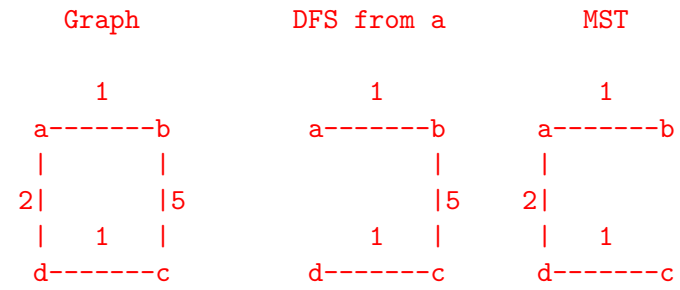
EECS 336: Introduction to Algorithms **Sample Final**

80 minutes

1. Suppose you have an undirected graph with weighted edges, and perform a depth-first search, such that the edges going out of each vertex are always explored in order by weight, smallest first. Is the depth first search tree resulting from this process guaranteed to be a minimum spanning tree? Explain why, if it is, or, if it is not, provide a counterexample (specifying the start vertex for the DFS and showing both trees).



2. The Independent Set decision problem, i.e., deciding whether there is an independent set $S \subset V$ in a graph $G = (V, E)$ with cardinality at least $|S| \geq k$, is NP-complete. (Recall that an independent set is a set S that contains no pair of vertices that share an edge.) Reduce the problem of *finding* an independent set of size at least k to the decision problem.

   Given a black box to decide whether there is an independent set of size at least k, we can find an independent set of size at least k (if one exists) as follows:

   (a) if G has no independent set of size $\geq k$ return "none found", otherwise:

   (b) $S =$ empty set.

   (c) for each vertex $v \in V$:

      i. $G' =$ graph G with v and its neighbors removed

      ii. if G' has an independent set of size at least $k - 1$:

         A. add v to S

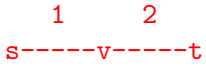
         B. set $G = G'$

         C. subtract one from k.

3. Given an arbitrary *network flow problem* specified by a directed graph $G = (V, E)$, integer capacities $c(e)$ for all $e \in E$, source s and sink t; answer the following true-false questions. If the answer is false, give a counter example. Recall that an edge is *saturated* by a flow f iff $f(e) = c(e)$.

   (a) For any integral max-flow f and e that is not saturated by f, if we were to increase the capacity of e by one then the max-flow value of G must increase.

   False. Let $e = (v, t)$ in the flow graph below.

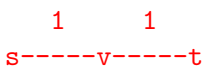


(b) For any integral max-flow f and e that is not saturated by f, if we were to increase the capacity of e by one then the max-flow value of G must not increase.

True.

(c) For any integral max-flow f and e that is saturated by f, if we were to increase the capacity of e by one then the max-flow value of G must increase.

False. Let $e = (v, t)$ in the flow graph below.

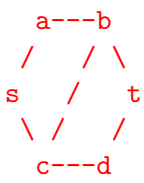


(d) For any s-t path P in G, if the capacity of each edge in P is increased by one then the max-flow value of G must increase.

True.

(e) For any s-t path P in G, if the capacity of each edge in P is increased by one then the max-flow value of G increases by at most one.

False. Consider this graph with capacities out of s and into t equal to 1 and all other capacities equal to zero. Increase the capacity of the s-a-b-c-d-t path and the max flow increases from 0 to 2.



4. Consider a variant on the Weighted Interval Scheduling Problem where instead of one machine there are two machines.

   Input:

   - n jobs $S = \{1, \ldots, n\}$.
   - two machines a and b.
   - start time s_i and end time f_i for each job i.
   - value v_i for object i if scheduled.

   Output: assignment of jobs to machines where

   - each machine has at most one job scheduled on it at any given time.
   - the total value of scheduled jobs is maximized.

   From each job's perspective, it does not matter which machine it is scheduled on, only whether it is scheduled or not.

   This problem can be solved with dynamic programming. Answer the following questions. Note: you are not asked to give the final algorithm.

(a) Identify a subproblem that will lead to an efficient dynamic program for the two-machine weighted interval scheduling problem. State that subproblem in English.

$\text{OPT}(i, j)$ = "the optimal way to schedule when jobs $\{i, \ldots, n\}$ are feasible for machine a and jobs $\{j, \ldots, n\}$ are feasible for machine b (sorted by start time)"

(b) Give a recurrence (formula) for calculating the total value of a subproblem from "smaller" subproblems.

Main idea: only consider scheduling a job when it is the last possible chance.

If $i = j$:

- $\text{OPT}(i, i) = \max(\text{OPT}(i+1, i+1), v_i + \text{OPT}(\text{next}(i), i+1))$
- justification: either don't schedule i, or without loss due to symmetry, schedule on machine a.

If $i < j$:

- $\text{OPT}(i, j) = \max(\text{OPT}(i+1, j), v_i + \text{OPT}(\text{next}(i), j))$
- justification: last chance to schedule i (only possible on a), either schedule it or don't.

If $i > j$:

- $\text{OPT}(i, j) = \max(\text{OPT}(i, j+1), v_j + \text{OPT}(i, \text{next}(j))$
- justification: symmetric with "$i < j$" case.

Instructor's Note: a simpler (but worse runtime) approach would have three parameters, two for the availablility on each machine and one for the remaining jobs to be scheduled. The approach above is complicated to prove correct because the argument that jobs are not scheduled for both a and b is non-trivial (but follows from the 'main idea' above). However, no proof or justifification was requested so the recurrences alone would be sufficient for full credit on this question.

(c) What is the runtime of the algorithm that would result from your approach (expressed in terms of n; you do not have to give the algorithm)?

$O(n^2)$

(d) If you were to generalize your algorithm to $k > 2$ machines, what would be the runtime's dependence on k (expressed in terms of n and k; you do not need to give the algorithm)?

$O(n^k)$

5. The Cycle Cover Problem is defined as follows: given a directed graph $G = (V, E)$, determine whether there exists a collection $\{C_1, C_2, \cdots, C_k\}$, where each C_i is a *simple directed cycle* in G (i.e., a cycle with no repeated vertices), such that for *every* vertex v in G, there is a *unique* cycle C_i containing it.

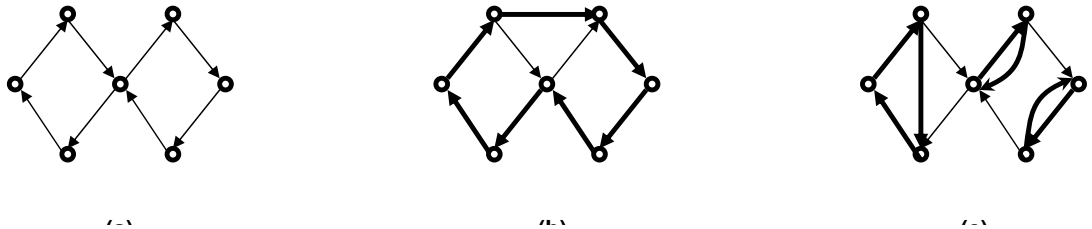
   The 3-Cycle Cover Problem is defined identically to the Cycle Cover Problem except for the additional constraint that each cycle must have length at most three, i.e., $|C_i| \leq 3$ for all i.

Examples of "Yes" and "No" instances for the Cycle Cover and 3-Cycle Cover problems. (The cycle covers are displayed in bold.)

(a) A "No" instance for Cycle Cover and 3-Cycle Cover;

(b) A "Yes" instance for Cycle Cover, but a "No" instance for 3-Cycle Cover;

(c) A "Yes" instance for both Cycle Cover and 3-Cycle Cover.



Prove that Cycle Cover is in **P**.

We reduce Cycle Cover to Bipartite matching. Bipartite Matching is in **P** so therefore Cycle Cover is as well. To do so, we construct a bipartite graph $G' = (A, B, E')$ and use a perfect matching in G' to construct a cycle cover.

(a) Let $A = V$ and $B = V$ be "representatives" of the vertices of the original graph.

(b) For $(u, v) in E$ in the original graph, add edge e' to E' from the representative of u in A to the representative of v in B.

(c) A perfect matching in G' implies a cycle cover in G:
Consider a vertex $v \in V$ in the original graph. It has representatives in A and B and since the matching is perfect, these each have incident edges. Thus, v has exactly two incident edges selected by the matching. A graph in which each vertex has exactly two incident edges is a graph of cycles. Therefore, we have a cycle cover.

(d) A cycle-cover in G implies a perfect matching in G':
Traverse each edge in each cycle. Add the corresponding edge to the matching. The resulting matching is perfect. This is because in traversing each cycle in the cycle cover we enter and leave each vertex exactly once. Leaving a vertex matches its representative in A and entering it matches its representative in B. Thus, each vertex in A and B is matched.

6. Prove that 3-Cycle Cover (defined in Problem 5) is **NP**-complete.

3-Cycle Cover is in **NP**: The certificate of a "yes" instance is the cover. Checking that it is a cover can be done by inspecting that each cycle is indeed a cycle and by making sure each vertex is covered. This algorithm is linear time.

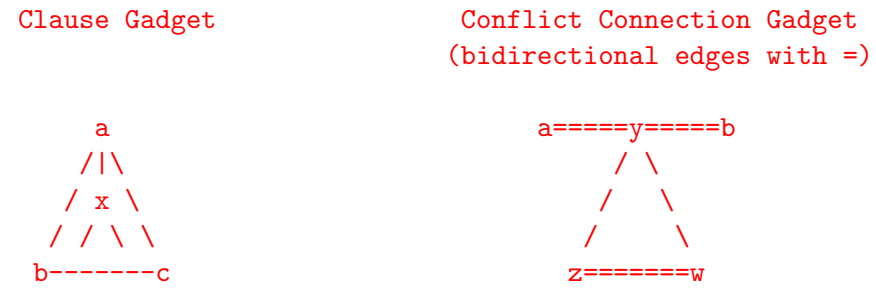
To show 3-Cycle Cover is **NP**-hard We reduce from 3-SAT using the same basic idea as we did in class for Independent Set. A gadget for clauses with a vertex needing to be covered

signifying a true literal. These the literals in these clause-gadgets are then connected to indicate that conflicting literals cannot both be true. Shown is a clause gadget on literals a, b, and c; and a conflict connection between literals a and b (which is "not a"). Notice that if a is not covered by in the clause gadget then we can cover it with a two-cycle in the conflict connection. But we cannot both cover a and b on the conflict connections. Notice that the only two and three cycles in the of clauses gadgets with conflict connections are the ones explicitly represented in the individual gadgets and connections.

Runtime: This construction can be computed easily in quadratic time.

Correctness: f is SAT $\Rightarrow$ G has 3-cycle cover. In a satisfying assignment each clause has a true literal. E.g., in the diagram suppose a is the true literal, then select cycle (b, x, c) in the clause gadget and (a, x) in the conflict connection gadget. This covers all clause gadget vertices. Moreover, in the conflict connection gadgets because the assignment was satisfied each y is covered at most once (i.e., the literal corresponding to a and b cannot both be true as a and b conflict). If y is covered, we can cover this gadget by just choosing cycle (z, w). If y is not covered, we can cover this gadget by just choosing cycle (z, y, w).

G has a 3-cycle cover $\Rightarrow$ f is SAT. Each conflict connection gadget y can be covered only once. Consider each variable, if there is a conflict connection gadget that corresponds to a literal of this varaible where (a, y) or (y, b) is covered then set the variable to true if it is a and false if it is b; otherwise, arbitrarily set the variable to true (it doesn't matter). By construction all conflict connection gadgets must agree for this variable. Consider each clause. The only way to cover a clause gadget is to have a three cycle on x and two of (a,b, or c) and a two cycle with the remaining end-point in a conflict connection. Thus each clause is satisfied by the chosen assignment.



Instructor's Note: This solution is much more verbose than your solution would be expected to be if this were an exam question.