CSCI 3104 Fall 2021
Instructor: Profs. Grochow and Waggoner

Problem Set 2

Due Date ..... **September 14, 2021**
Name ..... **Michael Ghattas**
Student ID **109200649**
Collaborators ..... **Me, Myself, and I**

Contents

1 Instructions

- The solutions **must be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Canvas page** only. Please submit one PDF file, compiled using this LaTeX template.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit this document with no fewer pages than the blank template (or Gradescope has issues with it).

- You are welcome and encouraged to collaborate with your classmates, as well as consult outside resources. You must **cite your sources in this document. Copying from any source is an Honor Code violation. Furthermore, all submissions must be in your own words and reflect your understanding of the material.** If there is any confusion about this policy, it is your responsibility to clarify before the due date.
- Posting to **any** service including, but not limited to Chegg, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.
- You **must** virtually sign the Honor Code (see Section 2). Failure to do so will result in your assignment not being graded.

2 Honor Code (Make Sure to Virtually Sign)

Problem 1.
- My submission is in my own words and reflects my understanding of the material.
- Any collaborations and external sources have been clearly cited in this document.
- I have not posted to external services including, but not limited to Chegg, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

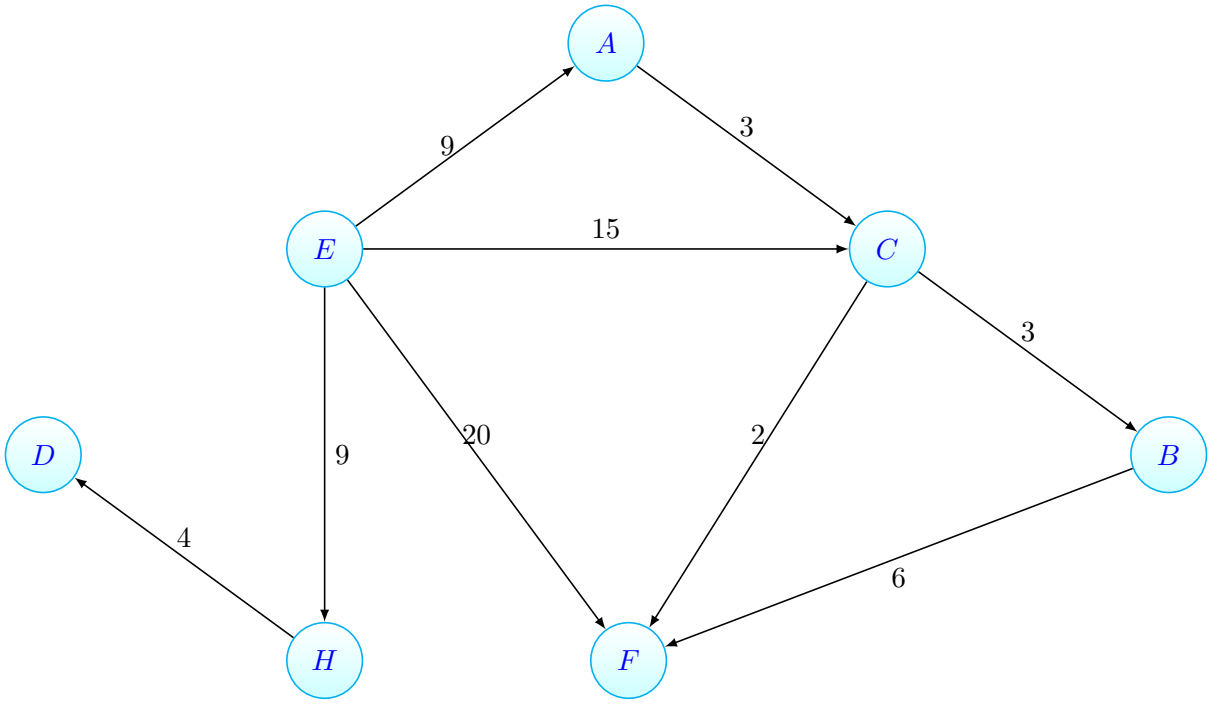
I agree to the above, Michael Ghattas. □

3 Standard 3- Dijkstra's Algorithm

3.1 Problem 2

Problem 2. Consider the weighted graph $G(V, E, w)$ pictured below. Work through Dijkstra's algorithm on the following graph, using the source vertex E.

- Clearly include the contents of the priority queue, as well as the distance from E to each vertex at each iteration.
- If you use a table to store the distances, clearly label the keys according to the vertex names rather than numeric indices (i.e., `dist['B']` is more descriptive than `dist['1']`).
- You do **not** need to draw the graph at each iteration, though you are welcome to do so. [This may be helpful scratch work, which you do not need to include.]



Answer:

Please see the solution Graph and Table on the next page.

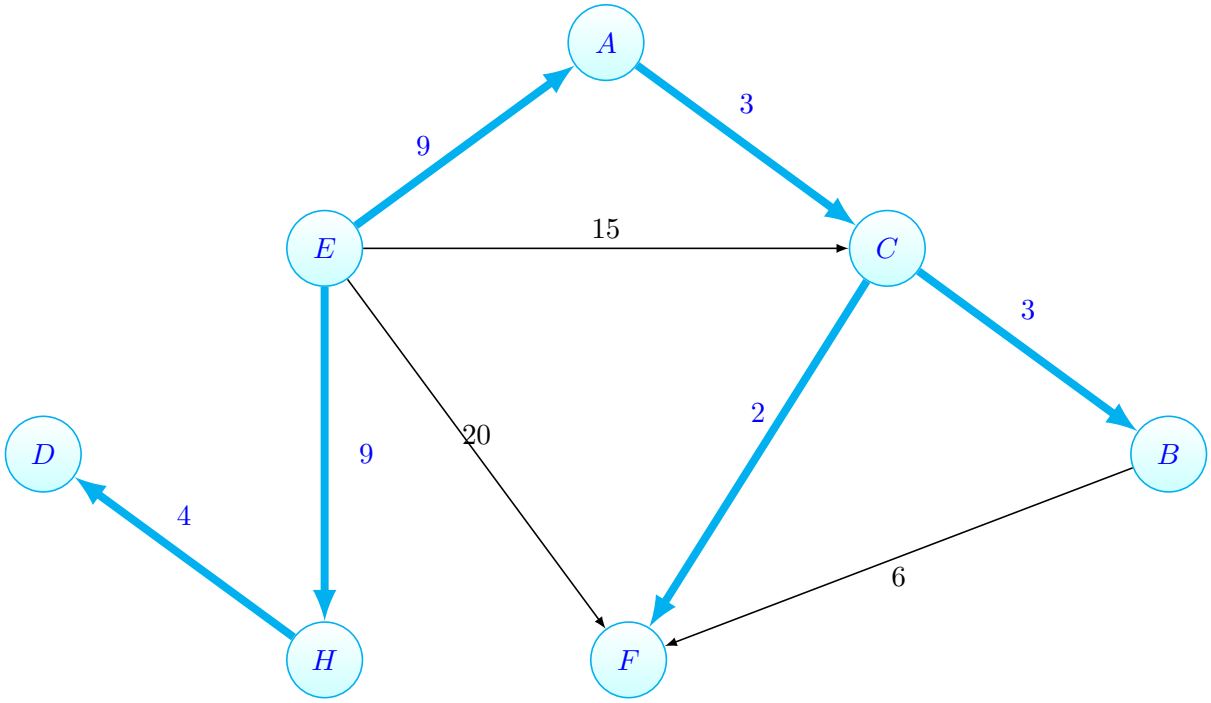


Table 1: **Priority Queue**

	dist[**A**]	dist[**B**]	dist[**C**]	dist[**D**]	dist[**E**]	dist[**F**]	dist[**H**]
1. *Source Vertex* $[E]$	(9)	(—)	(15)	(—)	(0)	(20)	(9)
2. $[EH]$	(9)	(—)	(15)	(13)	(0)	(20)	(9)
3. $[EH] + [EA]$	(9)	(—)	(12)	(13)	(0)	(20)	(9)
4. $[EHD] + [EA]$	(9)	(—)	(12)	(13)	(0)	(20)	(9)
5. $[EHD] + [EAC]$	(9)	(15)	(12)	(13)	(0)	(14)	(9)
6. $[EHD] + [EACF]$	(9)	(15)	(12)	(13)	(0)	(14)	(9)
7. $[EHD] + [EACF] + [EACB]$	(9)	(15)	(12)	(13)	(0)	(14)	(9)

□

3.2 Problem 3

Problem 3. You have three batteries, with 4200, 2700, and 1600 mAh (milli-Amp-hours), respectively. The 2700 and 1600-mAh batteries are fully charged (containing 2700 mAh and 1600 mAh, respectively), while the 4200-mAh battery is empty, with 0 mAh. You have a battery transfer device which has a "source" battery position and a "target" battery position. When you place two batteries in the device, it instantaneously transfers as many mAh from the source battery to the target battery as possible. Thus, this device stops the transfer either when the source battery has no mAh remaining or when the destination battery is fully charged (whichever comes first).

But battery transfers aren't free! The battery device is also hooked up to your phone by bluetooth, and automatically charges you a number of cents equal to however many mAh it just transfered.

The goal in this problem is to determine whether there exists a sequence of transfers that leaves exactly 1200 mAh either in the 2700-mAh battery or the 1600-mAh battery, and if so, how little money you can spend to get this result.

Do the following.

3.2.1 Problem 3(a)

(a) Rephrase this is as a graph problem. Give a precise definition of how to model this problem as a graph, and state the specific question about this graph that must be answered. [**Note:** While you are welcome to draw the graph, it is enough to provide 1-2 sentences clearly describing what the vertices are and when two vertices are adjacent. If the graph is weighted, clearly specify what the edge weights are.]

Answer:

The graph representing the problem would consist of multiple nodes, where each node holds the details of the current status of all the batteries. i.e. **Node(n)**=$[(4200mAh : Status)|(2700mAh : Status)|(1600mAh : Status)]$ **Further more, the edges between adjacent vertices would represent the phase transition between the batteries through charger transfer. Finally, each edge weight would be calculated based on the amount of charge transferred between two batteries. Accordingly, please see Table-2 and Table-3 for more detailed examples.**

Table 2: **Nodes** (*Based on charge distributions*)

Vertex	**4200 mAh** *Capacity*	**2700 mAh** *Capacity*	**1600 mAh** *Capacity*
V(**A**) *Source*	(**Empty**)	(**Full**)	(**Full**)
V(**B**)	(**2700**)	(**Empty**)	(**Full**)
V(**C**)	(**1600**)	(**Full**)	(**Empty**)
V(**D**)	(**Full**)	(**100**)	(**Empty**)
V(**E**)	(**Full**)	(**Empty**)	(**100**)
V(**F**)	(**2700**)	(**1600**)	(**Empty**)
V(**G**)	(**1600**)	(**1100**)	(**Full**)
V(**H**)	(**3200**)	(**1100**)	(**Empty**)
V(....)	(....)	(....)	(....)
V(**etc.**)	(**etc.**)	(**etc.**)	(**etc.**)

□

Table 3: **Edges** (*Based on transfer cost*)

Edge	**4200 mAh** *Capacity*	**2700 mAh** *Capacity*	**1600 mAh** *Capacity*	**Weight**
AB	(**2700**)	(**0**)	(**1600**)	(**2700**)
AC	(**1600**)	(**2700**)	(**0**)	(**1600**)
CD	(**4200**)	(**100**)	(**0**)	(**2600**)
BE	(**4200**)	(**0**)	(**100**)	(**1500**)
BF	(**2700**)	(**1600**)	(**0**)	(**1600**)
CG	(**1600**)	(**1100**)	(**1600**)	(**1600**)
GH	(**3200**)	(**1100**)	(**0**)	(**1600**)
....	(....)	(....)	(....)	(....)
etc.	(**etc.**)	(**etc.**)	(**etc.**)	(**etc.**)

3.2.2 Problem 3(b)

(b) Clearly describe an algorithm to solve this problem. If you use an algorithm covered in class, it is enough to state that. If you modify an algorithm from class, clearly outline any modifications. Make sure to explicitly specify any parameters that need to be passed to the initial function call.

Answer: **Using Dijkstra's Algorithm, we initialization of all nodes with distance =** $infinity$**, while initialization the starting node to 0. We mark the distance of the starting node as permanent, and all other distances as temporarily. Then, we set the starting node as active, and perform calculations of the temporary distances of all neighboring nodes of the active node by summing up the distance based on the weights of the edges. If the calculated distance of a node is smaller than the current one, we update the distance and set the current node as predecessor. We then set the node with the minimal temporary distance as active, and mark its distance as permanent. We repeat steps 4 to 7 until there aren't any nodes left with a permanent distance, and which neighboring nodes still have assigned temporary distances.**

function $Dijkstra(Graph, source)$:

for each vertex v in $Graph$:

dist$[v]$:= $infinity$

previous$[v]$:= $undefined$

dist$[source]$:= 0

Q := the set of all nodes in $Graph$

while Q is not $empty$:

u := $node$ in Q with $smallest$ dist[ ]

remove u from Q

for each neighbor v of u:

alt := dist$[u]$ + $DistBetween(u, v)$

if $alt <$ dist$[v]$

dist$[v]$:= alt

previous$[v]$:= u

update(Q)

return $Graph$; □

3.2.3 Problem 3(c)

(c) Apply that algorithm to the question. Report and justify your answer. Here, justification includes the sequences of vertices visited and the total cost.

Answer:

While there are sure to be multiple solutions possible through vertices with different charge combinations, using the information from part(a) and part(b) we should get the following lowest cost of operation and visited vertices:

V(*)[4200mAh-Capacity:$Charge$**, 2700mAh-Capacity:**$Charge$**, 1600mAh-Capacity:**$Charge$]

1. **V(1)[4200:**0**, 2700:**2700**, 1600:**1600]

   Edge: $V(1) \rightarrow V(2) := weight(2700)$

2. **V(2)[4200:**2700**, 2700:**0**, 1600:**1600]

   Edge: $V(2) \rightarrow V(3) := weight(1500)$

3. **V(3)[4200:**4200**, 2700:**0**, 1600:**100]

   Edge: $V(3) \rightarrow V(4) := weight(2700)$

4. **V(4)[4200:**1500**, 2700:**2700**, 1600:**100]

   Edge: $V(4) \rightarrow V(5) := weight(1500)$

5. **V(5)[4200:**1500**, 2700:**1200**, 1600:**1600]

Cost = weight(2700) + weight(1500) + weight(2700) + weight(1500) = 8400_c

□

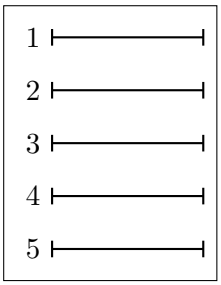
4 Standard 4- Examples Where Greedy Algorithms Fail

4.1 Problem 4

Problem 4. Recall the Interval Scheduling problem, where we take as input a set of intervals $\mathcal{I}$. The goal is to find a maximum-sized set $S \subseteq \mathcal{I}$, where no two intervals in S intersect. Consider the greedy algorithm where we place all of the intervals of $\mathcal{I}$ into a priority queue, ordered earliest start time to latest start time. We then construct a set S by adding intervals to S as we poll them from the priority queue, provided the element we polled does not intersect with any interval already in S.

Provide an example with at least 5 intervals where this algorithm fails to yield a maximum-sized set of pairwise non-overlapping intervals. Clearly specify both the set S that the algorithm constructs, as well a larger set of pairwise non-overlapping intervals.

You may explicitly specify the intervals by their start and end times (such as in the examples from class) or by drawing them. **If you draw them, please make it very clear whether two intervals overlap.** You are welcome to hand-draw and embed an image, provided it is legible and we do not have to rotate our screens to grade your work. Your justification should still be typed. If you would prefer to draw the intervals using LaTeX, we have provided sample code below.



Answer:

Below is an example how the greedy algorithm fails:

Let our set of intervals be int$[StartTime_i : EndTime_i]$.

Setting the order of intervals from earliest to latest start-time, left-to-right, we get our *PriorityQueue***:**

$\mathcal{I}$ = [int(1 : 12) , int(2 : 3) , int(4 : 5) , int(6 : 7) , int(8 : 9), int(10 : 11)]

Let our *answer* set of intervals be (S).

We start to populate our stack, and we first stack int(1 : 12), thus S = [int(1 : 12)]

Next, due to overlap with [int(1 : 11)], we realize that we are unable to add to S any

of the other intervals; [int(2 : 3) , int(4 : 5) , int(6 : 7) , int(8 : 9), int(10 : 11)].

Thus our final solution is S = [int(1 : 12)]

However, this is clearly not the optimal solution-set where we can stack multiple intervals to our queue. The optimal solution would consider the earliest end times instead.

Let $O = (Optimal\ Solution)$:

Then the optimal solution would instead be: (O) = [int(2 : 3) , int(4 : 5) , int(6 : 7) , int(8 : 9), int(10 : 11)]

Accordingly we can see how our example and provided greedy algorithm failed to provide us with the optimal solution-set for the given problem! □

4.2 Problem 5

Problem 5. Consider now the Weighted Interval Scheduling problem, where each interval i is specified by

$$([\text{start}_i, \text{end}_i], \text{weight}_i).$$

Here, the weight is an assigned value that is independent of the length $\text{end}_i - \text{start}_i$. Here, you may assume $\text{weight}_i > 0$. We seek a set S of pairwise non-overlapping intervals that maximizes $\sum_{i \in S} \text{weight}_i$. That is, rather than maximizing the number of intervals, we are seeking to maximize the sum of the weights.

Consider a greedy algorithm which works identically as in Problem 4. Draw an example with at least 5 appointments where this algorithm fails. Show the order in which the algorithm selects the intervals, and also show a subset with larger weight of non-overlapping intervals than the subset output by the greedy algorithm. The same comments apply here as for Problem 4 in terms of level of explanation.

Answer:

Below is an example how the greedy algorithm fails:

Let our set of intervals be $\text{int}[StartTime_i : EndTime_i, Weight_i]$.

Setting the order of intervals from highest to lowest weight, left-to-right, we get our $PriorityQueue$**:**

$\mathcal{I} = [\text{int}(2:12, 200)\ , \text{int}(9:10, 100)\ , \text{int}(7:8, 75)\ , \text{int}(5:6, 50)\ , \text{int}(3:4, 25), \text{int}(1:3, 10)]$

Let our *answer* set of intervals be (S).

We start to populate our stack, and we first stack $\text{int}(1:3, 10)$**, then replace it with** $\text{int}(2:12, 200)$ **due to the higher weight and overlap. Thus initially** $S = [\text{int}(2:12, 200)]$

Next, due to overlap with $[\text{int}(2:12, 200)]$, **we realize that we are unable to add to** S any of the other intervals; $[\text{int}(9:10, 100)\ , \text{int}(7:8, 75)\ , \text{int}(5:6, 50)\ , \text{int}(3:4, 25)]$, or bring back $\text{int}(1:3, 10)$.

Thus our final solution is $S = [\text{int}(2:12, 200)] = 200$.

However, this is clearly not the optimal solution-set where we can stack to our queue multiple intervals that can add-up to a larger total weight. The optimal solution would consider the higher weight along with the earliest end times instead.

Let $O = (Optimal\ Solution)$:

Then the optimal solution would instead be: $(O) = \{[\text{int}(9:10, 100) + \text{int}(7:8, 75) + \text{int}(5:6, 50) + \text{int}(3:4, 25)] = [100 + 75 + 50 + 25 = 250]\} > \{[S = \text{int}(2:12, 200)] = [200]\}$, Thus $[(O) > (S)]$.

Accordingly we can see how our example and provided greedy algorithm failed to provide us with the optimal solution-set for the given problem!

□

5 Standard 5- Exchange Arguments

5.1 Problem 6

Problem 6. Recall the Making Change problem, where we have an infinite supply of pennies (worth 1 cent), nickels (worth 5 cents), dimes (worth 10 cents), and quarters (worth 25 cents). We take as input an integer $n \geq 0$. The goal is to make change for n using the fewest number of coins possible.

Prove that in an optimal solution, we use at most 2 dimes.

Answer:

We will be using the below greedy algorithm, along with the notes from the class lecture (Chapter 3 - 3.1.1). Here we take in the input (n) as per the given problem, which is the number of cents provided. i.e. Let $n \in \mathbb{N}$ be the amount for which we wish to make change. In an optimal solution, we have at most one nickel.

Let (S) be the multi-set of coins used to make change for (n). Suppose that (S) contains $k > 1$ nickels. The key idea is that we may exchange each pair of nickels for a single dime. For example, based on our above division greedy Algorithm, $k = [(2j) + r]$, where $j \in \mathbb{N}$ and $r \in [0, 1]$. When $k > 1$, $j \geq 1$. We exchange $(2j)$ nickels for (j) dimes, and so on, to obtain a new optimal solution set $|S'|$, where $|S'| = [(|S| - j) < |S|]$.

Algorithm pseudocode :

function $CoinSort(n)$:

```
Count = [Quarters, Dimes, Nickels, Pennies]
Quarters = 0
Dimes = 0
Nickels = 0
Pennies = 0
while (n != 0) :
    if (n%25) :
        n = (n − 25)
        Quarters++
    else if (n%10) :
        n = (n − 10)
        Dimes++
    else if (n%5) :
        n = (n − 5)
        Nickels++
    else
        n = (n − 1)
        Pennies++
return(Count);
```

□

5.2 Problem 7

Problem 7. Consider the Interval Projection problem, which is defined as follows.

- Instance: Let $\mathcal{I}$ be a set of intervals on the real line.
- Solution: A minimum sized set S of points on the real line, such that (i) for every interval $[s, f] \in \mathcal{I}$, there exists a point $x \in S$ where x is in the interval $[s, f]$. We call S a *projection set*.

Do the following.

5.2.1 Problem 7(a)

(a) Find a minimum sized projection set S for the following set of intervals:

$$\mathcal{I} = \{[0, 1], [0.5, 1], [1, 1.5], [1.4, 2], [1.6, 2.3]\}.$$

Answer:

Let $\mathcal{I}$ be made of the intervals:
A = [0,1]
B = [0.5,1]
C = [1,1.5]
D = [1.4,2]
E = [1.6,2.3]

Let $\mathcal{K}$ = [Set of overlapping intervals]

$\mathcal{K}$ = {[0.5,1] , [1.4,1.5] , [1.6,2]}

Let $\mathcal{K}$ be made of the intervals:
α = [0.5,1]
β = [1.4,1.5]
γ = [1.6,2]

We can note that:
The interval α has the point **1** $\in$ (A, B & C)
The interval γ has the point **2** $\in$ (D & E)

If $\mathcal{S}$ represents the projection set, then we can clearly note that the set of points {1,2} represents the intervals $\in \mathcal{I}$. Thus:

$\mathcal{S}$ = {1, 2}

$|\mathcal{S}|$ = (2) □

5.2.2 Problem 7(b)

(b) Fix a set of intervals $\mathcal{I}$, and let S be a projection set. Prove that there exists a projection set S' such that (i) $|S'| = |S|$, and (ii) where every point $x \in S'$ is the right end-point of some interval $[s, f] \in \mathcal{I}$.

Answer:

Let $\mathcal{I}$ is made of a set of intervals, where each $interval_i$ is made of a set of points encapsulated by a start-point (s_i) and an end-point (f_i).

We assume each $interval_i$ is made of $[s_i, ..., f_i] \in \mathbb{R}$, where $s_i \leq x_i \leq f_i$.

We assume $\forall$ $interval_i \in \mathcal{I}$ ($f_1 \leq f_i \leq f_{i+1} \leq f_n$).

We assume $x_i \in \forall$ $interval_i \in \mathcal{I}$, then $x_i \in \mathcal{S}$.

Let δ = [Interval with least end-point]. i.e. $f_1 \in \delta$.

Let $\delta(x_i) = \epsilon = (f_1)$, where we have intervals intersecting at ϵ, and $\epsilon = f_\delta = f_1$.

Then $\epsilon \in \mathcal{S}$, and $\epsilon \in \mathcal{S}'$.

Repeating this process (n) times will give us $\mathcal{S}$, where $|\mathcal{S}| = |\mathcal{S}'|$. □