MINERVA
UNIVERSITY

14 Mint Plaza, Suite 300
San Francisco, CA 94103, USA

www.minerva.edu

The Final Project–where your CS110 journey wraps up!

At the end of the semester, you should be equipped with the tools to address computational problems requiring formulating and implementing algorithmic approaches. For your final project, you will implement several algorithmic strategies, analyze their complexity, and contrast the several approaches with alternative algorithms that could also solve the same problem. You will have an opportunity to demonstrate mastery of the course LOs. We have designed two projects allowing you to exercise the course LOs, all comparable in scope and depth. The large number of questions in each option serves to provide thorough guidance for your work so that you never miss opportunities to provide strong applications for every LO of the course.

Please refer to the CS110 Guide to check all the components you should submit with your deliverables.

You need to choose one of the two project options outlined below. For your "Pseudo Project 2 - Choose your Final Computational Application" assignment, you need to fill in the workbook provided, where you will specify your choice of topic for the final project.

OPTION 1. A day in the life of a Minervan Part II

You have n activities that you would like to schedule in a day, with the following basic attributes:

`id`	`description`	`duration`	`dependencies`	`status`

Figure 1. These attributes describe each task. The *status* specifies whether the task is `not_yet_started`, `in_progress`, or `completed at any given time during the day`.

These activities can be assigned a profit/utility/preference value, which you will compute dynamically (possibly as a function of all the other tasks included in the schedule). Please check this whitepaper to review the #utility HC and for extra motivation on how to consider utility functions to address this part.

Here are a few metrics your algorithm will need to consider:

- The profit value measures the priority that a given activity should have. The convention you will use is that the **higher** the profit value, the more important it is to schedule that activity first. You will need to choose the appropriate data structure according to this convention.
- Time constraints restrict the number of activities you can perform to, say, x activities, where $time(x) \leq n$, and n the total time you are allowed to do tasks (because everyone needs to sleep!).
- In this context and with the assumptions presented, you will need to explain what it means to have an optimal scheduler. Perhaps if you are planning several activities across town, you would like to save money on fuel/bus tickets, or you would like to spend the least time doing several tasks in a single day.

1. Carefully explain how you construct a utility value from the task features given in Figure 1. You should exemplify how the utility calculation applies to some of your specific tasks. Recall that this utility value will represent a preference for the task's outcome. Once you assign a preference level, this utility value is the only key to determining which tasks should be scheduled first.
   a. Please spend as much time as needed on this question, and do not rush; having a well-defined setup will guarantee that you can tackle the subsequent questions more efficiently.
   b. Note: you should have already done this in the previous assignment, so use this as an opportunity to improve your response.

2. Review your previous assignment, where you crafted an algorithm and test-drove it. Using the feedback you have received in the LO applications and everything you have learned up to this point in the semester, perform a critical and thoughtful analysis of your algorithm:
   a. Write a 200-300 word summary evaluating the applicability of all the assumptions in the worked implementation and all the possible algorithmic failure modes that could be addressed with an improved version. You do not need to provide the improved implementation at this stage, but your answer is expected to go beyond “my algorithm doesn’t handle multi-tasking” or “my algorithm doesn’t work for this type of input.”
   b. Please provide a word count for this question.

3. To move beyond the simplistic assumptions of the previous assignment, you realize that some of the tasks in your schedule can be multi-tasked! In other words, *many* of your daily tasks can be performed simultaneously (e.g., sipping a local beverage while chatting with a friend at a cafe, taking pictures while riding a bus, or walking in a park). While some tasks can be multi-tasked, others may demand your full attention (e.g., CS110 pre-class work, which cannot be multi-tasked and is set to happen at a specific pre-determined time).

Your input for each activity now includes an extra feature:

- `multi_tasking`: boolean field for whether a task can be multitasked.

id	description	duration	dependencies	multi_tasking	status

Figure 2 . To enable multitasking, we add an extra attribute (`multi_tasking`) to each task.

Criteria for scheduling non-multi-tasking activities—For tasks that require your full attention, your program cannot schedule any other tasks while that task’s status is `in_progress`. For instance, CS110 pre-class work cannot be multi-tasked and is set to happen at a specific pre-determined time). Note that if such a task (CS110 pre-class work, which we shall refer to as “A”) is part of the dependencies of another task (participating in a CS110 session, “B”), then once A’s status is set to `completed`, you will be required to update that property on the dependencies of B and the corresponding priorities.

Criteria for scheduling Multitasking tasks—For tasks that can be done simultaneously, they may have a different duration and terminate at different times. You will need to keep track of the remaining time for every task processed in multi-tasking mode and update the scheduler clock accordingly. When two or more multi-tasking activities are being executed, the remaining time for every activity needed is when one gets completed.
Notice that multi-tasking activities *can* but *need not* be executed simultaneously. For example, ‘eat a delicious pizza’ and ‘brushing your teeth’ don’t make sense together, but you can ‘eat a pizza’ while ‘talking to a friend,’ and you can ‘brush your teeth’ while ‘listening to a podcast.’ *How do you include these constraints in your schedule?*

🎥In a <4 minute video (you must provide the **shareable link and check it is accessible at the time of submission**), where your face is shown at all times, describe how you can design an algorithmic strategy that handles multitasks. To guide your composition, consider the following points that you must address.

A. Describe as clearly as you can any changes you will need to make to the first version of the scheduler to include multi-tasking activities. Your explanation should be ***conceptual***, focusing on the algorithmic strategies (how does your scheduler handle these events and why is it guaranteed to work). You can use, if you wish, schematics or flowcharts, or even make reference to a coding structure (but please note that this question is not asking us to provide any implementation, for now).
B. Describe how the priority values can still be used to accommodate multi-tasking. Are they sufficient, or do you require extra functionality?
C. Describe how the original priority queue specifically handles multitasking. Your explanation should be clear enough that anyone reading it will have no queries about how to implement it in Python (when asked to do so; again, the coding implementation is **not** the focus of this prompt). Interesting aspects you can consider would be, for instance, if a second priority queue is helpful. Why or why not?

Feel free to go beyond the guidance these prompts suggest when describing the algorithmic strategy if you think it will make your description clearer.

4. Combining dynamic programming and greedy techniques with your critical evaluation from questions 1 and 2, propose **at least two** improved algorithmic strategies to solve the scheduling problem with non-multitasking activities: **one** should implement a dynamic programming strategy, whereas the **other** should implement a greedy one. The optimization characterization (what your algorithm optimizes for) should be the same for both algorithms, but you are free to set it up as you wish.
   a. Design each algorithmic strategy, carefully explaining why they fall into the greedy and dynamic programming paradigm. Make sure to explain any assumptions you make carefully, including what data structures best suit each of these strategies. Will you still use a priority queue? Why or why not?
      i. Note: feel free to use any means you deem appropriate to convey the algorithmic strategies as clearly as possible (for example, you may consider flowcharts or pseudocode and the text description).
   b. Implement your new algorithmic strategies in working Python code. Ensure that your code works as intended and provide at least one test case (you need to demonstrate that the code is correct, so it needs to be clear that your test case is appropriate). Do the outputs agree? Comment on what the different outputs tell you about the algorithmic strategies.
   c. Study the algorithmic time and space complexity of these new versions. You will need to provide both theoretical arguments and experimental validation.
   d. How would you incorporate multi-tasking for each of these two algorithmic approaches? Explain in as much detail as possible. [Optional Challenge]: implement the improved algorithms in Python, and determine which algorithmic formulation is better and why. You will need to carefully and thoroughly document your work.

5. Write a computational critique of your schedulers (greedy and dynamic-programming driven) and contrast them with the algorithm you have produced for the previous assignment. Some relevant points to consider in your analysis warrant thoughtful explanations:
   a. Do all three implementations use the same data structures?
   b. Do these new formulations (greedy and dynamic-programming driven) address all the concerns raised in question 1?
   c. What metrics are you using to compare the schedule and the scheduler efficiencies?
   d. What experimental results support the theoretical arguments that favor one algorithmic strategy over another? You need to include at least one plot where we can three curves (one for each algorithm) that illustrate their scaling behavior.
   e. What is the best strategy (what is best?), and why?

6. Appendix material:
    a. In 80 words or fewer, identify a specific concept or discussion from class that clarified your problem-solving approach for this assignment. Briefly explain how it assisted you.
    b. Justify the application of the core course LOs application (there are 6) in <50 words each (please include a word count and use that metric to refine your justification). Provide **evidence** that you have taken the feedback provided throughout the course into account to further your understanding of the course materials.
    c. Identify the HC that you argued would be helpful for this assignment and explain how you have explicitly applied in this assignment in 50-70 words (please include a word count and again use that metric to refine your justification).

OPTION 2. Decoding the relationships between genes

A biotechnology company has hired you to work on a genealogical mutation sequencing research project. A gene is described by a string of characters where there is a small probability of inserting a new character, deleting an existing one, or randomly changing it. The probabilities `p_insertion`, `p_deletion`, and `p_mutation` represent forms of broadly described string mutation.

Now, suppose you start with a given string representing a gene sequence (top-down perspective). This sequence is then duplicated, resulting in two identical strings. Each of these strings undergoes the mutation process independently (where each character can suffer an extra insertion, deletion or mutation). These mutated strings are the 'child' strings of the original one. Subsequently, each child string undergoes further mutation, creating two 'grandchild' strings each. We can easily visualize this process if we were to draw a ***perfect*** genealogy binary tree relating a total of seven strings, two of which are parents and one a grandparent (from a bottom-up perspective).

Sadly, the order of the strings has been lost due to a glitch in the gene-sequencing generation program. Your goal is to recover the original genealogy tree for the seven strings labeled with lowercase letters:

```
[('a',
'ATGGTGCGAAAGCATCTCTTTTCGTGGCGTGATAAGTTTTATGGTATCCCCGGACGTTGGCTACTACAATTCTCCGAAGTATAAGTGAGTAGGATATGTCAATAACAAGA
GGGGATGCGTGACGCATTAGCACCAACTGAATCAAACGATAACTAACGTGGTTTCAGTGAGCGTATGTGGCAAAGGATTGGATACATTTTTCGAGCACGTCTACATAATGA
CCGTGACAATACTGGAGACTCCGTACCGTCATCTTGACACTCCT'), ('b',
'TGGTGCGAAAGCATCTCTTTTCCGTGGCGTATAGTTTTATGGTATCCCCGGAACGCTGGCTACTACAATCTCCGAAGTATAGAGTGAGTAGATTTAATTAACAGAGGGCG
TCGTTGACGCATTAGCACCAACTGAATCAACCGATAACTTAACGTGGGTTTCAGTGACTATAGGGCAAAGGATGAACATTTTCGAGCAGCTCTAATAATGAGCGTGACAAT
ATGAATCCACACCGTCATCTTGAACTCCT'), ('c',
'TCTGTGCGATATACATCTCTATCGTTGCGGTATGTTTTATGTGCATCACCCCACGCGCTGGCTACAGTACAATCTGCTGGAAGTACTAGGTGGTAGTTAATAACTAGGGT
GCGTCGTTGCGCATTACACAACTGGACAACCACTTAACTGGGGTAATCAGTGTTTAGGGCAGACAAGATGAAAACAAGTTTTCGAGCAGGCTCCTATAATGAGGACGGAAC
GTTAATAAATCCAACACCGCACTGCTTCGTAACCCT'), ('d',
'ATGAGGCGCAAAATTCTCTTTCTCGTGGCGCTGATTAAGTTTTATGTATCCCCGGACGTTGGCTACTGACAATTGCTCCGAAGTATAAAGTAGTAGGATATGTCAATAAC
AAAGACGGGGATAGCGTGACAGCATTAGAACGCAACTGGAATCAAACGTAACCTAAAGGGTTGTCAGGAGCGTATGTGGTCAAAAAGGATTGGATGACATTTTTCGACACG
TCTACATAATGACCTGTGACAAACTAGGAGACCTCCTACTCGGTCAATCTTGACGACTCCT'), ('e',
'TGGTGCGATATACATCTCTTTTCGTGCGTATGTTTTATGGTGATCACCCGGAACCGCTGGCTACATACAATCTCTGGAAGTACTAGGTGGTAGTTTAATAACTAGAGGTG
CGTCGTTGACGCATTACACAACTGGATCAACCGAACTTAACTGGGTATCAGTGATATAGGGCGACAAGATGAACAATTTTCGAGCAGCTCCTGAATAATGAGACGGAACGT
ATAATCCAACACCGTCACTGCTTCGAACCCT'), ('f',
'GGGGGAAAGCGATCCCTTATCGTGGCTGTGATAAGTTTTTATCGGGTATCCGCCGGACGTTGGCGTACTACAATTCTCCGAAGTTAAGTGAGTTAGGGATATAGTCAATA
ACAAGAGGGGATTGTCGTGACGCATAGCACACAACTGAATCAAATCGATAACTAAACGGGTTTCAGTAGAGCGTTGTGGCAAAGATTGGATACATTTTTCGCAGGACGTCT
TACCTAATGACGTGGACAATAACTGGCAGACGTCCGTACCGTCATCTTGACCACTCCCT'), ('g',
'TGGTGCGATATACATCCTCTTTTCGTGCGTATGTTTTAGGTACACCGGATACGCCTGGCTTACAAGTACCAATCTCTGAGAAGTCACTGAGGTGGTAGTTTAATAACTAG
AAGGGTGCGTCGGACGCATTCACACATACTGGATCAACCGAGACTTAACTGGGGTATCAGTGATTGATAGGGCGACAAGATATACAATTTTCGAGCAGCTCCCTGAATAAG
TGAAGAACGGAGACGTATAATCCAACACGATTCACTGCTTCGAACCCT')]
```

Figure 3. The set, named **set_strings**, comprises all the strings in this problem.

Let **set_strings** refer to the set of strings created with the gene-sequencing generation program.

1. Write Python code that, given any two arbitrary strings, outputs **all** of the Longest Common Subsequences (LCSs) for those two strings and their corresponding lengths. Include several test cases that demonstrate that your code is correct.

```
def longest_common_subsequences(x, y):
    """Gives the length of the longest common substring between strings x and y

    Inputs
    ----------
    x, y: strings
        Strings to compute the LCS
    Returns
    ----------
    all_lcs: tuple ([LCS1, LCS2, …], len(LCS1))
        Tuple of a list of all the possible LCS and the corresponding length (size)
    """
    # YOUR CODE HERE
    return all_lcs

#test cases
x1, y1 = 'ABCBDAB', 'BDCABA'
x2, y2 = 'abc', ''
x3, y3 = 'abc', 'a'
x4, y4 = 'abc', 'ac'

assert longest_common_subsequences(x1, y1) == (['BDAB', 'BCBA', 'BCAB'], 4)
assert longest_common_subsequences(x2, y2) = (None, 0)
assert longest_common_subsequences(x3, y3) = ([‘a’], 1)
assert longest_common_subsequences(x4, y4) = ([‘ac’], 2)
```

(please change the order of the output in the assert statements outlined here to be compatible with the output from your implementation)

Figure 4. Skeleton code for `longest_common_subsequences.`

2. How many LCSs lengths are there in **set_strings**? Generate the matrix of the lengths of the LCS for every pair of strings in **set_strings**. Make sure that your matrix obeys the following properties:
    a. The matrix should be cast as a two-dimensional numpy array. Store this 2D numpy array to a variable named **len_lcs_matrix**.
    b. Your 2D array len_lcs_matrix should have dimension (7,7), and **len_lcs_matrix[i,j]** should give the length of the LCS for the ith and jth strings. For example, **len_lcs_matrix[0,3]** gives the length of the LCS for string a and string d.
    c. Manually examine the matrix you obtained above.
        i. Explain how you would infer (directly from this matrix or on other matrices obtained from this one) which strings are more strongly related to each other.
        ii. Could you infer the resulting genealogy binary tree?
3. You will now examine the precise relationships between strings (i.e., explicitly identify grandparent, parent, and child strings). You will do so by proposing two strategies: one that is local and another that is global. Note:
    a. A *local* strategy infers the location of a particular string in the tree based on a greedy property (of your choice) of the node itself.
        i. Specifically, we compare each node with its immediate neighbors. For instance, we compare the root with its children only, but not with its grandchildren. This comparison can be achieved using a greedy approach, a dynamic programming approach, or a combination of both.
    b. A *global* strategy infers the whole tree based on a metric obtained by considering all the relationships involved in that tree at once.
        i. This is very similar to the dynamic programming approach we discussed to solve the coin problem in class, or how the regression fit is measured by R^2, which takes into account how all the individual datapoints offset from the line of best fit. You may need to research what a good metric could describe global relationships (i.e., beyond what was discussed in class).

    c. Regardless of the combination of techniques, you need to explain how they work in the context of this problem. For instance, when you argue that an algorithm has a greedy approach, you do need to explain what features of the algorithm make it greedy.

🎥In a <4 minute video (you must provide the **shareable link and check it is accessible at the time of submission**), where your face is shown at all times, describe how both of these strategies work. Your explanation should be clear to the extent that anyone will be able to grasp what the underlying strategy of your approaches is. You need to focus on explaining how and why each works, and what makes them local and global. Feel free to use any resources you deem helpful for your explanation.

4. Now implement your strategies, and draw the resulting genealogy binary tree(s) associated with **set_strings** resulting from each strategy. Comment on whether the results (the tree predictions) are expected to be the same or different. Provide a concise comparison between these results and the insights you have included in your answer to question 2.c.
5. For each algorithmic approach, what is your solution's computational complexity to produce genealogy binary trees? You can consider M the length of a gene and N the number of genes. Produce at least one experimental plot where you can induce the scaling growth of both of these algorithms and compare it with your theoretically derived results. You will need to vary the number of strings for this.
6. How would you estimate the probabilities of insertions, deletions, and mutations?
    a. Obviously, you don't have enough data to obtain meaningful estimates, but this small dataset has enough information to intuitively obtain estimates for these probabilities. Explain your algorithmic approach in as much detail as possible.
    b. Produce an estimation in Python that would take your algorithmic strategy into practice. Are the results you obtained reasonable?
7. Appendix material:
    a. In 80 words or fewer, identify a specific concept or discussion from class that clarified your problem-solving approach for this assignment. Briefly explain how it assisted you.
    b. Justify the application of the core course LOs application (there are 6) in <50 words each (please include a word count and use that metric to refine your justification). Provide **evidence** that you have taken the feedback provided throughout the course into account to further your understanding of the course materials.
    c. Identify the HC that you argued would be helpful for this assignment and explain how you have explicitly applied in this assignment in 50-70 words (please include a word count and again use that metric to refine your justification).