

Practice Quiz 2

Autumn 2025

Name _____

Net ID _____ (@uw.edu)

Academic Integrity: You may not use any resources on this quiz except for your one-page (front and back) reference sheet, writing instruments, your own brain, and the exam packet itself. This quiz is otherwise closed notes, closed neighbor, closed electronic devices, etc.

Instructions: Before you begin, **Put your name and UW Net ID at the top of this page.** Make sure that your name and ID are LEGIBLE. Please ensure that all of your answers appear within the boxed area provided.

The last page of this exam serves as a reference sheet as well as scratch paper. Please detach the last page from the exam packet. No markings on the last page will be graded.

(1 ESNU) Question 1: Most Important Course - Graphs

Help us settle a debate on which course at UW is most important.

Some courses require others as prerequisites. We say that course c_a is in the prerequisite chain of course c_b if taking c_b requires that c_a be taken at some point before it (i.e. if c_a is a prerequisite of c_b , or c_a is a prerequisite of a prerequisite of c_b , etc.)

We say that the most important course is the one that appears in the prerequisite chain of the most other courses.

Given a directed graph where nodes are courses, and there is an edge from c_a to c_b if c_a is a prerequisite of c_b , give an algorithm which returns the most important course.

To receive a grade of E, your algorithm should run in time $O(xn + xm)$ where x represents the number of courses that have no prerequisites, n is the number of courses, and m is the number of edges in the graph. Any correct algorithm could receive up to a grade of S.

- Give psuedocode for your algorithm. You're welcome to reference other algorithms from class without reproducing their psuedocode (e.g. you could describe cycle detection as "do a DFS and return that a cycle is present if you ever see an in progress node").

- Identify the running time of your algorithm (justification is not necessary)

(1 ESNU) **Question 2: No Parents - Dynamic Programming**

Suppose you are given an integer array of length n which represents a complete binary tree T ("complete" meaning every node either has 2 children or it is a leaf) such that the root of the tree is at index 1, its left child is at index 2, its right child is at index 3, etc. Let v_i represent the value of the node found at index i . Your goal is to find a subset U of vertices from T such that:

- U does not contain any node and its parent
- The sum of the values of all nodes in U is maximized.

For convenience, you may assume you have access to constant-time subroutines $L(i)$ and $R(i)$ which return the index of the left child of node i and the right child of node i (respectively) if it exists, and -1 otherwise (and so $L(-1) = R(-1) = -1$).

The questions below work towards dynamic programming algorithm to solve this problem.

- a) Write a recursive definition for $\text{OPT}(i)$, which should represent the maximum sum of U you can get for the subtree rooted at node i .

- b) How would you store the solutions to the subproblems that you encounter? How large is your storage?

- c) For an efficient iterative (i.e. bottom-up) dynamic programming solution, in what order would you evaluate these subproblems?

- d) What would be the running time of your iterative algorithm?

(2 ESNUs) **Question 3: Cake Off - Greedy**

Note: each of subproblem a) and b) will receive its own ESNU grade. It is NOT necessary to get part a) correct in order to get part b) correct.

To make a cake, a baker must first make all of the cake's components (e.g. sponge, icing, jam, etc.). After making the components, the baker will assemble the cake. Each component requires "active time" followed by "passive time" to prepare. During its "active time", the baker will work uninterrupted. During its "passive time" the baker may work on another component. The cake assembly cannot begin until both the active and passive times elapse for all components.

For example, the icing may take 5 minutes of active time to mix, then 5 minutes of passive time to chill in the fridge. Jam may take 15 minutes of active time to cook and then 25 minutes of passive time to cool. So while the icing is cooling, the baker can get started on the jam. A baker can only actively work on one component at a time and must finish that component's active work before moving onto the next. Your task will be to write a greedy algorithm which selects an order to make the components that minimizes the total time required.

The input to your algorithm is n components of the cake. Each component $c_i = (a_i, p_i)$ requires a_i minutes of active time followed by p_i minutes of passive time to complete.

If the components were baked in the order c_1, c_2, \dots, c_n then the start time of component c_i is $s_i = \sum_{j < i} a_j$ (i.e. the sum of all active times before it). The finish time of component c_i is then $f_i = s_i + a_i + p_i$ (its start time, plus its active time, plus its passive time). Your task is to identify the order that minimizes the maximum f_i (i.e. minimizes $\max_{i \leq n}(f_i)$).

- a) We provide three options for a greedy algorithm below, none of which produce a correct solution. provide a counterexample for each option. Your counterexample must identify $\max_{i \leq n}(f_i)$ (the latest finish time among all components) for the schedule that greedy algorithm produces AND give a different schedule with better solution.

- Select the component with the largest active time first (i.e. sort in descending order by a_i .)

Counterexample:

- Select the component with the smallest active time first (i.e. sort in ascending order by a_i .)

Counterexample:

- Select the component with the largest total time first (i.e. sort in descending order by $a_i + p_i$.)

Counterexample:

- b) The correct algorithm would be to select the component with the largest passive time first (i.e. sort in descending order by p_i). For the final question you will complete an exchange argument showing that this greedy choice is correct.

To demonstrate correctness, it suffices show that the following statement. Let O be a solution that does not necessarily match your greedy solution. We will show that undoing adjacent inversions in O will not make a worse schedule. In other words, if O listed c_k immediately before c_{k+1} , but your greedy solution put c_{k+1} somewhere before c_k , then swapping c_k and c_{k+1} within O will not increase largest finish time. If s represents the start time of component c_k in schedule O , Then the following represent the finish times of c_k and c_{k+1} :

- (1) $f_k = s + a_k + p_k$
- (2) $f_{k+1} = s + a_k + a_{k+1} + p_k$.

Next, if we swap c_k and c_{k+1} in O , so that active work on c_{k+1} is scheduled before c_k . the new post-swap finish times of c_k and c_{k+1} :

- (3) $f'_k = s + a_{k+1} + a_k + p_k$
- (4) $f'_{k+1} = s + a_{k+1} + p_{k+1}$

Because only c_k and c_{k+1} change after swapping, the value of $\max_{i \leq n}(f_i)$ will only change if $\max_{i \leq n}(f_i) = f_k$ or $\max_{i \leq n}(f_i) = f_{k+1}$, so it's sufficient to show

- (5) $\max(f_k, f_{k+1}) \geq \max(f'_k, f'_{k+1})$

Use equations (1), (2), (3), and (4) above and your greedy choice to show that statement (5) is true.

Reference

Nothing written on this page will be graded.

Logs

$$\begin{aligned}x^{\log_x(n)} &= n \\ \log_a(b^c) &= c \log_a(b) \\ a^{\log_b(c)} &= c^{\log_b(a)} \\ \log_b(a) &= \frac{\log_d(a)}{\log_d(b)}\end{aligned}$$

Asymptotic Notation

- $f(n)$ is $O(g(n))$ provided that after some input size n_0 , $f(n) \leq c \cdot g(n)$ for some constant c .
- $f(n)$ is $\Omega(g(n))$ provided that after some input size n_0 , $f(n) \geq c \cdot g(n)$ for some constant c .
- $f(n)$ is $\Theta(g(n))$ provided that $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$

Graph Algorithm Running Times

n is the number of nodes, m is the number of edges

Breadth-first search: $O(n + m)$

Depth-first search: $O(n + m)$

Kruskal's algorithm: $O(n \log n)$

Prim's algorithm: $O(n^2)$ or $O(m + n \log n)$.

Dijkstra's algorithm: $O(n^2)$ or $O(m + n \log n)$

Scratch Work

Nothing written on this page will be graded.