# CSCI 3104 PS3b

Jonathan Phouminh

TOTAL POINTS

## 46 / 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 6 / 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) >= p(i+1, J*), and correctly proved that if p(i, j) >= p(i+1, J*), then p(i+1, J) >= 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) >= 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) >= 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 10 / 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

Proof should be more rigorous.

3.2 10 / 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

Name: Jonathan Phouminh

ID: 106054641

**CSCI 3104, Algorithms** **Profs. Hoenigman & Agrawal**

**Problem Set 3b (50 points)** **Fall 2019, CU-Boulder**

---

**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.

---

Name: Jonathan Phouminh

ID: 106054641

**CSCI 3104, Algorithms** **Profs. Hoenigman & Agrawal**

**Problem Set 3b (50 points)** **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 $l_1 < l_2 < \cdots < l_n$ the locations of the charging pods along the route with $l_i$ the distance from the C4C to the $ith$ 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.

| cost | time |
|---|---|
| c1 | 1 |
| c2 | 1 |
| c3 | 1 |
| c4 | n + 1 |
| c5 | n + 1 |
| c6 | n + 1 |

   Find time complexity of this algorithm by adding up all the times of each cost and dropping all constants and only keep highest growing term. Thus, $T(n) = n$

Name: Jonathan Phouminh
ID: 106054641



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

$\boldsymbol{BaseCase}$ **:** For the first element added to the solution set, our greedy algorithm will select one element that is the greatest value in the list that does not exceed current value of k, thus it will insert one element or none at all. Therefore

$$|solutionset_{algorithm}| = 1$$

or

$$|solutionset_{algorithm}| = 0$$

In either case the solution set of our algorithm will be minimal therefore less than or equal to the optimal solution set. Base case holds . . .

$\boldsymbol{InductiveHypothesis}$ **:** Assume that for the $i^{th}$ iteration our algorithm selects and inserts the next greatest element in the list that does not exceed k, thus

$$|solutionset_{algorithm} \leq |solutionset_{optimal}|$$

$\boldsymbol{Proof}$ **:** For our solution set not to be optimal would mean that

$$|solutionset_{algorithm}| > |solutionset_{optimal}|$$

, meaning that our algorithm must have selected another element that wasn't the greatest value before exceeding k at some iteration i. This would violate our inductive hypothesis

$$|solutionset_{algorithm} \leq |solutionset_{optimal}|$$

therefore by contradiction, our algorithm is optimal.

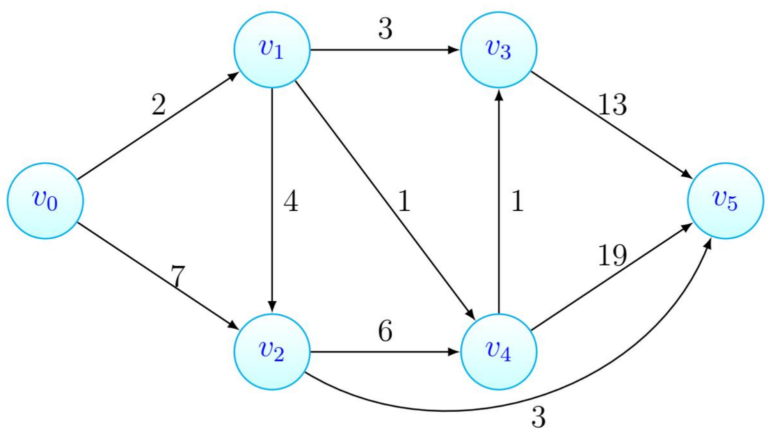
Name: Jonathan Phouminh

ID: 106054641

**CSCI 3104, Algorithms** **Profs. Hoenigman & Agrawal**

**Problem Set 3b (50 points)** **Fall 2019, CU-Boulder**

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.




Iteration 1: $V_0 = 0$
Iteration 2: $V_0 = 0$ , $V_1 = 2$
Iteration 3: $V_0 = 0$ , $V_1 = 2$ , $V_4 = 3$
Iteration 4: $V_0 = 0$ , $V_1 = 2$ , $V_4 = 3$ , $V_3 = 4$
Iteration 5: $V_0 = 0$ , $V_1 = 2$ , $V_4 = 3$ , $V_3 = 4$, $V_2 = 6$
Iteration 6: $V_0 = 0$ , $V_1 = 2$ , $V_4 = 3$ , $V_3 = 4$, $V_2 = 6$ , $V_5 = 9$
end

Name: Jonathan Phouminh

ID: 106054641

**CSCI 3104, Algorithms** **Profs. Hoenigman & Agrawal**
**Problem Set 3b (50 points)** **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 $v_1 < v_2 < \cdots < v_r$ 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.* .

```
//Algorithm will greedily take as many players with highest win rate as it can
//until we get desired amount of wins with minimal players
def getPlayers(winCount)
    if (wincount < 0): return -1
    solutionset = [[]for i in range(5)] //initialize 2d array for wins of 5 columns
   bool canSubstract = true
    while cansubtract:
            if (wincount - 16) ≥ 0
                wincount -= 16
                solutionset [[]4].append(1) //add player to that subset of 16 wins
            elif (wincount - 16) < 0
                canSubtract = false
```

   . . . we will do this process for all win rates $[1, 2, 4, 8, 16]$

   .

   .

   .

[Additional space for solving Q3a]

```
canSubstract = true
while canSubstract
        if (wincount -1) > 0
            wincount -= 1
            solutionset[[]0].append(1)
         elif (wincount -1) ¡ 0
            canSubract = false
```

//At this point we just loop through the 2D array and count all
//elements that are in the 2d array and that will give us a count for
//how many players we need to achieve the desired winCount, and it
//will be minimized.

```
playerCount = 0
for i in range(5)
        for j in range(len(solutionset[[i][]]):
            playerCount = playerCount + solutionset[i][j]
```

***ProvingOptimality***
For our algorihtm to be optimal we would say that our algorithm's solution set size is less than or equal to the size of the optimal solution set.
***BaseCase*** **:** Our algorithm will pick the player with the highest wins and add them to the solution set and it will have at most one element. Thus

$|algorithmset| = 1$ which is trivially $\leq |optimalset|$

Base case holds.
***InductiveHypothesis*** **:** Assume that $r = wincount$ (some value that holds amount of wins needed) our algorithm will add the player whose win value is greater than all subsequent win values and can fit in r will be added to the solution set. Thus,

$$|algorithmset| \leq |optimalset|$$

***Proof*** **:** If $|algorithmset|$ weren't optimal there would be more elements in our algorithms solution set than the optimal solution set. meaning that our algorithm did not select a player with the highest win value that could fit in some r. Since

$$|algorithmset| \leq |optimalset|$$

that extra player chosen cannot have existed in our solution set, so our algorithm is optimal.

Name: Jonathan Phouminh
ID: 106054641



(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.*

We can show where our algorithm fails by considering a scenario and actually showing how the the solution we will get from our algorithm is not the same size or less than the optimal solution.

Consider the set of wins of [1,25,26] with $wincount = 50$. Our algorithm will select 26 and wincount will then be 24. Then our algorithm will continually select 1 until wincount is equal to zero. Our algorithms solution set will be of size 25.

The optimal solution would have been to pick 2 players of value 25 and the solution set would only be of size 2, therefore,

$$|algorithmset| \text{ is not less than or equal to } |optimalset|$$

Thus this is a case where our greedy algorithm fails.

Collaborated with: Zachary Chommalla, Bao Nguyen

Name: Jonathan Phouminh

ID: 106054641

**CSCI 3104, Algorithms** **Profs. Hoenigman & Agrawal**
**Problem Set 3b (50 points)** **Fall 2019, CU-Boulder**

**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.*