Homework 1: Algorithm Design Basics

Handed out Thu, Sep 6. Due at the start of class Thu, Sep 20. Late homeworks are not accepted, but you may drop your lowest homework score.

References: For reference information on asymptotics, summations, and recurrences, see either the text, by Cormen, Leiserson, Rivest, and Stein or the text by Kleinberg and Tardos. Also see the class handout on Background Information.

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.

A Note about Writing Algorithms: When presenting algorithms, more detail is not necessarily better. Remember that your algorithm will be read by a human, not a compiler. You may assume that the reader is familiar with the problem. Be sure that your pseudo-code is sufficiently detailed that your intentions are clear and unambiguous. Avoid adding extraneous details and confusing jargon. (E.g., It is much clearer to say "Insert x at the end of the list" rather than `list.insertAtEnd(x)`.) You may make use of any standard data structures (linked lists, binary trees, heaps, etc.) without explaining how to implement them. If you have any questions, check with us.

In addition to presenting pseudo-code, explain your general strategy in English. (This way, if you make a coding error, the grader can ascertain your real intent and give partial credit.) It is also a good idea to provide a short example to illustrate your approach. Even if you are not explicitly asked, you should always provide a justification of correctness of your algorithm and analysis of its running time.

Problem 1. Consider the following simpler algorithm for the stable marriage problem. As in the standard problem, there are n men and n women, and each man and each woman has an n-element preference list that orders all the members of the opposite sex. This algorithm ignores woman preferences, and simply pairs each man with the first available woman on his list.

```
for (i = 1 to n) {
    let (w[1], ..., w[n]) be the women of m[i]'s preference list (from high to low)
    j = 1
    while (j <= n and m[i] is not yet engaged) {
        if (w[j] is not yet engaged) {
            match m[i] with w[j] (and both are now engaged)
        }
        else j = j + 1
    }
}
```

(Note that in this algorithm, once a woman accepts a man's proposal, she will never break it off.) We will explore the correctness of this algorithm by answering the following questions.

(a) Is this algorithm guaranteed to produce a perfect matching (that is, every man is paired exactly one woman and vice versa)? If so, give a proof, and if not, give a counterexample.

(b) If your answer to (a) was "no", skip the rest of the problem. Otherwise, is the matching produced by this algorithm guaranteed to be stable? If so, give a proof, and if not, present a counterexample.

(c) If your answer to (b) was "yes", skip this part. Otherwise, suppose that all the women in this system have exactly the same sets of preferences, and in particular, they rank the men in (decreasing preference) order $\langle m_1, m_2, \ldots, m_n \rangle$. (Each man's list contains all the women, but otherwise each man's preferences are arbitrary.) Under this restriction, is the matching produced by this algorithm guaranteed to be stable? As before, either give a proof or present a counterexample.

(Note: Throughout the semester, whenever you are asked to present a counterexample, you should strive to make your counterexample as short and clear as possible. In addition to giving the input for the counterexample, briefly explain what the algorithm does when run on this input and why it is wrong.)

Problem 2. Consider the following summation, which holds 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 identity holds for all $n \geq 0$, by induction on n. (Recall that by convention, for $n = 0$ we have an empty sum, whose value is defined to be the additive identity, that is, zero.)

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

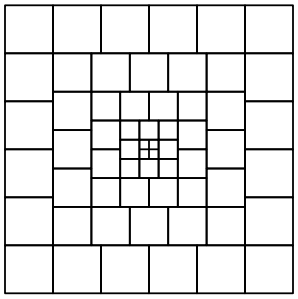


Figure 1: Problem 2.

Problem 3. For each of the parts below, list the functions in increasing asymptotic order. In some cases functions may be asymptotically equivalent (that is $f(n)$ is $\Theta(g(n))$). In such cases indicate this by writing $f(n) \approx g(n)$. When one is asymptotically strictly less than the other (that is, $f(n)$ is $O(g(n))$ but $f(n)$ is not $\Theta(g(n))$), express this as $f(n) \prec g(n)$. For example, given the set:

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

the first function is $\Theta(n^2)$ and the other two are $\Theta(n \log n)$, and therefore the answer would be

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

Explanations are *not* required, but may be given to help in assigning partial credit.

(a)	$(3/2)^n$	$3^{(n/2)}$	$2^{(n/3)}$
(b)	$\lg n$	$\ln n$	$\lg(n^2)$
(c)	$n^{\lg 4}$	$2^{\lg n}$	$2^{(2 \lg n)}$
(d)	$\max(50n^2, n^3)$	$50n^2 + n^3$	$\min(50n^2, n^3)$
(e)	$\lceil n^2/20 \rceil$	$\lfloor n^2/20 \rfloor$	$n^2/20$

Problem 4. The purpose of this problem is to design a more efficient algorithm for the previous larger element problem, as introduced in class. Recall that we are given a sequence of numeric values, $\langle a_1, a_2, \ldots, a_n \rangle$. For each element a_i, for $1 \leq i \leq n$, we want to know the index of the rightmost element of the sequence $\langle a_1, a_2, \ldots, a_{i-1} \rangle$ whose value is strictly larger than a_i. If no element of this subsequence is larger than a_i then, by convention, the index will be 0. Here is naive the $\Theta(n^2)$ algorithm from class.

```
previousLarger(a[1..n]) {
    for (i = 1 to n)
        j = i-1;
        while (j > 0 and a[j] <= a[i]) j--;
        p[i] = j;
    }
    return p
}
```

There is one obvious source of inefficiency in this algorithm, namely the statement `j--`, which steps through the array one element at a time. A more efficient approach would be to exploit p-values that have already been constructed. (If you don't see this right away, try drawing a picture.) Using this insight, design a more efficient algorithm. For full credit, your algorithm should run in $\Theta(n)$ time. Prove that your algorithm is correct and derive its running time.

Challenge Problem. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

An *in-place algorithm* is one that is given its input in a chunk of memory (e.g., as an array) which it is allowed to modify in the course of the algorithm, it stores its output in this same chunk of memory. Otherwise, it is only allowed to use a constant amount of additional working storage (e.g., local variables). It *cannot* declare any additional arrays (including strings) and it *cannot* allocate memory dynamically. It is also *not allowed* to make recursive calls (since doing so would allow it to use the system's recursion stack implicitly for working storage). The in-place restriction is often important in applications where the inputs are very large arrays, that barely fit into physical memory.

Suppose that you are given an array $X[1..n]$ of numbers, and you are given an index i, where $1 \leq i \leq n-1$. Consider the two subarrays, $X[1..i]$ and $X[i+1..n]$. Of course, these subarrays are not necessarily of the same size. Give an $O(n)$ time *in-place* algorithm which, given as input the array X and i, swaps these two subarrays within X, placing the contents of $X[i+1...n]$ before $X[1..i]$. The order of elements within each subarray should not be changed. An example is presented below.

	n	i	X
Input:	11	4	`[ 6  9  2 10  3  7 -4 18  5  1  0]`
Output:			`[ 3  7 -4 18  5  1  0  6  9  2 10]`

Homework 2: Greedy Algorithms (Revised!)

Handed out Tue, Oct 25. Due at the start of class Tue, Oct 9. (Note the new due date.) Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. In class we presented a greedy algorithm for scheduling a set of n tasks, in which each task is given a duration t_i and deadline d_i. We showed that scheduling the tasks in increasing order of deadline minimizes the *maximum* lateness. (Recall that if task i is scheduled at time $s(i)$, then its lateness is $\ell_i = \max(0, s(i) + t_i - d_i)$.) Define the *average lateness* to be $(1/n)\sum_{i=1}^{n} \ell_i$.

(a) Provide a counterexample to show that scheduling tasks in increasing order of *deadline* does not minimize average lateness. Briefly explain your example.

(b) Provide a counterexample to show that scheduling tasks in increasing order of *duration* does not minimize average lateness. Briefly explain your example.

(c) Suppose that we redefine lateness to be $\ell_i = s(i) + t_i - d_i$ (ignoring the "max"). Thus, if the task finishes before the deadline, its lateness is negative. (Intuitively, this can be thought of as a bonus, which can be applied to reduce the positive lateness of some other task.) Prove that if tasks are scheduled in increasing order of duration, then average lateness (under this modified definition) will be minimized. (Hint: Use the same sort of exchange argument we used in class to prove that earliest deadline first minimizes maximum lateness.)

Remark: When constructing a counterexample, try to make the counterexample as simple as possible. For example, a counterexample with three tasks is better than one with five tasks, because it is easier to understand. Also, avoid ambiguous situations. For example, if your algorithm schedules the earliest deadline first, you should not have two identical deadlines and then impose assumptions about which one the algorithm will choose first.

Problem 2. This problem involves the question of computing change for a given coin system. A coin system is defined to be a sequence of coin values $v_1 < v_2 < \ldots < v_n$, such that $v_1 = 1$. For example, in the U.S. coin system we have six coins with values $\langle 1, 5, 10, 25, 50, 100\rangle$. The question is what is the best way to make change for a given integer amount A.

(a) Let $c \geq 2$ be an integer constant. Suppose that you have a coin system where there are n types of coins of integer values $v_1 < v_2 < \ldots < v_n$, such that $v_1 = 1$ and, for $1 < i \leq n$, $v_i = c \cdot v_{i-1}$. (For example, for $c = 3$ and $n = 4$, an example would be $\langle 1, 3, 9, 27\rangle$.) Describe an algorithm which given n, c, and an initial amount A, outputs an n-element vector that indicates the minimum number of coins in this system that sums up to this amount. (Hint: Use a greedy approach.)

(b) Given an initial amount $A \geq 0$, let $\langle m_1, \ldots, m_n\rangle$ be the number of coins output by your algorithm. Prove that the algorithm is correct. In particular, prove the following:

(i) For $1 \leq i \leq n$, $m_i \geq 0$

(ii) $\sum_{i=1}^{n} m_i \cdot v_i = A$

(iii) The number of coins used is as small as possible

Prove that your algorithm is optimal (in the sense that of generating the minimum number of coins) for any such currency system.

(c) Give an example of a coin system (either occurring in history, or one of your own invention) for which the greedy algorithm may fail to produce the minimum number of coins for some amount. Your coin system must have a 1-cent coin.

Problem 3. You are given an undirected graph $G = (V, E)$ in which the edge weights are highly restricted. In particular, each edge has a positive integer weight of either $\{1, 2, \ldots, W\}$, where W is a constant (independent of the number of edges or vertices). Show that it is possible to compute the single-source shortest paths in such a graph in $O(n + m)$ time, where $n = |V|$ and $m = |E|$. (Hint: Because W is a constant, a running time of $O(W(n + m))$ is as good as $O(n + m)$.)

Problem 4. You are the mayor of a beautiful city by the ocean, and your city is connected to the mainland by a set of k bridges. Your city manager tells you that it is necessary to come up with an evacuation plan in the event of a hurricane. Your idea is to add a sign at each intersection pointing the direction of the route to the closest of the k bridges. You realize that this can be modeled as a graph problem, where the street intersections are nodes, the roads are edges, and the edge weights give the driving time between two adjacent intersections. Note that some of the roads in your city are one-way roads.

(a) Explain how to solve the evacuation-route problem in $O(m \log n)$ time, where n is the number of intersections and m is the number of streets connecting two adjacent intersections. Note that the number of bridges k is *not* a constant, that is, it may depend on n and m. Therefore, a running time of $O(k \cdot m \log n)$ is not an acceptable solution. (Although I will give half credit for such a solution.) The output of your algorithm will be a labeling of the intersections, with an arrow pointing to the road to take to the closest bridge.

(b) More disasters for the mayor! From each of the k bridges, a huge mass of crazed, migrating lemmings have come scurrying into the city. These voracious arctic rodents are organized into k different *gangs*,[1] each entering the city over a different bridge. To complicate matters, each of the k gangs runs at a different speed. For $1 \leq i \leq k$, let s_i denote the speed (in units per second) of the rodent gang streaming over the ith bridge.

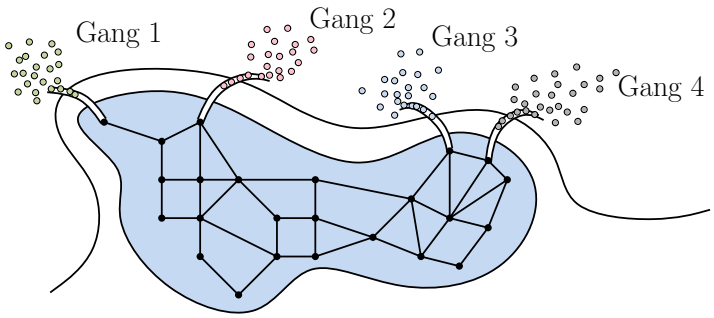


Figure 1: Problem 4(b).

The first gang of lemmings that arrives at an intersection claims the intersection as their own, and any other lemming gang arriving at this intersection is blocked from moving any further. Once a lemming gang arrives first at an intersection, they then start streaming out along each of the incident streets (again at the same speed of s_i units per second) in an attempt to claim other intersections. (By the way, lemmings don't read, so effectively there are no one-way streets.) Each gang has essentially an unlimited supply of lemmings, so they will eventually populate all the intersections of the city.

Your job as a member of the city's Lemming Response Task Force is to write an algorithm, which given the city's streets and the s_i values, labels each intersection with the index i, where $1 \leq i \leq k$, of the lemming gang that will first arrive and claim this intersection. It should also label the intersection with the time this gang arrives. For full credit, your algorithm should run in $O(m \log n)$ time, irrespective of k.

[1] I call them gangs because I don't know what the proper name is for a group of lemmings. Herd? Horde? Flock? Gaggle? Someone needs to research this!

Challenge Problem 1. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

Given an undirected graph $G = (V, E)$ and a source vertex s, present an algorithm that computes the length (number of edges) of the shortest *simple* cycle that contains s. (Since the cycle is simple, it cannot visit any edge twice.) Your algorithm should run in time $O(n+m)$, where $n = |V|$ and $m = |E|$. (Hint: Modify BFS.) Prove that your algorithm is correct. (I intentionally chose G to be an undirected graph. I think that the problem is a bit simpler for directed graphs. If you don't see the solution for undirected graphs, you can get partial credit by giving a solution to the case of a directed graph.)

Challenge Problem 2. (This is not really an algorithms problem.) In CMSC 451, Prof. M. has assigned each student a seat number for the final exam. The ever devious Prof. M.'s has decided to play a prank on his poor students. There are n students in the class and exactly n seats in the room. When the first student walks into the class, Prof. M. tells this student to ignore the seat assignment and just select a random seat. After this, as each student arrives, if their assigned seat is empty, they sit there. If their assigned seat is taken, they select a random empty seat to sit in.

As luck would have it, on the day of the final exam your alarm clock fails to go off. You arrive *last* to the exam. Since there are n seats and $n - 1$ students so far, exactly one seat is empty. What is the probability that your seat is the final seat available? To avoid messy computations, you are given three options. Justify whichever option you pick.

- In the limit, as $n \to \infty$, the probability that your seat is available on your arrival is 1.
- In the limit, as $n \to \infty$, the probability that your seat is available on your arrival is 0.
- In the limit, as $n \to \infty$, the probability that your seat is taken is some constant p, where $0 < p < 1$. If you select this option, what is p?

Homework 3: Greedy Algorithms and Divide and Conquer

Handed out Tue, Oct 9. Due at the start of class Tue, Oct 23. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. Consider an alphabet with m symbols $X = \{x_1, \ldots, x_m\}$. For each $x \in X$, let $p(x)$ denote its associated probability. Let T denote the tree that results by running Huffman's algorithm on X, and recall that $d_T(x)$ denotes the depth (number of edges from the root) of x in T. In class we showed that a string of length n can be encoded using $n \cdot B(T)$ bits, where

$$B(T) = \sum_{x \in X} p(x) d_T(x).$$

(This is a bit different from the definition given in class.)

The purpose of this problem is to investigate the relationship between $B(T)$ and a fundamental concept from information theory, called *entropy*. Given X as above, define the *entropy* of X, denoted $H(X)$, to be

$$H(X) = -\sum_{x \in X} p(x) \lg p(x).$$

(Recall that lg denotes the base-2 logarithm.) Note that, since $0 < p_i < 1$, we have $\lg p_i < 0$. Therefore, $H(X)$ is a nonnegative value.

Suppose that, in addition to the basic requirements that the probabilities are nonnegative and sum to 1, we have the additional property that they are all *powers of 2*. For example, suppose that X consists of 10 elements whose probabilities are $\left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{32}, \frac{1}{32}, \frac{1}{32}, \frac{1}{64}, \frac{1}{64}\right\}$.

(a) Show the tree that results by running Huffman's algorithm on the above example, shown as black dots.

(b) Letting T denote tree from (a), what relationship do you observe between $d_T(x)$ and $p(x)$? (If this is not clean and simple, double-check that you built the tree properly.)

(c) Prove that your observation from part (b) holds for *any* set consisting of probabilities that are powers of 2. (Hint: Use induction on m.)

(d) Based on your answers to (b) and (c), what can you infer about the relationship between $B(T)$ and $H(X)$, when the probabilities of X of powers of 2?

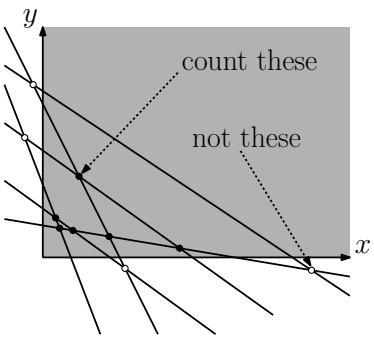
Problem 2. Given a list $A = \langle a_1, \ldots, a_n \rangle$ of integers, an *off-parity inversion* is a pair a_i and a_j such that $i < j$, $a_i > a_j$, and a_i and a_j are of different parities. (In other words, it is an inversion in which one number is even and the other is odd.)

Design an $O(n \log n)$ time algorithm that counts the number of off-parity inversions in a list A containing n elements. Justify your algorithm's correctness and derive its running time.

Problem 3. You are given a set of n lines in the plane. Each line is given by a pair of numbers (a_i, b_i), which represents the line $y = a_i x + b_i$. That is, a_i gives the slope of the line and b_i gives its y-intercept. You are told that $a_i < 0$ (all slopes are negative) and $b_i > 0$ (all intercepts are positive). You are asked to count the number of intersections that occur in the positive (x, y)-quadrant. (For example, in Fig. 1, there are six intersections in the first quadrant, shown as black dots.)

Of course, this would be easy to do in $O(n^2)$ time, but I want you to develop an answer for this problem that runs in $O(n \log n)$ time.

For simplicity, you may assume that the a_i values are all distinct and the b_i values are also all distinct. You may also assume that no two lines intersect exactly on the x-axis or the y-axis.



answer: 6

Figure 1: Problem 3.

Problem 4. This in computer graphics it is often desirable to compute the silhouette of a collection of objects. In this problem, we'll consider a simple example involving rectangles.

You are given a collection of (possibly overlapping) rectangles that extend upwards from the x-axis. Each rectangle is defined by a triple (a_i, b_i, h_i) where a_i and b_i denote the rectangle's left and right x-coordinates and h_i denotes the height of the rectangle. The union of these rectangles defines an *upper envelope*, which consists of a sequence of nonoverlapping intervals from left to right along the x-axis. Each interval is associated with a height value of the tallest rectangle that spans this interval.

For example, the input for the rectangles shown in Fig. 1 might be as follows. (Note that the rectangles are not given in any particular order.)

$$(1, 4, 2), (11, 12, 4), (8, 10, 7), (7, 13, 5), (2, 6, 4), (9, 15, 3), (3, 5, 3).$$

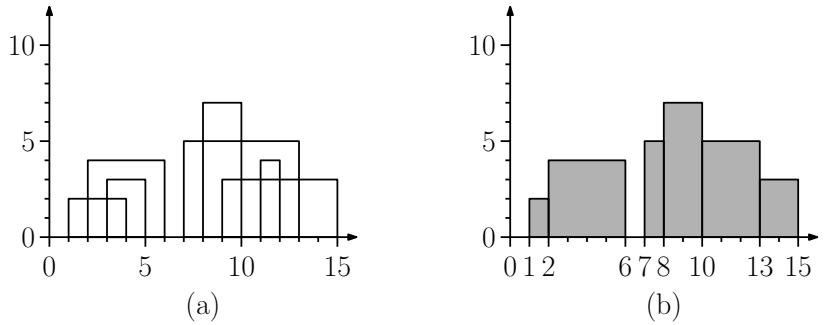


Figure 2: Problem 4.

The output consists of seven intervals (including one interval of height 0). Suppose that the output consists of m intervals. We represent this as an array $x[1..m+1]$ such that the ith interval spans $x[i]$ to $x[i+1]$, and an array $t[1..m]$ where $t[i]$ is the height of the tallest rectangle spanning the ith interval. The output for the above input would be:

$$x = \langle 1, 2, 6, 7, 8, 10, 13, 15 \rangle \qquad \text{and} \qquad t = \langle 2, 4, 0, 5, 7, 5, 3 \rangle.$$

Design an $O(n \log n)$ algorithm which, given a sequence of n such triples, computes the upper envelope of these union of the rectangles. Derive the running time of your algorithm. You may assume that the values a_i, b_i, and h_i are all distinct.

Challenge Problem. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

Given a positive integer n consider a square *grid* consisting of n rows and n columns. We can index each element of the grid as $g[i, j]$, where $1 \leq i, j \leq n$. An $n \times n$ *grid graph* is an undirected graph whose nodes form an $n \times n$ grid, and each grid node is connected by an edge to the (up to four) nodes immediately north, south, east, and west of it (assuming they exist). See Fig. 3(a).

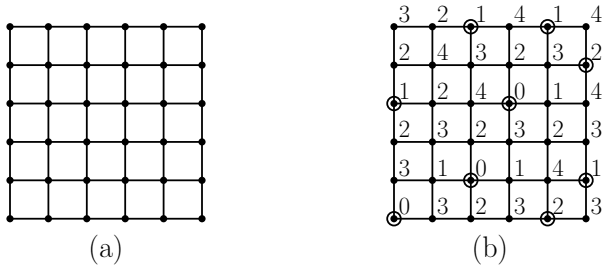


Figure 3: Challenge Problem: (a) a grid graph and (b) local depressions in a grid graph with elevation labels.

Grid graphs are often used to represent terrains, where each node is labeled with an elevation $e[i, j]$. You may assume that there are are no *flat spots*, in the sense that, if two nodes are adjacent, they have different elevation values. A node is called a *local depression* if all of its neighbors have strictly higher elevation values than its own (circled in Fig. 3(b)). Suppose you are given an $n \times n$ grid graph $G = (V, E)$. (Observe that G has n^2 vertices and $O(n^2)$ edges.)

Design an algorithm that finds *any one* local depression in G. Of course, this would be easy to do in $O(n^2)$ time. To get credit, you must solve the problem in $O(n \log n)$ time or, even better, $O(n)$ time. (This may seem impossible, since you do not have enough time to inspect every node of the graph, but it can be done. The fact that there are no flat spots is critical.)

Homework 4: Dynamic Programming and Network Flows

Handed out Tue, Nov 13. Due at the start of class Tue, Nov 20. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. The following algorithm is used in spelling checkers and correctors, as well as in determining string similarity in genetics research. You are given two strings, $X = \langle x_1 \ldots x_m \rangle$ and $Y = \langle y_1 \ldots y_n \rangle$. Define the *edit distance* between X and Y to be the minimum number of single character insertions, deletions, and single-character replacements that are applied to X to make it equal to Y. For example, if $X = \langle$barney$\rangle$ and $Y = \langle$crony$\rangle$ then the edit distance is 4. The sequence of changes is:

Replace x_1 ('b') with y_1 ('c').
Delete x_2 ('a').
Insert y_3 ('o') after x_3 ('r').
Delete x_5 ('e').

The objective of this problem is to derive an $O(mn)$ time and $O(mn)$ space algorithm for this problem.

(a) Give a dynamic programming formulation (the recursive rule) for determining the minimum edit distance between X and Y. (Hint: Modify the rule for the longest common subsequence.)

(b) Use your recursive rule to generate pseudo-code for a dynamic programming algorithm for this problem. Your algorithm should also contain the necessary "hooks" so that you can extract the actual edit operations (see part (c)).

(c) Using the algorithm from (b), give pseudo-code for an algorithm that outputs the sequence of editing operations (in a form similar to the one given above).

Problem 2. A thief is robbing a store. There are n items in the store. The ith item has a weight of w_i and a dollar value v_i. The thief has a knapsack that can hold a total of W units of weights before ripping open. All weights and values are positive integers. The problem is to determine the greatest value of goods that the thief can carry away in his knapsack. The thief may either leave an object or take the entire object. (So, he cannot steal a fraction of an object for a fraction of the value and weight.)

(a) Give a recursive dynamic programming rule for this problem. (Hint: For $0 \leq i \leq n$ and $0 \leq u \leq W$, let $V[i, u]$ be the maximum value that the thief could steal assuming that he may select only among the first i objects and that he has a knapsack of capacity u.)

(b) Give the pseudo-code for a dynamic programming algorithm to solve this problem. Your algorithm need not determine the actual items to be stolen, just the maximum value. Your algorithm should run in $O(nW)$ time and $O(nW)$ space.

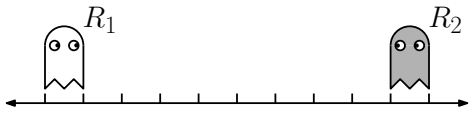
Problem 3. Describe a polynomial time algorithm for the following problem. You are given a flow network $G = (V, E)$ with source s and sink t, and whose edges all have unit capacity, that is, $c(u, v) = 1$ for all $(u, v) \in E$. Your algorithm is also given a parameter k, where $1 \leq k \leq m$, where $m = |E|$. The objective is to delete k edges from G so as to reduce the maximum s-t flow value as much as possible. In other words, compute a subset $E' \subseteq E$ of size k such that the flow in $G' = (V, E \setminus E')$ is as low as possible.

Justify your algorithm's correctness. (Hint: Your solution may involve computing a network flow in G or some variant of G.) Let $T_f(n, m)$ denote the running time of a good algorithm for network flow on a network with n vertices and m edges. Express the running time of your algorithm as a function of n, m, and $T_f(n, m)$.

Problem 4. Network flow issues come up in dealing with natural disasters and other crises, since major unexpected events often require the movement and evacuation of large numbers of people in a short amount of time. Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location (so different people will have different options for hospitals, depending on where they are right now). At the same time, one doesn't want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospitals is balanced: Each hospital receives at most $\lceil n/k \rceil$ people. Give a polynomial-time algorithm that takes the given information about the people's locations and determines whether this is possible.

Challenge Problem. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

You have two robots R_1 and R_2 that can move along the integer line (which is infinite in both directions). The robots start at different positions along the line, but you have no idea what these positions are (see Fig. 1). (In particular, a robot does not know which position it is in, it does not know whether it is to the left or right of the other robot, and it does not know how far away the other robot is.)



(a)

Figure 1: Challenge Problem.

Write a program to help the robots find each other. The two robots are synchronized and execute one step every second. A robot may do any of the following things at each step of my program:

- Move itself left/right one position
- Mark the spot that it is on, but this action can be done only once per robot
- Test whether the spot the current robot is on has been marked
- Test whether the other robot is currently on the same spot as this robot

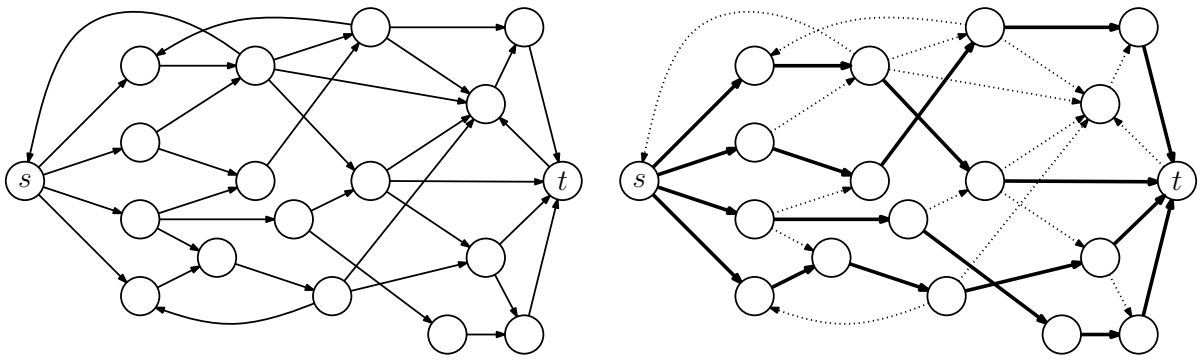
Each robot has only a finite amount of bits of local memory. The initial distance between the two robots may be much much larger than any number that can be stored in the small memory of the robots. Note that marks cannot be erased. The running time should be $O(n)$, where n is the (unknown) distance between the robot's initial positions.

Hints: It is possible to have both robots execute exactly the same program. Don't try to count, you don't have enough local memory.

Homework 5: Network Flows and NP-Completeness

Handed out Thu, Nov 29. Due at the start of class Tue, Dec 11. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. Present an efficient algorithm for the following problem. Given a directed graph $G = (V, E)$ and two vertices s and t, determine the maximum number of paths between s and t, such that, other than s and t, these paths do not share any vertices in common. (In Fig. 1, there are 4 such paths.)



(a) (b)

Figure 1: Problem 1.

(Hint: As a start, devise a way to compute network flows such that the total flow through any vertex is at most 1.)

Problem 2. The Conflict Set Problem (CONF) is as follows. The input is a pair $(\mathcal{S}, k)$ consisting of a collection of sets $\mathcal{S} = \{S_1, \ldots, S_n\}$ over some finite domain, and a positive integer k. The question is whether there exists a set C of size k such that every set of $\mathcal{S}$ has a nonempty intersection with C. That is, whether for all $1 \leq i \leq n$, $S_i \cap C \neq \emptyset$. (We say that C *conflicts* with S_i.)

For example, let $S_1 = \{1, 2, 3\}$, $S_2 = \{1, 4, 5\}$, $S_3 = \{2, 4, 6\}$, $S_4 = \{2, 5, 7\}$, $S_5 = \{3, 7, 9\}$, and let $\mathcal{S} = \{S_1, \ldots, S_5\}$. There exists a conflict set of size 3 consisting of $C = \{2, 5, 7\}$. Therefore $(\mathcal{S}, 3) \in$ CONF. However, there is no conflict set of size 2 for $\mathcal{S}$ (since no matter which two elements you pick, some set will fail to contain at least one of them), and therefore $(\mathcal{S}, 2) \notin$ CONF.

The goal of this problem is to show that CONF is NP-Complete.

(a) Briefly explain why CONF is in NP.

(b) Prove that CONF is NP-hard by showing that the vertex cover problem, VC, is polynomially reducible to CONF.

Problem 3. Given an undirected graph $G = (V, E)$ and a subset $V' \subseteq V$, 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 $(u, v) \in E$. The *acyclic subgraph problem* (AS) is as follows. Given a directed graph $G = (V, E)$ and an integer k, does G contain a subset V' of k vertices such that the induced subgraph on V' is acyclic? (For example, in Fig. 2(a), we show a graph that has an acyclic subgraph of size $k = 8$. I don't believe a larger acyclic subgraph exists. Fig. 2(b) shows the acyclic induced subgraph.)

The goal of this problem is to show that IS is NP-Complete. (Next page.)

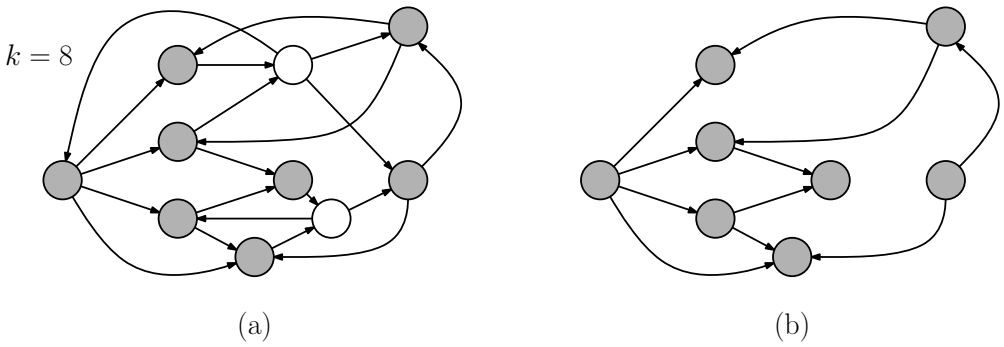


Figure 2: Problem 2.

(a) Briefly explain why AS is in NP.

(b) Prove that AS is NP-hard by showing that the independent set problem, IS, is polynomially reducible to AS.

Problem 4. A sequence of positive integers $A = \langle a_1, \ldots, a_n \rangle$ is given along with a positive integer B. A subset $S \subseteq A$ is said to be *feasible* if the sum of numbers in S does not exceed B, that is

$$\sum_{a_i \in S} a_i \leq B.$$

This sum is called the *total value* of S. Our objective is to find the feasible subset S of A of maximum total value. For example, given $A = \langle 5, 9, 13, 15, 19, 27 \rangle$ and $B = 35$, the subset $S = \{5, 13, 15\}$ has a total value of 33. (This is the best solution I found, but frankly I'm too lazy to try them all out.)

Consider the following heuristic for producing a feasible solution. Initialize $S = \emptyset$ and $t = 0$. For i running from 1 to n, if $t + a_i \leq B$, then set $t = t + a_i$ and add a_i to S. For example, on the above example, the output would be $S = \{5, 9, 13\} = 27$. After this, the addition of any of the remaining elements would result in a sum larger than 35. (Note that the elements of A are considered in exactly the order in which they are given.)

For any input (A, B), let t^* denote the total value of an optimal solution, and let t be the total value produced by the above heuristic. The *performance ratio* is $\rho = t^*/t$. The *ratio bound* is the largest achievable ratio over all possible inputs.

(a) Show that this heuristic could generate arbitrarily bad solutions, in the sense that for any $\rho \geq 1$, there exists an input for which the heuristic has a performance ratio at least as large as ρ.

(b) Show how to modify the above heuristic (or devise a new heuristic of your own) that achieves a performance ratio bound of at most 2. Your heuristic should run in $O(n \log n)$ time. Present a proof of your heuristic's ratio bound.

Challenge Problem. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

Consider the following graph, called H_4. It has 16 vertices. Each vertex is labeled with a different four-element bit vector. We place an edge between two vertices if they differ in exactly one bit position. For example, the vertex labeled 1011 is adjacent to the vertices 0011, 1111, 1001, and 1010.

(a) Show that H_4 has a Hamiltonian cycle.

(b) Prove that for any $k \geq 2$, H_k has a Hamiltonian cycle. (Hint: Use induction on k.)

Practice Problems for the Midterm

The midterm will be on Tues, Oct 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. For example, we have covered some topics this year that were not covered in previous semesters. So, just because a topic is not covered here, do not assume it will not be on the exam.

Problem 0. You should expect one problem in which you will be asked to work an example of one of the algorithms we have presented in class on a short example.

Problem 1. Short answer questions.

(a) Consider the code $a = \langle 0 \rangle$, $b = \langle 01 \rangle$, $c = \langle 11 \rangle$, $d = \langle 101 \rangle$. Is this a prefix code? Explain.

(b) 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 $d[v] \geq d[u] + 2$? Is it possible that $d[u] \geq d[v] + 2$? Explain briefly.

(c) As a function of n, give an exact closed-form solution (no embedded sums) to the following summation:

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

(d) Consider the recurrence $T(n) = 4T(n/2) + n$. Using Θ-notation, give a tight asymptotic bound on $T(n)$. (E.g., $T(n)$ is $\Theta(n)$ or $\Theta(n \log n)$, etc.)

(e) Given a connected undirected graph $G = (V, E)$ in which all edges have weight 1, what is the asymptotically fastest way (that we know of) for computing a minimum spanning tree for G?

(f) What is the maximum number of edges in an undirected graph with n vertices, in which each vertex has degree at most k?

(g) You are given a connected, undirected graph $G = (V, E)$ with positive edge weights. You form another graph G' by squaring the weight of every edge of G. True or false: A spanning tree T is a minimum spanning tree of G if and only if T is a minimum spanning tree of G'. Explain briefly.

(h) You are given a connected, undirected graph $G = (V, E)$ in which each edge has a numeric edge weight, and all edge weights are distinct. Let e_1, e_2, and e_3 be the edges with smallest, second smallest, and third smallest weights among all the edges of G. Among these three edges, which *must* be in the minimum spanning tree (MST) of G and which *might* be in the MST.

Problem 2. The flaky Professor Hubert J. Farnsworth drives from College Park to Miami Florida along I-95. He starts with a full tank and can go for 100 miles on a full tank. Let $x_1 < x_2 < \ldots < x_n$ denote the locations of the various gas stations along the way, measured in miles from College Park. Present an algorithm that determines the *fewest number* of gas stations he needs to stop at to make it to Miami without running out of gas along the way. Give a short *proof of the correctness*.

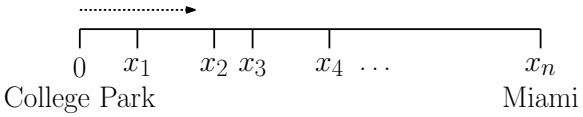


Figure 1: Example for Problem 3.

Problem 3. A pharmacist has W pills and n empty bottles. Let $\{p_1, \ldots, p_n\}$ 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? Assume that you pay only for the fraction of the bottle used. For example, if the ith bottle is half filled, you pay only $c_i/2$. (No need to prove correctness.)

Problem 4. On the planet of Smurf, the adorable, gentle Smurf folk hold a convention every four years. In case you don't know, Smurfs are all descended from a common ancestor, the venerable "Papa Smurf." 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 you are given n Smurfs, named Smurf-1 through Smurf-n. For $1 \leq i \leq n$, let t_i denote the travel cost of sending Smurf-i to the convention. You may assume the Smurfs have been numbered, so that if Smurf-i is the father of Smurf-j, then $i < j$. (Thus, Papa Smurf is Smurf-1.)

Give an algorithm to determine (1) the minimum total travel costs for all the Smurfs, subject to the above attendance requirement, and (2) output the set of Smurfs attending the convention (see Fig. 2). The inputs to your algorithm are two arrays C and t, where $C[i]$ is the list of the children of Smurf-i, and $t[i]$ is the associated travel cost.

Derive the running time of your algorithm. ($O(n)$ time is possible.) For example, in Figure 2, we give a sample example. 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 $5.2 + 7 = 12.2$.

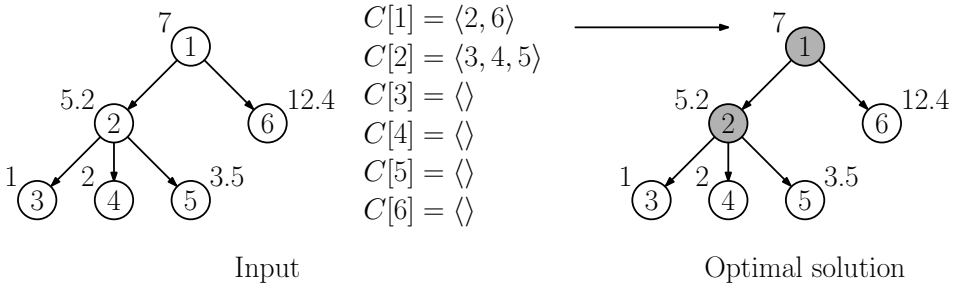


Figure 2: Example for Problem 4. Next to each node of the tree is its travel cost.

Hint: I know a few different solutions. One involves dynamic programming and another is a greedy solution. You are free to present any $O(n)$ time algorithm. In any case, be sure to show your algorithm's correctness.

Problem 5. For each part, either give a short proof of the correctness of your claim (if true) or give a counterexample (if false).

(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 6. You are given a connected undirected graph $G = (V, E)$ in which each edge's weight is either 1 or 2. Present an $O(n + m)$ time algorithm to compute a minimum spanning tree for G, where $n = |V|$ and $m = |E|$. Explain your algorithm's correctness and derive its running time. (Hint: This can be done by a variant of DFS or BFS.)

Problem 7. 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 brilliant but flaky Professor Hubert J. Farnsworth 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 $O(m \log n)$ algorithm to do this, where $n = |V|$ and $m = |E|$. Briefly justify your algorithm's correctness and derive its running time.

Problem 8. 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 $X = \langle \text{ABCBABA} \rangle$ and $Y = \langle \text{BCAABAB} \rangle$, then $Z = \langle \text{ABCAABABA} \rangle$ is an SCS of both X and Y.

Give an $O(mn)$ time 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 9. In this problem we will consider two variations of the longest common subsequence problem. You are given three sequences X, Y, and Z, where $m = |X|$, $n = |Y|$ and $|Z| = k$. You want to know whether the sequences X and Y can be shuffled together (without changing the relative order of elements within either sequence) in order to form the sequence Z. For example, if $X = \langle \text{ABC} \rangle$, $Y = \langle \text{BACA} \rangle$, and $Z = \langle \text{ABBACCA} \rangle$ the answer is "yes," ($Z = \langle \text{A}_x\text{B}_x\text{B}_y\text{A}_y\text{C}_x\text{C}_y\text{A}_y \rangle$) but if $Z = \langle \text{ABCBAAC} \rangle$ the answer is "no." You may assume that $k = m + n$, for otherwise answer is definitely "no."

(a) The eminent but flaky Professor Hubert J. Farnsworth claims that the following simple algorithm works for this problem. First, compute the LCS of X and Z, and remove these elements from Z. Let Z' be the resulting sequence. If $Y = Z'$ then the answer is yes, and otherwise, the answer is no. Is Farnsworth's algorithm correct? Either prove that it is or present a counterexample.

(b) Present an $O(nm)$ time algorithm that answers this decision problem. (Whether right or wrong, you cannot reuse Farnsworth's algorithm.)

Problem 10. You are given a weighted digraph $G = (V, E)$ with no negative weight cycles. As usual, the shortest path between two vertices is the path that minimizes the total weight of edges. Let m be the maximum, over all pairs of vertices $u, v \in V$, of the minimum number of edges in any shortest path from u to v.

(a) Describe a simple modification to the Bellman-Ford algorithm that allows it to terminate in at most $m + 1$ stages.

(b) Prove that your modified algorithm is correct (that is, on termination $d[u]$ is the same as it would be for the original Bellman-Ford algorithm), and that it terminates in at most $m + 1$ stages.

Midterm Exam

This exam is closed-book and closed-notes. You may use one 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. (15 points) Consider the alphabet {a, b, c, d} with the following probabilities:

Symbol	a	b	c	d	e
Probability	0.20	0.15	0.30	0.15	0.20

(a) Show the final tree produced by Huffman's algorithm on this input.

(b) Given your tree, what would be the encoding of the string "decade".

(c) If you encode a string of length 100, what is the expected length of the result?

Problem 2. (25 points; 5–10 points each.) Short answer questions.

(a) Consider the following ranking of preferences:

Men:	*Bob*	*Ted*
	Alice	Carol
	Carol	Alice

Women:	*Carol*	*Alice*
	Bob	Bob
	Ted	Ted

True or false: The pairing (Bob,Carol) and (Ted,Alice) is stable. Explain.

(b) As a function of n, give the (tight) asymptotic running time of the following three nested loops using Θ-notation. Briefly justify your answer.

```
for (i = 1 to n)
    for (j = 1 to n - i)
        for (k = j to i + j)
            ...something taking constant time...
```

(c) Recall that in the interval scheduling problem, you are given n requests, each having a start time s_i and finish time f_i and the objective is to schedule the maximum number of nonconflicting tasks. Which of the following greedy strategies is optimal? (List all that apply. No explanation needed.)

(i) Earliest start time first (select in increasing order of s_i)

(ii) Earliest finish time first (select in increasing order of f_i)

(iii) Latest start time first (select in decreasing order of s_i)

(iv) Latest finish time first (select in decreasing order of f_i)

(v) Shortest activity first (select in increasing order of $f_i - s_i$)

(vi) Lowest conflict first (select the activity that has the minimum number of conflicts with the remaining tasks)

Problem 3. (20 points) Given a list $A = \langle a_1, \ldots, a_n \rangle$ of positive integers, a *strong inversion* is a pair a_i and a_j such that $i < j$, $a_i > 2a_j$. (In other words, it is an inversion in which one number is more than twice as large as the other.)

Design an $O(n \log n)$ time algorithm that counts the number of strong inversions in a sequence A containing n elements. You may assume that A is presented to you as an array. (If you prefer, you may express your answer by describing just the modifications to the inversion counting algorithm given in class.) Justify your algorithm's correctness and derive its running time.

Problem 4. (25 points) Recall that in the longest common subsequence (LCS) problem the input consists of two strings $X = \langle x_1, \ldots x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$ and the the objective is to compute the longest string that is a subsequence of both X and Y.

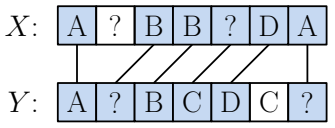
For each of the following variations, present a modification of the recursive rule. It may help to recall the following recursive rule for the *standard LCS problem*:

For $0 \le i \le m$, and $0 \le j \le n$, let $\text{lcs}(i, j)$ be the length of the longest common subsequence of $X_i = \langle x_1, \ldots, x_i \rangle$ and $Y_j = \langle y_1, \ldots, y_j \rangle$.

$$\text{lcs}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ \text{lcs}(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(\text{lcs}(i-1, j),\ \text{lcs}(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

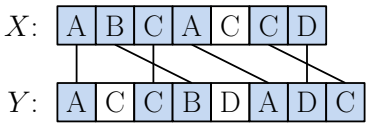
(a) (LCS with wild cards) Each of the strings X and Y may contain a special character "?", which is allowed to match *any single character* of the other string, *except* another wild-card character (see the figure below (a)).

(b) (LCS with swaps) Any two consecutive characters of either string are allowed to be swapped before matching in the LCS (see the figure below (b)).



$\text{LCS}(X, Y) = \text{ABBCDA}$

(a)



$\text{LCS}(X, Y) = \text{A}\{\text{BC}\}\text{A}\{\text{CD}\}$

(b)

In all cases, your revised rule should admit an $O(mn)$ time solution.

Problem 5. (15 points) An undirected graph $G = (V, E)$ is a *quasi-tree* if it is connected and has at most $n + 8$ edges, where $n = |V|$. Give an algorithm with running time $O(n)$ that is given a quasi-tree G with weighted edges, and returns a minimum spanning tree of G. You may assume that the edge weights are distinct. Prove that your algorithm is correct and derive its running time. (Hint: It may be simpler to consider first how to solve the problem on a connected graph that has exactly n edges.)

Practice Problems for the Final Exam

The final will be on Fri, Dec 14, 8-10am. The exam will be closed-book and closed-notes, but you will be allowed two sheets of notes (front and back of each sheet).

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. For example, we have covered some topics this year that were not covered in previous semesters. So, just because a topic is not covered here, do not assume it will not be on the exam.

Problem 0. You should expect one problem in which you will be asked to work an example of one of the algorithms we have presented in class or some NP-complete reduction we covered.

Problem 1. Short answer questions.

(a) Technically advanced aliens come to Earth and show us that some known NP-hard problem cannot be solved faster than $O(n^{100})$ time. Does this resolve the question of whether P = NP? (Explain briefly.)

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

(c) 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.

(d) Suppose that $A \leq_P B$, and there is a factor-2 approximation to problem B, then which of the following necessarily follows:

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

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

(iii) We cannot infer anything about our ability to approximate A.

(e) You are given a connected, undirected graph $G = (V, E)$ in which each edge has a numeric edge weight, and all edge weights are distinct. Let e_1, e_2, and e_3 be the edges with smallest, second smallest, and third smallest weights among all the edges of G. Among these three edges, which *must* be in the minimum spanning tree (MST) of G and which *might* be in the MST.

(f) The worst-case running time of the Ford-Fulkerson network flow algorithm is: (select any/all that apply.)

(i): $O(V^3)$ (ii): $O(V^2(E + V \log V))$ (iii): Depends on the edge capacities.

Problem 2. (Bucket redistribution) You are given a collection of n blue buckets, and n red buckets. These are denoted B_i and R_i for $0 \leq i \leq n-1$. Initially each of the blue buckets contains some number of balls and each red bucket is empty. The objective is to transfer all the balls from the blue buckets into the red buckets, subject to the following restrictions.

The input to the problem consists of two sequences of integers, $\langle b_0, b_1, \ldots, b_{n-1}\rangle$ and $\langle r_0, r_1, \ldots, r_{n-1}\rangle$. Blue bucket B_i holds b_i balls initially, and at the end, red bucket R_i should hold exactly r_i balls. The balls from blue bucket B_i may be redistributed only among the red buckets R_{i-1}, R_i, and R_{i+1} (indices taken modulo n). You may assume that $\sum_i b_i = \sum_i r_i$.

Design a polynomial time algorithm which given the lists $\langle b_0, b_1, \ldots, b_{n-1}\rangle$ and $\langle r_0, r_1, \ldots, r_{n-1}\rangle$, determines whether it is possible to redistribute the balls from the blue buckets into the red buckets according to these restrictions.

Problem 3. You are given a collection of n points $U = \{u_1, u_2, \ldots, u_n\}$ in the plane, each of which is the location of a cell-phone user. You are also given the locations of m cell-phone towers, $C = \{c_1, c_2, \ldots, c_m\}$. A cell-phone user can connect to a tower if it is within distance Δ of the tower. For the sake of fault-tolerance each cell-phone user must be connected to at least three different towers. For each tower c_i you are given the maximum number of users, m_i, that can connect to this tower.

Give a polynomial time algorithm, which determines whether it is possible to assign all the cell-phone users to towers, subject to these constraints. Prove its correctness. (You may assume you have a function that returns the distance between any two points in $O(1)$ time.)

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 (see Fig. 1). A directed *Hamiltonian path* is a path that visits every vertex in a digraph exactly once.

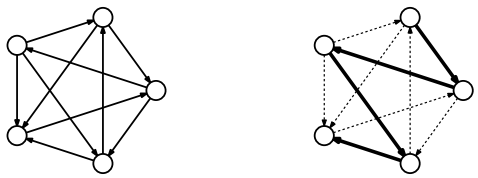


Figure 1: Hamiltonian path in a tournament.

(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. 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.

Hint: Reduction from the standard 3-coloring problem (3COL).

Problem 6. In ancient times, King Arthur had a large round table around which all the knights would sit. Unfortunately, some knights hate each other, cannot be seated next to each other. There are n knights altogether, $\{v_1, v_2, \ldots, v_n\}$, and the king has given you a list of pairs of the form $\{v_i, v_j\}$, which indicates that knights v_i and v_j hate each other.

You are asked to write a program to determine if it is possible to seat the knights about the table, called the *angry knight seating problem* (AKS). Prove that AKS is NP-complete.

Problem 7. This is a variant of the Clique problem. You are given an undirected graph $G = (V, E)$. We say that a subset $V' \subseteq V$ is a *pseudo-clique* if for any two vertices $u, v \in V'$ the distance from u to v in G is at most two (that is, either there is an edge between them or they share a common neighbor). The *Pseudo-Clique Problem* (PC) is, given an undirected graph $G = (V, E)$ and an integer k, does G have a pseudo-clique of size k. Prove that PC is NP-complete. (Hint: Reduction from Clique.)

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

Problem 9. Recall the following problem, called the *Interval Scheduling Problem*. We are given a set $S = \{1, 2, \ldots, n\}$ of n *activity requests*, each of which has a given start and finish time, $[s_i, f_i]$. The objective is to compute the maximum number of activities whose corresponding intervals do not overlap. In class we presented an optimal algorithm greedy algorithm. We will consider some alternatives here.

(a) **Earliest Activity First** (EAF): Schedule the activity with the earliest start time. Remove all activities that overlap it. Repeat until no more activities remain.

Give an example to show that, not only is EAF not optimal, but it may be arbitrarily bad, in the sense that its approximation ratio may be arbitrarily high.

(b) **Shortest Activity First** (SAF): Schedule the activity with the smallest duration $(f_i - s_i)$. Remove all activities that overlap it. Repeat until no more activities remain. Give an example to show that SAF is not optimal.

(c) Prove that SAF has an approximation ratio of 2, that is, it schedules at least half as many activities as the optimal 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. (10 points)

(a) Recall the reduction of 3SAT to Independent Set (IS). Show the result of the reduction on the following boolean formula. Be sure to give both the graph G and the integer k that result.

$$F = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3).$$

(b) Show (e.g., by circling the vertices) an independent set corresponding to the satisfying assignment $x_1 = 1$, $x_2 = 1$, $x_3 = 0$.

Problem 2. (20 points; 3–7 points each.) Short answer questions. (Unless otherwise specified, explanations are not required.)

(a) Yes or No: Suppose an algorithm runs in $O(n \cdot 2^{(2\log_2 n)})$ time. Is this a *polynomial time* algorithm?

(b) You are given a digraph G that has edge weights with the values of either $+1$ or -1. It has *no* negative cost cycles. Which of the following can be used to compute shortest paths in G from a single source vertex? (List all that apply): *breadth-first search*, *Dijkstra's algorithm*, *Bellman-Ford algorithm*.

(c) Explain why Kruskal's MST algorithm made use of a Union-Find data structure.

(d) True or False: One of the principal advantages of a *memoized implementation* of a dynamic programming algorithm is that no table is needed to store the results of the intermediate subproblems.

(e) Huffman's algorithm is an example of (select one): *greedy algorithm*, *divide-and-conquer*, *dynamic programming*.

Problem 3. (15 points) A shipping company wants to ship n objects of weights $\{w_1, \ldots, w_n\}$. Each weight is a positive integer. The company wants to partition these objects between two different ships, so that the total weight of the two ships is as similar as possible. In particular, if W_1 is the total weight of objects on Ship 1, and W_2 is the total weight on Ship 2, then the objective is to minimize the *weight ratio*,

$$\frac{\max(W_1, W_2)}{\min(W_1, W_2)}.$$

Observe that this ratio is never smaller than 1, and it equals 1 if and only if the two ships are carrying identical total weights.

For example, suppose the inputs are $w_1 = 40$, $w_2 = 70$, $w_3 = 20$, $w_4 = 30$, $w_5 = 60$, and $w_6 = 50$. If we partition the elements as Ship-1 $= \{2, 5\}$ and Ship-2 $= \{1, 3, 4, 6\}$, then the total weights are $70 + 60 = 130$ and $40 + 20 + 30 + 50 = 140$. The final weight ratio is $140/130 \approx 1.077$.

This is called the *Partition Problem*. Present an efficient algorithm, which given the weights $\{w_1, \ldots, w_n\}$, computes the optimum weight ratio. You can express your running time as a function of both n and the total weight $W = \sum_{i=1}^{n} w_i$.

(Hints: Use Dynamic Programming. You are not required to give the entire DP algorithm, just a recursive formulation. You need only compute the optimum weight ratio, not the actual partition. Justify your algorithm's correctness and derive its running time. Note that $O(n \cdot W)$ time is possible. It suffices to focus on computing the total weight carried by just one of the ships, since the other must carry all the remaining weight.)

Problem 4. (15 points) As part of an international exchange program, the university would like to pair up local students with international students. The local students are $\{u_1, \ldots, u_m\}$, and the international students are $\{v_1, \ldots, v_n\}$. There are many more local students than international students. For each pair u_i and v_j, the university knows whether their schedules are *compatible*, and if so, they can be paired.

Present a polynomial time algorithm that determines whether it is possible to generate a pairing such that:

- Each local student is paired with *exactly* 1 international student.
- Each international student is paired with at least 1 but no more than 5 local students.
- Students are paired only if their schedules are compatible.

A sample input and possible output is shown in Fig. 1. (You may use any algorithm given in class to help solve this.)

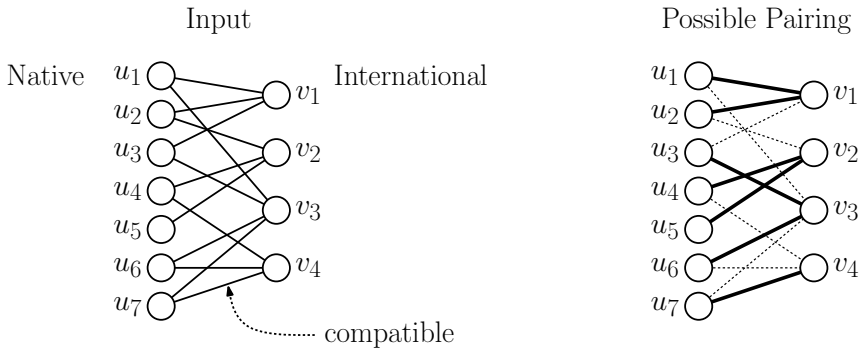


Figure 1: Problem 4: Pairing local and international students.

Problem 5. (20 points) In the High-Degree Independent Set (HDIS) problem, you are given a graph $G = (V, E)$ and an integer k, and you want to know whether there exists an independent set V' in G of size k such that each vertex of V' is of degree at least k. (For example, the graph in Fig. 2 has an HDIS for $k = 3$, shown as the shaded vertices. Note that it does *not*

have an HDIS for $k = 4$. Although adding the topmost vertex would still yield an independent set, this vertex does not have degree at least four.)

(a) Show that HDIS is in NP.

(b) Show that HDIS is NP-hard. (Hint: Use standard independent set (IS).)

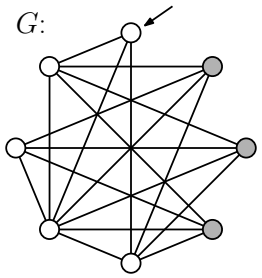


Figure 2: Problem 5: High-degree independent set.

Problem 6. (20 points) Recall the Partition Problem (Problem (3) above). Consider the following heuristic for this problem. First, sort the objects by *decreasing order* of their weights. Initially both ships have 0 weight. Repeatedly, take the next object from the sorted list, and put it on the ship whose total weight is the smaller of the two. (If they are tied, put it on Ship-1.) Update the weight of this ship.

For example, suppose that after sorting, the weights are $w_1 = 70$, $w_2 = 60$, $w_3 = 50$, $w_4 = 40$, $w_5 = 30$, and $w_6 = 20$. The heuristic would put object 1 on Ship-1, object 2 on Ship-2. Since Ship-2 is now lighter, it would put object 3 on Ship-2, and so on. The final assignment would be Ship-1 $= \{1, 4, 5\}$ and Ship-2 $= \{2, 3, 6\}$. The total weights are $70 + 40 + 30 = 140$ and $60 + 50 + 20 = 130$. The final weight ratio is $140/130 \approx 1.077$ (which is actually optimal).

(a) (5 pts) Briefly explain how to implement this in $O(n \log n)$ time.

(b) (5 pts) Give a small example that shows that this is not optimal.

(c) (10 pts) Prove that for any input, this heuristic achieves a performance ratio of at most 2. That is, the weight ratio produced by this heuristic is at most twice as large as the weight ratio of the optimal solution.