University of New Mexico
Department of Computer Science

Final Examination

CS 362, Data Structures and Algorithms II
Spring, 2021

Name:
Email:

- This exam lasts 4 hours. It is open book, notes and Internet. However, you are not allowed to discuss problems with any person.
- PLEASE: Start each main problem (i.e. Problems 1-5) at the top of a new page!
- Give yourself extra time to properly upload your solutions. Late exams may be penalized.
- *Show your work!* You will not get full credit if we cannot figure out how you arrived at your answer.

Question	Points	Score	Grader
1	20		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **Short Answer (2 points each)**

   Answer the following questions using the *simplest possible* Θ notation. Assume as usual, that $f(n)$ is $\Theta(1)$ for constant values of n.

   (a) Solution to the recurrence $T(n) = T(n-1) + 2$?

   (b) Expected number of nodes in a skip list storing n items?

   (c) Solution to the recurrence $T(n) = 4T(n/4) + n$?

   (d) Solution to the recurrence: $f(n) = 2f(n-1) - f(n-2) + 1$.

   (e) Time to determine if a graph with n nodes and m edges has a 5-clique?

(f) A stack has standard push and pop operations, and also a popLessThan(x) operation which repeatedly pops the top item off the stack until either the stack is empty or the top item on the stack has value at least x. Over n calls to push, pop and popLessThan, what is the worst case cost of any single call?

(g) In the problem above, what is the worst case *amortized* cost of any operation?

(h) Worst-case runtime of Kruskal's algorithm when using a union-find data structure where Make-Set, Find-Set and Union all have amortized cost $O(\log^2 n)$? Assume the graph has n nodes and m edges.

(i) A set of n people schedule vaccination appointments independently and uniformly at random on n different days. What is the expected number of pairs that will have appointments on the same day?

(j) What is the best time to solve the activity selection problem if the start and finish times of all activities are selected independently and uniformly at random from time 0 to some time T?

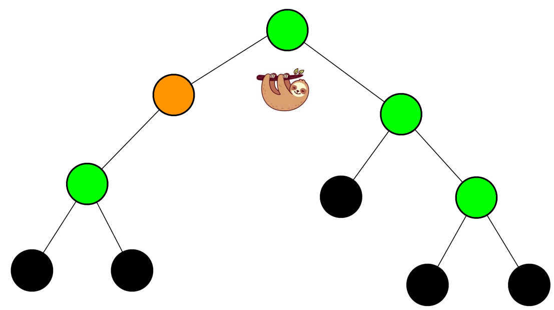
2. **Proof by Induction**

(a) (8 points) Consider the following recurrence relation: $f(1) = 2$ and $f(2) = 4$, and for all $n \geq 3$, $f(n) = f(n-1) * f(n-2)$. Prove by induction that $f(n) \geq 2^n$ for all $n \geq 1$.

(b) (12 points) Consider a new type of s-tree where every node is colored green, black, or orange, with the follow rules:

- A green node 2 children
- An orange node has 1 child
- A black node has no children.

Example:



Prove, by induction on the number of nodes, n, that this s-tree always has a number of black nodes that is 1 more than the number of green nodes. Don't forget to include the BC, IH, and IS.

3. **MIN-4COLORING** In the MIN-4COLORING problem, you are given a graph G. In a 4-coloring, each node of G is colored with one of 4 colors. An edge is called *satisfied* if both endpoints are colored differently. In MIN-4COLORING, you must output the minimum number of unsatisfied edges achievable by any 4-coloring.

   (a) (5 points) Show that MIN-4COLORING is NP-Hard by a reduction from one of the following: 3-SAT, VERTEX-COVER, SUBGRAPH-ISOMORPHISM, INDEPENDENT-SET, 3-COLORABLE, 4-COLORABLE, HAMILTONIAN-CYCLE, or CLIQUE.

   (b) (7 points) Let m be the number of edges in G. Consider the algorithm that colors each node independently with a color selected uniformly at random from the 4 colors. Compute the expected number of unsatisfied edges as a function of m for this algorithm using indicator random variables and linearity of expectation.

(c) (4 points) Use Markov's inequality and your result from part (b) to bound the probability that there are at least $m/3$ unsatisfied edges after running the algorithm in part (b).

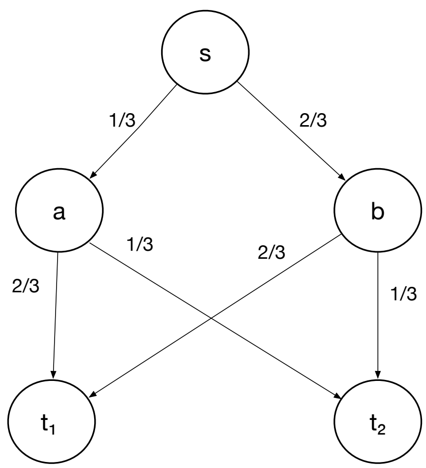
(d) (4 points) Using your result from part (c), bound the expected number of times you would need to run the algorithm in part (b) before you get a coloring that has less than $m/3$ unsatisfied edges. *Hint: Recall that for a random variable Y taking on positive integer values, $E(Y) = \sum_{i=1}^{\infty} Pr(Y \geq i)$ (see slides 18-19 from our "Randomized Data Structures" lecture).*

4. **Dynamic Programming and Graphs**

In this problem, you are given a directed graph with no directed cycles (often called a directed acyclic graph (DAG)). This graph $G = (V, E)$ has a special *source* vertex s with no incoming edges; and ℓ special *sink* vertices $t_1, t_2, \ldots, t_\ell$, all of which have no outgoing edges. All edges are weighted with probabilities that obey the following rules:

- For every edge $u \to v$: $0 \leq w(u \to v) \leq 1$.
- For every node u that is not a sink, the weights on the out edges sum to 1: $\sum_{u \to v \in E} w(u \to v) = 1$

The edge weights specify probabilities for a random walk that starts at the source vertex, s, and ends at one of the sink vertices. In this walk, when at any non-sink vertex u, the probability of leaving u along edge $u \to v$ is given by $w(u \to v)$. For example, in the graph below, when at s, the probability of going to a is $1/3$ and the probability of going to b is $2/3$.



(a) (5 points) Describe and analyze an algorithm to compute a *least likely* path from s to any sink vertex. In the example graph, your algorithm should return the path s, a, t_2. What is the runtime of your algorithm as a function of n, the number of nodes in the graph and m, the number of edges?

(b) (5 points) Describe and analyze an algorithm to compute a *most likely* path from s to any sink vertex. In the example graph, your algorithm should return the path s, b, t_1. What is the runtime of your algorithm as a function of n, the number of nodes in the graph and m, the number of edges?

(c) (10 points) Describe and analyze a dynamic program to calculate, for each sink vertex, the probability that the random walk reaches that vertex. For example, in the figure, the probability of visiting a is $1/3$ and b is $2/3$. Thus, the probability of reaching t_1 is $1/3*2/3+2/3*2/3 = 2/3$; and the probability of reaching t_2 is $1/3*1/3+2/3*1/3 = 1/3$.

5. **Dynamic Programming - Hard**

Assume you are given a string of n numbers $S = s_1, s_2, \ldots s_n$. You want to find the smallest product of any non-empty substring of these numbers. For example if you are given $S = 1, 4, -1, -5, 2, 0, -8, -2, 2, 1$ then you should return the value -10 which is the product of the substring $-5, 2$.

Clearly, you can solve this with a brute-force algorithm in $O(n^2)$ time. To achieve $o(n^2)$ requires clever dynamic programming. You will use 3 functions, defined as follows:

- $minEndingAt(i)$ returns the smallest product of any substring ending at index i
- $maxEndingAt(i)$ returns the largest product of any substring ending at index i
- $minUpTo(i)$ returns the smallest product of any substring ending at index no more than i

(a) (6 points) Fill in the remaining values for the example sequence in the table below.

S	1	4	-1	-5	2	0	-8	-2	2	1
minEndingAt	1	4	-4	-5	-10					
maxEndingAt	1	4	-1	20	40					
minUpTo	1	1	-4	-5	-10					

(b) (12 points) Write recurrence relations for minEndingAt(i), maxEndingAt(i) and minUpTo(i) that can be calculated efficiently to fill in a table like the one above.

(c) (2 points) In 2 sentences describe an efficient dynamic program that uses your recurrences to solve this problem. What is the run-time of your dynamic program?