Problem Set 3
data structures and algorithms 2 - fall 2023

cfr5spw
due october 20, 2023 at 11:59 pm

Collaboration Policy: You are encouraged to collaborate with up to 4 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite them. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: hhv6bx

Sources: https://www.geeksforgeeks.org/quick-sort/#, https://en.wikipedia.org/wiki/Greedy_algorithm, https://www.geeksforgeeks.org/greedy-algorithms/#, Lecture slides and notes

problem 1 *Scenic Highways*

In this problem we'll describe something similar to solving a problem with Dijkstra's algorithm, and ask you some questions about this new problem and a possible algorithm.

You are taking a driving vacation from town to town until you reach your destination, and you don't care what route you take as long as you maximize the number of scenic highways between towns that are part of your route. You model this as a weighted graph G, where towns are nodes and there is an edge for each route between two towns. An edge has a weight of 1 if the route is a scenic highway and a 0 if it is not.

You need to find the path between two nodes s and t that maximizes the number of scenic highways included in the path. You design a greedy algorithm that builds a tree like Dijkstra's algorithm does, where one node is added to a tree at each step. From the nodes adjacent to the tree-nodes that have already been selected, your algorithm uses this greedy choice:

Choose the node that includes the most scenic highways on the path back to the start node s.

When node t is added to the tree, stop and report that path found from s to t. Reminder (in case it helps you): the greedy choice for Dijkstra's algorithm was:

Choose the node that has the shortest path back to the start node s.

1. What is the key-value stored for each item in the priority-queue? Also, what priority-queue operation must be used when we need to update a key for an item already stored in the priority-queue?

   Solution:

   key-value information stored for each item is the current maximum number of scenic highways included in the path from the starting node to its specific node. This value is adjusted continuosly as the algo explores different routes it can take. when we need to update this value for an item already in the queue we can use the decrease key operation.

2. Draw one or more graphs to convince yourself that the greedy approach describes above does not work. Then explain in words as best you can the reason this approach fails or a situation that will cause it to fail (i.e., a counterexample).

   Solution:

   To find the path of the most scenic route (most scenic highways) we'd select the greedy choice: A to B (scenic) then B to D to F which allows for a direct scenic route to F.
   so we would have A to B to D to F that would have 2 scenic highways.

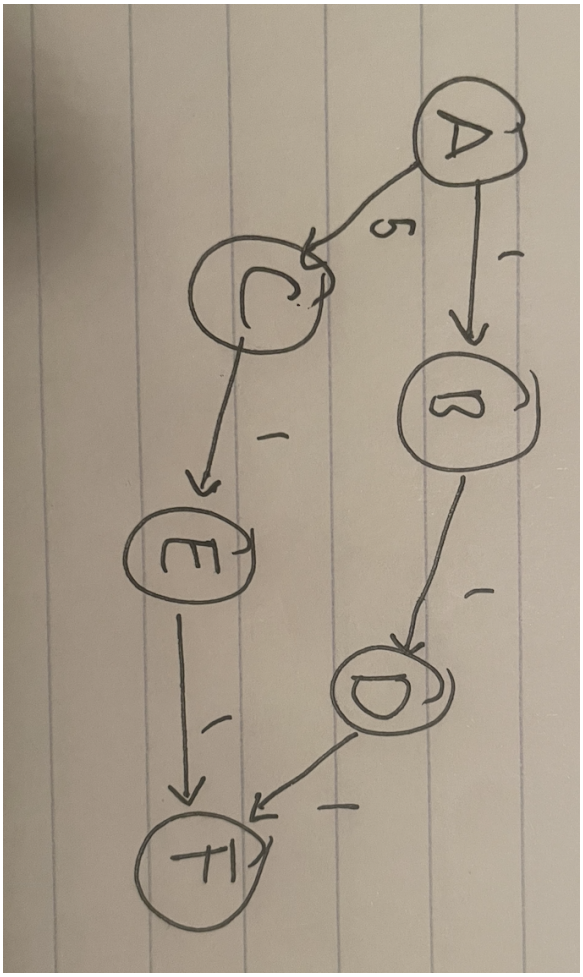

Figure 1: Graph

Optimal Solution:

taking the route: A to C to E to F would have 3 scenic highways

starting with a highway that is not scenic in the optimal solution allows us to have more scenic routes later on. In the greedy solution we only get 2 as we start with a scenic highway which results in a failed optimization.

PROBLEM 2 *As You Wish*

Buttercup has given Westley a set of n tasks $T = t_1, \ldots, t_n$ to complete on the farm. Each task $t_i = (d_i, w_i)$ is associated with a deadline d_i and an estimated amount of time w_i needed to complete the task. To express his undying love to Buttercup, Westley strives to complete all the assigned tasks as early as possible. However, some deadlines might be a bit too demanding, so it may not be possible for him to finish a task by its deadline; some tasks may need extra time and therefore will be completed late. Your goal (inconceivable!) is to help Westley minimize the deadline overruns of any task; if he starts task t_i at time s_i, he will finish at time $f_i = s_i + w_i$. The deadline overrun (or lateness) of tasks—denoted L_i—for t_i is the value

$$L_i = \begin{cases} f_i - d_i & \text{if } f_i > d_i \\ 0 & \text{otherwise} \end{cases}$$

Design a polynomial-time algorithm that computes the optimal order W for Westley to complete Buttercup's tasks so as to minimize the maximum L_i across all tasks. That is, your algorithm should compute W that minimizes

$$\min_{W} \max_{i=1,\ldots,n} L_i$$

In other words, you do not want Westley to complete *any* task *too* late, so you minimize the deadline overrun of the task completed that is most past its deadline.

1. What is your algorithm's greedy choice property?

   Solution: The greedy choice property for this problem dictates selecting the task with the earliest deadline first. By prioritizing tasks with earlier deadlines, the algorithm minimizes the risk of tasks becoming overdue. Focusing on immediate deadlines allows the algorithm to gradually construct a solution aiming to minimize the maximum lateness across all tasks. **Algorithm:** 1. Sort the tasks based on their deadlines in ascending order. 2. Schedule tasks in the sorted order.

2. What is the run-time of your algorithm?

   Solution: The algorithm's runtime is primarily determined by the sorting step. Since tasks are sorted based on their deadlines, the runtime complexity is $O(n \log n)$, where n represents the number of tasks.

3. Consider the following example list of tasks:

   $$T = \{(2,2), (11,1), (8,2), (1,5), (20,4), (4,3), (8,3)\}$$

   List the tasks in the order Westley should complete them. Then argue why there is no better result for the given tasks than the one your algorithm (and greedy choice property) found.

   Solution:

   $$T_{\text{Westley Order}} = \{(1,5), (2,2), (4,3), (8,2), (8,3), (11,1), (20,4)\}$$

   Task Schedule: - start time for task 1: 0 - end time: $0 + 5 = 5$ - task 2 start at 5 and finish at 7 - task 3 start at 7 and finish at 10 - task 4 start at 10 and finish at 12 - task 5 start at 12 and finish at 15 - task 6 start at 15 and finish at 16 - task 7 start at 16 and finish at 20 Maximum lateness at task 1,$5 - 1 = 4$. We can prove that there's no better result, we can consider that any permutation of tasks that doesn't prioritize the earlier deadlines would risk having those tasks be late, hence we are increasing the maximum lateness. By sorting tasks based on the deadline we are ensuring tasks with tighter deadlines are completed first which reduces the risk of a larger maximum lateness.

PROBLEM 3 *Course Scheduling*

The university registrar needs your help in assigning classrooms to courses for the spring semester. You are given a list of n courses, and for each course $1 \leq i \leq n$, you have its start time s_i and end time e_i. Give an $O(n \log n)$ algorithm that finds an assignment of courses to classrooms which minimizes the *total number* of classrooms required. Each classroom can be used for at most one course at any given time. Prove both the correctness and running time of your algorithm.

Solution:

We can use a quick sort algorithm to sort the Start(s) and End(e) times of each course from earliest to latest.

Quick sort Algo for O(nlogn) from geeksforgeeks:

QuickSort(arr, low, high):
if low ¡ high:
pivotIndex = Partition(arr, low, high)
QuickSort(arr, low, pivotIndex - 1)
QuickSort(arr, pivotIndex + 1, high)

Partition(arr, low, high):
pivot = arr[high] // Choose the rightmost element as the pivot
i = low - 1
for j = low to high - 1:
if arr[j] ¡= pivot:
i = i + 1
Swap(arr[i], arr[j])
Swap(arr[i + 1], arr[high])
return i + 1

Once sorted array of Course start and end times should look like this:timePeriods = [s1,s2,e1,s3,e2..];

We will also have an array of available classrooms: classrooms = [c1,c2,c3,c4..];

We can now iterate from i=0 while i¡length through the array of timePeriods.

if timePeriods[i] is a start time we can allocate it to the available classroom at the end of the array so that we can then pop that class room off the array as it is no longer availble until the end of that course.
we will continue to iterate assigning a start course to a available classrooom. -¿ if timePeriods[i] is an end time we will push the class room that this course was occupying back on to the classrooms array sto signify that it is now available again.
since we are pushing this classroom back onto the array it will be the first to become available for a new course which is starting. This will help reduce the total number of classrooms used.

In the case of not having an available classroom we can designate a course to an array of classes that were not able to be scheduled and remove its corresponding endtime from the array or simply account for its endtime by providing a conditional such that if an endtime of a course is not found within a class room simply move on.
Note: Helper methods would also be required for sorting algorithm such as: Swap(a, b)) and others required for the algorithm which might be: createNewClassroom(), assign(course, classroom), getAssignedClassroom(time)

PROBLEM 4 *Ubering in Arendelle*

After all of their adventures and in order to pick up some extra cash, Kristoff has decided to moonlight as the sole Uber driver in Arendelle with the help of his trusty reindeer Sven. They usually work after the large kingdom-wide festivities at the palace and take everyone home after the final dance. Unfortunately, since a reindeer can only carry one person at a time, they must take each guest home and then return to the palace to pick up the next guest.

There are n guests at the party, guests $1, 2, ..., n$. Since it's a small kingdom, Kristoff knows the destinations of each party guest, $d_1, d_2, ..., d_n$ respectively and he knows the distance to each guest's destination. He knows that it will take t_i time to take guest i home and return for the next guest. Some guests, however, are very generous and will leave bigger tips than others; let T_i be the tip Kristoff will receive from guest i when they are safely at home. Assume that guests are willing to wait after the party for Kristoff, and that Kristoff and Sven can take guests home in any order they want. Based on the order they choose to fulfill the Uber requests, let D_i be the time they return from dropping off guest i. Devise a greedy algorithm that helps Kristoff and Sven pick an Uber schedule that minimizes the quantity:

$$\sum_{i=1}^{n} T_i \cdot D_i.$$

In other words, they want to take the large tippers the fastest, but also want to take into consideration the travel time for each guest. Prove the correctness of your algorithm. (Hint: think about a property that is true about an optimal solution.)

Solution:

We want to minimize the quantity

$$\sum_{i=1}^{n} T_i \cdot D_i.$$

we would want pick the guests with the largest ratio of tip/unit time (to get to the destination and back)
We can use this as our Greedy Choice: At each step, pick up the guest with the largest ratio of tip per unit time.

Optimal solution

we can prove this is optimal by showing that the greedy choice property is consistent and holds at each step.
we can prove the property as if we suppose an optimal solution exisits and that it does not include a guest with the highest tip to time ratio then by considering a guest, g, from the greedy approach that is not in the set of guests from the optimal solution we can show our quantity as such:

$$\sum_{i=1}^{n} T_i \cdot T_g.$$

since g is the guest with the highest ratio of tip over unit time we know

$$\frac{T_g}{D_g} \geq \frac{T_i}{D_i}$$

so adding g to the set of guests from the optimal solution will not decrease the total tips obtained.
Adding g to the set will not increase the total time as g is the next guest to be dropped off.

since this solution is also optimal this contradicts our assumption that g is not included in the optimal solution, so, the algorithm always chooses the guest with the highest ratio of tip per unit time at each step, ensuring that the chosen schedule minimizes the quantity,

$$\sum_{i=1}^{n} T_i \cdot D_i.$$

.