Homework 1: Algorithm Design Basics

Handed out Thu, Sep 12. Due at the start of class Tue, Sep 24. Late homeworks are not accepted, but you may drop your lowest homework score.

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 and the material discussed in class. Be sure that your description or 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 me.

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. In our statement of the Gale-Shapley algorithm, we assumed that each person honestly lists his/her preferences individually, without coordinating with the other participants. The purpose of this problem is to consider whether it is possible to do better by working together as a group.

(a) Suppose that the men get together and decide amongst themselves the final pairing that they want as a group. By carefully arranging their individual preference lists, can the men force the GS algorithm to produce the pairing that they want, irrespective of what the women do? (If so, describe how the men should set up their preference lists, and explain why the GS algorithm will produce the desired result. If not, explain how the women can spoil their plans, no matter how cleverly the men try to succeed.)

(b) Answer (a) again, but this time with the genders reversed. That is, can the women join forces to cause the GS algorithm to produce whatever pairing they desire as a group?

(c) Men are dumb and women are clever. (This is well known.) Suppose that all the men use the same preference list, and the women obtain a copy of this shared list. Using this information, can the women coordinate as a group to force the GS algorithm to produce whatever pairing they desire as a group? (As in part (a), either present the women's scheme and explain why it works, or else prove that it cannot be done.)

Problem 2. You are given a set of real numbers $A = \{a_1, \ldots, a_n\}$, and an integer k, where $1 \leq k \leq n$. Present an algorithm that computes the smallest interval $[x, y]$, such that at least k elements of A lie within this interval. (The *size* of interval $[x, y]$ is $y - x$.) For example, if

$$A = \{3.1, 6.2, 10, 5.8, 1.0, 2.2, 6.0, 7.3\} \qquad \text{and} \qquad k = 4,$$

then the smallest interval containing k elements is $[5.8, 7.3]$ of size 1.5, which contains the elements $\{5.8, 6.0, 6.2, 7.3\}$.

Your algorithm should run in $O(n \log n)$ time. Justify your algorithm's correctness, and derive its running time.

Problem 3. Like many of the algorithms we will describe this semester, our presentation of the Gale-Shapley (GS) algorithm was very high-level. As competent programmers, I will usually assume that you can add the necessary (hopefully easy) implementation details. For the sake of concreteness, let's consider how we would do that for this algorithm.

(a) Consider the pseudo-code below for the GS algorithm. Describe what data structures (lists, arrays, stack, queues, hash tables, etc.) you would use for implementing the code below.

(b) Using the data structures from part (a), explain how to implement that GS algorithm so that it runs in $O(n^2)$ time, where n is the number of men and the number of women in the system.

```
1:  Initially all men and all women are unengaged
2:  while (there is an unengaged man who hasn't yet proposed to every woman) {
3:      Let m be any such man
4:      Let w be the highest woman on his list to whom he has not yet proposed
5:      if (w is unengaged) then (m, w) are now engaged
6:      else {
7:          Let m' be the man w is engaged to currently
8:          if (w prefers m to m') {
9:              Break the engagement (m', w)
10:             Create the new engagement (m, w)
11:             (m' is now unengaged)
12:         }
13:     }
14: }
```

Problem 4. For each pair of functions $f(n)$ and $g(n)$ below, indicate whether $f(n)$ is Θ, o, or ω of $g(n)$ (meaning that $f(n)$ is asymptotically equal, strictly smaller, or strictly larger than $g(n)$, respectively). The notation lg means logarithm base 2. Briefly explain your answers (e.g., by reducing each to its simplest asymptotic form).

	$f(n)$	$g(n)$
(a)	$10n^3 + n \lg n$	$n^3 + n \lg^2 n$
(b)	2^n	$3^{(n/2)}$
(c)	$\lg(2^n)$	$\lg(3^n)$
(d)	$\lg \sqrt{n}$	$\sqrt{\lg n}$
(e)	$n^{\lg 4}$	$2^{\lg n}$
(f)	$n(\lg \lg n)^2$	$n\sqrt{n}$

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.

Recall that the input to the stable marriage problem consists of a set X of n men and Y of n women, where each man and each woman has an n-element list of preferences. For any given input, there are generally many possible stable matchings. Given $x \in X$, define $V(x)$ to be the subset of women with whom x is paired with in some stable matching, and for $y \in Y$, define $V(y)$ to be the subset of men with whom y is paired with in some stable matching. (We call these the *viable* sets of mates.) Prove that in the Gale-Shapley algorithm, a woman will *never* reject a proposal from one of her viable mates. (Such a rejection may occur either when a viable man first proposes and she turns it down or when she is already engaged to a viable man, but then she later breaks off the engagement.)

Homework 2: Greedy Algorithms

Handed out Thu, Oct 3. Due at the start of class Thu, Oct 17. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. Let $\{f_1, \ldots, f_n\}$ be a collection of n files to be stored on a tape. File f_i requires s_i bytes of storage. The tape is long enough to store all the files. We are told that the probability of accessing file f_i is p_i, where $0 \leq p_i \leq 1$, and $\sum_{i=1}^{n} p_i = 1$ (see Fig. 1(a)). We rewind the tape before each access, and so the time to access any file is proportional to the distance from the front of the tape to the end of the file.

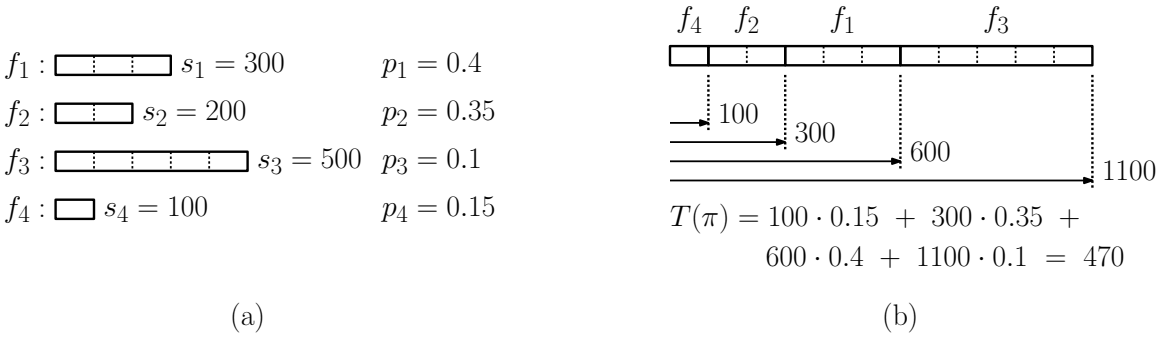


Figure 1: Problem 2.

A *layout* of files on the tape is given by a permutation $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ of the numbers $\{1, \ldots, n\}$. (For example, Fig. 1(b) shows the layout permutation $\pi = \langle 4, 2, 1, 3 \rangle$.) Given a layout π, the *individual cost* of accessing the ith file on the tape is the product of its access probability and the distance from the start of the tape to the end of the file, that is, $C_i(\pi) = p_{\pi_i} \cdot \left(\sum_{j=1}^{i} s_{\pi_j} \right)$. The *total cost* of a layout π is the sum of the individual costs, $T(\pi) = \sum_{i=1}^{n} C_i(\pi)$.

(a) Present a (short) counterexample so show that laying out the files on the tape in increasing order of size (s_i) is not optimal.

(b) Present a (short) counterexample so show that laying out the files on the tape in decreasing order of access probability (p_i) is not optimal.

(c) Present an algorithm, which given s_i's and p_i's, determines a layout π of minimum total cost. Prove your algorithm's correctness and derive its running time. (Hint: Use a greedy approach. See the correctness proofs given in class for the greedy scheduling algorithms for examples of what I am expecting.)

Problem 2. You are working for a delivery company, where you must drive trucks from a central dispatch center to various locations. The road network is modeled by a weighted digraph $G = (V, E)$, where the weight of each edge (u, v) is the distance location u to v in miles. Recently, the department of transportation has erected toll booths in various locations. This is modeled by associating a nonnegative toll value $t(u)$ with each vertex u of the digraph. Based on your level of gas consumption, you determine that you are willing to pay $\$X$ dollars in tolls in order to save $20 \cdot X$ miles. For example, if you have three possible paths, of lengths 100, 150, and 200 miles, involving \$10, \$5, and \$3 in tolls, respectively, you would prefer the second option (150 miles and \$5 in tolls).

Devise an efficient algorithm that computes the minimum cost path from the dispatch center s to every node of the digraph. As always, justify your algorithm's correctness and derive its running time. (Hint: This can be solved as fast as Dijkstra's algorithm.)

Problem 3. You are given a connected, undirected graph $G = (V, E)$, where each edge $(u, v) \in W$ is labeled with a nonnegative weight $w(u, v)$. Suppose that we run Prim's MST algorithm starting at some source node $s \in V$. Let T be the resulting tree. You would like to know whether T is also the shortest path tree for the single-source node s. Of course, you could run Dijkstra's algorithm and compare the two trees, but this would take additional $O(n \log n + m)$ time, where $n = |V|$ and $m = |E|$.

Show that (once computed) it is possible to determine whether T is the shortest path tree in only $O(n + m)$ time. Your algorithm takes as input the graph G, a source vertex s, and (as in the output of Prim's algorithm) the tree T is represented by the values pred$[v]$, for each $v \in V$. Your algorithm need only answer "yes" or "no". To avoid messy situations involving ties, you may assume that all edge weights and shortest path lengths are distinct. As always, justify your algorithm's correctness and derive its running time.

Problem 4. You are building the navigation system for a computer game. In this game, a player is located on an $N \times N$ grid and wishes to move from some cell s to another cell t. (Let's assume that cells are identified by their $[i, j]$ row-column indices.) On the grid, certain $[i, j]$ pairs are *forbidden*, meaning that the player is not allowed to pass through or land on these cells (see Fig. 2(a)). The player moves by *teleporting* from the current cell to any "unforbidden" cell that is horizontal or vertical from this cell, provided that there is no forbidden cell between.

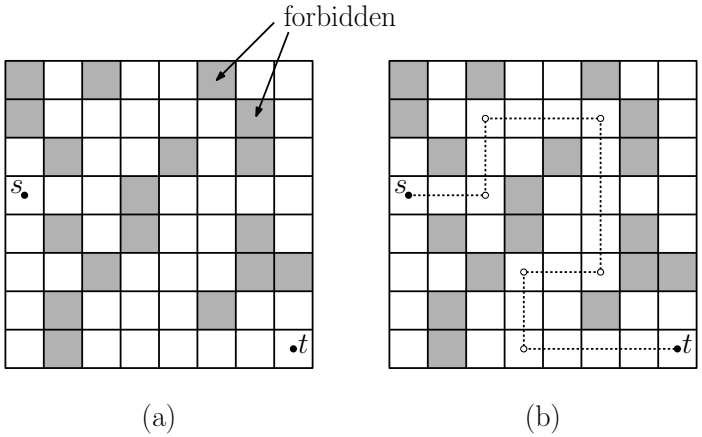


Figure 2: Problem 4.

Your problem is to devise an algorithm that computes the minimum number of moves to get from s to t. (For example, in Fig. 2(b) we show that it is possible to get from s to t using only 7 moves.) By the nature of teleportation, the distance traveled on each move does not matter, only the number of moves. The input to your program is the value N and a list of the $[i, j]$ pairs of the forbidden cells. The output is the number of moves to reach t. (It is not necessary to output the path.)

Let $n = N^2$ denote the total number of cells on the grid. Justify the correctness of your algorithm and derive its running time as a function of n. The credit you will receive will depend on your algorithm's correctness and running time. (Hint: $O(n^2)$ is pretty poor, but better than nothing. $O(n \log n)$ is good. I believe that $O(n)$ 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.

Challenge Problem 1: You are given an connected, undirected graph $G = (V, E)$, where $n = |V|$ and $m = |E|$. Each edge of G is labeled either as being *red* or *blue*. You are also given an integer k, where $1 \leq k \leq n - 1$. A wealthy (republican?) donor wants you to build a spanning tree for G, but this donor requires that *at least* k edge of your spanning tree be red edges.

(a) (Easier) Show how to efficiently compute spanning tree for G where *at least* k edges are red (and hence at most $n - 1 - k$ edges are blue), or output that no such spanning tree exists.

(b) (Harder) Show how to efficiently compute spanning tree for G where *exactly* k edges are red (and hence exactly $n - 1 - k$ edges are blue), or output that no such spanning tree exists.

Note there are no edge weights, so there are no costs to be minimized here.

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 unpredictably 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. Every one of the 52 cards must be in exactly one of the two stacks.

Here is the challenge: You are blindfolded, and cannot tell which cards are face-up or face-down by touch or any other means.

Homework 3: Dynamic Programming and Network Flows

Handed out Thu, Nov 14. Due at the start of class Tue, Nov 26. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. 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.

Given the initial configuration, (A, B, C), and with the assumptions that Jen plays first and both players play as well as possible, determine which of the two players can force a win. (Since there is no element of chance, and the game is finite in length, one of the two can always force a win.)

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), respectively) leaving Ben without a move.

(a) Derive a (recursive) dynamic programming rule to determine the winner, given the initial board (A, B, C). (Be sure to include a description of the basis cases.) Justify the correctness of your formulation. (For this part I do not want a full algorithm, just the recursive rule.)

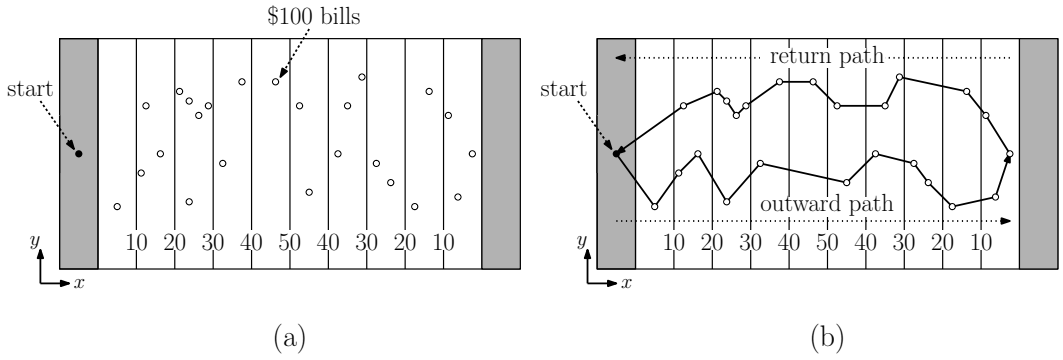
(b) Present an implementation of recursive rule of part (a). (You may use memoization or the bottom-up method.) Express your running time as a function of A, B, and C.

Problem 2. A popular game show has come to campus, and you have been selected to be one of the participants (lucky you!). The host of the show explains how the game works. You are given a starting point in the end-zone of the football field in Byrd Stadium, and a number of \$100 bills have been placed throughout the field. You are to dress up like a chicken and run and pick up all the bills as fast as you can and then return to the starting point. Your running speed is constant, and if you take too long, you will be dropped into a vat of boiling oil. Therefore, you want to compute a path that hits all the bills and is as short as possible.

There is one additional constraint. Your path must consist of two parts, an *outward path* where you run only to the right (x-coordinates increasing) from the starting end-zone, and an *return path* where you run only to the left (x-coordinates decreasing). (See Fig. 1.)

The input consists of a set of points $P = \{p_1, \ldots, p_n\}$ where the bills are and a start point p_0. Each point is given by its (x, y)-coordinates, $p_i = (x_i, y_i)$. You may assume that the points are sorted from left to right (that is, by increasing x-coordinates). You may assume you have access to a function $\text{dist}(p_i, p_j)$, which computes the distance between any two points.

Give an efficient algorithm that computes the length of the shortest path that hits all the bills and has the desired outward-return structure. (Hint: Use DP, working from left to right. Build both paths, outward and return, simultaneously. $O(n^2)$ time is possible.)

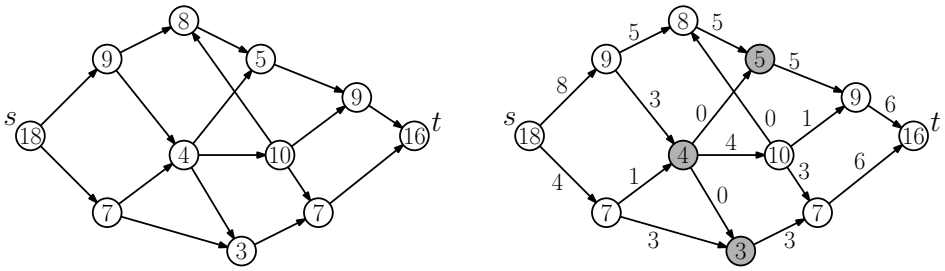


(a) (b)

Figure 1: Problem 2.

Problem 3. In this problem we will consider network flows involving networks with node capacities, rather than edge capacities.

(a) Suppose you are given a directed s-t network $G = (V, E)$, in which every node $u \in V$ is associated with a nonnegative capacity $c(u)$. We call this a *node-capacitated network*. A *flow* in G is defined the same as for a standard network except the capacity constraint applies to the flow into and the flow out of each node (including s and t), rather than to the edges. (An example of such a network is shown in Fig. 2(a) and a possible flow is shown in Fig. 2(b).)



(a) (b)

Figure 2: Problem 3. The shaded nodes in (b) form a cut.

Explain how to modify any node-capacitated network G into an equivalent edge-capacitated network G' so that, computing the maximum flow in G' (e.g., by any standard max-flow algorithm) yields the maximum flow in G.

(b) Define a *cut* in a node-capacitated s-t network to be a subset of of nodes $U \subseteq V \setminus \{s, t\}$, such that every path from s to t passes through at least one node of U (see Fig. 2(b)). Define the *capacity* of U to be the sum of the capacities of the nodes of U. Prove the equivalent of the Max-Flow/Min-Cut Theorem for node-capacitated networks, namely that the maximum value of any flow in G is equal to the minimum capacity of any cut in G. (Hint: You can either do this by modifying the proof given in class, or you can appeal to the standard Max-Flow/Min-Cut Theorem applied to your construction from (a).)

Problem 4. The FBI has received a tip that a band of diabolical crooks will be stealing valuable merchandise from a top-secret government facility. (It is the secrete warehouse where Chris Christie's massive doughnut repository is located.) You have been asked by the FBI to help assist them set up roadblocks

at various intersections in order to catch the crooks. The FBI has provided you with a map of the city represented as a directed graph $G = (V, E)$, where each node is an intersection and each edge is a road (see Fig. 3(a)). They have told you that the secret facility where the theft took place is located at node p, and the crooks hideout is located at node q. Your task is to determine the locations of a minimum number of roadblocks such that no matter what route the crooks take from p to q, they must pass through at least one of your roadblocks (see Fig. 3(b)). Note that you cannot place a roadblock either at p or q. (That would be too easy.)

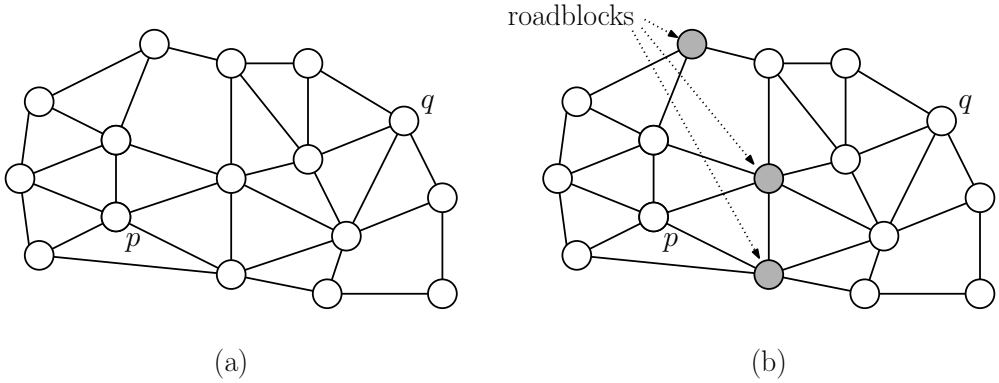


Figure 3: Problem 4.

(a) Present an efficient algorithm to compute these roadblocks. Justify your algorithm's correctness. Explain your reduction (including how your network is constructed, what the source and sink nodes are, what the capacities are, and how the output of the flow algorithm is interpreted.) Let $T(n, m)$ denote the time to compute the maximum flow on a network with n nodes and m edges. As a function of n, m, and $T(n, m)$, what is the running time of your algorithm?

(b) The FBI has received an update that the theft may take place at any one of a number of nodes $P = \{p_1, \ldots, p_k\}$, and the hideout may be at any one of the nodes $Q = \{q_1, \ldots, q_\ell\}$. You may assume that P and Q are disjoint. (In this case you can place roadblocks at the elements of P and/or Q, but if k and ℓ are large, this may not be optimal.)

Explain how to modify your solution to part (a) to deal with this new complication. In particular, you wish to compute a minimum number of roadblocks so that any path from any $p_i \in P$ to any $q_j \in Q$ must pass through at least one of your roadblocks. As a function of k, ℓ, n, m, and $T(n, m)$, what is the running time of your modified algorithm?

(Hint: The results of Problem 3 may be of use.)

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.

Challenge Problem 1. You are given an array of positive integers $A[1..n]$. Each entry of A differs from its successor by at most 1, that is $|A[i] - A[i + 1]| \leq 1$. A *upward run* of length k is a sequence of $k + 1$ consecutive elements of A that are strictly increasing in value. (The length of the run is one less than the number of elements.) A *downward run* is defined similarly, but the values are strictly decreasing. A *V-shape* of length k consists of a downward run of some length k_1 followed immediately by an upward run of some length k_2, where $k_1 + k_2 = k$ (see Fig. 4).

Present an algorithm which given A determines whether A contains a V-shape of total length at least $n/4$. (You may not make any assumptions about the lengths of the upward and downward runs. For

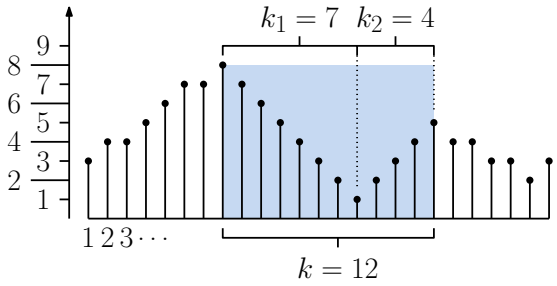


Figure 4: Challenge Problem 1, showing a V-shape of length 12. The height of each segment is the value of $A[i]$.

example, it might be that $k_1 = k_2 = k/2$, or it might be that $k_1 = 0$ and $k_2 = k$, or all possibilities in between.) Your algorithm should run in $O(\log n)$ time.

(Hint: Begin by showing that given any index i, it is possible to find the longest upward run containing the element $A[i]$ in $O(\log n)$ time.)

Challenge Problem 2. This is not an algorithms problem, but I found it interesting.

You and your roommate are contestants in a game of wits. A guy dressed up like the devil gives each of you a card with a positive integer written on it. Each of you cannot see the other person's card, but he tells you that the difference in the two numbers is 1. For example, if your number is "53", then you know that your roommate may have the number "52"or "54", but you don't know which. Otherwise, all you know about the possible numbers is that they must be 1 or larger.

This devilish fellow tells you that if either of you can guess the number on your roommate's card, you will receive a "shiney fiddle made of gold" (proving that the devil listened to country rock music from the 1970's). Otherwise, you and your roommate will have to pay the devil's $2.50 parking bill. You and your roommate are pretty smart, so you take the devil's challenge. The devil starts his game:

- The devil asks you whether you know the number on your roommate's card. After thinking, you answer "no".
- The devil then asks your roommate whether he/she knows your number. After thinking, your roommate answers "no".
- The devil is nice enough to give you another chance. After thinking, you again say "no".
- The devil gives your roommate another chance. After thinking, your roommate answers "no".

At this point, the devil gives up in disgust, and asks you two to cough up the $2.50. Suddenly, you exclaim, "I know the number on my roommate's card!" Your roommate says the same, and you both get your golden fiddles.

Explain how each of you determined the number on your card. (Hint: There is no trick. Just simple logic, but this works only because the devil was foolish enough to give you a particularly nice pair of numbers.)

Homework 4: Network Flow, NP-Completeness, and Approximation

Handed out Tue, Dec 3. Due at the start of class Thu, Dec 12. Late homeworks are not accepted, but you may drop your lowest homework score.

Problem 1. 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.)

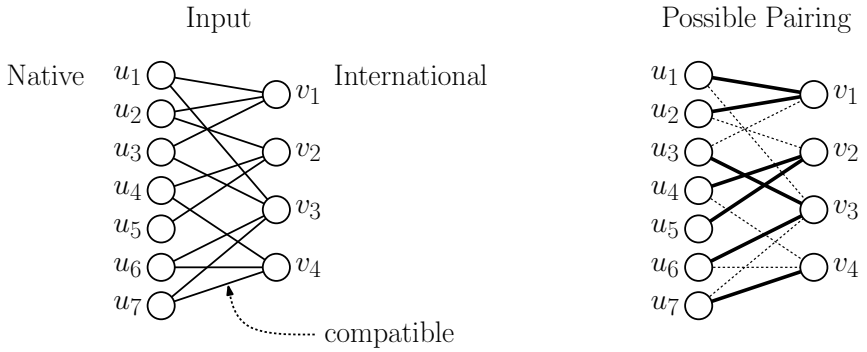


Figure 1: Problem 1: Pairing local and international students.

Problem 2. Highly intelligent aliens from another world come to Earth and tell us (1) that the 3-Coloring problem (which is NP-complete) is solvable in $O(n^9)$ time, and (2) there is no algorithm for 3-Coloring that runs faster than $\Omega(n^7)$ time in the worst case. (Here n denote the number of nodes in the graph.)

For each of the following assertions, indicate whether it follows from the information the aliens have given us. Also, provide a short explanation in each case.

(i) All NP-complete problems are solvable in polynomial time.

(ii) All problems in NP, even those that are not NP-complete, are solvable in polynomial time.

(iii) All NP-hard problems are solvable in polynomial time.

(iv) All NP-complete problems are solvable in $O(n^9)$ time.

(v) No NP-complete problem can be solved faster than $\Omega(n^7)$ time in the worst case.

(vi) There is no algorithm for the 4-Coloring problem that runs in time faster than $\Omega(n^7)$ time.

Problem 3. In the High-Degree Independent Set (HDIS) problem, you are given an undirected 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: Reduction from the standard independent set problem (IS).)

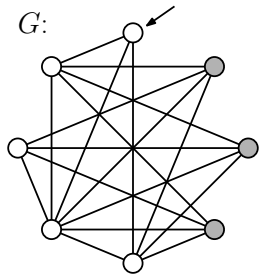


Figure 2: Problem 3: High-degree independent set.

Problem 4. Recall that the TSP (optimization) problem is: Given a complete undirected graph $G = (V, E)$ with weighted edges, find the cycle of minimum total cost that visits every node exactly once. Recall that this problem is NP-complete, with the reduction from the Hamiltonian Cycle problem (given an undirected graph $G = (V, E)$, does there exist a simple cycle that visits all the nodes). In class we showed that if the edge weights satisfy the triangle inequality, then there exists a factor-2 approximation.

The goal of this problem is to explore the approximability of TSP if the graph's edge weights *do not* satisfy the triangle inequality.

(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 four vertices.)

(b) Suppose that for some constant $c > 1$, there existed a polynomial time, factor-c approximation algorithm for TSP for graphs with arbitrary edge weights (that is, not satisfying

the triangle inequality). Show that this is very unlikely, by proving that this procedure could be applied to solve the Hamiltonian Cycle problem in polynomial 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.

It is a fact that all people love to see pumpkins smash (except perhaps pumpkin farmers). You have been assigned to a Presidential Gold-Ribbon Team to answer the following question: What is the lowest floor on the Empire State Building, such that if a pumpkin is dropped from this floor it will smash? (Let us assume that all pumpkins smash from the same floor, and if a pumpkin smashes when dropped from floor k, it will smash from all higher floors.)

Your team investigates this problem by dropping pumpkins from various heights and observing the results. Suppose that the Empire State Building has n floors (and n is potentially *very* large). Due to budget sequestration, your team is asked to use the smallest number of pumpkins possible to solve this problem. As the one computer scientist on the team, you know, for example, that the problem can be solved using $O(\log n)$ pumpkins and binary search.

(a) Suppose that the lowest floor from which a pumpkin smashes is k. Show that the problem can be solved using only $O(\log k)$ pumpkins. (In fact, I think that $\lceil \log_2 k \rceil$ suffices.)

(b) If you had only one pumpkin to work with, you could still solve the problem, but it would take multiple droppings of this pumpkin. You would drop it from the floors 1, 2, ..., until you saw it smash. This would involve k droppings, but it is the best you can hope for, since once the pumpkin is smashed, you are out of pumpkins.

Suppose however that you had *two* pumpkins to experiment with. As a function of n (or k if possible), what is the minimum number of droppings to answer the problem? (Hint: It is asymptotically smaller than n but larger than $\log n$).

(c) Generalize your solution for (b) to any constant c number of pumpkins.

Practice Problems for the Midterm

The midterm will be on **Thu, Oct 31**. 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 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) 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)

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

(f) 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.

(g) 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 eminent but 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 (see Fig. 1). 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*.

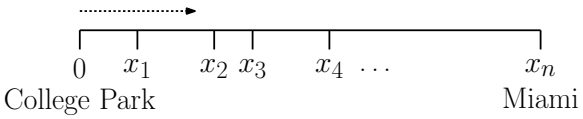


Figure 1: Example for Problem 2.

Problem 3. A pharmacist has W pills and n empty bottles. Let b_i denote the capacity of bottle i, that is, the number of pills it can hold. Let v_i denote the cost of purchasing bottle i. The objective is find the least expensive combination of bottles into which to place all W pills.

Describe a greedy algorithm, which given the number of pills W, the bottle capacities b_i, and the bottle costs v_i, determines the most inexpensive set of bottles needed to store all the pills. Assume that you pay only for the *fraction* of the bottle that is used. For example, if the ith bottle is half filled with pills, you pay only $v_i/2$. (This assumption is very important.) Prove the correctness of your solution.

Problem 4. 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.)

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 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 9. 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 objective is to compute the longest string that is a subsequence of both X and Y. For each of the following variations, present a short DP formulation. (It suffices to provide the recursive rule, similar to what we did with the standard LCS problem.)

(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)).

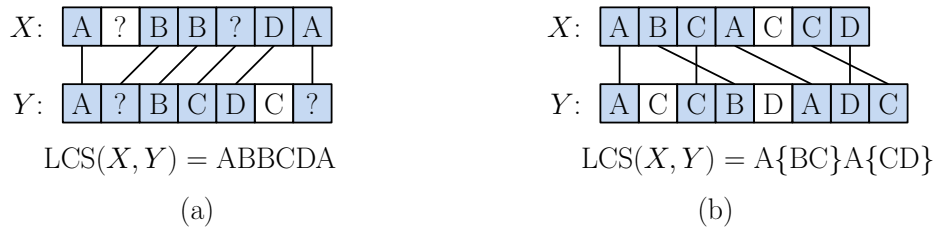


Figure 2: Problem 9.

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

Problem 10. Recall that a *bipartite graph* is an undirected graph G whose vertex set is partitioned into two sets $U = \{u_1, u_2, \ldots, u_m\}$ and $V = \{v_1, v_2, \ldots, v_n\}$, such that all edges have one endpoint in U and one endpoint in V. Two edges (u_i, v_j) and $(u_{i'}, v_{j'})$ are said to *cross* if either $i < i'$ and $j > j'$ or if $i > i'$ and $j < j'$. A *noncrossing subset* is subset of edges in G in which no two edges cross one another (see Fig. 3).

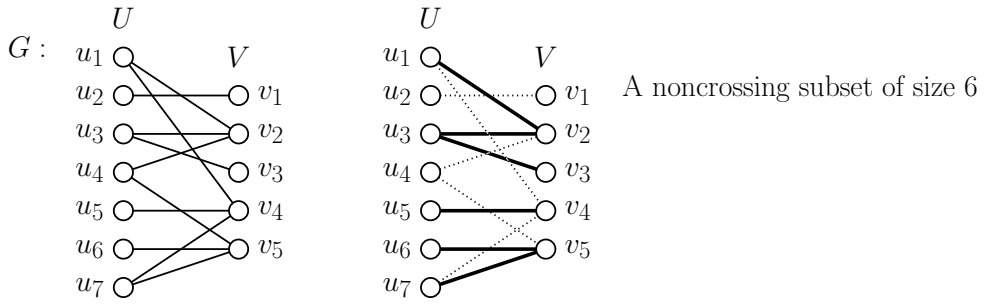


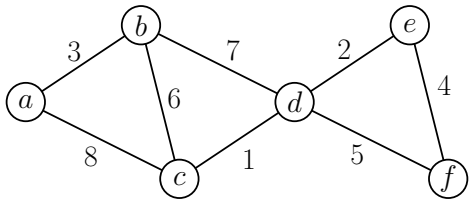
Figure 3: Problem 10.

Give a dynamic programming algorithm, which given a bipartite graph $G = (U, V, E)$, computes the size of the *maximum noncrossing subset* for G. It is sufficient to give just the recursive rule. (Hint: The algorithm is structurally similar to the LCS algorithm.)

Midterm Exam: Halloween Edition

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. (10 points) A friendly zombie presents you with the following graph:



He agrees not to eat you if you can do the following:

(a) List the edges of the minimum spanning tree *in the order* that they are added by *Kruskal's algorithm*. (List *only* the edges that are in the MST.) You may list edges either by their weight (e.g., "7") or by their endpoints (e.g., "(b, d)").

(b) Assuming that 'a' is the start vertex, list the edges of the minimum spanning tree *in the order* that they are added by *Prim's algorithm*. (List *only* the edges that are in the MST.)

Problem 2. (25 points; 3–6 points each.) Short answer questions. Explanations are not required, but may be given for partial credit.

(a) Suppose that in the Gale-Shapley algorithm, a *man's* proposal has just been accepted. **True or false:** He is guaranteed to remain engaged (to this person or someone else) for the remainder of the algorithm's execution.

(b) Suppose that in the Gale-Shapley algorithm, a *woman* has just accepted a proposal. **True or false:** She is guaranteed to remain engaged (to this person or someone else) for the remainder of the algorithm's execution.

(c) As a function of n, what is the asymptotic running time of the following function? (Express your running time using Θ notation.)

```
void scareMe(int n) {
  i = n;
  while (i > 0) {
    for (j = 1 to i) print("boo!\n");
    i = i/2;
  }
}
```

(d) Dijkstra's algorithm is run on a graph that has some *negative weight edges*. Which of the following statements it true? (Select all that apply.)

(i) The algorithm may return an incorrect result.

(ii) The algorithm is guaranteed to return an incorrect result.

(iii) If the graph has a negative cost cycle, the algorithm may loop infinitely.

(iv) The recent dead will come back to life and make a general nuisance of themselves.

(e) Recall that a sorting algorithm is *stable* if two equal elements remain in the same relative order after the sorting process is finished. **True or false:** Mergesort is stable.

(f) Let X be a string of length m, and let Y be a string of length n. Suppose that the length of their longest common subsequence is k. As a function of m, n, and k, what (if anything) can we say about the length of their shortest common super-sequence? (Recall that the *shortest common super-sequence* is the shortest string Z such that X and Y are both subsequences of Z.)

Problem 3. (20 points) A trick-or-treater breaks into candy store where there are bags of candy. He has a knapsack that can hold W pounds of candy. Each bag of candy has an associated weight w_i and an associated value v_i, both positive real numbers. Two bags of the same weight can have very different values. (Everyone knows that a pound of Reese's cups is worth a heck of lot more than a pound of Milky Way bars.) The trick-or-treater wants to maximize the total value, subject to the condition that the total weight of all the bags does not exceed W.

(a) (15 points) Suppose that it is possible to steal any *fraction* of a bag. If you take only $1/3$ of the ith bag, the weight is $w_i/3$ and the value is $v_i/3$.

For example, suppose that $W = 50$. Consider the following weights and values.

Bag	1	2	3	4	5
Weight	20	30	40	10	50
Value	\$15	\$75	\$10	\$90	\$100

By taking bags 2, 4, and one fifth of bag 5, the total weight is $30 + 10 + (50/5) = 50$ and the total value is $\$75 + \$90 + (\$100/5) = \$185$.

Present an algorithm which, given the knapsack capacity W, a sequence of n bag weights $\langle w_1, \ldots, w_n \rangle$, and the corresponding values $\langle v_1, \ldots, v_n \rangle$ determines the maximum total value. Briefly justify your algorithm's correctness and derive its running time.

Hint: $O(n)$ time is possible using a greedy approach. Only one bag needs to be split.

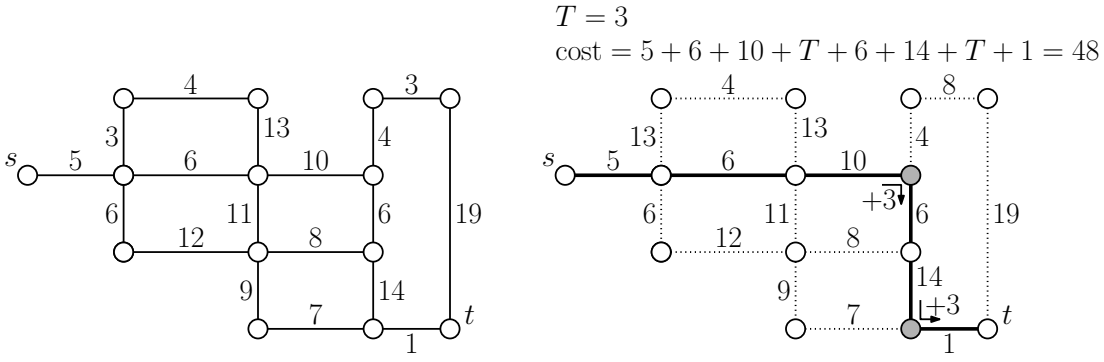
(b) (5 points) Suppose instead that each bag must be taken in its *entirety*. Present a short example to show that your greedy algorithm from part (a) is *not* optimal.

Hint: You can do this with just three bags.

Problem 4. (25 points) Google maps wants you to program a navigation algorithm to find the shortest route to a Halloween party. You are given a graph $G = (V, E)$ where nodes are intersections and roads are edges. This runs in an urban setting, and each road runs either north-south or east-west (see the figure below). Each edge is associated with a positive

weight $w(u, v)$, which indicates the time to travel from node u to v. Also, each edge (u, v) is associated with an *orientation* $o(u, v)$ which is either NS (for north-south) or EW (for east-west). Whenever a driver makes a *left or right turn*, there is an additional *time penalty* T added to the time, which does not apply when going straight through. (You wouldn't want to hit any trick-or-treaters!)

For example, suppose that $T = 3$. In the graph below, the edge cost of the path from s to t is 42, and it makes two turns, for a total time of 48.



Present an efficient algorithm, which given such a graph, a source vertex s, and a destination vertex t, and the value T, computes a minimum cost path from s to t. (It is sufficient to compute the cost of the path, not the path itself.)

Hint: It is possible to achieve the same running time as Dijkstra's algorithm. Briefly justify your algorithm's correctness and derive its running time.

Problem 5. (20 points) You operate a costume business that has two offices, one in Washington DC and one in Los Angeles. Each week, you need to decide whether you want to work in the DC office or the LA office. Depending on the week, your business makes more profit by having you at one office or the other. You are given a table of weekly profits, based on your location. Here is an example:

Week	1	2	3	4	5
DC	\$400	\$100	\$200	\$50	\$1100
LA	\$210	\$900	\$100	\$1500	\$20

Clearly, you would prefer to work at the location where you get the greater profit, but here is the catch. It costs \$1000 to fly from one office to the other. (For example, if you do the first job in DC, the next three in LA, and the last in DC, the total profit will be $\$400 - \$1000 + (\$900 + \$100 + \$1500) - \$1000 + \$1100 = \$2000$.

You are given two lists of length n, DC[1..n] and LA[1..n], where DC[i] is the profit for spending week i in DC, and LA[i] is the profit for spending week i in LA. Present an efficient algorithm, which given these two arrays, determines your maximum overall profit. You must *start* and *end* in DC, but you may travel back and forth any number of times. Briefly justify your algorithm's correctness and derive its running time.

Hint: $O(n)$ time is possible using dynamic programming. It suffices to give just the recursive rule. You will need to find a way to keep track of where you were the previous week.

Practice Problems for the Final Exam

The final will be **Tue, Dec 17, 8:00-10:00am** in our usual classroom. 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 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 set of 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.)

By the way, the Partition Problem is NP-hard. The above algorithm is only pseudo-polynomial, because the running time depends on the magnitude of the numbers, not on the logarithm of their magnitude.

Problem 5. Show that the following problem is NP-complete.

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. 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 8. 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.

Problem 9. Recall the Partition Problem (Problem (4) 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) Briefly explain how to implement this in $O(n \log n)$ time.

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

(c) 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.

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. (25 points: 3–8 points each) Short answer questions. (Unless otherwise specified, explanations are not required but may be given for partial credit.)

(a) What is the maximum number of edges in an undirected graph with n vertices, in which each vertex has degree at most k? (For full credit, given an exact answer, not asymptotic.)

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

(c) True or false: If the capacities of a network are integers, then there exists a maximum flow, such that the flow on each edge is an integer.

(d) Indicate whether each claim below is True, False, or "not known to science." Let A and B be any two decision problems.

(i) A and B are in P implies that $A \leq_P B$.

(ii) A and B are in NP implies that $A \leq_P B$.

(iii) A and B are NP-complete implies that $A \leq_P B$.

(e) Indicate whether each claim below is True, False, or "not known to science."

(i) Determining whether a graph *does not* have a Hamiltonian Cycle is in NP.

(ii) Suppose that problem X is in NP but not known to be NP-complete, and someone proves that X is not solvable in polynomial time. It follows that no NP-complete problem is solvable in polynomial time.

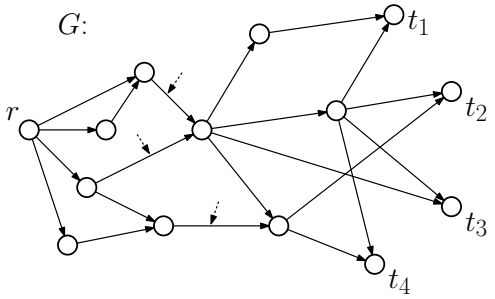
(iii) It is possible to determine in polynomial time whether a graph G has an independent set of size 100.

Problem 2. (20 points) Consider the following weighted variant of the longest-common subsequence problem (LCS). You are given two strings $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$. Each character z in your alphabet is associated with a nonnegative numeric weight $w(z)$. Present an efficient algorithm, which given X and Y computes the common subsequence of maximum total weight, called the *maximum weight common subsequence* (MWCS).

For example, suppose that $w(\text{A}) = w(\text{B}) = 1$, $w(\text{C}) = 5$, and $X = \langle \text{ABACB} \rangle$ and $Y = \langle \text{ACBAB} \rangle$. The standard LCS would be $\langle \text{ABAB} \rangle$ (of total weight 4) but the MWCS would be $\langle \text{ACB} \rangle$ (of total weight 7).

Your algorithm need only compute the *total weight* of the MWCS, not the actual sequence. (Hint: Use Dynamic Programming. You are not required to give the entire DP algorithm, just the recursive rule.) Briefly justify your algorithm's correctness and derive its running time.

Problem 3. (15 points) You are given a directed network $G = (V, E)$ with a root node r and a set of terminal nodes $T = \{t_1, \ldots, t_k\}$. Present a polynomial time algorithm to determine the minimum number of edges to remove so that there is no path from r to any of the terminals. (Hint: Use network flow.) Prove that your algorithm is correct.



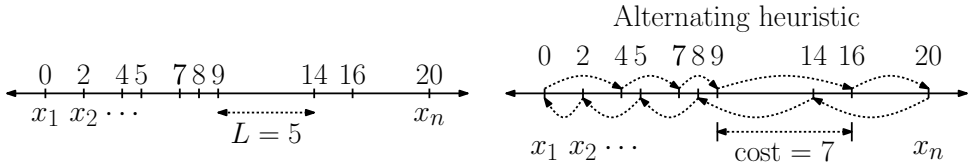
Problem 4. (25 points) In this problem, we will show that the following variant of the independent-set problem is NP-complete.

Half-sized Independent Set (HSIS): Given a graph G with n vertices, does G have an independent set of size $\lceil n/2 \rceil$?

(Hint: Remember to show both (i) that HSIS is in NP and (ii) HSIS is NP-hard. The reduction is from Independent Set (IS). Given the instance (G, k) for the independent set, there are two cases to consider: $k \leq n/2$ and $k > n/2$. For simplicity, you may assume that n is *even*. Your reduction should work correctly for any value of k, but I will give full credit if you give a proof of correctness only for the case $k \leq n/2$.)

Problem 5. (15 points) Suppose you have a sequence of points $X = \langle x_1, \ldots, x_n \rangle$ sorted from left to right along a line. The distance between two points x_i and x_j is just their absolute difference $|x_j - x_i|$. The *bottleneck TSP problem* is the following: Find a cycle that visits each point exactly once, such that *maximum* distance traveled between any two consecutive points is as small as possible.

Consider the following *alternating heuristic* for this problem: Travel from x_1 to x_n, skipping every other point. Then return from x_n to x_1 visiting the skipped points. (An example is shown in the figure below right.) The final cost is the longest segment traversed, which is the segment of length 7 between the points at positions 9 and 16.



Prove that this heuristic provides a factor-2 approximation to the bottleneck TSP problem (for points on a line). Hint: Let L be the maximum distance between any two consecutive points. Relate the costs of the optimum path and the heuristic path to L.

(Note: I believe that the alternating heuristic is actually optimal, but it is much easier to prove the factor-2 approximation bound.)