Practice Problems – CS 6515/4540 (Fall 2025)

Collection of problems for students to practice

1 Short Answer Practice

Unless explicitly mentioned, a short 1–2 line justification is required.

1.1

T/F Consider computing the nth Fibonacci number $F(n)$ defined by $F(0) = 1$, $F(1) = 1$ and $F(i) = F(i-1) + F(i-2)$ for all integers $i \geq 2$. We need $\Theta(n)$ space to compute $F(n)$ since we have to store the entire DP table.

Assume integers take $O(1)$ space to represent.

1.2

We have $f' \in O(f(n))$ and $g' \in O(g(n))$. Then $\frac{f'}{g'} \in O(\frac{f(n)}{g(n)})$?
What about taking θ instead of Big-O?

1.3

Given a markov decision process, the probability of going from one state s to another state s' given action a, $P_a(s, s') \in \{0, 0.5, 1\}$, is it possible to solve for $V^*(s,t)$ in $o(|S|^2|A|T)$ time.

1.4

True or false: The median finding algorithm with $M = \sqrt{n}$ will result in an $O(n)$ algorithm.

1.5

True or false: There exists a polynomial time algorithm for knapsack where for all jobs i, $v_i \in \{1, 2, 3, \ldots, 10\}$

1.6

True or False: If every edge weight in a connected, undirected graph is distinct, then the Minimum Spanning Tree (MST) of the graph is unique.

1.7

True/False Consider a matching M on a graph G that only has augmenting paths of length 11 or greater. Every path or cycle in the symmetric difference between M and M', the maximum matching, must have the ratio of blue edges (M) to red edges (M') to be $\geq \frac{5}{6}$

2 Divide and Conquer

The Strassen algorithm to multiply two $n \times n$ matrices utilizes seven (7) multiplications of $\frac{n}{2} \times \frac{n}{2}$ sized matrices and some constant number of matrix additions. If adding two $n \times n$ sized matrices takes $O(n^2)$ time, find the time complexity of the Strassen algorithm.

(a). State your recurrence for the runtime of Strassen's algorithm.

(b). Is the time complexity $O(n^{2.5})$?

(c). Is it $o(n^3)$?

3 Dynamic Programming

3.1

Assume you are given two integer arrays called $arr1$ of length m and $arr2$ of length n. Design a dynamic programming algorithm to find the longest non-mod-2 subsequences between the two arrays. The longest Non-Mod-2 subsequences are the longest equal-length subsequences S of $arr1$ and T of $arr2$ such that $(S[i] \mod 2) \neq (T[i] \mod 2)$ $\forall i$. The running time of your algorithm should be $O(n^2)$.

Example Input:
$arr1 = [5, 7, 6, 4, 1, 8, 9]$ and $arr2 = [3, 10, 7, 9, 4]$.

Example Output: 4
This is because $S = [5, 6, 8, 9]$ and $T = [10, 7, 9, 4]$. There are multiple non-mod-2 subsequences of length 4. But, $S = [7, 6, 4, 8]$ and $T = [10, 7, 9, 4]$ is not a potential answer because for index $i = 3$, $(4 \mod 2) = (8 \mod 2)$.

1. Define the entries of your table in words. E.g. $T(i)$ or $T(i, j)$ is ...
2. State recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct.
3. Analyze the running time of your algorithm.

3.2

You are given two integer arrays of the same length $present[0, 1, ...n - 1]$ and $future[0, 1, ..., n - 1]$ where $present[i]$ and $future[i]$ represent the present and future price of a stock i. You can buy and sell a stock at most once, and you are given a *budget* amount to maximize your profits. Provide an algorithm to maximize your profits under these constraints. Reason why your algorithm is correct and also provide its run-time analysis.

Example Input: present = [5,4,6,2,3], future = [8,5,4,3,5], budget = 10
Example Output: 6

Explanation:
One possible way to maximize your profit is to:
Buy the 0th, 3rd, and 4th stocks for a total of 5 + 2 + 3 = 10.
Next year, sell all three stocks for a total of 8 + 3 + 5 = 16.
The profit you made is 16 - 10 = 6.
It can be shown that the maximum profit you can make is 6.

1. Give your recurrence equation. Briefly explain why your algorithm works. Also write your algorithm.
2. Analyze its runtime.

3.3

Two arrays X and Y are considered *close* if for all i, $|X[i] - Y[i]| \leq 1$ and $|X| = |Y|$

Given two integer arrays A and B of length n find the length of the longest subsequence C and D of arrays A and B respectively such that C and D are *close*.

Array C is a subsequence of A if there exists a set of sorted indices, I such that $C[i] = A[I_i]$ for all $i \in \{1, 2, \ldots, \text{len}(C)\}$.

Your algorithm should run in $O(n^2)$ time.

4 Graph Algorithms

4.1

Given an $m \times n$ binary matrix mat, write an algorithm to return an updated matrix with the distance of the nearest 0 for each cell. Briefly describe why your algorithm is correct and also provide an analysis of its run-time complexity.
A cell is said to be at a distance of 1 to another cell if they are adjacent (i.e share a side). Note that diagonally connected cells are not at a distance of 1.

Example 1

$$\text{Input: } \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{Output: } \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Example 2

$$\text{Input: } \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \text{Output: } \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

4.2

Given a graph of $G = (V, E)$ where edges are colored red and blue. Among all paths from $s \in V$ to $t \in V$ with the minimum number red edges, find the path with the minimum number of blue edges.

4.3

In class we proposed the Ford Fulkerson algorithm for max-flow of a flow network. The idea is to continue finding some augmenting path P in the residual graph[1] and updating the flow such that one of the edges on P is saturated[2]. However, the algorithm for finding the augmenting path is not specified.

In the Edmonds-Karp algorithm, a variant of Ford-Fulkerson, we utilize BFS to find an augmenting path in the residual graph. It turns out that this algorithm terminates in polynomial time. It can be shown that for two vertices u and v in the network, the directed edge (u, v) never gets saturated in the residual graph more than $|V|/2$ times.

[1] Keep in mind the number of edges in the residual is twice that of the original network

[2] in class we used augmenting paths to increase the total flow 1 unit at a time. Instead, increase the flow as much as the augmenting path allows, i.e. increment it by the capacity of the lowest capacity residual edge on the augmenting path

Using the above, show that the Edmonds-Karp algorithm terminates in polynomial time. Provide the time complexity in terms of the number of vertices $|V|$ and edges $|E|$

4.4

Compute at least five (5) different augmenting paths of the matching shown below

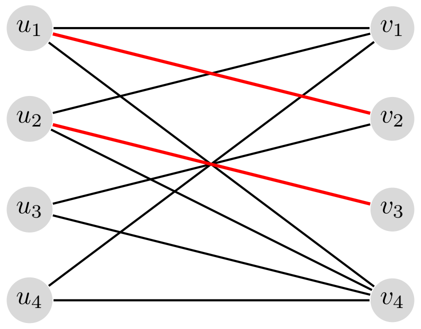

Figure 1: A bipartite graph with a matching highlighted in red.

4.5

We are given a bipartite graph $G = ((A, B), E)$ where the two parts are A and B, and the edge set is E.

1. Prove by contradiction that if G satisfies $|\mathsf{Nbr}(S)| \geq |S|$ for every $S \subseteq A$ (i.e., the number of neighbors of every subset S of A is at least the size of S), then any maximum matching M matches every vertex of A.

   Hint: Assume that the maximum matching M leaves some vertex $x \in A$ unmatched. Now consider all alternating paths starting at x and argue that we can get an augmenting path, hence a contradiction since the size of M can be increased by 1.

2. Suppose we are given a bipartite graph where every vertex has the same degree d (some positive integer like $d = 10$). Use the above first part to deduce that this graph has a perfect matching.

4.6

Let $G = (V, E)$ be a directed graph where every edge has capacity 1. For some given vertices $s \neq t \in V$, let let f be an integral st-max-flow vector (i.e., integer f_e denotes the flow through edge $e \in E$).

1. Some edge $e \notin E$ is added to the graph with capacity 1, so the flow f might no longer be a valid maximum flow. Construct an $O(|V| + |E|)$-time algorithm that, when given G, s, t, f, e, returns a new st-maximum-flow f' that is valid on graph $G' = (V, E \cup \{e\})$.

2. Some edge $e \in E$ is deleted from the graph, so the flow f might no longer be a valid maximum flow. Construct an $O(|V| + |E|)$-time algorithm that, when given G, s, t, f, e, returns a new st-maximum-flow f' that is valid on graph $G' = (V, E \setminus \{e\})$.

(For both parts, give algorithm description, proof of correctness, and complexity analysis.)