

Homework 1: Algorithm Design Basics

Handed out Thu, Feb 7. Due at the start of class Tue, Feb 19. 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.

Problem 1. The eminent but flaky Professor Hubert J. Farnsworth¹ proposed 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.

- (1) Repeat the following for i running from 1 to n :
 - (a) Man m_i proposes to the highest woman on his preference list.
 - (b) This woman accepts this proposal, and is immediately removed from the preference lists of all remaining men.
- (2) Output the resulting sets of mates.

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

- (a) Is the matching produced by Farnsworth's algorithm guaranteed to be stable? If so, give a proof. If not, present a counterexample. (See the note below.)
- (b) Suppose that all the woman in this system have exactly the same sets of preferences, and in particular, they rank the men in (decreasing preference) order $\langle m_1, m_2, \dots, m_n \rangle$. The men's preferences are not constrained. (Of course, each man's list must contain all the women.) Under this restriction, is the matching produced by Farnsworth's algorithm guaranteed to be stable? As before, either give a proof or present a counterexample.

(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 $(1^3 + 2^3 + \dots + n^3) = (1 + 2 + \dots + n)^2$.

- (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.

¹Famous inventor of the *Coolometer*, a hand-held device that measures a persons "coolness" as measured in "MegaFonzies."

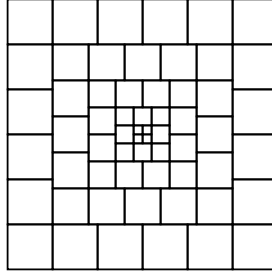


Figure 1: Problem 2.

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 $f(n) \equiv g(n)$. 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:

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

the answer would be

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

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

- | | | | |
|-----|------------------------|--------------------------|--------------------|
| (a) | $(3/2)^n$ | $3^{(n/2)}$ | $2^{(n/3)}$ |
| (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, $\langle a_1, a_2, \dots, a_n \rangle$. For each element a_i , for $1 \leq i \leq n$, we want to know the index of the rightmost element of the sequence $\langle a_1, a_2, \dots, a_{i-1} \rangle$ 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, but it is only allowed to use $O(1)$ additional working storage. That is, it can use a constant number of primitive variables, but it cannot declare any additional arrays (including strings) nor can it allocate memory dynamically. It is also not allowed to make recursive calls, since doing so would allow it to use the 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] |

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. However, 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)`.) In addition to presenting the 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 an 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.

Homework 2: BFS, DFS, and Greedy Algorithms

Handed out Thu, Feb 21. Due at the start of class Tue, Mar 4. Late homeworks are not accepted, but you may drop your lowest homework score.

As always, if you are asked to present an algorithm, in addition to the algorithm you must justify its correctness and derive its running time.

Problem 1. Consider the BFS algorithm presented in class. Given a digraph $G = (V, E)$ and a starting vertex $s \in V$, this algorithm computes for each vertex $u \in V$ the value $d[u]$, which is the length (number of edges) on the shortest path from s to u . For purposes of robust network communications, it is sometimes useful to know the number of shortest paths. The objective of this problem is to modify the BFS algorithm of class to compute the number of shortest paths from s to each vertex of G . We will do this in two steps.

- (a) First run the standard BFS on G , starting from s . Explain how to use the result of this BFS to produce a new digraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that every path in G' that starts at s , is a shortest path in G starting from s , and conversely, every shortest path in G starting from s is a path in G' .
- (b) Explain how to take the result of part (a) to compute for each vertex $u \in V'$, a quantity $c[u]$, which is the number of paths in G' from s to u . (Hint: This can be done by a modification of BFS.)

Both of your algorithms should run in $O(V + E)$ time.

Hint: Be careful in how you compute $c[u]$. In general, the number of shortest paths from one node to another can be exponential in the size of V . (You might think about why this is.) This means that your algorithm cannot generate the paths one by one. Rather, it must employ some method that can count up many paths at once.

Problem 2. Each of the following problems involves knowledge of cycles and paths in undirected and directed graphs. For all three parts, assume that the graph or digraph is represented using an adjacency list.

- (a) Present an algorithm which, given an undirected graph $G = (V, E)$, determines whether G contains a cycle in $O(V)$ time. (It is important that the running time of your algorithm is independent of E , which may generally be as high as $\Omega(V^2)$.)
- (b) The eminent but flaky Professor Hubert J. Farnsworth¹ claims that he has discovered an $O(V)$ time algorithm that determines whether a digraph has a cycle, assuming that the digraph is represented using a standard adjacency list. Prove that the Professor is wrong!
Hint: Assume that you have the professor's source code. Based on his code and given an arbitrary value V for the number of vertices, devise a digraph with $\Omega(V^2)$ edges, with the property that if his algorithm fails to look at *every* edge of your graph, you can trick his algorithm into producing the wrong result by altering one of the edges that his algorithm failed to look at.
- (c) A digraph $G = (V, E)$ is said to be semi-connected if, given any two vertices, $u, v \in V$, there exists a path from u to v , or there exists a path from v to u , but not necessarily both. Give an algorithm which, given a directed acyclic graph (DAG), determines whether it is weakly connected.

¹Noted inventor of the *electric frankfurter* and winner of the Academy of Sciences award for "Advances in superconductivity properties of sandwich meats."

Problem 3. The following two problems involve algorithms for connectivity in graphs and digraphs.

- (a) Recall that a *bridge* is an edge in a connected, undirected graph whose removal causes the graph to become unconnected. Present an $O(V + E)$ algorithm, which reports all the bridges in a graph.
- (b) The concept of a bridge can be extended to directed graphs. (I don't know that it has any particular name, so I'll make one up.) Given a digraph $G = (V, E)$, an edge (u, v) is called a *span* if there is no path from v to u in G . Present an $O(V + E)$ algorithm, which determines all the tree edges of the DFS forest that are spans. (Note that cross edges and forward edges might be spans as well, but your algorithm should only report on the DFS tree edges that are spans.)

Hint: Both algorithms can be implemented by an appropriate modification of the algorithm we gave in class for biconnected components.

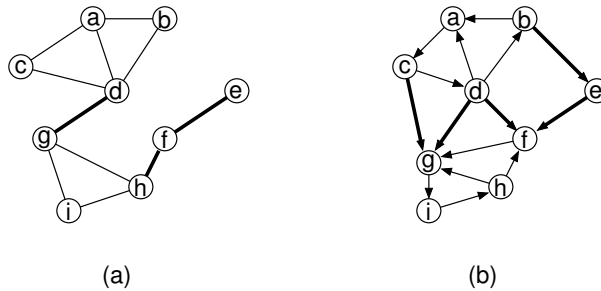
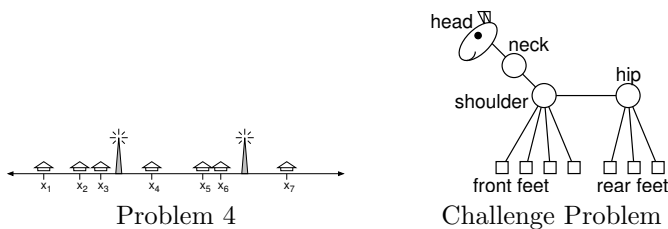


Figure 1: Problem 3(a) and 3(b). Bridge edges (shown as bold lines) in (a) are $\{d, g\}$, $\{f, h\}$, and $\{e, f\}$. Span edges (shown in bold in (b)) are (c, g) , (d, g) , (d, f) , (b, e) , and (e, f) .

Problem 4. Consider a long straight road with houses scattered very sparsely along its length. Let $x_1 < x_2 < \dots < x_n$ denote the locations of these houses. Design a greedy algorithm that will place the minimum number of cell towers along this road so that every house is within four miles of one of the towers. (Your algorithm should run in $O(n)$ time.) Prove that the result it produces is optimal. (Try to be rigorous in your proof, for example, by showing that any optimal solution that is not equal to the greedy solution can be converted into a solution of equal size that is closer to the greedy solution, in some sense.)



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 integers i and j , both greater than or equal to 2, we define an (i, j) -goat to be an undirected graph which consists of $n = i + j + 4$ vertices. The vertices consist of i front feet, j rear feet, and four extra vertices, the head, the neck, the shoulder, and the hip. These are connected as shown below (head-neck-shoulder-hip, and front feet to shoulder and rear feet to hip) (see the figure above).

You are given the $n \times n$ adjacency matrix for an n -vertex goat, but you are *not* told what the values of i and j are. Give an $O(n)$ time algorithm which, given the adjacency matrix of a goat, determines the

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

Homework 3: Kruskal, Dijkstra, and Dynamic Programming

Handed out Tue, Mar 11. Due at the start of class Thu, Mar 27. Late homeworks are not accepted, but you may drop your lowest homework score.

As always, if you are asked to present an algorithm, in addition to the algorithm you must justify its correctness and derive its running time.

Problem 1. Suppose that you wish to route flow through a network of pipes. We model the network as a connected, undirected graph $G = (V, E)$, in which each edge has a numeric value $c(u, v)$, which represents the edge's *capacity*, that is, the amount of flow it can take. Given any path $P = \langle u_1, u_2, \dots, u_k \rangle$, its *capacity* is defined to be the minimum capacity of any edge on the path, that is $\text{cap}(P) = \min(c(u_1, u_2), c(u_2, u_3), \dots, c(u_{k-1}, u_k))$. By convention, $c(u, u) = \infty$. For every $u, v \in V$, define $\text{cap}(u, v)$ to be the maximum capacity over all paths from u to v .

- (a) (Single-Source Capacity) Given a source vertex $s \in V$, present an algorithm that computes $\text{cap}(s, u)$ for all $u \in V$. Your algorithm should run in $O(E \log V)$ time. (Hint: One solution that I know of involves modifying Dijkstra's algorithm.)
- (b) (All-pairs capacity) Present an algorithm that computes $\text{cap}(u, v)$ for all $u, v \in V$. Your algorithm should run in $O(V^3)$ time. (Hint: One solution that I know of involves modifying the Floyd-Warshall algorithm.)

In both cases it is sufficient just to compute the capacity of the paths. It is not required to compute the actual paths.

Note: In both cases, be sure to establish the correctness of your algorithms. It is not sufficient to simply appeal to the correctness of related algorithms like Dijkstra's or Floyd-Warshall. Capacity is fundamentally a different optimization criterion than weighted path length, and so the proofs of these other algorithms will need to be modified to work in this context.

Problem 2. On the planet of Smurf, the adorable, gentle Smurf folk hold a convention every four years. In case you don't know, Smurf's never die, and they 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. 1). 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 1, 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 $5.2 + 7 = 12.2$.

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.

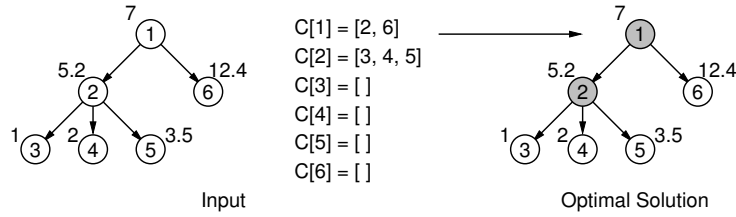


Figure 1: Example for Problem 2. Next to each node of the tree is its travel cost.

Problem 3. 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 $X = \langle ABC \rangle$, $Y = \langle BACA \rangle$, and $Z = \langle ABBACCA \rangle$ the answer is “yes,” ($Z = \langle A_x B_x B_y A_y C_x C_y A_y \rangle$) but if $Z = \langle ABCBAAC \rangle$ the answer is “no.” You may assume that $k = m + n$, for otherwise answer is definitely “no.”

- The eminent but flaky Professor Hubert J. Farnsworth¹ 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.
- Present an $O(nm)$ time algorithm that answers this decision problem. (Whether right or wrong, you cannot reuse Farnsworth’s algorithm.)
- Let us convert this decision problem into an optimization problem, by computing the smallest number of deletions from any of X , Y , or Z to make such a shuffle possible. For example, if $X = \langle ABC \rangle$, $Y = \langle BACA \rangle$, and $Z = \langle ABCBAAC \rangle$, then by deleting the last character of Y and the second to last character of Z , such a shuffle is possible. Thus, the answer would be 2. The running time of your algorithm should be $O(mnk)$ where $m = |X|$, $n = |Y|$, and $k = |Z|$. (Because characters can be deleted, you should not assume that that $k = m + n$.) Your algorithm should not only compute the minimum number of deletions needed, but it should also output the set of deletions to be performed.

Problem 4. A pizza delivery boy needs to make a series of deliveries. He is given the coordinates $p_i = (x_i, y_i)$ of a collection of n points $\{p_1, p_2, \dots, p_n\}$. You may assume that the points are given in increasing order of x -coordinate, that is, $x_1 < x_2 < \dots < x_n$. His pizza parlour is located at p_1 , and his route is subject to the restriction that he first travels strictly from left to right (along increasing x -coordinates) until getting to p_n , and then he returns to p_1 strictly from right to left (along decreasing x -coordinates). (See Fig. 2.) Present an efficient algorithm that computes the cost of the shortest delivery tour that visits all the points subject to the above left-right-left restriction. (You do not need to compute the path, just the cost.) You may assume that you have access to a function $dist(i, j)$, which returns the distance from point p_i to p_j .

Hint: Use DP. Rather than thinking of a outbound and inbound path, consider a formulation that builds two paths, both starting at p_1 and both traveling from left to right, such that each point will either be added to one path or the other. In my formulation, I used two variables i and j , where one path ended at point i and the other ended at point j . For full credit, your algorithm should run in $O(n^2)$ time. I suspect that there is a somewhat simpler $O(n^3)$ time algorithm.

¹Noted inventor of the *nose tuba*, a brass instrument that can be played with the nose. The nose tuba achieved notoriety as the only musical instrument to be featured in an article of the International Journal of the Medical Malpractice, because of its tendency to cause musicians to blow their brains out while attempting the tricky second movement of Mendelssohn’s Tuba Concerto in A minor.

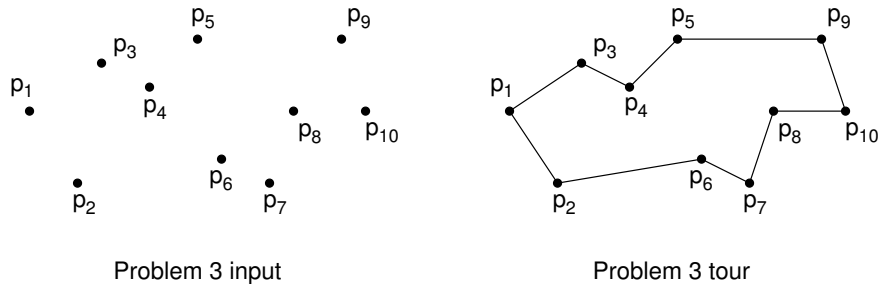


Figure 2: Problem 4.

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.

The following two problems are similar in spirit, and both solutions involve a similar trick. You can do either one or both (or neither!).

- (a) You are given a pointer to the head of a linked list of unknown length. Each node of the link list contains nothing but a pointer to the next node in the list. Either the linked list terminates in a null pointer, or else it loops back on itself (see Fig. 3(a).) Write an algorithm that determines which is the case. You are not allowed to alter the contents of any of the nodes of the linked list, and you only have access to a constant number of simple variables (no arrays or dynamic memory allocation allowed). Your algorithm should run in $O(n)$ time, where n is the (unknown) number of elements in the linked list. (For maximum credit you do not need any integer variables of more than constant size. My solution requires only two pointer variables and a boolean.)

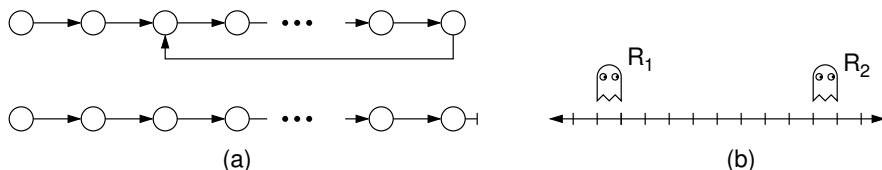


Figure 3: Challenge Problem.

- (b) 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. 3(b)). (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.)

Write a program to help the robots find each other. The two robots are synchronized and execute one step every second. As a robot I may do any of the following things at each step of my program: move myself left/right one position, mark the spot that I am on (this can be done only once per robot), test whether the spot I am on has been marked, and test whether the other robot is currently on my spot. The robots have only a finite amount of local memory. Note that the initial distance between the two robots may be much much larger than any number that can be stored in the local memory.

Hints: It is possible to have both robots execute exactly the same program. Marks cannot be erased. Don't try to count, you don't have enough local memory. The running time should be $O(n)$, where n is the (unknown) distance between the robot's initial positions.

Homework 4: Network Flow and NP-Completeness

Handed out Tue, Apr 15. Due at the start of class Tue, Apr 29. Late homeworks are not accepted, but you may drop your lowest homework score.

As always, if you are asked to present an algorithm, in addition to the algorithm you must justify its correctness and derive its running time.

Problem 1. The eminent but flaky Professor Hubert J. Farnsworth¹ has invented an alternative network flow algorithm. His algorithm is similar to Ford-Fulkerson, but Farnsworth's algorithm only *increases* flows along edges that have not yet reached their capacity—it never reduces flow on any edge by pushing flow along backward edges of the residual network.

- **(For partial credit)** Show that Farnsworth's algorithm is *not* optimal by presenting a counterexample, which consists of an s - t network G such that his algorithm terminates with a flow that is strictly less than the true maximum flow. You are free to select the augmenting paths however you like in forming your counterexample.
- **(For full credit)** Show that Farnsworth's algorithm is not merely suboptimal but can be *arbitrarily bad*. In particular, given any positive integer $b > 1$, show how to construct an s - t network G such that the ratio between the true maximum flow in G to Farnsworth's flow at least as large as b . (The size of your network and its capacities are allowed to be functions of b .)

Be sure to explain how your counterexample works.

Problem 2. 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 ($1 \leq k \leq |E|$). 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 $G' = (V, E - E')$ is as low as possible.

Justify your algorithm's correctness and derive its running time.

Problem 3. You are the manager of the computing staff at Mega State University, and one of your less pleasant duties involves assigning your staff to be on-call during holiday periods. There are n people in your staff and there are m holiday periods. Based on your calendar, you know that the j th holiday runs for $d[j]$ days. (You may assume that no two holidays overlap in time.) Here are the constraints that you must abide by:

- There must be one staff member on-call during every day of every holiday period.
- According to union rules, no staff member may be on-call for more than c total days over all the holiday periods.
- According to union rules, no staff member may be on-call for more than one day during any one holiday period.
- Each staff member $1 \leq i \leq n$ has given you a list $H[i]$ of holiday periods when they are available. (You may assume that if a person is listed as available for some holiday, he/she is available for any day of the holiday period.)

Describe a polynomial time algorithm, which given $d[1..m]$, c , and $H[1..n]$, generates an assignment of staff members to holidays. If no such assignment is possible, your algorithm should indicate this. Prove the correctness of your algorithm and give its running time.

¹Famous inventor of the *laser-guided ketchup dispenser*, a device capable of squirting ketchup to within 2.5 microns of any desired spot on your hamburger. (The GPS-enabled version cannot be sold to minors, except in Texas.)

Problem 4. Consider a standard instance of the network flow algorithm, that is, an s - t network $G = (V, E)$, where each edge (u, v) has a nonnegative integer capacity $c(u, v)$. Present a polynomial time algorithm which determines whether G 's minimum s - t cut is *unique*. For example, the network shown in Fig. 1 has two distinct minimum cuts, each of capacity 11.

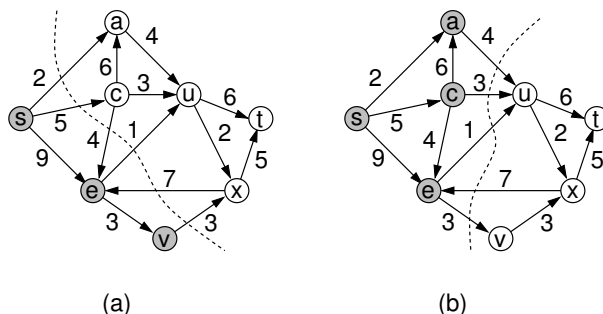


Figure 1: A network with (at least) two distinct minimum s - t cuts, each of value 11.

Justify your algorithm's correctness and derive its running time.

Hint: One approach involves invoking a network flow algorithm as a black-box (either applied to G directly or to some modification of G) followed by a subsequent analysis of the flow and/or the residual network. This process may be repeated a number of times.

Problem 5. This problem involves the observation that a solution to a decision problem can often be used to solve a related optimization problem.

Given an undirected graph $G = (V, E)$, it is said to be *3-colorable* if there is an assignment of the labels $\{1, 2, 3\}$ to the vertices of G such that no two adjacent vertices have the same label. (It is known that deciding 3-colorability is an NP-complete problem.) Suppose that (by some miracle) you have access to a procedure `is3Colorable(G)` which, given an undirected graph G , returns *true* if G is 3-colorable and *false* otherwise.

Using this procedure as a black box, present an algorithm which, given a undirected graph G determines whether G is 3-colorable, and if so outputs a valid coloring of the vertices. Your algorithm should run in polynomial time, and can make a polynomial number of calls to `is3Colorable`.

Hint: Consider how to make a series of modifications to the graph which successively restrict the possible ways of choosing colors, but without changing the fact that G is 3-colorable.

Challenge Problem. Some friends of yours who watch lots of movies have invented a new game. You start with a set X of n actresses and a set Y of n actors, and two players P_0 and P_1 . Player P_0 names an actress $x_1 \in X$, player P_1 names an actor $y_1 \in Y$ who has appeared in a movie with x_1 , player P_0 then names an actress x_2 who has appeared in a movie with y_1 , and so on. Thus, P_0 and P_1 collectively generate a sequence $\langle x_1, y_1, x_2, y_2, \dots \rangle$ such that each actor/actress in the sequence has appeared with the actress/actor immediately preceding. A player P_i ($i = 0, 1$) loses when it is P_i 's turn to move, and he/she cannot name a member of his/her set who has not been named before.

Suppose you are given a specific pair of such sets X and Y , with *complete information* on who has appeared in a movie with whom. A *strategy* for player P_i , in our setting, is an algorithm that takes a current sequence $\langle x_1, y_1, x_2, y_2, \dots \rangle$ and generates a legal next move for P_i (assuming it is P_i 's turn to move). Give a polynomial-time algorithm that decides which of the two players can force a win, in a particular instance of this game.

Hint: The material from Section 7.5 of KT on bipartite matchings, and Hall's Theorem in particular, may be useful.

Homework 5: NP-Completeness and Approximation Algorithms

Handed out Tue, Apr 29. Due at the start of class Tue, May 13. Late homeworks are not accepted, but you may drop your lowest homework score.

As always, if you are asked to present an algorithm, in addition to the algorithm you must justify its correctness and derive its running time.

Problem 1. The eminent but flaky Professor Hubert J. Farnsworth¹ has stunned the scientific community by announcing that he has proven that $P = NP$. In particular, he has discovered a polynomial time algorithm for the CLIQUE problem.

For any fixed $k \geq 1$, the k -CLIQUE problem is as follows: Given an undirected graph $G = (V, E)$, does it contain a clique of size k ? Farnsworth claims that for any $k \geq 1$, the following polynomial time algorithm solves k -CLIQUE.

Input the graph $G = (V, E)$. Let $n = |V|$. Enumerate all subsets of vertices of size k . There are $\binom{n}{k} = O(n^k)$ such subsets. In $O(n^2)$ time it is possible to check whether each such subset is a clique. Thus, in $O(n^{k+2})$ time, we can answer the k -CLIQUE problem. Since any instance of CLIQUE is an instance of k -CLIQUE for some k , CLIQUE can be solved in polynomial time, and therefore $P = NP$.

What is Farnsworth's mistake?

Problem 2. Consider the following problem, called the *zero-weight cycle* (ZWC). You are given a directed graph $G = (V, E)$ with weighted edges (which may be positive, negative or zero), and the question is whether there exists a simple cycle of total weight 0? (To prevent trivial solutions, we require that any solution consist of at least one edge.) Prove the ZWC is NP-complete. (Hint: Reduction from directed Hamiltonian cycle.)

Problem 3. This is a variant of the Independent Set problem. You are given an undirected graph $G = (V, E)$. We say that a subset $V' \subseteq V$ is a *strongly independent set* if for any two vertices $u, v \in V'$ the edge (u, v) is not in E and there is no path of two edges from u to v . The *Strongly Independent Set Problem* (SIS) is, given an undirected graph $G = (V, E)$ and an integer k , does G have a strongly independent set of size k . Prove that SIS is NP-complete. (Hint: Reduction from Independent Set (IS).)

Problem 4. This problem arises in applications of navigation, where a user wishes to go from one point to another, while escaping detection by avoiding revisiting the same regions twice. The problem is called the *evasive path problem* (EPP). You are given a directed graph $G = (V, E)$, a designated start node s , a target node t , and a collection of pairwise disjoint *zones* $Z_i \subseteq V$, for $1 \leq i \leq k$. The question is whether there exists a path from s to t that visits at most one vertex within each zone. Prove that EPP is NP-complete. (Hint: Reduction from directed Hamiltonian path.)

Problem 5. Suppose you are given an $n \times n$ *grid graph* $G = (V, E)$, as shown in Fig. 1. Each node is associated with a nonnegative integer weight. You may assume that all weights are distinct. Your objective is to compute an independent set $V' \subseteq V$ so that the sum of weights of the vertices of V' is as large as possible. Consider a simple greedy algorithm that works by selecting the vertex u of maximum weight from G and then deleting u and all its neighbors. This process is repeated until there are no more vertices left.

Prove that this algorithm produces an independent set for G whose total weight is at least $\frac{1}{4}$ the total weight of the optimal solution.

¹Famous inventor of the *solar-powered night light* and recipient of the prestigious award for *Most Idiotic Application of Green Technology*.

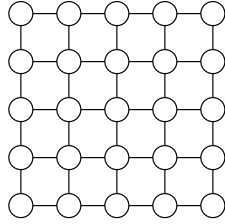


Figure 1: Problem 5.

Challenge Problem. The local video store has received a shipment consisting of a large number of video tapes. Among these is one special tape. Exactly 7 days after viewing this tape the viewer dies a mysterious death without warning or exception. (This might remind you of a scary movie, which came out a number of years ago.)

The video store manager wants to determine which of the tapes is the deadly one before the big sale coming up 8 days from now. He has managed to find a number of foolish people, who are willing to test the tapes for a hefty fee. Each tester will be given some subset of the tapes to view on the first day, and then he/she nervously waits for 7 days for the final results to develop. (There is no limit on the number of tapes that may be assigned to one tester, and each tape may be viewed many different testers.)

Suppose that there are n tapes total. The manager realizes that he can determine the deadly tape by arranging for n testers, each of whom will view one tape. A smart clerk informs him that he can do it with fewer testers. What is the minimum number of tape testers needed to determine which tape is deadly and how is the test conducted? (The stingy store manager does not care how many of the testers dies in the process; he just wants to pay the least number of testers.) If the deadly tape is random among the set of n tapes, then what is the expected number of testers that survive the process?

Practice Problems for the Midterm

The midterm will be on Tues, Apr 1. 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 $d[v] \geq d[u] + 2$? 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) In the DFS of a digraph there is an edge (u, v) such that $d[u] < d[v]$. What are the possible types of this edge? (tree, back, forward, cross.)
- (e) Given a connected undirected graph $G = (V, E)$ in which all edges have weight 1, what is the fastest and simplest way (that we know of) for computing a minimum spanning tree for G ?
- (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 graph with V vertices and E edges as an adjacency list. How fast can you sort the vertices in increasing order of their degrees? (Explain briefly.)
- (h) True or False: If a digraph has k distinct cycles, then for any depth-first search (DFS) of this digraph, there are at least k back edges.

Problem 2. Given two strings, $X = x_1x_2\dots x_m$ and $Y = y_1y_2\dots y_n$, the *shortest common supersequence* (SCS) is a minimum length string Z such that both X and Y are subsequences of Z . For example, if $X = \langle ABCBABA \rangle$ and $Y = \langle BCAABAB \rangle$, then $Z = \langle ABCAABABA \rangle$ is an SCS of both X and Y .

Give an $O(mn)$ dynamic programming algorithm which given X and Y computes the length of the SCS of X and Y . You do not need to determine the actual SCS, just its length. Be sure to give the DP formulation, and explain its correctness. (Hint: There is a cute, tricky solution. But to gain practice in doing DP problems, even if you see it, work through the full DP formulation anyway.)

Problem 3. A pharmacist has W pills and n empty bottles. Let $\{p_1, p_2, \dots, p_n\}$ denote the number of pills each bottle can hold.

- (a) Describe a greedy algorithm, which given W and the p_i 's determines the fewest number of bottles needed to store all the pills. (An informal description is sufficient.)
- (b) Argue that the *first* bottle chosen by your greedy algorithm will be in some optimal solution.
- (c) How would you modify your algorithm if each bottle also has an associated cost c_i , and you want to minimize the total cost of the bottles used to store all the pills? (No need to prove correctness.)

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

- (a) *Shortest request first:* Sort the requests in increasing order of duration $f_i - s_i$. Give a counterexample to show that this is *not* optimal.
- (b) *Latest start time first:* Sort the requests in decreasing order of start time. Give a short proof that this is optimal. (You may use any results proved in class.)

Problem 5. Let $G = (V, E)$ be an undirected graph. Write an $O(V + E)$ time algorithm to determine whether it is possible to direct the edges of G such that the indegree of every vertex is at least one. If it is possible, then your algorithm should show a way to do this. (Hint: Use DFS.)

Problem 6. 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 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 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 $O(E \log V)$ algorithm to do this. Briefly justify your algorithm's correctness and derive its running time.

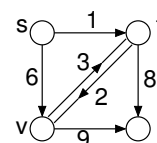
Midterm Exam

This exam is closed-book and closed-notes. You may use 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)

Show the execution of Dijkstra's shortest path algorithm on the digraph in the figure, where s is the source vertex. Indicate each of the following:

- The order in which the vertices are removed from the priority queue,
- The final distance value $d[u]$ for each vertex u ,
- The final predecessor pointer $\text{pred}[u]$ for each vertex u .



Problem 2. (30 points; 4–7 points each.) Short answer questions. Explanations are not always required, but may be given for partial credit.

- List the following functions in increasing asymptotic order. (No explanations required.)

$$f(n) = n^{\lg 8} \quad g(n) = 2^{(2 \lg n)} \quad h(n) = (\min(n, 2))^5.$$

If two functions are asymptotically equal (e.g., if $f(n) = \Theta(g(n))$), then indicate this. (Recall that \lg denotes logarithm base 2.)

- Recall that a digraph $G = (V, E)$ is *semi-connected* if, given any two vertices, $u, v \in V$, there exists a path from u to v , or there exists a path from v to u , but not necessarily both. Suppose you are given a semi-connected DAG G . True or false: The topological ordering of the vertices of G is unique. Explain briefly.
- Suppose that you run DFS on a directed graph $G = (V, E)$ and compute both discovery and finish times for each vertex. For some edge (u, v) you observe that $f[v] < d[u]$. What type of edge might this be? (List all the apply. Give a short explanation.)

(i) tree edge (ii) back edge (iii) forward edge (iv) cross edge

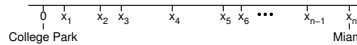
- You are given a connected, undirected graph $G = (V, E)$ with positive edge weights. You form another graph G' by squaring the weight of every edge of G . True or false: A spanning tree T is a minimum spanning tree of G if and only if T is a minimum spanning tree of G' . Explain briefly.
- Recall the following two rules used in the Floyd-Warshall algorithm for computing shortest paths in a directed graph with edge weights $w(i, j)$.

$$d_{ij}^{(0)} = w(i, j) \quad d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

Explain how to modify these rules for the following related problem. (You do not need to modify the entire algorithm, just the rules.) For each edge $(i, j) \in E$ you are given the

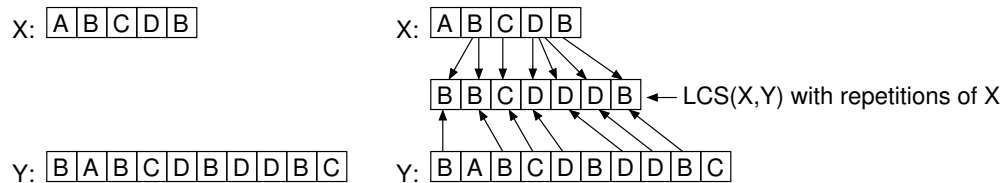
probability $0 < p(i, j) < 1$ that the edge is usable. The probability that a path is usable is the *product* (i.e., multiplication) of the probabilities of the edges on the path. For each i and j , the objective of your algorithm is to compute the path whose probability of usability is *maximum*.

Problem 3. (20 points) Professor 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 $x_1 < x_2 < \dots < x_n$ 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*.



Problem 4. (20 points) For each of the following two problems, present a short algorithm, briefly justify its correctness and derive its running time. (If you present a DP solution, it is sufficient to present the recursive DP formulation, rather than an entire algorithm.)

- (a) Consider the following modification of the longest common subsequence (LCS) problem. You are given two strings $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$. The objective is to compute the length of the longest common subsequence, in which each character of X (but not Y) is allowed to be *repeated* any number of times in the LCS.



Your program need only output the *length* of the LCS, not the actual subsequence.

- (b) Consider a further modification. In addition to X and Y , you are give a positive integer k . The problem is the same as in part (a), but each character of X can be repeated *at most k times* in the final LCS. (Characters of Y cannot be repeated.)

For example, if $k = 2$ in the above figure, you would only be allowed to use ‘D’ twice, rather than three times. (Hint: Add another index to your formulation of part (a).)

Problem 5. (15 points) You are given a connected undirected graph $G = (V, E)$ in which each edge’s weight is either 1 or 2. Present an $O(V + E)$ time algorithm to compute a minimum spanning tree for G . Explain your algorithm’s correctness and derive its running time. (Hint: This can be done by a variant of DFS.)

Practice Problems for the Final Exam

The final will be on Fri, May 16, 8-10am. The exam will be closed-book and closed-notes, but you will be allowed two pages of notes (front and back of each page).

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 the approximability of A .

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, $\langle b_0, b_1, \dots, b_{n-1} \rangle$ and $\langle r_0, r_1, \dots, r_{n-1} \rangle$. 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 $\langle b_0, b_1, \dots, b_{n-1} \rangle$ and $\langle r_0, r_1, \dots, r_{n-1} \rangle$, determines whether it is possible to redistribute the balls from the blue buckets into the red buckets according to these restrictions. (Hint: Use network flow.)

Problem 3. Given an undirected graph $G = (V, E)$, a *feedback vertex set* is a subset of vertices such that every simple cycle in G passes through one of these vertices. The *feedback vertex set problem* (FVS) is, given an undirected graph G and an integer k , does G contain a feedback vertex set of size at most k ? For example, the graph shown in Fig. 1 has a feedback vertex set of size 2 (shaded).

Show that FVS is in NP. That is, given a graph G that has a FVS of size k , give a certificate, and show how you would use this certificate to verify the presence of a FVS in polynomial time. (This is one of the few NP problems where verifying a certificate is not obvious.)

(You are *not* asked to show that FVS is NP-hard, but if you wanted to try this, as a hint the reduction is from Vertex Cover.)

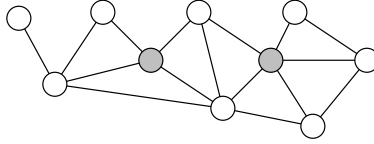


Figure 1: A feedback vertex set (FVS) of size 2.

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. 2.) A directed *Hamiltonian path* is a path that visits every vertex in a digraph exactly once.

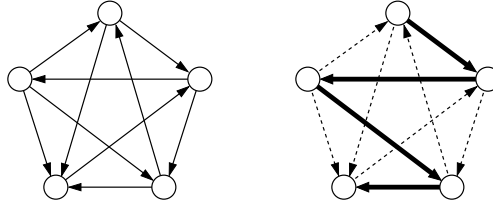


Figure 2: Hamiltonian path in a tournament.

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

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

Balanced 3-coloring (B3C): Given a graph $G = (V, E)$, where $|V|$ is a multiple of 3, can G 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 7. The *set cover* optimization problem is: Given a pair (X, S) , where X is a finite set and a $S = \{s_1, s_2, \dots, s_n\}$ 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. Prove that the Hamiltonian Path problem is NP-complete for undirected graphs. (Hint: Reduction from Directed Hamiltonian Path. Replace each vertex with a small gadget that splits the incident edges into two groups, such that if a path enters from one group of edges then it must leave along the others.)