

CSCI 3104 PS3b

Luna McBride

TOTAL POINTS

24 / 50

QUESTION 1

23 pts

1.1 10 / 10

- ✓ + 1 pts Accepts k and list of charging stations as parameter
- ✓ + 3 pts Correctly iterates through the list of charging stations
- ✓ + 3 pts Finds first charging station is further than distance k from current position
- ✓ + 2 pts Returns charging station immediately prior to the first one further than distance k from current position
- ✓ + 1 pts Returns or prints charging stations
 - 4 pts Code generates runtime errors
 - 2 pts Code produces incorrect solution for test case $k=5$, $list=[0,3,4,6,8,9,11]$
 - + 0 pts no solution/no submissions

1.2 3 / 3

- ✓ + 1 pts Recognized $O(n)$ complexity
- ✓ + 2 pts Justification regarding loop
 - 0.5 pts Clearly specify that the complexity is $O(n)$, and not "the time complexity is n " or " $T(n) = n$."
 - 0.5 pts Avoid dependence on specific input instances when possible.
- + 0 pts No answer

1.3 4 / 10

- ✓ + 1 pts Correct base case
- ✓ + 1 pts Correct inductive hypothesis
- ✓ + 2 pts Know greedy stays ahead and what to do in inductive step, but failed to prove it.
 - + 3 pts Inductive Step: Considered case that $p(i, J) \geq p(i+1, J^*)$, and correctly proved that if $p(i, j) \geq p(i+1, J^*)$, then $p(i+1, J) \geq p(i+1, J^*)$

+ 3 pts Inductive Step: Considered case $p(i, J) < p(i+1, J^*)$, and correctly proved that if $p(i, J) < p(i+1, J^*)$, then $p(i+1, J) \geq p(i+1, J^*)$.

+ 2 pts Prove that $|J| = |J^*|$ using contradiction.

+ 0 pts Incorrect/Not attempted.

+ 2 pts Considered case $p(i, J) < p(i+1, J^*)$, as well as case when $p(i, J) \geq p(i+1, J^*)$.

QUESTION 2

2 7 / 7

- ✓ + 1 pts Correct step-1 or 1st iteration to mention $v1$
- ✓ + 1 pts Correct step-2 or 2nd iteration to mention $v4$
- ✓ + 2 pts Correct step-3 or 3rd iteration to mention $v3$
- ✓ + 1 pts Correct step-4 or 4th iteration to mention $v2$
- ✓ + 2 pts Correct step-5 or 5th iteration to mention $v5$
 - 1 pts Minor calculation mistake in any one of the steps or the last step. Please refer to the solution file to check calculations.
 - 1 pts Minor calculation mistake in any two of the steps. Please refer to the solution file to check calculations.
 - 2 pts Minor calculation mistake in any three or more of the steps. Please refer to the solution file to check calculations.
 - + 0 pts Empty solution or incorrect approach mentioned. Please refer to the solution file.
 - 2 pts Please do provide all vertex pair (from $v0$) distances after each iteration explicitly. Please refer to the solution file to check calculations after each iteration.

QUESTION 3

20 pts

3.1 0 / 10

+ 4 pts Correct algorithm

+ 1.5 pts Observes that, for each $i = 0, 1, 2, 3$, we use at most one player of type 2^i .

+ 1.5 pts Attempted to argue by contradiction that the greedy algorithm was not optimal, by correctly assuming the conclusion is false

+ 1.5 pts Deduced that, if the greedy algorithm was not optimal, then more players than necessary would have been selected

+ 1.5 pts Achieved contradiction, noting that the extraneous players of type 2^i could have been exchanged for a player of type 2^{i+1} (where $i = 0, 1, 2, 3$).

✓ + 0 pts **Incorrect**

+ 0 pts Not attempted

+ 2 pts Partially correct algorithm

+ 1.5 pts Alternate approach: Shows that there is only one possible optimal answer

+ 3 pts Alternate approach: Shows that greedy solution is optimal through contradiction or other approach.

+ 1 pts Alternate approach: Progress towards showing the optimal solution is unique

3.2 0 / 10

+ 6 pts Gave valid counter-example

+ 4 pts Contrasted counter-example with optimal solution

+ 0 pts No answer

✓ + 0 pts **Invalid Example. See Solution.**

+ 0 pts The greedy algorithm is optimal for your example.

- 1 pts Clearly specify constructions for greedy algorithm and optimal solution.

- 1 pts Incorrect point present in your explanation.

+ 2 pts In the right direction but your answer is incorrect. See the solution

💡 Your algorithm should work for any set of inputs and give the correct solution using the greedy algorithm technique.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.
- Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.
- For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.
- You may work with other students. However, **all solutions must be written independently and in your own words**. Referencing solutions of any sort is strictly prohibited. You must explicitly cite any sources, as well as any collaborators.

CSCI 3104, Algorithms
Problem Set 3b (50 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

1. (23 pts) Imagine an alternate reality where CU has a small robot that travels around the campus delivering food to hungry students. The robot starts at the C4C and goes to whatever dorm or classroom has placed the order. The fully-charged battery of the robot has enough energy to travel k meters. On campus, there are n wireless charging pods where the robot can stop to charge its battery. Denote by $l_1 < l_2 < \dots < l_n$ the locations of the charging pods along the route with l_i the distance from the C4C to the i th charging pod. The distance between neighboring charging pods is assumed to be at most k meters. Your objective is to make as few charging stops as possible along the way.

- (a) (10 pts) Write a python program for an optimal greedy algorithm to determine at which charging pods the robot would stop. Your code should take as input k and a *list* of distances of charging pods (first distance in the list is 0 to represent the start point and the last is the destination and not a pod). Print out the charging pods where the robot stops using your greedy strategy.

Example 1 - If $k = 40$ and **Pods** = [0, 20, 37, 54, 70, 90]. Number of stops required is 2 and the output should be [37, 70].

Example 2 - If $k = 20$ and **Pods** = [0, 18, 21, 24, 37, 56, 66]. Number of stops required is 3 and the output should be [18, 37, 56].

Example 3 - If $k = 20$ and **Pods** = [0, 10, 15, 18]. Number of stops required is 0 and the output should be [].

- (b) (3 pts) Provide the time complexity of your python algorithm, including an explanation.

Solution.

```
p=k-- >1, c=len(stations) -- >1, stops=np.zeros(0,dtype=int)-- >1
```

```
count=0-- > 1
```

```
for i in range (1,c): -- > c (only c because the +1 from rechecking the loop is negated by a -1 of not starting at 0)
```

```
Statements in loop: 5 (for worst case)
```

```
return count, stops -- >1
```

Constants (non-loop): 5, non-constants (from loop): $5c$

$5c + 5$

$\mathcal{O}(c)$

CSCI 3104, Algorithms
Problem Set 3b (50 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

(c) (10 pts) Prove that your algorithm gives an optimal solution.

Solution.

Let $i_1 \dots i_n$ be the list of charge stations in the algorithm, $\text{len}(A)=n$

Let $j_1 \dots j_m$ be the list of charge stations in the optimal solution, $\text{len}(O)=m$

Base case: For $s=1$, there is one station.

—Case 1: The destination is within battery life, neither optimal nor algorithm needs to stop at a charging station.

—Case 2: The destination is not within battery life, in which both will need to charge at the station to reach the destination.

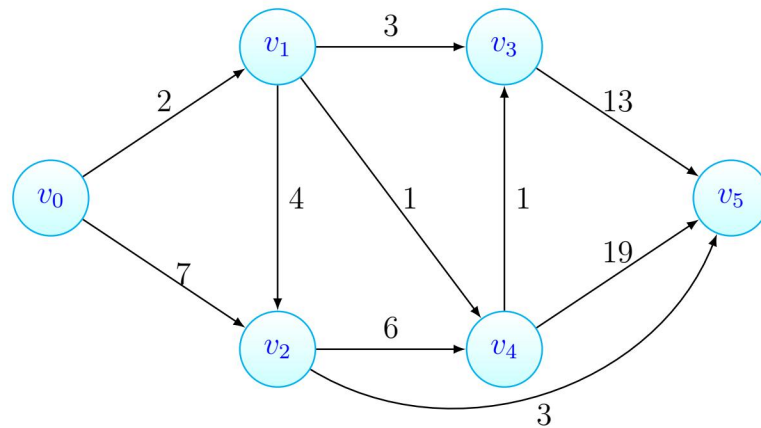
Inductive Hypothesis: Assume $i_s=j_s$, where s is the station they both share a recharge point at.

Inductive Step: Following how this question was defined, the optimal and algorithm both have an endpoint and start point at the same distance. The most optimal way to go about this is to go the furthest possible before recharging again, as to maximize battery life. That is, $j_{s+1}=j_s < \text{Max}(O[\text{rs}]) \leq k+j_s$, where rs is the random station we are checking and $\text{Max}(O[\text{rs}])$ is the furthest station less than or equal to $k+j_s$.

The algorithm I coded does the same calculation, using a variable p . $p=i_s+k$ and is used to search for the highest value that does not fit in the range from the last station. The one before must be the furthest station available in the range of the battery life, as the one after it in the list exceeded the distance. Since the optimal and my algorithm both check and get the absolute furthest value, with the same set of data, the absolute furthest must be the same for the given battery life. That is to say, $j_{s+1} = i_{s+1}$.

Therefore, by weak induction, this algorithm is an optimal solution.

2. (7 pts) Using Dijkstra's algorithm, determine the length of the shortest path from v_0 to each of the other vertices in the graph. Clearly specify the distances from v_0 to each vertex **after each iteration** of the algorithm.



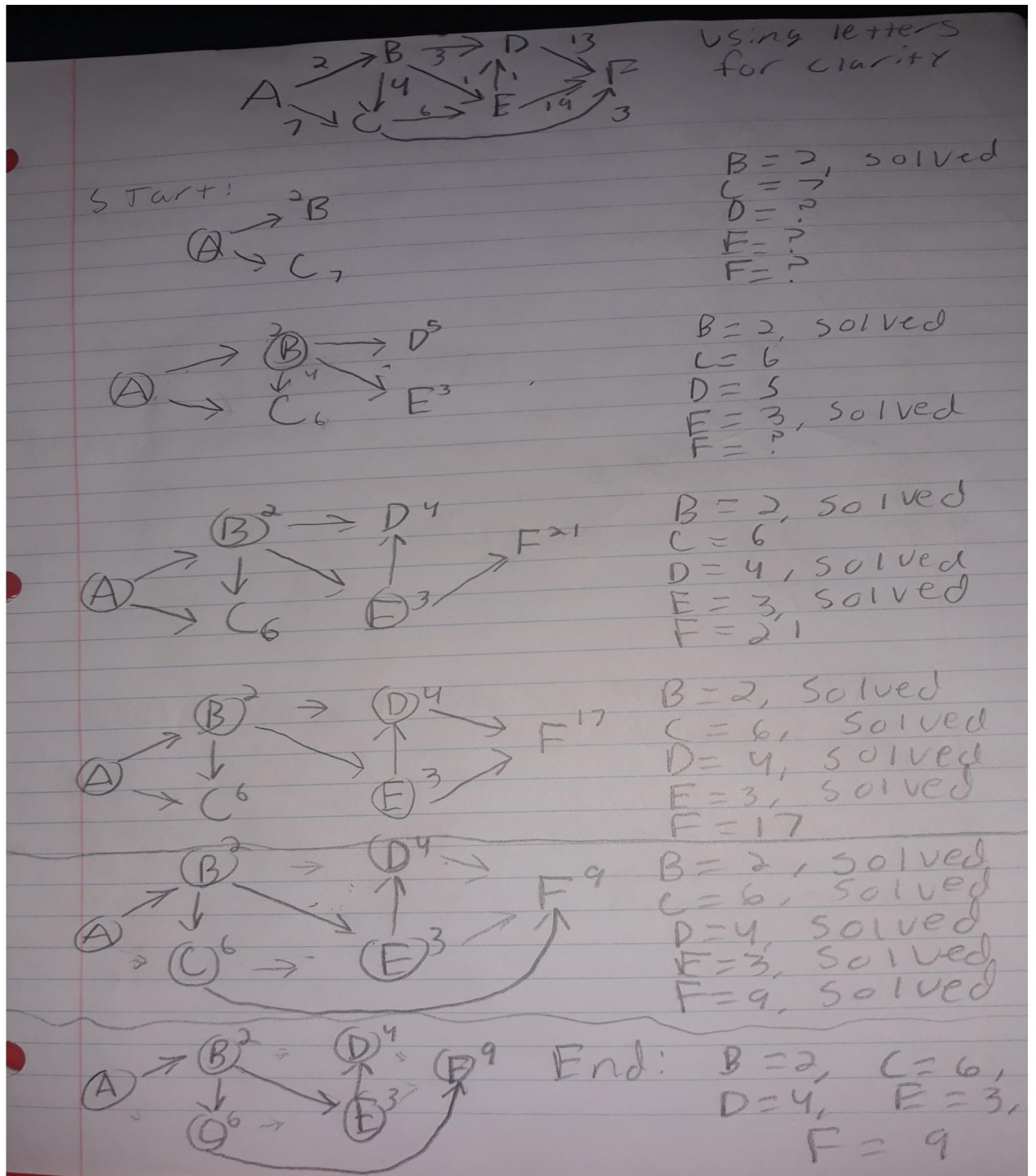
Solution.

Name: Luna McBride

ID: 107607144

CSCI 3104, Algorithms
Problem Set 3b (50 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder



3. (20 pts) After years of futility, the Colorado Rockies have decided to try a new approach to signing players. Next year, they have a target number of wins, n , and they want to sign the fewest number of players who can produce exactly those n wins. In this model, each player has a win value of $v_1 < v_2 < \dots < v_r$ for r player types, where each player's value v_i is a positive integer representing the number of wins he brings to the team. (Note: In a real-world example, All-Star third baseman, Nolan Arenado, contributed 4.5 wins this year beyond what a league-minimum player would have contributed to the team.) The team's goal is to obtain a set of counts $\{d_i\}$, one for each player type (so d_i represents the quantity of players with valuation v_i that are recruited), such that $\sum_{i=1}^r d_i = k$ and where k is the number of players signed, and k is minimized.

- (a) (10 pts) Write a greedy algorithm that will produce an optimal solution for a set of player win values of $[1, 2, 4, 8, 16]$ and prove that your algorithm is optimal for those values. Your algorithm need only be optimal for the fixed win values $[1, 2, 4, 8, 16]$. You do **not** need to consider other configuration of win values.

Solution.

(Code in python code file)

The code has a big O of n , as the loop creates a multiplier of n while all other items remain as constants

The optimal searches the list of items for the specific values $[1, 2, 4, 8, 16]$ and checks specifically if those are already chosen. The minimized player count will then be the chosen based off only one instance of each, putting just as many players as there are instances of each value.

This is what my code does, as it looks specifically for these values specifically, using numpy to easily check the values in the array and the math log function to get the index of the array for already used values. With the values coming out the same and being specifically tailored to this set of values, there is no doubt that this is an optimal algorithm for this playerbase.

CSCI 3104, Algorithms
Problem Set 3b (50 points)

Name: Luna McBride
ID: 107607144
Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

[Additional space for solving Q3a]

CSCI 3104, Algorithms
Problem Set 3b (50 points)

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

- (b) (10 pts) Find a set of win values where your algorithm does not produce the optimal solution and show where your algorithm fails for those values.

Solution.

As this algorithm is heavily tuned to the values [1,2,4,8,16], there is a lot to be desired for other values. Of course, there is the basic things (array counting already used values only going up to 5, for example), but that is to be expected from code like this in general. However, there are more parts that could break under any other value.

if l in values: -> This if checks if the value given is in the list of given values. This will trip up for any value not provided specifically in the list, such as 3 and 5.

p=int(math.log(l,2)), and subsequently if used[p]==1: -> The value p is set up to be specifically used as to not have to index through the array of listed values for keeping track of used values. Since the values [1,2,4,8,16] are all 2^n , it was easier to just log the 2 out to get the indecies than to loop through the set of necessary values. Of course, this could work out for anything else that follows the 2^n structure, however, this vastly neglects values like 3, 5, or any other value that is not of that form. I would have to restructure everything and maybe add another loop to make up for that issue.

Not to add more to the pile, but if we added decimal values in there in addition, it would turn into a nightmare. Perhaps a floor/ceiling function could be used to put those into regular numbers, but that would be disingenuous for us to do given the nature of these averaged wins.

Of course, a lot more could be done to make this optimal for all values, however, in its current state, this algorithm would not only be unoptimal for values outside our list, it would just flat out not work for anything outside the [1,2,4,8,16] range. It is easy to make something super optimal for just one thing like we have done here, but giving it new tricks will completely destroy the whole function if made too crazy optimal.

Ungraded questions - These questions are for your practice. We won't grade them or provide a typed solution but we are open to discuss these in our OHs and you should take feed backs on your approach during the OHs. These questions are part of the syllabus.

1. Suppose we have a directed graph G , where each edge e_i has a weight $w_i \in (0, 1)$. The weight of a path is the product of the weights of each edge.

- (a) Explain why a version of Dijkstra's algorithm cannot be used here. [**Hint:** We may think about transforming G into a graph H , where the weight of edge i in H is $\ln(w_i)$. It is equivalent to apply Dijkstra's algorithm to H .]

Solution.

- (b) What conditions does each edge weight w_i need to satisfy, in order to use Dijkstra's algorithm?

Solution.