EECS 376: Foundations of Computer Science **Due 8:00pm, February 7**
University of Michigan, Winter 2024 **(accepted until 9:59 pm, no credit after)**

Homework 4

This homework has 8 questions, for a total of 100 points and 8 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up.*

(0 pts) 0. **Before you start; before you submit.**

If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework; you may also find the video "walkthroughs" on Canvas helpful. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

Solution:

2. **Keeping things fresh with potpourri.**

(3 pts) (a) Consider the following function, which takes as input non-negative integers a and b that are powers of two.

```
function ALG($a$,$b$)
    if $a = 1$ or $b = 1$ or $a = b$ then return 0
    if $a > b$ then return ALG($a/2, 2b$)
    if $b > a$ then return ALG($2a, b/2$)
```

Either find a valid potential function to prove that ALG halts on all valid inputs a, b, or show that no such function exists by giving an input on which ALG runs forever.

Solution:

(3 pts) (b) Let $T(n)$ be the running time of the following function on an array of n entries. Write a recurrence for $T(n)$, and *briefly* justify your answer.

```
function FOO($A[1, \dots, n]$)
    if $n = 1$ then return $A[1] + 1$
    $x = 2 \cdot \textsc{Foo}(A[1, \ldots, n/2])$
    $y = 3 \cdot \textsc{Foo}(A[n/2+1, \ldots, n])$
    $tmp = 0$
    for $i = 1, \ldots, n$ do
        for $j = 1, \ldots, n$ do
            $tmp \leftarrow tmp + A[i] + A[j] - x + y$
    return $tmp$
```

Solution:

(4 pts) (c) Suppose there are algorithms A and B, where $T_A(n)$ and $T_B(n)$ are respectively the worst-case running times of A and B on inputs of size n. Suppose that T_A satisfies $T_A(n) = 2T_A(n/2) + T_B(n)$, and $T_B(n) = O(n^2)$ but $T_B(n) \neq O(1)$. Select all bounds for $T_A(n)$ that *could possibly hold* (for some choice of $T_B(n)$ satisfying the above bounds):

A. $\Theta(n^3)$
B. $\Theta(n^2)$
C. $\Theta(n \log n)$
D. $\Theta(n)$
E. $\Theta(1)$

Choose one answer you selected as possible, and give a specific choice of $T_B(n)$ that would yield that result. You do not need to provide any justifications, other than why your choice of $T_B(n)$ yields the claimed solution for $T_A(n)$.

Solution:

3. **Counting certain cupcakes.**

Lily has started a new job at Bakehouse 46 in Downtown Ann Arbor! Her responsibilities in opening up the store everyday include placing the store's cupcakes in the display case. The cafe makes several different flavors of cupcakes, and various numbers of each flavor. The manager requires that for each flavor, either *all* or *none* of the cupcakes of that flavor are displayed. Lily wants to determine whether it is possible to *completely* fill the display case with cupcakes while following this rule.

We formalize the problem as follows. We are given an array of non-negative integers $S[1, \ldots, n]$, where each $S[i]$ represents the number of cupcakes of the ith flavor, and a non-negative integer K representing the total number of cupcakes the display case can hold. We wish to determine whether there is a set of flavors that have exactly K cupcakes in total.

Given the array S, for suitable integers i, m (in ranges for you to determine) define $D(i, m)$ to be the Boolean value indicating whether there a subset of the first i flavors having exactly m cupcakes.

(10 pts) (a) Derive a recurrence relation, including base case(s), for $D(i, m)$ that is suitable for a dynamic-programming solution to this problem. Briefly justify its correctness.

Solution:

(5 pts) (b) Give pseudocode for a (bottom-up) dynamic programming algorithm that solves this problem, and analyze its running time.

Solution:

4. **Dynamic programming for dynamic shortest paths.**

Graphs that arise in real applications are often not fixed, but can change over time; such graphs are called "dynamic." (This is a totally different use of the word "dynamic" than in "dynamic programming". Computer scientists sometimes name things in confusing ways!) For example, roads and intersections can be added to or deleted from the road network, computers can be added to or removed from the Internet, and people can (un)follow each other on social networks. For many problems of interest, there are algorithms for dynamic graphs that are faster than just re-computing answers "from scratch" whenever the graph changes. In this problem you will give such algorithms for shortest-path problems.

Let $G = (V, E)$ be a weighted directed graph with $n = |V|$ vertices, where the weight of each edge (u, v) is denoted $\ell(u, v)$. (There is no negative-weight cycle in G.) Suppose that we have already computed the all-pairs distance table D_G of G. That is, for each vertex pair $u, v \in V$, $D_G(u, v)$ stores the distance (i.e., the length of a shortest path) from u to v in G.

Now, a new vertex v_{new} is added to the graph, together with its incident edges. Let $G_{new} = (V \cup \{v_{new}\}, E \cup E_{new})$ denote the updated graph, where E_{new} consists of all the incoming and outgoing edges for v_{new}. (There is no negative-weight cycle in G_{new}.) We aim to compute the updated distance table $D_{G_{new}}$ of G_{new}.

A straightforward approach is to compute $D_{G_{new}}$ from scratch using the Floyd-Warshall algorithm, in $\Theta(n^3)$ time. But we want to do better by exploiting the fact that we already have D_G. In this problem, you will obtain a faster $O(n^2)$-time algorithm.

(20 pts) (a) Write expressions for $D_{G_{new}}(v_{new}, u)$ and $D_{G_{new}}(u, v_{new})$ that hold for all $u \in V$, where the expressions are in terms of the already-known quantities $D_G(y, z)$ for $y, z \in V$, and $\ell(y, z)$ for $y, z \in V \cup \{v_{new}\}$. Evaluating your expressions should take $O(n)$ time for each vertex u, for a total of $O(n^2)$ time. Justify the correctness of your expressions and the evaluation time.

Hint: Consider why the Bellman-Ford algorithm is correct.

Solution:

(10 pts) (b) Write an expression for $D_{G_{new}}(u, v)$ that holds for all $u, v \in V$, where the expression is in terms of the already-known quantities listed in the previous part, as well as the quantities you computed in the previous part. Evaluating your expression for all $u, v \in V$ should take a total of $O(n^2)$ time. Justify the correctness of your expression and the evaluation time.

Observe that by combining the two parts, we have computed the entire new distance table $D_{G_{new}}$ of G_{new} in $O(n^2)$ time.

Hint: Consider why the Floyd-Warshall algorithm is correct.

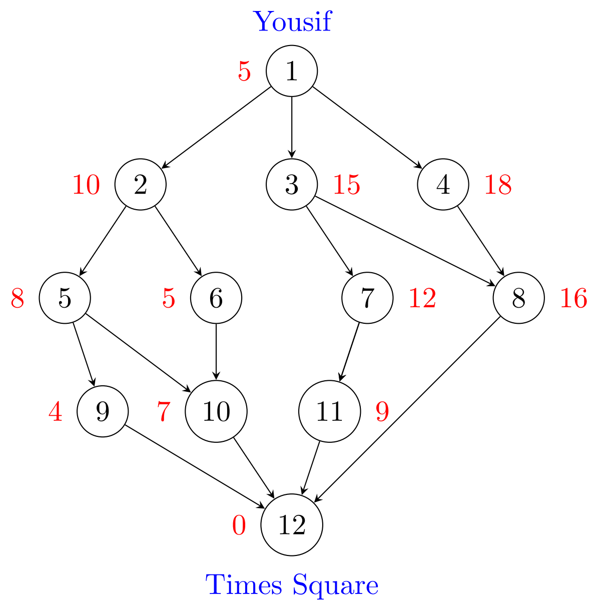
Solution:

5. **Maximizing money made in Manhattan.**

To manage the surge in New Year's Eve tourism, the New York City government has implemented traffic control measures: all roads have been made one-way; to prevent accumulating traffic, all intersections are reachable and there are no loops or dead ends (except at Times Square); and all tourists must use ride-shares. As the midnight countdown approaches, tourists at various intersections are "bidding" on rides to Times Square to witness the iconic Ball Drop, by posting the prices they are willing to pay for rides.

Yousif, a limousine driver, will drive through the city to Times Square, picking up all the tourists at each intersection along the way. (His limousine has unlimited passenger capacity and fuel.) Yousif's goal is to find a route to Times Square that maximizes the total money he earns.

We model this problem mathematically as follows (see the graph below for an example). The city's road network is given by a directed acyclic graph (DAG) $G = (V, E)$ with n vertices given in topological order as $V = \{1, \dots, n\}$ (representing intersections), and m directed edges $(i, j) \in E$ where $i < j$ (representing road segments). Every vertex $i < n$ has at least one outgoing edge (there is only one dead end), and every vertex $j > 1$ has at least one incoming edge (all intersections are reachable). For each vertex i there is an associated price $p_i \geq 0$ that the tourists at intersection i are willing to pay (in total). Yousif wishes to find a route from vertex 1 (his starting location) to vertex n (Times Square) that maximizes the total price along the route.



(2 pts) (a) Consider the following "greedy" algorithm for this problem: at each step, move to an adjacent vertex that has the maximum price, until arriving at Times Square. What total

price will this algorithm yield for the above graph? (The price for each vertex is shown next to it, in red.) Is it optimal, and why or why not?

Solution:

(4 pts) (b) Briefly but clearly describe a brute-force algorithm for this problem, which may run in exponential time in the number of vertices n. You do not need to give pseudocode, justify correctness, or analyze the running time.

Solution:

(5 pts) (c) Give, with justification and base case(s), a recurrence relation that is suitable for a dynamic programming solution to this problem.

Solution:

(7 pts) (d) Give a (bottom-up) dynamic programming algorithm, including pseudocode, that solves the "value version" of this problem in $O(n+m)$ time. (The "value version" is to find the maximum money that Yousif can earn, not necessarily a route that obtains it.) You may assume that for each $j \in V$, you are given a list of all its incoming edges $(i, j) \in E$.

Solution:

(2 pts) (e) Briefly describe in words (no pseudocode or correctness/runtime analysis needed) how to extend the above algorithm to output an optimum route.

Solution:

6. **Fixed-pattern paths.**

Let $G = (V, E)$ be a graph with n vertices and m edges. Also, suppose that each edge is colored either black or white. We say that a path $P = (e_1, \ldots, e_k)$ of edges has *color pattern* $(c_1, \ldots, c_k)$, where each $c_i \in \{\text{black}, \text{white}\}$, if each e_i has color c_i. In this problem, you will design an algorithm that, given two vertices s, t and a desired color pattern $(c_1, \ldots, c_k)$, determines in $O(mk)$ time whether there exists a path from s to t with that color pattern. Note that paths in this problem need not be *simple*; they can visit vertices or edges more than once.

(10 pts) (a) Derive a recurrence relation, including base case(s), for this problem. Briefly justify its correctness.

Solution:

(5 pts) (b) Give a bottom-up dynamic programming algorithm, including pseudocode, for this problem that has running time $O(k(n+m))$ (or just $O(km)$ under the assumption that $m \geq n$).

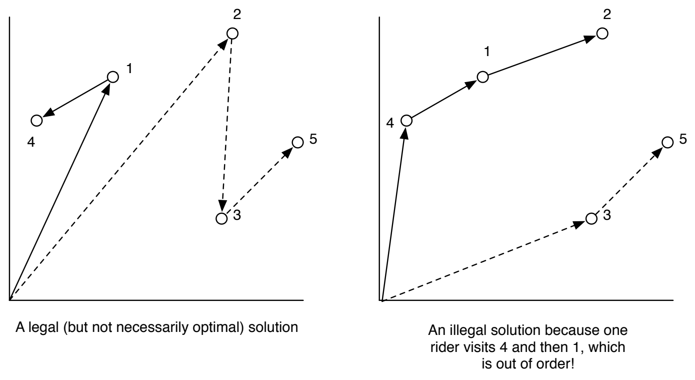

A legal (but not necessarily optimal) solution
An illegal solution because one rider visits 4 and then 1, which is out of order!
1
2
3
4
5

Figure 1: Two examples of routes for the two riders. The origin denotes the starting point p_0. One rider's route is shown with solid lines and the other with dashed lines. The route on the left is fair because points are visited by increasing indices. The route on the right is not fair because, for example, point p_4 is visited before point p_1.

Solution:

7. **Optional extra credit: Wheel deliver.**

Julie and Daphne have started a new unicycle-based meal delivery business called "Wheel Deliver" that works as follows. Residents of Ann Arbor submit their requests for a meal each day before noon using the Wheel Deliver App. In the afternoon, Julie and Daphne cook up a delicious meal in their kitchen and deliver the food on their unicycles.

Let n be the number of customers on a given day and $P = (p_1, \ldots, p_n)$ be the list of locations of these customers as points in the 2D plane, sorted in the order in which the customers submitted their requests: p_1 is the location of the first person to place an order, and p_n is the location of the last person to place an order. Let p_0 denote the location of the the kitchen from which Julie and Daphne depart. Let $d(p_i, p_j)$ denote the distance between p_i and p_j.

Julie and Daphne wish to split up the list of locations P, not necessarily evenly, so that each delivery is made by exactly one of them. In addition, "Wheel Deliver" has a fairness rule that goes like this: If customers at p_i and p_j for $i < j$ are assigned *to the same unicycle rider*, then the delivery to the customer at p_i must occur before the delivery to the customer at p_j—even if that makes the trip longer. (If p_i and p_j are assigned to different riders, then we don't care which delivery occurs first.) Subject to these constraints, the objective is to minimize the total distance travelled by the two unicycle riders.

Julie and Daphne have hired you to design an efficient algorithm that takes as input an array of points $P = (p_1, p_2, \ldots, p_n)$, sorted by the times in which their orders arrived, and outputs the sequences of delivery points for the two riders.

(4 EC pts) (a) Derive a recurrence relation, including base case(s), that is suitable for an efficient dynamic-programming solution for this problem. Briefly justify its correctness.

Solution:

(4 EC pts) (b) Give a (bottom-up) dynamic programming algorithm, including pseudocode, for this problem that has running time $O(n^2)$.

Solution: