CS161 Final Exam

Once you turn this page, your exam has officially started!

You have three (3) hours for this exam.

Instructions:

- This is a **timed**, **closed-book** exam.
- You must complete this exam within **180 minutes** of opening it.
- You may use two two-sided sheets of notes that you have prepared yourself. **You may not use any other notes, books, or online resources. You may not collaborate with others.**
- **If you have a question about the exam:** Try to figure out the answer the best you can, and clearly indicate on your exam that you had a question, and what you assumed the answer was.
- You may cite any result we have seen in lecture without proof, unless otherwise stated.
- This exam is printed two-sided. There are blank pages at the end for extra work. Do NOT tear off or unstaple the pages.
- **Please write your name at the top of all pages.**

Advice: If you get stuck on a problem, move on to the next one. Pay attention to how many points each problem is worth. Read the problems carefully.

Honor code: The following is a statement of the Stanford University Honor Code.

By signing your name below, you acknowledge that you have abided by the Stanford Honor Code while taking this exam.

Signature: ______________________________

Name: ______________________________

SUNetID: ______________________________

1 Multiple Choices (26 pt.)

For the following problems, fill in the circle(s) for the correct choice(s). No explanation is needed.

1.1 (6 pt.) Looking Back :)

Which of the following is true? **Select all that apply.**

- (A) A divide and conquer algorithm can always use memoization to reduce runtime at the cost of increased memory usage.
- (B) A randomized algorithm's expected runtime will always be faster than or equal to its worst-case runtime.
- (C) For a problem that can be solved using a greedy algorithm, making any choice other than those of the algorithm leads to a non-optimal solution.
- (D) A top-down dynamic programming (DP) algorithm is similar to a bottom-up DP algorithm, except the bottom-up version stores the results of previously encountered sub-problems, while the top-down version does not.

1.2 (6 pt.) Hashing

Let U be a finite set and n be a positive integer. Let S be the set of all functions with domain U and range $\{0, 1, 2, \cdots, n-1\}$. Assume $|U|$ is much much larger than n. Suppose that we want to make a hash table with S as our hash family. Which of the following is true? **Select all that apply.**

- (A) S is a universal hash family.
- (B) When we choose a hash function $\mathcal{H}$ from S at random, there always exist two elements that would be hashed into the same bucket by $\mathcal{H}$.
- (C) It takes $O(|U|)$ bits to store a hash function in S.
- (D) Assume that we need to hash n elements in total and looking up each hash value takes constant time. In the hash table constructed using a random function from S, the operations Insert, Delete, and Search would run in expected $O(1)$ time.

1.3 (4 pt.) Climb Stairs

Consider the CLIMB-STAIRS (CS) problem: there are n stairs to climb. Return the number of ways there are for a person standing at the bottom to climb to the top if they can climb either 1, 2, or 3 stairs at a time. What would be the recurrence relation for CS(n), assuming that $n \geq 3$? **Select the single correct answer.**

- (A) $\text{CS}(n) = \text{CS}(n-1) + \text{CS}(n-2) + \text{CS}(n-3)$
- (B) $\text{CS}(n) = \max\{\text{CS}(n-1), \text{CS}(n-2), \text{CS}(n-3)\}$
- (C) $\text{CS}(n) = \text{CS}(n-1) + \text{CS}(n-2) + \text{CS}(n-3) + 1$

(D) $\text{CS}(n) = \max\{\text{CS}(n-1), \text{CS}(n-2), \text{CS}(n-3)\} + 1$

1.4 (6 pt.) Negative Edge Weights

Which of the following graph algorithms can return the correct result on a graph with potentially negative edge weights (but there are no negative cycles)? Assume that all four algorithms can take weighted graphs as valid inputs. **Select all that apply.**

(A) DFS for finding all vertices that are connected to a vertex in an undirected graph

(B) Dijkstra's algorithm for finding the shortest path from a vertex to all other vertices in a directed graph

(C) Floyd-Warshall for finding All-Pairs Shortest Paths in a directed graph

(D) Kruskal's algorithm for finding a Minimum Spanning Tree in an undirected graph

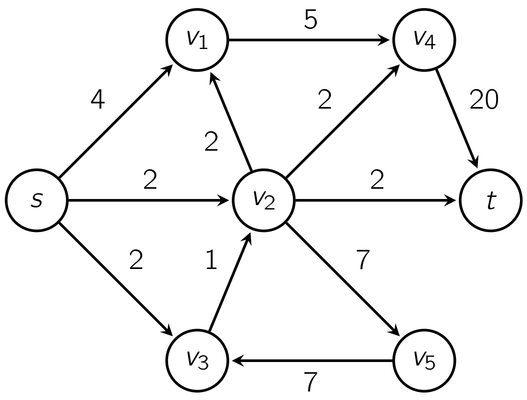
1.5 (4 pt.) Flows and Cuts

Consider a graph $G = (V, E)$ with a capacity c_e for every $e \in E$. Suppose there exists a flow from s to t (not necessarily the maximum) of value 5. What do we know about the size of the **minimum** s-t cut (denoted by $|C|$) in G? **Choose the single correct answer.**

(A) $|C| \leq 5$ (B) $|C| = 5$ (C) $|C| \geq 5$ (D) None of the above

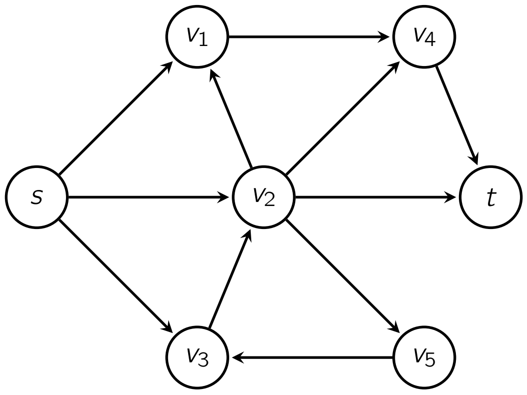
2 Short Answer (30 pt.)

2.1 (6 pt.)



Above you see a graph with edge capacities indicated next to each edge. Find a maximum flow in the above graph from s to t. You may fill out the flow on every edge in the below picture.

[We are expecting: Fill in the graph below with a maximum flow.**]**



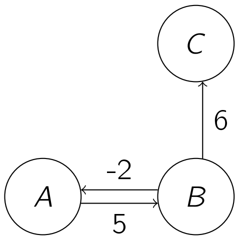
2.2 (6 pt.)

Suppose we are given an undirected, weighted, and connected graph G with n vertices and m edges. We wish to store the graph so that we can use Prim's algorithm to return the minimum spanning tree of the graph in the best possible asymptotic runtime (that is achievable by Prim's algorithm specifically). Do we want to use an adjacency list representation for our graph or an adjacency matrix representation? Justify your choice.

[We are expecting: A choice of representation and a short explanation**]**

2.3 (6 pt.)

Suppose we have a directed graph with n vertices and m edges. You start at some specified vertex, and each time cross an edge going out of the current vertex. Each edge has an integer dollar amount attached, such that your earnings increase by that dollar amount if you traverse the edge (or decrease if the dollar amount is negative). If you traverse an edge multiple times, your earnings increase by the dollar amount multiple times. For example, in the graph below, if you start at A, then move to B, then move back to A, then move back to B, then move to C, you will earn $5 + (-2) + 5 + 6$ dollars.



You want to know whether your earnings are bounded (i.e., there is a path that earns the maximum amount of money, beyond which you cannot earn more). More specifically, you want an algorithm that can take in any graph and will return true if your earnings are bounded and false if there is the potential to earn unlimited money. Explain an algorithm to do this in $O(mn)$ time.

[We are expecting: An English description of your algorithm, likely referencing an algorithm from class, and a brief explanation of why it works**]**

2.4 (6 pt.)

Suppose we have a directed acyclic graph with n vertices and m edges where each vertex has an associated integer score. We also care, however, about a vertex's descendantScore, which is the highest score of the vertex itself or of any of its descendants (i.e., the vertices it has a path to). Explain how to calculate the descendantScore of all the vertices in our graph in $O(n + m)$ time.

[We are expecting: Pseudocode OR a brief English description of your algorithm. You do not need to justify the runtime.**]**

2.5 (6 pt.)

Suppose for a given graph, we have access to a second graph representing the DAG of the strongly connected components of the first graph. We also have access to a function getSCC, which takes as input a vertex in the first graph and returns the vertex corresponding to its strongly connected component in the second graph. The function runs in $O(1)$ time. If there are b strongly connected components, devise an $O(b^2)$ algorithm that takes as input vertices u, v in the original graph and identifies whether there exists a path from u to v.

[We are expecting: Pseudocode OR a brief English description of your algorithm, as well as a brief justification of runtime. You do not need to justify the runtime of any algorithms from the lecture.**]**

3 Greedy for Compensation (49 pt.)

3.1 (8 pt.)

Lucky recently discovered an opportunity in the psychology department, where students are paid to participate in studies. There are n ongoing studies numbered from 1 to n, with no new studies starting. Today is day 0, and each study i will end on day $d_i \geq 0$. Lucky is limited to participating in each study only once and has the flexibility to choose which day to participate, up to and including the study's end date. Each study requires only one day of participation, and Lucky can participate in only one study per day.

Payouts can differ between studies. Study i pays out $p_i \geq 0$ dollars. Lucky's goal is to maximize the total payout. Lucky has devised the following greedy strategy: every day, Lucky participates in the remaining study with the highest payout and adds that to his total. A pseudocode implementation of this strategy is shown below.

Algorithm 1: Lucky's algorithm

Input: A set of studies studies $= \{(p_i, d_i) \mid 1 \leq i \leq n\}$, where p_i is the payout and d_i is the end day for study i.

Output: Lucky's total payout.

```
payout ← 0
day ← 0
sortedStudies ← studies sorted by decreasing order of payout
for $(d, p) \in$ sortedStudies do
    if day $\leq d$ then
        day ← day + 1
        payout ← payout + $p$
return payout
```

Prove that Lucky's strategy may not result in the maximum payout.

[We are expecting: A concise proof by counterexample which provides a list of (d_i, p_i) pairs on which the algorithm does not achieve maximum payout.**]**

3.2 (25 pt.)

Plucky suggests a fix to Lucky's algorithm. Plucky's strategy still processes studies in decreasing order of payout but schedules each at the last available day possible rather than the first. A pseudocode implementation of this strategy is shown below.

Algorithm 2: Plucky's algorithm

Input: A set of studies studies $= \{(p_i, d_i) \mid 1 \leq i \leq n\}$, where p_i is the payout and d_i is the end day for study i.

Output: Maximum payout from studies.

```
payout ← 0
studySchedule ← an array of length $\max\{d_1, \ldots, d_n\} + 1$ with all entries set to “available”
sortedStudies ← studies sorted by decreasing order of payout
for $(d, p)$ ∈ sortedStudies do
    for day = $d, d-1, \ldots, 0$ do
        if studySchedule[day] = “available” then
            studySchedule[day] ← “booked”
            payout ← payout + $p$
            break // from the inner for loop
return payout
```

Prove that Plucky's algorithm correctly returns the maximum payout Lucky can achieve.

[We are expecting: A formal proof of correctness.**]**

Extra space for problem 3.2. Questions continue on the next page . . .

3.3 (16 pt.)

Lucky has resolved to be a more ethical lemur. Lately, he's been reading up on utilitarianism, and decides that rather than pursuing only his own profit, he should instead participate in the studies that are most valuable by utilitarian standards.

3.3.1 (4 pt.)

How would a utilitarian calculate the value of each study? Would Lucky's personal payout (assuming he participates) factor into the calculation? Why or why not?

[We are expecting: (1) a 1 sentence description, in general terms, of how a utilitarian calculates value, and (2) a 1 sentence explanation of whether and why Lucky's personal payout does or does not factor into the calculation**]**

3.3.2 (4 pt.)

What changes would Lucky need to make to Plucky's algorithm in order to return maximum utilitarian value rather than maximum personal payout?

[We are expecting: 2-3 sentences describing the ways in which Plucky's algorithm would need to change to output maximum utilitarian value**]**

3.3.3 (4 pt.)

Why might it be difficult to actually do the kind of utilitarian calculation you described in 3.3.1? Please support your answer by giving at least one specific example of something relevant to the utilitarian calculation that could be difficult to measure.

[We are expecting: 2-3 sentences clearly describing a measurement-related barrier to doing the utilitarian calculation, either focusing on or supported by a specific example of a difficult-to-measure quantity that the calculation relies on**]**

3.3.4 (4 pt.)

Suppose that one of the studies violates the privacy rights of its participants, although the participants never find out about this violation. Would this fact affect a utilitarian's evaluation of the study (assuming the utilitarian was aware of it)? Why or why not?

[We are expecting: 2-3 sentences explaining why this fact is or is not relevant to the utilitarian's evaluation**]**

4 Optimal Matrix Multiplication (35 pt.)

We consider the problem of deciding multiplication order for matrix multiplication. In particular, given n matrices $A_0, A_1, \ldots, A_{n-1}$, we want to decide in which order to multiply them, to take the least time. For this problem, we assume the matrix dimensions are given as an $(n+1)$-sized array d, where the dimensions of the matrix A_i are $d[i] \times d[i+1]$ for $i \in \{0, 1, \ldots, n-1\}$. Assume that it takes $i \cdot j \cdot k$ time to multiply two matrices of sizes $i \times j$ and $j \times k$; the resulting matrix is going to be of size $i \times k$.

For example, for $d = [1, 2, 3, 1]$, we have three matrices A_0, A_1, A_2 with dimensions 1×2, 2×3, 3×1, respectively. There are two possible ways of multiplying these three matrices which can be shown as $((A_0A_1)A_2)$ and $(A_0(A_1A_2))$. In other words, we could multiply the first two matrices first and the result with the third, or multiply the last two matrices first. It takes 9 time for the first way and 8 time for the second way, so we prefer the second way.

4.1 (5 pt.)

Find the optimal time (and how to achieve this time) for multiplication in the following examples:

Example 1: $d = [1, 10, 1, 10, 1]$ Example 2: $d = [1, 10, 1, 10, 10, 1]$

[We are expecting: Optimal time and order of multiplications for each of the examples (you can use parentheses to indicate the order).**]**

4.2 (15 pt.)

Let us define $M(i, j)$ as the optimal time to multiply the matrices between i and j (both inclusive), i.e., the optimal time to compute $A_iA_{i+1} \cdots A_j$. Write a recursive relation for $M(i, j)$.

[We are expecting: Recursive formulation and 2-3 sentences of justification.**]**

4.3 (15 pt.)

Write an algorithm, with a runtime of $O(n^3)$, to find the optimal time to multiply our n matrices.

[We are expecting: Pseudocode AND English description, justification for runtime, and proof of correctness.**]**

5 Graph Mania! (40 pt.)

For each of the following algorithm design tasks, please provide the following:

- A clear English description of how you would do the task, and
- A justification for why your proposed algorithm meets runtime requirements.

In this problem, all graphs will have n vertices and m edges.

Please note the following:

- Pseudocode is not required, but please feel free to include it to make your algorithm clearer. **It is not necessary to provide a formal proof of correctness.**
- Any result or algorithm seen in class is fair game to cite without justification.
- If you want to do a graph task using a graph algorithm from class, be very explicit about which algorithm/procedure you are using, what task you want to accomplish, and what your input/output is.
- All runtimes are worst-case, deterministic runtimes.
- Space is not constrained, but do not use prohibitively large amounts of space.
- It is not necessary, but if it helps, you may assume you can easily access any node's neighbors in constant time (in a directed graph, incoming and outgoing neighbors).

5.1 (10 pt.)

We define the **diameter** of a strongly connected directed graph $G = (V, E)$ as the largest distance between any two vertices in V. That is,

$$\text{diameter}(G) = \max\{\text{distance}(u, v) \mid u, v \in V\}.$$

Recall that distance(u, v) is the cost of the shortest directed path from u to v.

Take $G = (V, E)$ to be a weighted strongly connected directed graph. G may have negative edges, but you are guaranteed that G contains no negative cycles. **Design an algorithm that finds the diameter of G in time $O(n^3)$.**

[We are expecting: A clear English description (pseudocode optional) AND a brief justification of runtime.**]**

5.2 (13 pt.)

Let $G = (V, E)$ be a directed unweighted graph (i.e., every edge has weight 1). Among a set of vertices S, we say $v \in S$ is the closest to u if, among those vertices in S that are reachable from u, vertex v is the one reachable via a path of the shortest total cost. **Given two distinct vertices $x, y \in V$, design an algorithm that finds the closest vertex to x that is also reachable by y. Your algorithm must run in $O(m)$ time.**

Please note the following:

- It may be the case that the closest vertex to x reachable by y is actually x itself.
- You may assume that at least one vertex is reachable by both x and y.
- You may assume that $m > 0$.

[We are expecting: A clear English description (pseudocode optional) AND a brief justification of runtime.**]**

5.3 (17 pt.)

Let $G = (V, E)$ be a directed unweighted graph. We say G is "somewhat-connected" if for every pair of vertices $u, v \in V$, either there is a directed path from u to v in G, there is a directed path from v to u in G, or both.

Design an algorithm to determine if G is "somewhat-connected" in $O(m + n)$ time.

[We are expecting: A clear English description (pseudocode optional) AND a brief justification of runtime.**]**

This is the end!

Left intentionally blank as scratch paper or for extra space for any question. Please indicate in the relevant problem if you have work here that you want graded, and label your work clearly.

Left intentionally blank as scratch paper or for extra space for any question. Please indicate in the relevant problem if you have work here that you want graded, and label your work clearly.

Left intentionally blank as scratch paper or for extra space for any question. Please indicate in the relevant problem if you have work here that you want graded, and label your work clearly.