. However, this is not true for a couple of reasons. First, when working in a group it is important the final document reflect a common vision of the group, resolving any differences among the group members. Second, the exams (which are worth much more than the homeworks) require on a deep understanding of the methods used in solving homework problems. Group members that only have a superficial understanding of the material will pay a heavy price on the exams. In-depth knowledge only comes by struggling (often for hours) with different approaches and solution strategies.

Academic Dishonesty: You may *discuss* homework problems and general solution strategies with classmates (whether inside or outside your group), but when it comes to formulating and writing solutions you must work alone or only with your fellow group members. If you make use of other sources in coming up with your answers you must cite these sources clearly. (This includes papers or books in the literature, friends or classmates, and information downloaded from the web.) Instances of academic dishonesty will be dealt with harshly, and usually result in a hearing in front of a student honor council, and a grade of XF.

Syllabus: The topics and order listed below are tentative and subject to change.

Review of algorithm analysis and sorting: (Chapts 1–9) Review of asymptotics, summations, recurrences, sorting algorithms, selection, and lower bounds for sorting. (1 week)

Dynamic Programming and Greedy Algorithms: (Chapts 15–16) Longest common subsequence, chain multiplication, minimum weight triangulation, Huffman trees. (2 weeks)

Basic Graph algorithms: (Chapts 22–23) Review of basic graph traversals (DFS and BFS), directed acyclic graphs, topological sort and strong components, minimum spanning trees. (2 weeks)

Shortest Paths: (Chapt 24) Dijkstra's algorithm and Bellman-Ford algorithm. (1 week)

Network Flow: (Chapt 26) Ford-Fulkerson and Edmonds-Karp algorithms, and applications to maximum matching. (1 week)

Computational Geometry: (Chapt 33) Closest pair, convex hulls. (1 week)

NP-completeness: (Chapts 34) Basic terminology, polynomial reductions, examples of NP-complete problems (SAT, independent set, vertex cover, clique, Hamiltonian path, TSP), approximation algorithms. (2.5 weeks)

Approximation Algorithms: (Chapt 35) Vertex cover, TSP k-center approximations. PTASs and the knapsack approximation. (1.5 weeks)

CMSC 451: Some Useful Background Information

Asymptotic Forms: The following gives both the formal "c and n_0" definitions and an equivalent limit definition for the standard asymptotic forms. Assume that f and g are positive functions.

Asymptotic Form	Relationship	Limit Form	Formal Definition
$f(n) \in \Theta(g(n))$	$f(n) \equiv g(n)$	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	$\exists c_1, c_2, n_0, \forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n).$
$f(n) \in O(g(n))$	$f(n) \preceq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	$\exists c, n_0, \forall n \geq n_0,\ 0 \leq f(n) \leq cg(n).$
$f(n) \in \Omega(g(n))$	$f(n) \succeq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$	$\exists c, n_0, \forall n \geq n_0,\ 0 \leq cg(n) \leq f(n).$
$f(n) \in o(g(n))$	$f(n) \prec g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$	$\forall c, \exists n_0, \forall n \geq n_0,\ 0 \leq f(n) \leq cg(n).$
$f(n) \in \omega(g(n))$	$f(n) \succ g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	$\forall c, \exists n_0, \forall n \geq n_0,\ 0 \leq cg(n) \leq f(n).$

L'Hôpital's rule: If $f(n)$ and $g(n)$ both approach 0 or both approach ∞ in the limit, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)},$$

where $f'(n)$ and $g'(n)$ denote the derivatives of f and g relative to n.

Polylog-Polynomial-Exponential: For any a, b, and c, where $b > 0$ and $c > 1$.

$$\log^a n \prec n^b \prec c^n.$$

Common Summations: Let $x \neq 1$ and $n \geq 0$.

Name of Series	Formula	Closed-Form Solution
Constant Series	$\sum_{i=a}^{b} 1$	$= \max(b - a + 1, 0)$
Arithmetic Series	$\sum_{i=0}^{n} i = 0 + 1 + 2 + \cdots + n$	$= \frac{n(n+1)}{2}$
Geometric Series	$\sum_{i=0}^{n} x^i = 1 + x + x^2 + \cdots + x^n$	$= \frac{x^{n+1} - 1}{x - 1}$
Quadratic Series	$\sum_{i=0}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2$	$= \frac{2n^3 + 3n^2 + n}{6}$
Linear-geometric Series	$\sum_{i=0}^{n-1} ix^i = x + 2x^2 + 3x^3 \cdots + nx^n$	$= \frac{(n-1)x^{(n+1)} - nx^n + x}{(x-1)^2}$
Harmonic Series	$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$	$\approx \ln n$

Summations with general bounds: For $1 \leq a \leq b$

$$\sum_{i=a}^{b} f(i) \;=\; \sum_{i=0}^{b} f(i) - \sum_{i=0}^{a-1} f(i).$$

Approximate using integrals: Let $f(x)$ be any *monotonically increasing function* (the function increases as x increases).

$$\int_0^n f(x)dx \;\leq\; \sum_{i=1}^{n} f(i) \;\leq\; \int_1^{n+1} f(x)dx.$$

(Simplified) Master Theorem for Recurrences: Let $a \geq 1$, $b > 1$ be constants and let $T(n)$ be the recurrence

$$T(n) = aT(n/b) + cn^k,$$

defined for $n \geq 0$.

Case 1: $a > b^k$ then $T(n)$ is $\Theta(n^{\log_b a})$.

Case 2: $a = b^k$ then $T(n)$ is $\Theta(n^k \log n)$.

Case 3: $a < b^k$ then $T(n)$ is $\Theta(n^k)$.

Comparison-Based Sorting Algorithms: A *stable* sorting algorithm preserves the relative order of equal elements. An *in-place* sorting algorithm uses no additional array storage (although $O(\log n)$ additional space is allowed for the recursion stack).

Algorithm	Time	Stable	In-place
BubbleSort	$\Theta(n^2)$	Yes	Yes
InsertionSort	$\Theta(n^2)$	Yes	Yes
MergeSort	$\Theta(n \log n)$	Yes	No
HeapSort	$\Theta(n \log n)$	No	Yes
QuickSort*	$\Theta(n \log n)$	Yes/No	No/Yes

*There are two versions of QuickSort, one which is stable but not in-place, and one which is in-place but not stable.

Lower Bound for Comparison-Based Sorting: Any comparison-based sorting algorithm has worst-case running time at least $\Omega(n \log n)$.

Non-Comparison-Based Sorting Algorithms: All of these algorithms are stable, but not in-place.

Algorithm	Assumptions	Time	Space
CountingSort	Sorts n integers over a range of size k	$\Theta(n+k)$	$\Theta(n+k)$
RadixSort	Sorts n d-digit integers where each digit is over a range of size k	$\Theta(d(n+k))$	$\Theta(n+k)$
BucketSort	Sorts n uniformly distributed numbers	$\Theta(n)$ (expected case)	$\Theta(n)$

Note that RadixSort can be modified to sort a set of n integers over a range of size $O(n^d)$ in $\Theta(dn)$ time and $\Theta(n)$ space.

Selection: For any k, $1 \leq k \leq n$, the kth smallest element of a list of size n can be computed in $O(n)$ time.

Homework 1: Prerequisites

Handed out Tuesday, Sep 2. Due at the start of class Thursday, Sep 11. Late homeworks are not accepted, but you may drop your lowest homework score.

The prerequisites for this class are mentioned in the syllabus, and correspond roughly to Chapters 1–9 of our course textbook, by Cormen, Leiserson, Rivest, and Stein (CLRS). We will review some of this information in the first couple of lectures. This homework is designed to refresh your memory about some of these elements.

Notation: Throughout the semester, we will use $\lg x$ to denote logarithm of x base 2 ($\log_2 x$) and $\ln x$ to denote the natural logarithm of x. We will use $\log x$ when the base does not matter.

Problem 1. Around 100 A.D. Nicomachus of Greece proved the following remarkable identity. For all $n \geq 0$,

$$\sum_{i=1}^{n} i^3 \;=\; \left(\sum_{i=1}^{n} i\right)^2$$

That is $(1^3 + 2^3 + \cdots + n^3) = (1 + 2 + \cdots + n)^2$.

(a) Prove this by induction on n.

(b) Explain why the following figure provides an informal "proof" of this identity.

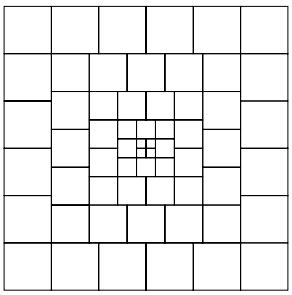


Figure 1: Problem 1

Problem 2. For each part, list the functions in increasing asymptotic order. Indicate whether the two functions are asymptotically equivalent ($f(n) \equiv g(n)$) or one is strictly less than the other ($f(n) \prec g(n)$). For example, given the list

$$n^2, \quad n \log n, \quad 3n + n \log n,$$

the answer would be

$$n \log n \;\equiv\; 3n + n \log n \;\prec\; n^2.$$

Explanations are *not* required, but may be given to help in assigning partial credit. (Hint: Review Chapt. 3 in CLRS, especially the section on logarithms.)

(a)	$5 + 104n^2$	$5n^3 + \log n$	2^n
(b)	$(1.5)^n$	$3^{(n/2)}$	$2^{(n/3)}$
(c)	$\lg n$	$\ln n$	$\log_{1.2} n$
(d)	$\lg(\lg n)$	$\sqrt{\lg n}$	$\lg \sqrt{n}$
(e)	$n^{\lg 4}$	$2^{\lg n}$	$2^{(2 \lg n)}$
(f)	$\max(n^2, 200n)$	$n^2 + 200n$	$\min(n^2, 200n)$
(g)	$\lceil n^2/20 \rceil$	$\lfloor n^2/20 \rfloor$	$n^2/20$

Problem 3. Consider the following recurrences defined for $n \geq 1$. In each case, apply the Master Theorem to derive an asymptotic formula for the recurrence.

$$\begin{array}{lrcl} (a) & T(n) & = & 3T(n/2) + 2n \\ (b) & T(n) & = & 5T(n/2) + n^4 \\ (c) & T(n) & = & 8T(n/2) + 3n^2 + n^3 \\ (d) & T(n) & = & 2T(n/4) + n\sqrt{n} \\ (e) & T(n) & = & 2T(n/3) + 5^{\log_2 n} \end{array}$$

Try to present your answer in the simplest form you can. Show how you derived your answer in each case.

Problem 4. For this problem you are given an array $A[1..n]$ of numbers and an integer k ($1 \leq k \leq n$). You may not assume that the elements of A are distinct, nor that they are sorted, nor that they are small integers. (The fact that they are not small integers means that you cannot sort the elements of A in $O(n)$ time.)

(a) Give an algorithm which, given an array $A[1..n]$ of numbers and an integer k ($1 \leq k \leq n$), outputs the k smallest elements of A (in any order). For example, if the elements of A are:

$$A[1..8] = \langle 6.4,\ 0.3,\ 5.0,\ 1.5,\ 1.5,\ 3.2,\ 2.5,\ 1.5 \rangle,$$

and $k = 3$, the algorithm should output the set $\{0.3, 1.5, 1.5\}$ (in any order).

(b) Give an algorithm which, given a list $A[1..n]$ of numbers, a positive integer k ($1 \leq k \leq n$) and number X, outputs the k numbers of A that are closest to X in value. (These numbers may be greater than, less than, or equal to X.) For example, given the array elements:

$$A[1..8] = \langle 9.5,\ 1.3,\ 5.2,\ 0.1,\ 7.5,\ 6.6,\ 2.5,\ 9.1 \rangle,$$

$k = 4$ and $X = 8$, the algorithm should output the set $\{9.5, 7.5, 6.6, 9.1\}$ (in any order).

Briefly explain your algorithms (include an example), and derive their running times. You may assume that you have access to any of the algorithms of Chapters 1–9 of CLRS. Your algorithms should run in $O(n)$ time, irrespective of the value of k.

Challenge Problem. An *in-place algorithm* is one that is given an array of length n as input, but it is allowed to use only $O(1)$ additional working storage. In other words, it can use any number of regular variables, but it is not allowed to declare arrays, to dynamically allocate new storage, and it is not allowed to make recursive calls (since doing so would allow it to use the recursion stack for working storage). The in-place restriction is often important in applications where the inputs are very large arrays, that barely fit into memory.

Suppose that you are given an array $X[1..n]$ of numbers, and you are given an index i, $1 \leq i \leq n-1$. This implicitly splits X into two subarrays, $X[1..i]$ and $X[i+1..n]$. These subarrays are not necessarily of the same size. Give an $O(n)$ time *in-place* algorithm which, given the array X and i, swaps these two subarrays within X, without changing the order of elements within each subarray. Your algorithm should run in $O(n)$ time.

A Note about Writing Algorithms: When writing pseudo-code do not give a complete program. Give just enough detail that your intentions are clear and unambiguous, but do not provide extraneous details (complex syntax and/or variable declarations) which are otherwise obvious. English explanations, examples, and figures should be given to illustrate your intentions (and can help in assigning partial credit). Even if you are not asked, you should always provide a justification of correctness and analysis of running time.

Homework 2: Dynamic Programming and Greedy Algorithms

Handed out September 16 (revised Sept 23). Due at the start of class Tuesday, September 30. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1.

(a) Demonstrate the execution of the dynamic programming algorithm for LCS (as given in class or in CLRS) on the following example:

$$X = \langle ABCDABE \rangle \qquad Y = \langle CABE \rangle$$

Show just the contents of the $c[i, j]$ length table and not the back pointers. What is the final LCS and its length?

(b) True or False: Suppose that we know the first 3 characters of X are distinct and match the first 3 characters of Y, e.g., $X = \langle ABC \ldots \rangle$ and $Y = \langle ABC \ldots \rangle$. Then we may assume that the LCS of X and Y begins with these three common characters, that is, $LCS(X, Y) = \langle ABC \ldots \rangle$. Explain your answer.

(c) True or False: Suppose that we know the first 3 characters of X are distinct and match characters 2–4 of Y, e.g., $X = \langle ABC \ldots \rangle$ and $Y = \langle QABC \ldots \rangle$. Then we may assume that the LCS of X and Y begins with these 3 common characters. That is, $LCS(X, Y) = \langle ABC \ldots \rangle$. Explain your answer.

(d) Repeat part (c), but this time, suppose that the first 999 characters of X are distinct and match characters 2-1000 of Y.

Problem 2. The objective of this problem is to write a dynamic programming algorithm to play a game. Two players, called Jen and Ben alternate in taking moves, with Jen always going first. Initially the board consists of three piles of diamonds, which we denote (A, B, C), meaning that there are A diamonds in the first pile, B in the second pile, and C in the third. The board always consists of three numbers that are nonnegative. During a move a player can do any one of the following:

(1) Remove 1 diamond from pile 1.
(2) Remove either 1 or 2 diamonds from pile 2.
(3) Remove either 2 or 3 diamonds from pile 3.

The first player who cannot make a move loses. (And the winner gets all the diamonds.) That is, if it is a player's turn to move and the board is either $(0, 0, 0)$ or $(0, 0, 1)$ then he/she loses.

Write a program that will, given numbers (A, B, C), determine whether Jen or Ben has a winning strategy, given this starting board and the assumption that Jen plays first. That is, your program should determine which player wins, assuming both players play as well as is possible.

For example, suppose that the initial board is $(0, 1, 4)$. In this case Jen can force a win. She uses rule (3) to remove 2 diamonds from pile 3, resulting in the board $(0, 1, 2)$. Ben's only choices are to remove 1 from pile 2 or 2 from pile 3, resulting in the possible boards $(0, 0, 2)$ and $(0, 1, 0)$. In either case, Jen can remove the final diamonds (using either rules (3) or (2)) leaving Ben without a move.

Derive a dynamic programming formulation to determine the winner, given the initial board (A, B, C). Justify the correctness of your formulation. (You do not need to give an entire algorithm, just the DP formulation.) Argue that the running time (if implemented) would be a polynomial function of A, B, and C.

Hint: Consider a boolean table $W[a, b, c]$, where $0 \leq a \leq A$, $0 \leq b \leq B$, and $0 \leq c \leq C$. The entry $W[a, b, c]$ is true if a player making the next move with this board can force a win and false otherwise. Consider the results of all possible moves starting from this board, assuming that the results of all smaller boards are already computed.

Problem 3. This is an actual problem that arises in video processing, called *temporally consistent assignment*. You are given two videos, each represented as a sequence of images. Both videos were taken of the same scene at roughly the same time period. Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ be one sequence of images and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be the other. The cameras are not synchronized and may run at significantly different speeds. The objective is to assign each image of the X video sequence with its most similar image in the Y video sequence, subject to the constraint that the assignments are consistent in time.

In order to determine how similar two images are, as part of the input, you may assume that you are given a table $D[i, j]$ which contains a numeric dissimilarity value between these two images x_i and y_j. The lower $D[i, j]$ is, the more similar the images are. Do not worry how $D[i, j]$ is computed.

An *assignment* of sequence X to sequence Y is a sequence of m indices $A = \langle j_1, j_2, \ldots, j_m \rangle$ meaning that, for $1 \leq i \leq m$, image x_i is assigned to image y_{j_i}. The *cost* of an assignment is a sum of the dissimilarities between the assigned images. That is,

$$\text{cost}(A) = \sum_{i=1}^{m} D[i, j_i].$$

An assignment A is *temporally consistent* if $j_i \leq j_{i+1}$, for $1 \leq i < m$. In other words, if x_i is assigned to some image y_{j_i} then the next image in the sequence x_{i+1} must be assigned to an image appearing no earlier in the Y image sequence. (This makes sense, the time runs forward for both cameras.) We allow two images of X to be assigned to the same image of Y. The problem is: given the video sequences X, Y and the cost table $D[i, j]$, compute the minimum cost temporally consistent assignment of X to Y.

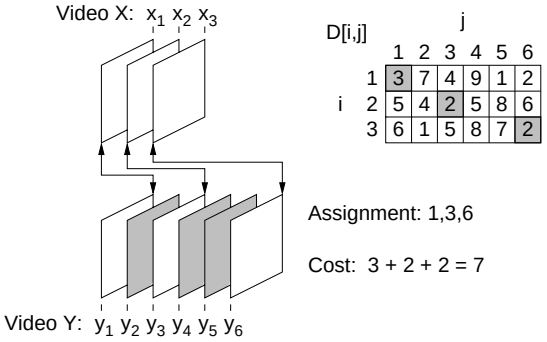


Figure 1: Example for Problem 3. The dissimilarity matrix is shown for the two sequences. The assignment $\langle 1, 3, 6 \rangle$ has a cost of $D[1, 1] + D[2, 3] + D[3, 6] = 3 + 2 + 2 = 7$.

(a) Present a dynamic programming formulation for the problem of computing the cost of the optimal temporally consistent assignment.

Hint: Let $X_i = \langle x_1, x_2, \ldots, x_i \rangle$ and $Y_j = \langle y_1, y_2, \ldots, y_j \rangle$ denote the subsequences conisting of the first i images of X and first j images of Y, respectively. (If i or j is zero, then the resulting subsequence X_i or Y_j is the empty sequence.) For $0 \leq i \leq m$ and $0 \leq j \leq n$, let $C[i, j]$ be the cost of the optimum assignment for the subsequences X_i and Y_j. Give a formula for computing $C[i, j]$. Be sure to include the basis cases.

(b) Express your answer to (a) as an algorithm (in pseudocode).

(c) What is the running time of your algorithm, as a function of m and n.

Problem 4. Let $F_1, F_2, \ldots, F_n$ be a collection of n files to be stored on a tape. File F_i requires s_i kilobytes of storage. The tape is long enough to store all the files. We know that, on average, a fraction p_i of requests to the tape are for file F_i, where $0 \leq p_i \leq 1$, and $\sum_{i=1}^{n} p_i = 1$. After a file is loaded from the tape, the tape is rewound. In order to access a file on the tape, it is necessary to read through all the files that precede it.

The problem is, given s_i's and p_i's determine the best way in which to order the files on the tape, so that the expected retrieval time is minimized.

(a) Let $i_1, i_2, \ldots, i_n$ denote the final order in which files are stored on the tape. Explain (clearly) why the expected time of accessing a file is proportional to:

$$T = \sum_{j=1}^{n} \left(p_{i_j} \sum_{k=1}^{j} s_{i_k} \right).$$

(b) Present a greedy algorithm that determines the optimal ordering of the files on the tape. Prove your algorithm's correctness and derive its running time. (Hint: Find the best way to order the files based on s_i and p_i. Show that any different way of ordering the files cannot be optimal, by showing how to decrease the expected access time by swapping an appropriate pair of adjacent files on the tape.)

Challenge Problem. You have just been hired as the quality-control engineer for the ACME Wonderful Widget corporation. The widgets are supposed to have identical weight. You are given a set of n widgets, and are told that *at most one* (possibly none) of the n widgets is either too heavy or too light (but you do not know which). Your task is to develop an efficient test procedure to determine which of the n widgets is defective, or report that none is defective. To do this test you have a scale. For each measurement you place some of the widgets on the left side of the scale and some of the widgets on the right side. The scale indicates either (1) the left side is heavier, (2) the right side is heavier, or (3) both subsets have the same weight. It does not indicate how much heavier or lighter.

(a) Present an method to determine the defective widget using at most $(\lceil \log_3(1+2n) \rceil + c)$ scale measurements, where c is a constant (independent of n). Try to make c as small as possible. Explain your algorithm's correctness. (If you cannot succeed in this, then try to get at least $c\lceil \log_3(1+2n) \rceil$, for a small constant c, for partial credit.)

(b) Prove that in the worst-case the minimum number of measurements using the scale is at least $\lceil \log_3(1+2n) \rceil$. (Hint: Use a decision tree argument.)

To simplify your arguments, you may assume, for example, that $1+2n$ is a power of 3 or that n is a power of 3.

Homework 3: Graph Algorithms

Handed out October 9. Due at the start of class Tuesday, October 21. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. Show the result of executing both BFS and DFS on the graph shown in Fig. 1 (left). Given a choice of which vertex to visit next, always select the next eligible vertex in alphabetical order. (Start with "a".) For BFS: show the final BFS tree and the $d[u]$ and $pred[u]$ values. For DFS: show the final DFS tree, give the $d[u]$ and $f[u]$ values for each node. In both cases indicate which edges are tree edges (solid) and which edges are back or cross edges (dashed).

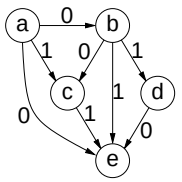
Figure 1: Problem 1 (left) and Problem 2 (right).

Problem 2. Recall that a free tree is a connected, acyclic undirected graph. Suppose that you are given a free tree $G = (V, E)$ represented by its adjacency list, and a starting vertex u_0. Give an $O(V + E)$ time algorithm that outputs the vertices as they would be visited by a *twice-around tour* starting at vertex u_0. This is a traversal of the tree that walks down and then up each branch of the tree. See, for example, in Figure 1 (right) the twice-around tour starting at a, would produce $\langle acdchfhcagbgega \rangle$. Briefly explain your algorithm and derive its running time.

Problem 3. You are given a DAG (directed acyclic graph) $G = (V, E)$ in which each edge has a label $L(u, v)$ which is either 0 or 1. An *alternating path* is defined to be a path containing at least one edge, such that the edges along the path alternate in label between 0 and 1. The path may begin with a 0-label edge or a 1-label edge. Note: The empty path is *not* an alternating path. A path with a single edge is always an alternating path. Paths need not be of maximal length.

Give an $O(V + E)$ algorithm which, given a DAG $G = (V, E)$ with 0-1 edge labels, computes for each vertex $u \in V$ the number $A[u]$ of alternating paths originating from u. Briefly justify your algorithm's correctness and running time.

Hint: Maintain two counts. The number of alternating paths starting with 0 and the number starting with 1. Show how to compute these in a bottom-up manner using DFS.



Vertex	A
a	6
b	5
c	1
d	1
e	0

Figure 2: Example for problem 3.

Problem 4. On the planet of Smurf, the tiny Smurf folk hold a convention every 4 years. It is ancient Smurf tradition that for every father-son pair, at least one of the two must attend the convention, and both may attend. This means that, if a Smurf decides to stay home, his father and all his children are required to attend. Assume that there are n Smurfs altogether, named Smurf 1 through Smurf n. Let $T[i]$ denote the travel cost of sending Smurf i to the convention. You may assume the nodes have been numbered, so that if Smurf i is the father of Smurf j, then $i < j$. (So, "papa Smurf" is Smurf 1.)

(a) Give an algorithm to determine the minimum total travel costs for all the Smurfs, subject to the above attendance requirement. The inputs to the algorithm are two arrays C and T, where $C[i]$ is the list of the children of Smurf i, and $T[i]$ is his travel cost.

(b) Modify your algorithm from (a) to output which Smurfs should attend the convention, in order to achieve this minimum travel cost.

Derive the running time of your algorithms. $O(n)$ time is possible. For example, in Figure 3, we give a sample input. The distances are listed next to each node of the Smurf family tree. The optimal solution is to send Smurfs 1 and 2 to the meeting, for a total transportation cost of $5.2 + 7 = 12.2$.

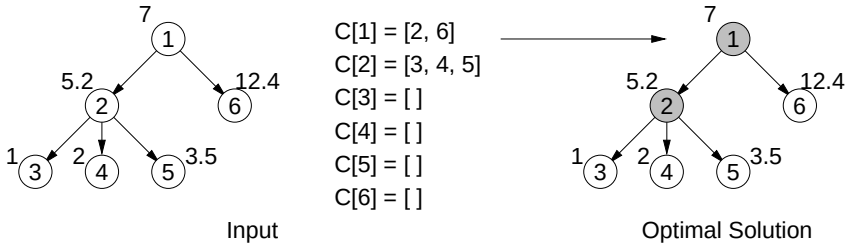
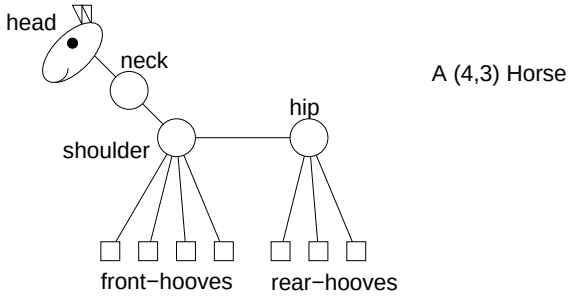


Figure 3: Example for Problem 4. Next to each node of the tree is its travel cost.

Hint: This can be solved by a DP approach. Think of C as defining a tree. Let S_i denote the subtree rooted at Smurf i. Let $G[i]$ be the minimal transportation cost for all the Smurf's in subtree S_i if Smurf i goes to the meeting, and let $H[i]$ be the minimal transportation cost for this subtree if Smurf i stays home.

Challenge Problem. Given integers i and j, both greater than or equal to 2, we define an (i, j)-*horse* to be an undirected graph which consists of $n = i + j + 4$ vertices. The vertices consist of i front-hooves, j rear-hooves, and four extra vertices, the head, the neck, the shoulder, and the hip. These are connected as shown in the figure below (head-neck-shoulder-hip, and front-hooves to shoulder and rear-hooves to hip). You are given the $n \times n$ adjacency matrix for an n-vertex horse, but you are not told what the values of i and j are, and you may assume nothing about the vertex ordering.

Give an $O(n)$ time algorithm which, given the adjacency matrix of a horse, determines the values of i and j, and labels all the vertices according to their type (head, neck, shoulder, hip, front-hoof, or rear-hoof). Note that because the adjacency matrix has n^2 entries, you cannot scan the entire matrix in $O(n)$ time. Explain how your algorithm works. (Try to keep it as simple as possible.)



Practice Problems for the Midterm

The midterm will be on Tues, Nov 4. The exam will be closed-book and closed-notes, but you will be allowed one cheat-sheet (front and back).

Disclaimer: These are practice problems, which have been taken from old homeworks and exams. They do not necessarily reflect the actual length, difficulty, or coverage for the exam.

Problem 1. Short answer questions.

(a) What is the worst-case running time of BucketSort? How might this worst case arise?
(b) Consider the code $a = 0$, $b = 01$, $c = 11$, $d = 101$. Is this a prefix code? Explain.
(c) Consider a breadth-first search (BFS) of a directed graph $G = (V, E)$. Let (u, v) be any edge in G. Is it possible that $d[v] \geq d[u] + 2$? Explain briefly.
(d) Suppose that you are given a connected, undirected graph, $G = (V, E)$ with integer edge weights in which every vertex has degree 2. How fast can you compute an MST for this graph? Explain briefly.
(e) Use the Master Theorem, give an asymptotic solution to the recurrence: $T(1) = 1$ and $T(n) = 5T(n/3) + n^4$, for all $n > 1$, where n is a power of 3.
(f) As a function of n, give an exact closed-form solution (no embedded sums) to the following summation:

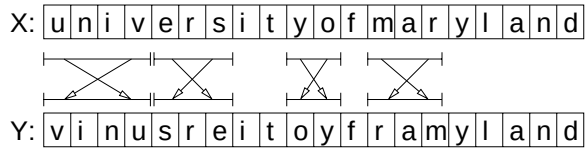
$$T(n) = \sum_{i=n}^{2n} i.$$

(g) In the DFS of a digraph there is an edge (u, v) such that $d[u] < d[v]$. What are the possible types of this edge? (tree, back, forward, cross.)
(h) Given a connected undirected graph $G = (V, E)$ in which all edges have weight 1, what is the fastest and simplest way (that we know of) for computing a minimum spanning tree for G?
(i) What is the maximum number of edges in an undirected graph with n vertices, in which each vertex has degree at most k?

Problem 2. Given two strings, $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$, the *shortest common supersequence* (SCS) is a minimum length string Z such that both X and Y are subsequences of Z. For example, if $X = \langle ABCBABA \rangle$ and $Y = \langle BCAABAB \rangle$, then $Z = \langle ABCAABABA \rangle$ is an SCS of both X and Y.

Give an $O(mn)$ dynamic programming algorithm which given X and Y computes the length of the SCS of X and Y. You do not need to determine the actual SCS, just its length. Be sure to give the DP formulation, and explain its correctness.

Problem 3. Given a string X, define a *substring reversal* to be an edit operation that replaces the substring $x_i x_{i+1} \ldots x_{j-1} x_j$ with its reversal, $x_j x_{j-1} \ldots x_{i+1} x_i$. Two substring reversals are said to be *independent* if the reversed substrings do not overlap each other. The figure below gives an example involving 4 independent reversals. Note that this operation does not alter the lengths of the strings.



Give an efficient algorithm to solve the following problem. Given two strings X and Y, each of length n, determine the minimum number of independent reversals needed to convert X into Y (or ∞ if it is impossible). (Hint: Use dynamic programming. Let $R[0..n]$ be an array where $R[i]$ is the minimum number of independent reversals needed to convert X_i into Y_i. If this is impossible, then $R[i] = \infty$. You need only give the DP formulation.) Justify the correctness of your algorithm.

Problem 4. A pharmacist has W pills and n empty bottles. Let $\{p_1, p_2, \ldots, p_n\}$ denote the number of pills each bottle can hold.

(a) Describe a greedy algorithm, which given W and the p_i's determines the fewest number of bottles needed to store all the pills. (An informal description is sufficient.)

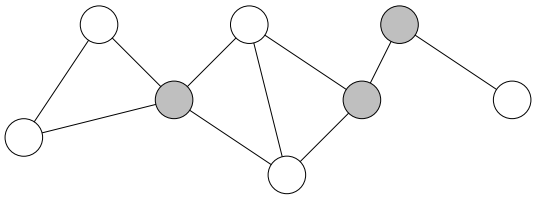
(b) Argue that the *first* bottle chosen by your greedy algorithm will be in some optimal solution.

(c) How would you modify your algorithm if each bottle also has an associated cost c_i, and you want to minimize the total cost of the bottles used to store all the pills? (No need to prove correctness.)

Problem 5. Design an algorithm, which given a sequence of n positive integers, $d_1, d_2, \ldots, d_n$, such that $\sum_i d_i = 2n - 2$, constructs an (undirected) free tree with n vertices, such that the degrees of the vertices are exactly $d_1, d_2, \ldots, d_n$. Your program may output the tree, for example, as a list of edges. (Hint: $O(n^2)$ time is not too hard, but $O(n)$ time is possible.)

Problem 6. Let $G = (V, E)$ be an undirected graph. Write an $O(V + E)$ time algorithm to determine whether it is possible to direct the edges of G such that the indegree of every vertex is at least one. If it is possible, then your algorithm should show a way to do this. (Hint: Use DFS.)

Problem 7. Let $G = (V, E)$ be a connected undirected graph. A *cut vertex* is defined to be any vertex whose removal (together with the removal of any incident edges) causes the graph to become disconnected. For example, in the figure below the cut vertices are shaded.



(a) Consider any vertex $u \in V$. Perform a DFS of G starting from u (thus u is the root of the DFS tree). How can we determine whether u is a cut vertex based on the the structure of the resulting DFS tree? Explain your answer.

(b) Can a cut vertex be a leaf of a DFS tree? If so, give an example. If not explain why not.

Problem 8.

(a) Consider a weighted undirected graph G. Suppose you replace the weight of every edge with its negation (e.g. $w(u, v)$ becomes $-w(u, v)$), and compute the minimum spanning tree of the resulting graph using Kruskal's algorithm. True or False: The resulting tree is a *maximum* cost spanning tree for the original graph.

(b) Consider a weighted digraph G and source vertex s. Suppose you replace the weight of every edge with its negation and compute the shortest paths using Dijkstra's algorithm. True or False: The resulting paths are the *longest* (i.e., highest cost) simple paths from s to every vertex in the original digraph.

Problem 9. You are given an undirected graph $G = (V, E)$ where each vertex is a gas station and each edge is a road with an associated weight $w(u, v)$ indicating the distance from station u to v. The evil Professor Mount wants to drive from vertex s to vertex t. Since his car is old and may break down, he does not like to drive along long stretches of road. He wants to find the path from s to t that minimizes the *maximum* weight of any edge on the path. Give an $O(E \log V)$ algorithm to do this. Briefly justify your algorithm's correctness and derive its running time.

Midterm Exam

This exam is closed-book and closed-notes. You may use a sheet of notes (front and back). Write all answers in the exam booklet. You may use any algorithms or results given in class. If you have a question, either raise your hand or come to the front of class. Total point value is 100 points. Good luck!

Problem 1. (10 points) Show the execution of Huffman's algorithm on the following set of 6 symbols and associated probabilities:

Symbol	Probability	Symbol	Probability
A	0.15	D	0.05
B	0.50	E	0.15
C	0.10	F	0.05

Show the final Huffman tree. Also, show the final prefix code for your tree. (Intermediate steps are not required, but help in assigning partial credit.)

Problem 2. (30 points; 3–6 points each.) Short answer questions. Explanations are not always required, but may be given for partial credit.

(a) Which of the following sorting algorithms are both *stable* and *in-place*? MergeSort, HeapSort, QuickSort, and BubbleSort. (List all that apply.)

(b) What is the *principal of optimality* as it applies to dynamic programming?

(c) You are given a graph with V vertices and E edges as an adjacency list. How fast can you sort the vertices in increasing order of their degrees? (Explain briefly.)

(d) True or False: If a digraph has k distinct cycles, then for any depth-first search (DFS) of this digraph, there are at least k back edges.

(e) Consider an edge (u, v) in a digraph, and in some DFS we find that $d[u] < d[v]$. Then this edge could be a: tree edge, back edge, cross edge, forward edge. (List all that apply.)

(f) True or False: If a digraph has a negative cost cycle, Dijkstra's algorithm may go into an infinite loop.

Problem 3. (10 points) Consider a variant of the diamonds game. The initial board consists of two piles of diamonds, containing A and B diamonds, respectively. During a move a player can do either of the following:

(1) Remove 1 diamond from pile 1.

(2) Remove 1 diamond from pile 1 and 1 diamond from pile 2, but only if the number of diamonds currently in pile 1 is even.

The first player who cannot make a move loses. Give a dynamic programming formulation for for determining which player wins, assuming that the initial board has A and B diamonds. Briefly explain your formulation. A complete algorithm is not required. (Hint: Let $W[a, b]$ be true if the next player to go has a winning strategy.)

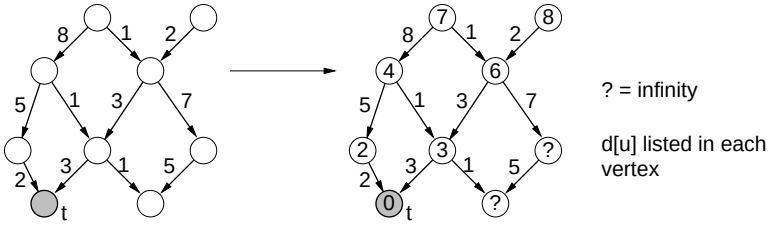
Problem 4. (20 points) Recall that in the activity scheduling problem, we are given a set of n activities that are to be scheduled to use some resource, where each activity is defined by a start/finish time interval (s_i, f_i). The objective is to schedule the maximum number of nonoverlapping activities. Consider the following greedy strategies. In both instances, we sort the tasks according to the given criterion. Then, we repeatedly schedule the next task from the order that does not overlap in time with a previously scheduled task.

(a) *Shortest activity first:* Sort the activities in increasing order of duration $f_i - s_i$. Give a counterexample to show that this is *not* optimal.

(b) *Latest start time first:* Sort the activities in decreasing order of start time. Give a short proof that this is optimal. (You may use any results proved in class.)

Problem 5. (20 points) The following problem assume that $G = (V, E)$ is a DAG (directed acyclic graph) with nonnegative edges weights, $w(u, v)$ for each $(u, v) \in E$.

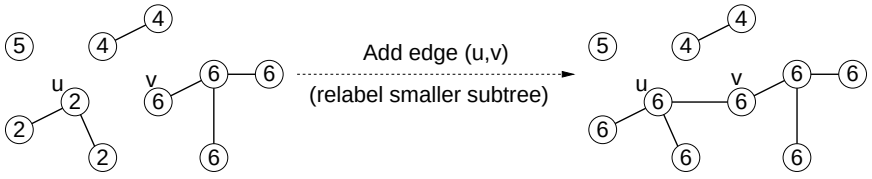
(a) Present an $O(V + E)$ *single-sink* shortest path algorithm. Given G and a sink vertex $t \in V$, it computes the cost of the shortest path from every vertex u to t. (Hint: Let $d[u]$ be the distance from u to t. Compute the d values for every vertex by DFS.) Briefly justify the correctness of your algorithm.



(b) How would you modify your algorithm of part (a) to compute the *single-source* shortest paths from some vertex s to every vertex in G?

Problem 6. (10 points) Recall that we used the Union-Find data structure in Kruskal's algorithm to determine whether two vertices are in the same subtree. This problem investigates an alternative method, based on vertex labeling.

Initially, each vertex is given a unique label. At all times in Kruskal's algorithm, all the vertices within the same subtree have the same label. To achieve this, whenever an edge is added, which merges two subtrees T_1 and T_2, we visit all the vertices of the smaller subtree, and change their labels to match the labels of the larger subtree. (If both subtrees are of the same size, pick one arbitrarily.) This is illustrated in the figure below.



Let $R(u)$ denote the number of times that vertex u has been assigned a new label. Prove that $R(u)$ is $O(\log V)$, where V is the total number of vertices. (Hint: Consider what happens when two trees are merged. It may simplify things to first consider merging trees of equal sizes.)

Homework 4: Paths and NP-Completeness

Handed out November 11. Due at the start of class Tuesday, November 25. (Because of Thanksgiving, there will be no chance for an extension, so plan accordingly.) Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. Describe how to modify the Floyd-Warshall DP formulation algorithm to solve the following problems, all of which involve some sort of all-pairs path problem. Throughout, $G = (V, E)$ is a directed graph, given as an adjacency matrix, in which each edge $(i, j) \in E$ is associated with a numeric weight $w(i, j)$.

In each case give the DP formulation (not a full algorithm) and justify its correctness. Each formulation should lead to an $O(V^3)$ time algorithm. Be sure to specify how you assign the $w_{i,j}$ values if (i, j) is a self-loop or nonexistent edge.

(a) In this problem, the weight $w(i, j)$ denotes the probability that the edge (i, j) exists. (Nonexistent edges of G are treated having probability 0.) Each weight is in the range from 0 to 1. The probability that a path exists is defined to be the product of the edge probabilities along the path. For each pair of vertices, $i, j \in V$, compute the (probability of) the maximum probability path from i to j, denoted $P_{i,j}$.

(Note: You may discover that there is a way to reduce this problem directly to the Floyd-Warshall algorithm, but I would like you to solve this problem by dynamic programming. You may give the reduction for extra credit.)

(b) Assume in this case that G is a DAG, and $w(i, j) = 1$ if $(i, j) \in E$ and $w(i, j) = 0$ otherwise. Two paths are said to be *distinct* if they differ in any way. For each pair of vertices $i, j \in V$, compute the total number of distinct paths from i to j, denoted $C_{i,j}$. Why did we need to assume that G is a DAG? (Hint: Pay special attention to the basis case here, because it is easy to get it wrong.)

Problem 2. Consider the following problem, which (superficially) is very similar to the problems of Problem 1. Let G be a digraph with numeric edge weights, $w(i, j)$. Define the cost of a path to be sum of edge weights along the path (this is the usual definition). For each pair of vertices $i, j \in V$ compute the cost of the *maximum cost path* from i to j.

(a) Explain why this problem makes no sense if G has any positive weight cycles (that is, the sum of edge weights on the cycle is strictly positive).

(b) One way to get around the above problem, is to require that the path being computed is *simple*, that is, it does not visit any vertex more than once. Derive an "obvious" modification of the Floyd-Warshall DP formulation to compute the longest simple path in a weighted digraph. (Note: Read part (c) before attempting this.)

(c) The formulation you gave in part (b) is incorrect! Explain why. In particular, what crucial property holds for the problems in Problem 1 that fails to hold for this problem.

Problem 3. For each part, indicate whether the claim is True, False, or "Not known to science." Let A and B be any two decision problems.

(a) If A and B are both in P then $A \leq_P B$.

(b) If A and B are both in NP then $A \leq_P B$.

(c) If $A \leq_P B$ and there is an $O(n^6)$ algorithm for B, then there is an $O(n^6)$ algorithm for A.

Problem 4. The *Set Cover* (SC) problem is: Given a triple (X, S, k), where X is a finite set and a $S = \{s_1, s_2, \ldots, s_n\}$ is a collection of subsets of X, and k is a positive integer, does there exist a subset $C \subseteq S$ of k sets whose union equals X, that is

$$X = \bigcup_{s_i \in C} s_i.$$

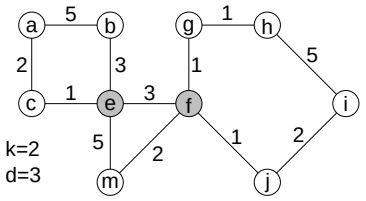
For example, consider the system of sets given below, where $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

$$\begin{array}{llllllllll} s_1 & = & \{1,4,6\} & s_2 & = & \{4,9\} & s_3 & = & \{7,8\} & s_4 & = & \{5,8\} \\ s_5 & = & \{2,6,8\} & s_6 & = & \{3,5,9\} & s_7 & = & \{7,9\} \end{array}$$

The system has a cover of size four consisting of sets $C = \{s_1, s_3, s_5, s_6\}$, since their union equals X.

Prove that the set cover problem is NP-complete. Show both (a) that SC $\in$ NP, and (b) that some known NP-complete problem is polynomially reducible to SC. Prove the correctness of your reduction, and give an example to illustrate your reduction. (Hint: Use vertex cover.)

Problem 5. The *k-center problem* (KC) is: Given an undirected graph $G = (V, E)$ with integer edge weights, an integer k, and a distance bound d, does there exist a subset V' of k vertices such that every vertex in G is within distance d of some vertex in V'? For example, in the figure below, there is a k-center set of size 2 for distance bound 3, since every vertex is within distance 3 of the two vertices e and f.



Remember to show both that KC $\in$ NP and that some known NP-complete problem is polynomially reducible to KC. (Hint: Reduction from dominating set.)

Challenge Problem. The 2SAT problem is, given a boolean formula in CNF form in which each clause contains exactly two literals, determine whether there is an variable assignment such that the formula evaluates to true. Prove that 2SAT is solvable in polynomial time. (In fact, linear time is possible.)

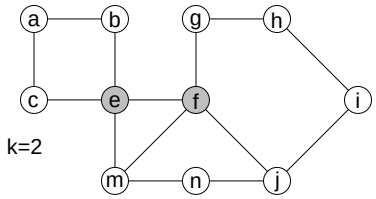
(Hint: This can be reduced to a digraph problem by observing that the clause $(x \vee y)$ is equivalent to the implications $(\overline{x} \Rightarrow y)$ and $(\overline{y} \Rightarrow x)$. Construct a digraph whose vertices correspond to literals, either x or $\overline{x}$, and for each clause $(x \vee y)$, add the directed edges $(\overline{x}, y)$ and $(\overline{y}, x)$ to the digraph. What does the existence of a directed path running from x to $\overline{x}$ or $\overline{x}$ to x imply about satisfiability? What algorithm have we studied this semester that will allow us to test this property efficiently?)

Homework 5: NP-Completeness and Approximations

Handed out Tue, Dec 2. Due by 2:00pm, Monday Dec 15. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. Recall that a *Hamiltonian cycle* is a simple cycle that visits every vertex exactly once. Suppose that you have access to a function HamCycle(G), which given an undirected graph G returns true if G has a Hamiltonian cycle and false otherwise. Using this function, present an algorithm that either determines that G has no Hamiltonian cycle, or if it does, outputs the sequence of vertices in some such cycle. Your algorithm should make at most $O(E)$ calls to the HamCycle function, where E is the number of edges in G. Briefly explain how your algorithm works. (Note: Be sure to explain why your algorithm is correct even if G has multiple Hamiltonian cycles.)

Problem 2. Given an undirected graph $G = (V, E)$ a *2-Hop Dominating Set* (2HDS) is a subset of vertices $V' \subseteq V$ such that every vertex is either in V' or is connected to a vertex of V' by a path of length at most 2. In other words, a 2HDS is a generalization of a standard Dominating Set (DS) in which a vertex of V' covers not only itself and its neighbors, but also its neighbors' neighbors. The 2HDS problem is, given a graph G and integer k, does G have a 2HDS of size k? For example, the graph below has a 2HDS of size 2.



Show that 2HDS is NP-hard. Be sure to give a careful proof of correctness. (Hint: Reduction from either Vertex Cover or Dominating Set. The reduction from DS is easy, but the correctness proof is hard. The reduction from VC is harder, but the correctness proof is easier.)

Problem 3. Recall the *activity scheduling problem* from Lecture 7. We are given a set $S = \{1, 2, \ldots, n\}$ of n *activities*, which are to be scheduled to use some resource. Each activity has a start time s_i and finish time f_i. We say that two activities i and j *overlap* if their (open) start-finish intervals have a nonempty intersection, that is, if $(s_i, f_i) \cap (s_j, f_j) \neq \emptyset$. (Note that if one activity finishes exactly when the other starts, they are not overlapping.) The problem is to select a maximum number of nonoverlapping events. (The resource utilization time is not considered.)

In class we presented an optimal greedy strategy (*earliest finish time first*), henceforth called *OPT*. Consider the following (nonoptimal) strategy, called *shortest activity first*, or *SAF*. Sort the activities in increasing order of duration $(f_i - s_i)$. For each activity in this order, if it does not overlap any previously scheduled activities, then add this activity to the schedule.

(a) Give a series of counterexamples that show that there are arbitrarily large inputs for which the SAF strategy can be at least twice as bad as the optimum. In particular, show that for each $k \geq 1$, there is an input such that SAF generates a schedule with k activities, but OPT generates a schedule with at least $2k$ activities.

(b) Prove that the SAF strategy has a performance ratio bound of 2. In other words, give a proof that for any input S, if SAF schedules k activities, then OPT schedules at most $2k$ activities. (Hint: First, show that at most two activities of OPT can overlap any activity of SAF.)

Problem 4. A complete graph $G = (V, E)$ with edge weights $c(u, v)$ is said to satisfy the *triangle inequality* if for each triple $u, v, w \in V$, $c(u, w) \leq c(u, v) + c(v, w)$. Consider a variant of the TSP problem, where we are given a complete weighted graph $G = (V, E)$ whose edge weights satisfy the triangle inequality. We are also given a designated *start vertex* s and *target vertex* t. The objective is to compute a Hamiltonian path from s to t whose total cost is at most twice the cost of the minimum cost Hamiltonian path from s to t.

Hint: The construction is very similar to the TSP approximation given in class, based on a twice-around tour of the MST. Root your MST at s. Show how to build the twice-around tour in a more careful manner, so that it ends at the vertex t.

Challenge Problem 1. The following problem is based on the movie "The Ring."

The local video store has received a shipment consisting of n video tapes. Among these is exactly one deadly tape. Exactly 7 days after viewing this tape the viewer dies a mysterious death (without warning or exception). The video store manager wants to determine which of the tapes is the deadly one before the big sale coming up 8 days from now. To help him, he has found a number of very greedy (but very foolish) people who have volunteered to serve as the tape testers for a very high fee.

Each tester is given some subset of the tapes to view, and then nervously waits for 7 days for the final results to develop. (These are short tapes and each viewer can watch any number of tapes, and each tape can be viewed by any number of viewers.) The manager realizes that he can determine the deadly tape by paying for n testers. A smart clerk informs him that he can do it with fewer testers. What is the minimum number of tape testers needed to determine which tape is deadly and how is the test conducted? (The stingy manager does not care how many of the testers dies in the process; he just wants to pay the least number of testers.) If the deadly tape is random among the set of n tapes, then what is the expected number of testers that survive the process?

Challenge Problem 2. The following has nothing to do with algorithm design, but I thought it was a cute puzzle.

You are given a stack of 52 playing cards. Most of the cards are face-down, but exactly 10 of the cards have been turned face-up. These 10 cards are scattered randomly throughout the deck, and you do not know where they are located. Your task is to split the deck into two stacks, such that the number of face-up cards in the first stack is equal to the number of face-up cards in the second stack. (This number need not be 5.) Every one of the 52 cards must be in exactly one of the two stacks.

And here is the catch: You are blindfolded, and cannot tell face-up from face-down by touch or any other means.

Practice Problems for the Final Exam

The final exam will be on Sat, Dec 20, 10:30-12:30. The exam will be closed-book and closed-notes, but you will be allowed one cheat-sheet (front and back).

Disclaimer: These are practice problems, which have been taken from old homeworks and exams. They do not necessarily reflect the actual length, difficulty, or coverage for the exam.

Problem 1. Short answer questions.

(a) You are given a list of n numbers, where each number is an integer in the range from 1 to 2^n. How fast can you sort these numbers?

(b) Your smart friend in Montana tells you that he has discovered a comparison-based sorting algorithm that runs in $O(n)$ time, and he has found a polynomial time solution to 3SAT. Which result is more unbelievable?

(c) Aliens show us an $O(n^k)$ time algorithm which determines whether a graph with n vertices has a vertex cover of size k. Does this mean P = NP?

(d) A smarter group of aliens come to Earth and show us that some known NP-hard problem cannot be solved faster than $O(n^{100})$ time. Does this mean P $\neq$ NP?

(e) Suppose that $A \leq_P B \leq_P C$ and the first reduction runs in $O(n^2)$ time, the second runs $O(n^3)$ time, and C can be solved in $O(n^4)$ time. What can we infer about the time needed to solve A?

Problem 2. Describe a variant of DFS on digraphs, which not only traverses the graph, but which uses the colors and the discovery and finish times to classify the type of each edge: tree edge, cross edge, forward edge, or back edge. Justify your algorithm's correctness. If possible, try to output the classification when the edge is first seen (rather than making two passes over your digraph.) Justify your algorithm's correctness.

Problem 3. Describe an algorithm to determine whether a directed graph with edge weights has a negative cost cycle. Your algorithm should run in $O(n^3)$ time. (Hint: The running time should suggest which algorithm should be adapted for this.)

Problem 4. A *tournament* is a digraph $G = (V, E)$ in which for each pair of vertices u and v, either there is an edge (u, v) or an edge (v, u) but not both. A directed *Hamiltonian path* is a path that visits every vertex in a digraph exactly once.

(a) Prove that for all $n \geq 1$, every tournament on n vertices has a directed Hamiltonian path. (Hint: Use induction on the number of vertices.)

(b) Give an $O(n^2)$ algorithm which given a tournament, finds a Hamiltonian path. (You may assume either an adjacency list or an adjacency matrix representation of G.)

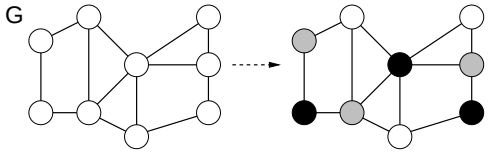
Problem 5. Prove that the following problem, called the *acyclic subgraph problem* (AS) is NP-complete. Given a directed graph $G = (V, E)$ and an integer k, determine whether G contains a subset V' of k vertices such that the induced subgraph on V' is acyclic. Recall that the *induced subgraph* on V' is the subgraph $G' = (V', E')$ whose vertex set is V', and for which $(u, v) \in E'$ if $u, v \in V'$ and $(u, v) \in E$. (Hint: Reduction from Independent Set. Think of a reduction that maps undirected edges to directed cycles.)

(continued on next page)

Problem 6. Show that the following problem is NP-complete. Remember to show (1) that it is in NP and (2) that some known NP-complete problem can be reduced to it.

Balanced 3-coloring (B3C): Given a graph $G = (V, E)$, where $|V|$ is a multiple of 3, can G can be 3-colored such that the sizes of the 3 color groups are all equal to $|V|/3$. That is, can we assign an integer from $\{1, 2, 3\}$ to each vertex of G such that no two adjacent vertices have the same color, and such that all the colors are used equally often. (An example is shown in the figure below, where the colors are shown as white, black, and gray.)

Hint: Reduction from the standard 3-coloring problem (3COL), which is defined on page 1019 of CLRS.



Problem 7. Do problem 35.2–3 on page 1032 of CLR. (Prove that the closest point hueristic gives a TSP tour that is within a factor 2 of optimal.) You may assume that the cost function satisfies the triangle inequality. (Hint: Relate the sum of closest point distances to the cost of the minimum spanning tree.)

Problem 8. The *set cover* optimization problem is: Given a pair (X, S), where X is a finite set and a $S = \{s_1, s_2, \ldots, s_n\}$ is a collection of subsets of X, find a minimum sized collection of these sets C whose union equals X. Consider a special version of the set-cover problem in which each element of X occurs in *at most* three sets of S. Present an approximation algorithm for this special version of the set cover problem with a ratio bound of 3. Briefly derive the ratio bound of your algorithm.

Final Exam

This exam is closed-book and closed-notes. You may use two sheets of notes (front and back). Write all answers in the exam booklet. You may use any algorithms or results given in class. If you have a question, either raise your hand or come to the front of class. Total point value is 100 points. Good luck!

Problem 1. (40 points; 4–7 points each.) Short answer questions. Explanations are not always required, but may be given for partial credit.

(a) Which of the following is the fastest algorithm for sorting all the students enrolled at the University of Maryland according to their birth year? (List one.)

BubbleSort QuickSort CountingSort

(b) A connected undirected graph G has V vertices and E edges. As a function of V and E, how many back edges are there in the DFS tree of G? (If you believe that it cannot be determined from just V and E, and depends on the order in which DFS visits the vertices of G, then say so.)

(c) What is a *prefix code*? Give one advantage of a prefix code over a non-prefix code.

(d) True or False: If a graph G has a vertex cover of size k, then it has a dominating set of size k or smaller.

(e) Suppose that $A \leq_P B$, and there is a factor-2 approximation to problem B, then

(A) There is a factor-2 approximation for A.

(B) There is a constant factor approximation for A, but the factor might not be 2.

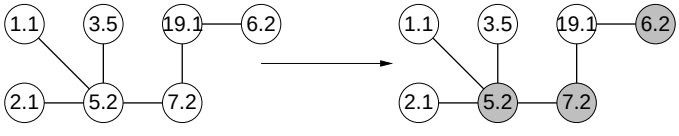
(C) We cannot infer anything about the approximability of A.

(f) True or False: Both A and B are NP-hard and $A \leq_P B$. This implies that $B \leq_P A$.

(g) The great wizard Gandalf tells you that he has discovered a polynomial time algorithm that can determine whether a graph has a Hamiltonian cycle, provided that the number of vertices of the graph is a prime number less than 1,000,000. Does Gandalf's claim imply that P = NP? Why or why not?

Problem 2. (10 points) Describe how to modify the Floyd-Warshall DP formulation to solve the following problem in $O(n^3)$ time. Let $G = (V, E)$ be a directed graph. For each pair of vertices, $i, j \in V$, compute the reachability matrix R, which is defined as follows. $R[i, j] = 1$ if there is a path from i to j and $R[i, j] = 0$ otherwise. (There is always a path from i to itself.) Briefly explain how you derived your formulation.

Problem 3. (15 points) Let $G = (V, E)$ be a *free tree*, that is, an undirected, connected, acyclic graph. Each vertex $u \in V$ is associated with a cost $c(u)$. A *vertex cover* is a subset V' of vertices such that every edge in E is incident to at least one vertex in V'. The cost of a vertex cover is the sum of the costs of the vertices of V'.

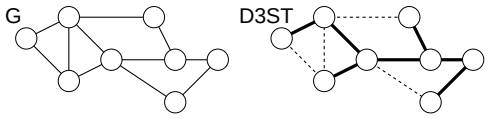


Give an $O(V + E)$ time algorithm that computes the cost of the minimum cost vertex cover for G. You need only compute cost, not the actual vertices in the cover. (Hint: This can be done by either dynamic programming or depth-first search. If you solve it by DP, you need only give the DP formulation.)

Problem 4. (15 points) Show that the following problem is NP complete. Remember to show that the problem is in NP and that some known NP-complete problem is reducible to this problem. Briefly prove the correctness of your reduction.

Degree-3 Spanning Tree (D3ST): Given an undirected graph $G = (V, E)$, does G have a spanning tree T in which every vertex in the spanning tree is of degree 3 or less? (An example is shown in the figure below.)

(Hint: Reduction from Hamiltonian path.)



Problem 5. (20 points) The goal of this problem is to explore the approximability of TSP if the graph's edge weights do NOT satisfy the triangle inequality. Recall that the TSP problem is: given a complete graph with weighted edges, find the cycle of minimum total cost that visits every vertex exactly once.

(a) Give an example of a complete undirected weighted graph (whose edge weights do NOT satisfy the triangle inequality) such that the TSP approximation given in class results in a tour whose cost is strictly more than twice the cost of the optimum TSP tour. Show each of the steps of the approximation algorithm on your graph. (Hint: This can be done with as few as 4 vertices.)

(b) Suppose that for some constant $c > 1$, there exists a polynomial time, factor-c approximation algorithm for TSP for graphs with arbitrary edge weights. Show how to use such a procedure to solve the Hamiltonian Cycle problem in polynomial time.