CMSC 451 Midterm Exam

This exam is closed-book and closed-notes. You may use one sheet of notes (front and back). You may use any algorithms or results given in class. The total point value is 100 points. Good luck!

Problem 1. (10 points) Consider the DAG shown in Fig. 1.

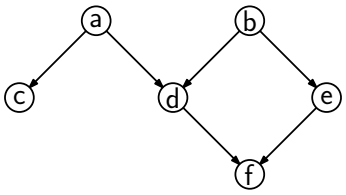

Figure 1: Depth-first search.

(a) (7 points) Show the result of running DFS on this graph. Whenever you have a choice, select the next vertex in *alphabet order*. Show the DFS tree along with the *start and finish times* for each vertex.

(b) (3 points) What is the *topological order* that results by enumerating vertices in reverse order of finish times?

Problem 2. (30 points) Short answer questions. *Explanations are not required*, but may be given for partial credit.

(a) (2 points) As a function of n, what is the *asymptotic running time* of the following code? (Use Θ notation.)

```
for (L = 2 to n):
    for (i = 1 to n-L+1):
        j = i + L - 1
        for (k = i to j-1):
            print("Hello")
```

(b) (6 points) You are given an undirected graph (no self-loops, no multi-edges) with n vertices and n edges. Which of the following must hold? (Select all that apply)

(i) This graph must be *connected*

(ii) This graph must have *at least one cycle*

(iii) The *average vertex degree* in this graph is $O(1)$

(c) (6 points) Suppose you run Dijkstra's algorithm (as presented in class) on a directed graph with a *negative cost cycle*. Which of the following might occur? (Select all that apply)

(i) It may generate incorrect distance values

(ii) It may go into an infinite loop

(iii) It may abort due to indexing an array out of bounds

(d) (8 points) Which of the following greedy strategies are *optimal* for (unweighted) interval scheduling? (Select all that apply.)

(i) EFF (Earliest finish-time first)

(ii) SDF (Shortest duration first)

(iii) FCF (Fewest conflicts first)

(iv) LSF (Latest start-time first - Selects the request with the latest start time and schedules it first)

(e) (4 points) Given a point set P in a metric space, the *diameter* of P, denoted $\text{diam}(P)$, is the maximum distance between any pair of points in P. Suppose we run Gonzalez's algorithm for two iterations, yielding centers c_1 and c_2. Which of the following assertions must hold? (Select one)

(i) $\text{diam}(P) = \text{dist}(c_1, c_2)$

(ii) $\text{diam}(P) \leq 2 \cdot \text{dist}(c_1, c_2)$

(iii) $\text{diam}(P) \leq c \cdot \text{dist}(c_1, c_2)$, for some constant c, but not 2

(iv) $\text{diam}(P)$ could be arbitrarily large compared to $\text{dist}(c_1, c_2)$

(f) (2 points) In the Floyd-Warshall algorithm, the DP formulation called for computing the quantity $d^{(k)}(i, j)$, where i is the source vertex and j is the destination vertex. In a sentence, what does k represent?

(g) (2 points) In the the residual network for the Ford-Fulkerson algorithm, we distinguished between *forward edges* and *backward edges*. Explain how pushing a flow along a forward edge (u, v) affects the flow between u and v. Do the same for a backward edge.

Problem 3. (15 points) In standard network flow, every vertex other that s and t must satisfy flow conservation, that is, the total flow into the vertex must equal the total flow out. In this problem, we'll consider a variant where vertices are allowed to "leak" so that some of the incoming flow is lost due to leakage.

A *leaky network* is the same as a standard flow network (that is, a digraph $G = (V, E)$, source s and sink t, and edge capacities $c(u, v)$ for each $(u, v) \in E$), but in addition, each vertex $v \in V$, is associated with a nonnegative *leak capacity*, denoted $h(v)$, which indicates the maximum leakage that can occur at this vertex (see Fig. 2(a)).

A flow f in a leaky network is valid if it satisfies the usual capacity constraint, but (other than s and t) the flow out of a vertex can be smaller than the flow in by up to $h(v)$, that is,

$$f^{\text{in}}(v) - h(v) \;\leq\; f^{\text{out}}(v) \;\leq\; f^{\text{in}}(v)$$

The value of a flow f is the flow out of the source, $f^{\text{out}}(s)$. (see Fig. 2(b)).)

(a) (10 points) Explain how to transform any leaky network G into an equivalent (non-leaky) network G' so that, for any valid leaky flow f in G there exists a flow f' of equivalent value in G', and vice versa. (Note: Your transformation is given G, the edge capacities, $c(u, v)$, and the leak capacities, $h(v)$, but not the flow.) Justify the correctness of your construction.

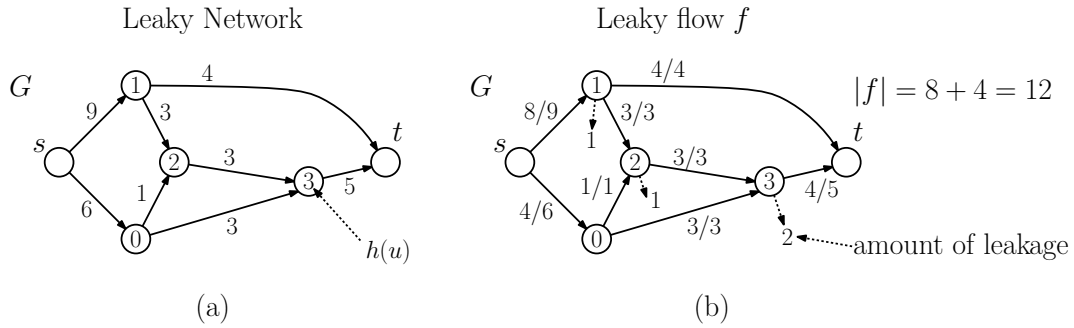

Figure 2: (a) A leaky network (each vertex u contains its leak capacity $h(u)$) and (b) a flow in the network (dashed lines indicate leak amounts).

(b) (5 points) Demonstrate how your transformation works on the following leaky network. (Just show the transformed non-leaky network. You do not need to show the flow.)

Problem 4. (25 points) This is a 1-dimensional variant of the paper pinning problem from Homework 2. You are given a collection of intervals $I = \{[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\}$ along the real line (see Fig. 3(a)). You may assume that the interval endpoints are distinct. Your objective is to place the minimum number of pins $X = \{x_1, x_2, \ldots, x_k\}$ to stab all of these intervals (see Fig. 3(b)). A pin x is said to *stab* an interval $[a_i, b_i]$ if $a_i \leq x \leq b_i$. (Note that you can place a pin through either of the interval endpoints.)

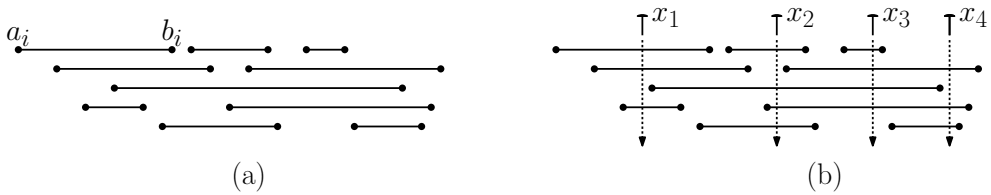

Figure 3: One-dimensional pinning.

(a) (5 points) Here is an obvious *depth-based greedy heuristic*. Place a pin that stabs the maximum number of intervals, remove all the intervals that this pin stabs, and then repeat on the remaining set of intervals.

Present a counterexample that shows that this depth-based greedy heuristic is *not* optimal. It suffices to give a drawing and a brief explanation. (Hint: There is an example involving 6 intervals where the optimum pinning set consists of 2 pins, but greedy generates 3 pins. Of course, any valid counterexample is acceptable.)

(b) (4 points) Given a set of n intervals I, let $\text{opt}(I)$ denote the size of the minimum stabbing set and let $\text{depth}(I)$ denote the number of pins generated by the above depth-based. Explain why the depth-based heuristic is essentially the same as the greedy set-cover heuristic. (What are the elements and what are the sets?) Given this, what can you infer about the approximation ratio $\text{depth}(I)/\text{opt}(I)$?

(c) (16 points) Devise an optimal greedy algorithm for this problem. Justify your algorithm's

correctness (both feasibility and optimality). (Note: If you don't see how to do this by a greedy approach, you can solve it through dynamic programming for half credit.)

The running time is not critical. We will give full credit to any polynomial time algorithm.

Problem 5. (20 points) In this problem, we will consider two variants of the typesetting problem. Recall that you are given a set of n words of lengths $W = \langle w_1, \ldots, w_n \rangle$, which we want to segment into lines. In each case, the objective is to minimize the maximum penalty, but penalties will be computed differently.

(a) (12 points) In typical typesetting systems, the words are distributed on a line so that the spaces between words are equal (see Fig. 4(a)). We will implement a simplification of this rule. Given a line of length L, define the *gap penalty* to the amount of unused space on the line, divided by the number of words on that line. For example, given a line of total length $L = 100$ with 3 words of total length 80 the gap penalty is $(100 - 80)/3 = 6.667$. Given a segmentation of words to lines, the *max-gap* is the maximum gap over all the lines (including the last line).

Present a DP formulation for an efficient algorithm that, given the line length L and word lengths $w[1..n]$, computes the optimum (smallest) max-gap. Briefly explain your formulation. (You don't need to implement your algorithm.)

You may assume that you have access to a function $\text{len}(i, i') = \sum_{j=i}^{i'} w_j$.

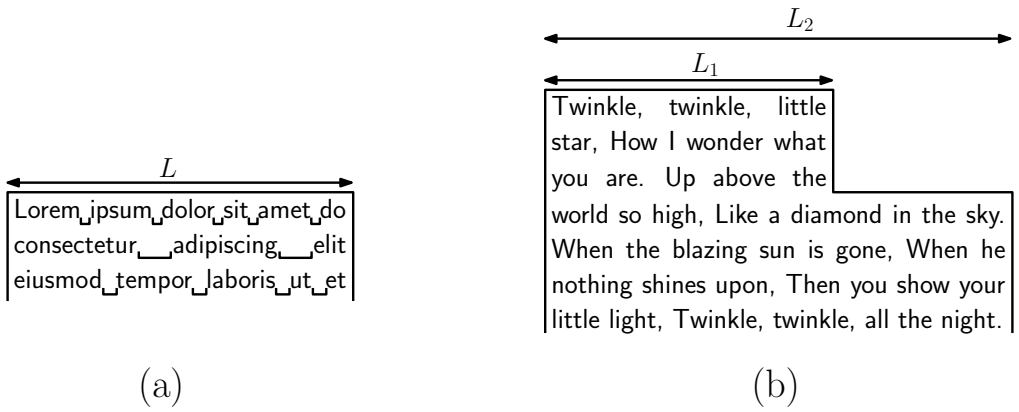

Figure 4: Optimal typesetting.

(b) (8 points) You are given two line lengths L_1 and L_2, where $L_1 \leq L_2$. The *first three lines* are typeset using the line length L_1, and the remaining lines are typeset using line length L_2.

Present a polynomial time algorithm for the same max-gap optimization problem as in part (a), but with the two line sizes. (Hint: Anything goes. You can modify your DP from (a) or design an entirely new approach, which may or may not use DP. Running time is a not a major issue. Any polynomial time algorithm can be given for full credit.) Explain your algorithm.