Final Project

December 20, 2019

1 CS110: COMPUTATION: SOLVING PROBLEMS WITH ALGORITHMS

1.1 Final Assignment

1.1.1 Improved Task Scheduler

2 1). Activity Scheduler

2.0.1 How the scheduler works

We are given a set of n activities with start time, duration, and the profit associated with executing the activity. We want to get such subset of x activites that is $x <= n$, does not contain overlapping activities, and that maximizes the maximum profit from all the activities performed. To start off, we need to sort our activities by their end time, which we calculate by adding the duration to the start time. Sorting the input in such a way allows us to use the Dynamic Programming. We have two variables: i, and j, which will be used to change the subsets of activites that we analyze. We also define a function $p(j)$, which will constitute the rightmost activity i, where i and j are non-overlapping and $i < j$. Using this function will give us the first activity to the left of j that isn't overlapping with it. For the purpose of locating this activity, we could use different approaches, however, binary search significantly decreases the running time. If we used an iterative solution, our algorithm could yield even $O(n * 2^n)$ time complexity which is extremely unfavorable. Solution bsed on binary search gives us $O(nlog(n))$ complexity - because we run binary search over all j's, whose amount is n - therefore, $nlog(n)$. Next, we can infer that the last rightmost subset of activities either is a part of globally optimal solution or is not. Such a dichotomy leads us to two cases: if n is a part of the optimal solution, then none of the smaller subsets will be better than the result from this solution (since all the other subsets are just subsets of n). On the contrary, if n does not belong to the optimal solution, then we know that our optimal solution has to be somewhere in the smaller subset of activities being $ 1, 2, . . . , p(n)$. These two cases support the idea that this problem reflects the optimal subproblem structure. Also, knowing them, allows us to formulate a strategy for finding the globally optimal solution using the Dynamic Programming. We will look at the bigger and bigger subsets at each iteration and at each step save the best potential solution out of the two cases (starting from subset of 1 activity and ending with all of them). In other words, we will look for the optimal solution by taking the maximum out of the two cases, as represented in this pseudocode: $optimum(j) = maximum(profit_j + optimum(p(j)), ooptimum(j-1))$. While solving the recurrence, we will store the results in a table for the purposes of memoization. In the last step, these memoized subsolutions are used

2.0.2 Scheduler's Time Complexity Analysis

Let's start with decomposing our algorithm and analyzing the time complexity of its chunks.

- First, we are sorting the activities by their end times. Depending on the algorithm we choose, the time complexity will differ. In this case, we use the python built-in function .sort, which by default uses quicksort. Hence, its time complexity will be: $\Omega(nlog(n))$, $\Theta(nlog(n))$, $O(n^2)$.
- Next, we calculate the values for the p function. Using the binary search method, as me ntioned in the upper part, yields $O(logn)$ complexity. Since we do it for all the n number of items, we have to multiply the complexities, reaching $O(nlog(n))$.
- Using the memoization and bottom-up approach, we get the linear $O(n)$ time complexity of calculating the values of all the optimal subproblems. -In the last step, we traverse the list of solutions to subproblem, which also has a linear time $O(n)$. Finally, we are able to get the time complexity of the entire algorithm. It is dominated by the $O(nlog(n))$ complexity.

In terms of the space complexity, our algorithm is also doing quite well. It is not working in the in-space manner, meaning that it needs additional space than only that taken by the input. It is because the algorithm uses the memoization technique. However, maintianing this table will still be much less space demanding than e.g. using a dataframe for a scheduler, as was in my LBA scheduler implementation.

In [330]:
```
#The code is mostly based on the implementation by Victor Farazdagi
#which can be found under this link:
#https://farazdagi.com/2013/weighted-interval-scheduling/?fbclid=IwAR0YNAhQYO8W5NmAB
import collections
import bisect

#defining the Activity class with the activity's name, start time, duration, and pro
#I will use the end-time rather than duration to implement the Dynamic Programming
class Activity(object):

    def __init__(self, name, start_time, duration, profit):
        self.name = name
        self.start_time = start_time
        self.end_time = start_time+duration
        self.profit = profit


#Function that turns the array of activities into class types
def create_activities(set_activities):
    global activities
    activities=[]
    for i in range(len(set_activities)):
        activities.append(Activity(name=set_activities[i][0],
                                   start_time=set_activities[i][1],
                                   duration=set_activities[i][2],
```

```
                                        profit=set_activities[i][3]))

    return activities


def compute_previous_activities(activities):
    #This function computes the latest activity that
    #is compatible with the activity j
    #arrays filled with starting and ending times
    start = [i.start_time for i in activities]
    end = [i.end_time for i in activities]

    p = []
    #the bisect is used as an alternative to binary search which
    #decreases the time complexity from O(n^2)
    #for the recursive solution to O(nlog(n)) in this implementation
    for j in range(len(activities)):
        i = bisect.bisect_right(end, start[j]) - 1
        p.append(i)

    return p

def schedule_weighted_activities(activities):
#Scheduler using Dynamic Programming
    #sorting the activities by their end times using the quicksort
    activities.sort(key=lambda x: x.end_time)
    #calling the previously defined function to keep the
    #rightmost activities
    p = compute_previous_activities(activities)

    # compute subsolutions
    sub_solutions = collections.defaultdict(int)
    sub_solutions[-1] = 0
    sub_solutions[0] = 0
    #using the dichotomy discussed earlier to calculate optimal subproblems
    for j in range(1, len(activities)):
        sub_solutions[j] = max(activities[j].profit + sub_solutions[p[j]],
                               sub_solutions[j - 1])

    #finding the final optimal solution from our subproblems
    solution = []
    def compute_solution(j):
        if j >= 0:
            if activities[j].profit + sub_solutions[p[j]] > sub_solutions[j - 1]:
                solution.append(activities[j])
                compute_solution(p[j])
            else:
                compute_solution(j - 1)
```

```
        compute_solution(len(activities) - 1)

        #sort the solutions to get the correct order
        solution.sort(key=lambda x: x.end_time)

        return solution


    #creating the input
    list_of_activities=[['Brush teeth',1, 5, 10], ['Meditate',10, 20, 20],
                        ['Prepare and eat a healthy breakfast',60, 30, 20],
                        ['Pre-class work for CS110',250, 200, 70],
                        ['Go to CS110 class', 455, 90, 100],
                        ['Work for my internship', 650, 120, 60],
                        ['Hang out with friends', 260, 20, 5],
                        ['Listen to my favorite podcast', 700, 30, 10],
                        ['Go to the gym', 100, 100, 50],
                        ['Talk to my parents', 900, 15, 10],
                        ['Reflect of the day', 1000, 15, 6],
                        ['Explore the city', 540, 120, 50]]

    #defining the function that combines all the required steps
    def scheduler(list_of_activities):
        activities=create_activities(list_of_activities)
        solution = schedule_weighted_intervals(activities)
        print("The optimal schedule is:")
        for i in range(len(solution)):
                print(solution[i].name)
```

```
In [331]: scheduler(list_of_activities)
```

```
The optimal schedule is:
Brush teeth
Meditate
Prepare and eat a healthy breakfast
Go to the gym
Pre-class work for CS110
Go to CS110 class
Work for my internship
Talk to my parents
Reflect of the day
```

---

3 2). Testing LBA

```
In [336]: #copy-pasting the code from LBA
          import pandas as pd
```

```
from random import randint

#Naming the columns
column_names=['Activity','Task ID','Task description', 'Task duration in minutes', '
#Adding the data of tasks to the data list composed of lists (each nested list is a
data=[['CARE','CARE|01','Morning meditation',20,False,None,'NOT-YET-STARTED',0],
      ['CARE','CARE|02','Taking a shower',15,False,None,'NOT_YET_STARTED',0],
      ['CARE','CARE|03','Setting my goals for the day',10,False,None,'NOT_YET_STARTE
      ['CARE','CARE|04','Preparing a healthy breakfast',15,True,None,'NOT_YET_STARTE
      ['CARE','CARE|05','Reflecting on the day',15,False,None,'NOT_YET_STARTED',0],
      ['CARE','CARE|06','Going to the gym',120,False,None,'NOT_YET_STARTED',0],
      ['WORK','WORK|01','Data exploration of a new dataset for my internship',90,Tru
      ['WORK','WORK|02','Setting the roadmap for building the model',15,True,'WORK|0
      ['WORK','WORK|03','Update the manager',5,False,['WORK|01','WORK|02'],'NOT_YET_S
      ['WORK','WORK|04','Do some work-study',60,True,None,'NOT_YET_STARTED',0],
      ['WORK','WORK|05','Prepare for the meeting with Civic Partner',70,True,None,'NO
      ['ACD','ACD|01','Do the readings for CS110',60,True,None,'NOT_YET_STARTED',0],
      ['ACD','ACD|02','Pre-class work for CS110',90,True,'ACD|01','NOT_YET_STARTED',0
      ['ACD','ACD|03','Take the CS110 class',90,False,['ACD|01','ACD|02'],'NOT_YET_ST
      ['ACD','ACD|04','Do the readings for CS111A',60,False,None,'NOT_YET_STARTED',0]
      ['ACD','ACD|05','CS111A pre-class',45,True,'ACD|04','NOT_YET_STARTED',0],
      ['ACD','ACD|06','CS111A class',90,False,['ACD|04','ACD|05'],'NOT_YET_STARTED',0
      ['ACD','ACD|07','AH111 Readings',100,True,None,'NOT_YET_STARTED',0],
      ['ACD','ACD|08','AH111 Pre-class',100,True,['ACD|07'],'NOT_YET_STARTED',0],
      ['SOC','SOC|01','Hang out with friends',120,True,None,'NOT_YET_STARTED',0],
      ['SOC','SOC|02','Grab a dinner with roommates',35,True,None,'NOT_YET_STARTED',0
      ['SOC','SOC|03','Meet my civic partner',120,False,None,'NOT_YET_STARTED',0],
      ['SOC','SOC|04','Talk to my parents',15,False,None,'NOT_YET_STARTED',0],
      ['GEM','GEM|01','Listen to my favorite podcast',35,True,None,'NOT_YET_STARTED'
      ['GEM','GEM|02','Watch some youtube',20,True,None,'NOT_YET_STARTED',0],

      ]


#Creating a dataframe of our tasks
df = pd.DataFrame(columns=column_names, data=data)

def priority_evaluation(i):
    #Checking if the task has any dependencies.
    #If it has none, +50 to the priority since that
    #indicates the first task from the activity
    if df['Dependencies'][i]==None:
        df['Priority'][i]+=50

    #If it has, +5 to the priority of the task that it is
    #dependent upon will be added
    if df['Dependencies'][i]!=None:
        for j in range(len(df['Dependencies'][i])):
```

```
            df.loc[df['Task ID']==df['Dependencies'][i][j], ['Priority']]+=5
    #If the task can be executed simultanously with another one, +1 added to priorit
    if df['Multitasking'][i]==True:
        df['Priority'][i]+=1

    #If it's a long task, +3 added to priority
    if df['Task duration in minutes'][i]>120:
        df['Priority'][i]+=3
    #If it's a medium-long task, +2 added to priority
    elif ((df['Task duration in minutes'][i]<=120) & (df['Task duration in minutes']
        df['Priority'][i]+=2
    #If it's a short task, +1 added to priority
    elif ((df['Task duration in minutes'][i]<60) & (df['Task duration in minutes'][i
        df['Priority'][i]+=1

    #If it's one of the GEM type of tasks, it's priority is set to -1
    #It is because we will use this tasks complementary to the others
    #and we don't want to include them in the regular heap sorting
    if 'GEM' in df['Task ID'][i]:
        df['Priority'][i]=(-1)

#Running the priority_evaluation functions on the entire dataframe
for i in range(len(df)):
    priority_evaluation(i)

#
# Defining some basic binary tree functions
#
def left(i):          # left(i): takes as input the array index of a parent node in t
    return 2*i + 1    #          returns the array index of its left child.

def right(i):         # right(i): takes as input the array index of a parent node in
    return 2*i + 2    #           returns the array index of its right child.

def parent(i):        # parent(i): takes as input the array index of a node in the bi
    return (i-1)//2   #            returns the array index of its parent


# Defining the Python class MaxHeapq to implement a max heap data structure.
# Every Object in this class has two attributes:
#           - heap : A Python list where key values in the max heap are stored
#           - heap_size: An integer counter of the number of keys present in the max
class MaxHeapq:
    """
    This class implements properties and methods that support a max priority queue d
    """
    # Class initialization method. Use: heapq_var = MaxHeapq()
    def __init__(self):
```

```
    self.heap       = []
    self.heap_size  = 0

# This method returns the highest key in the priority queue.
#   Use: key_var = heapq_var.max()
def maxk(self):
    return self.heap[0]

# This method implements the INSERT key into a priority queue operation
#   Use: heapq_var.heappush(key)
def heappush(self, key):
    """
    Inserts the value of key onto the priority queue, maintaining the max heap i
    """
    self.heap.append(-float("inf"))
    self.increase_key(self.heap_size,key)
    self.heap_size+=1

# This method implements the INCREASE_KEY operation, which modifies the value of
# in the max priority queue with a higher value.
#   Use heapq_var.increase_key(i, new_key)
def increase_key(self, i, key):
    if key[7] < self.heap[i]:
        raise ValueError('new key is smaller than the current key')
    self.heap[i] = key
    while i > 0 and self.heap[parent(i)][7] < self.heap[i][7]:
        j = parent(i)
        holder = self.heap[j]
        self.heap[j] = self.heap[i]
        self.heap[i] = holder
        i = j

# This method implements the MAX_HEAPIFY operation for the max priority queue. T
# the array index of the root node of the subtree to be heapify.
#   Use heapq_var.heapify(i)
def heapify(self, i):
    l = left(i)
    r = right(i)
    heap = self.heap
    if l <= (self.heap_size-1) and heap[l][7]>heap[i][7]:
        largest = l
    else:
        largest = i
    if r <= (self.heap_size-1) and heap[r][7] > heap[largest][7]:
        largest = r
    if largest != i:
        heap[i], heap[largest] = heap[largest], heap[i]
        self.heapify(largest)
```

```
    # This method implements the EXTRACT_MAX operation. It returns the largest key i
    # the max priority queue and removes this key from the max priority queue.
    #   Use key_var = heapq_var.heappop()
    def heappop(self):
        if self.heap_size < 1:
            raise ValueError('Heap underflow: There are no keys in the priority queue
        maxk = self.heap[0]
        self.heap[0] = self.heap[-1]
        self.heap.pop()
        self.heap_size-=1
        self.heapify(0)
        return maxk

#Checking which activity has the highest priority
priorities_sum=df.groupby("Activity")['Priority'].sum()
priorities_sum.sort_values(ascending=False).index

#Below part of code from the code used in LBA assignment was highly dependent
#on the input. I needed to change it according to the new input
#It is worth noting that this shows one of the weaknesses of the previous
#implementation. I will tackle this problem in the improved version of the scheduler
"""
#Creating a priority queues for each activity in
#the sorted order by the activity sum priority
GBP_heap = MaxHeapq()
GBP_list=df[df['Activity']=='GBP'].values.tolist()
for key in GBP_list:
    GBP_heap.heappush(key)

SFL_heap = MaxHeapq()
SFL_list=df[df['Activity']=='SFL'].values.tolist()
for key in SFL_list:
    SFL_heap.heappush(key)

BSB_heap = MaxHeapq()
BSB_list=df[df['Activity']=='BSB'].values.tolist()
for key in BSB_list:
    BSB_heap.heappush(key)

SFW_heap = MaxHeapq()
SFW_list=df[df['Activity']=='SFW'].values.tolist()
for key in SFW_list:
    SFW_heap.heappush(key)

DDP_heap = MaxHeapq()
DDP_list=df[df['Activity']=='DDP'].values.tolist()
for key in DDP_list:
```

```
    DDP_heap.heappush(key)

BBQ_heap = MaxHeapq()
BBQ_list=df[df['Activity']=='BBQ'].values.tolist()
for key in BBQ_list:
    BBQ_heap.heappush(key)

GEM_heap = MaxHeapq()
GEM_list=df[df['Activity']=='GEM'].values.tolist()
for key in GEM_list:
    GEM_heap.heappush(key)

#Adding all the activity heaps to a list
all_heaps=[GBP_heap, SFL_heap, BSB_heap, SFW_heap, DDP_heap, BBQ_heap, GEM_heap]"""

CARE_heap = MaxHeapq()
CARE_list=df[df['Activity']=='CARE'].values.tolist()
for key in CARE_list:
    CARE_heap.heappush(key)

WORK_heap = MaxHeapq()
WORK_list=df[df['Activity']=='WORK_heap'].values.tolist()
for key in WORK_list:
    WORK_heap.heappush(key)

ACD_heap = MaxHeapq()
ACD_list=df[df['Activity']=='ACD'].values.tolist()
for key in ACD_list:
    ACD_heap.heappush(key)

SOC_heap = MaxHeapq()
SOC_list=df[df['Activity']=='SOC'].values.tolist()
for key in SOC_list:
    SOC_heap.heappush(key)

GEM_heap = MaxHeapq()
GEM_list=df[df['Activity']=='GEM'].values.tolist()
for key in GEM_list:
    GEM_heap.heappush(key)


#Adding all the activity heaps to a list
all_heaps=[CARE_heap, WORK_heap, ACD_heap, SOC_heap, GEM_heap]

time=0 #setting the time to 0

for i in range(len(all_heaps)): #running the function through all the activities
    #running the function through all the tasks within an activity
```

```
while all_heaps[i].heap_size>0:
    #choosing a random number between 0 and 2.
    #It will decide whether or not we will execute a multitask alongside the
    #main one
    luck = randint(0,4)
    #popping the root of the heap - the task with the highest priority
    task=all_heaps[i].heappop()
    task[6]='IN-PROGRESS' #setting the status to "IN-PROGRESS"
    #Creating the condition for multitasking
    if task[4]==True:
        gem=None
        #gem is the task that can be executed simultanously with any other
        #task in the list
        if GEM_heap.heap and (luck==1 or 2): #40% of having luck, i.e.
                                             #executing multitask
            gem=GEM_heap.heappop()

        if gem==None: #if none is found, the function continues with
                      #executing only one task
            time+=task[3] #adding the duration of the task to the total time
            task[6]='COMPLETED' #setting the status to COMPLETED
            print(task[2]) #printing the statement of the action
            print('Time taken:',task[3])
            print('Total time:',time)
            print('----------------------------------------')

        else:
            if gem[3]>task[3]: #setting the execution flow when the duration of
                               #GEM is longer than the main task
                duration=gem[3]#saving the time taken to execute the side task
                gem[3]-=task[3]#subtractting the time of the task from the gem
                               #time
                time+=task[3]*2 #adding the time of task*2 because we also count
                           #the time taken to accompllish the part of GEM
                gem[6]='IN-PROGRESS'
                GEM_heap.heappush(gem)#pushing back the gem to its heap
                task[6]='COMPLETED'
                print(task[2])
                print('Multitasking with', gem[2], 'for',gem[3], 'minutes')
                print('Time taken:',(task[3]*2))
                print('Total time:',time)
                print('----------------------------------------')

            #setting the condition for when task duration
            #is bigger than the gem's
            elif gem[3]<task[3]:
                duration=gem[3]+task[3]
                task[3]-=gem[3]
```

```
                    time+=gem[3]*2
                    gem[6]='COMPLETED'
                    time+=task[3]
                    print(task[2])
                    print('Multitasking with', gem[2], 'for',gem[3], 'minutes')
                    print('Time taken:',duration)
                    print('Total time:',time)
                    print('----------------------------------------')

                #setting the condition for when task duration
                #is equal to that of the gem
                else:
                    time+=task[3]*2
                    task[6]='COMPLETED'
                    gem[6]='COMPLETED'
                    print(task[2])
                    print('Multitasking with', gem[2], 'for', gem[3], 'minutes')
                    print('Time taken:',task[3]*2)
                    print('Total time:',time)
                    print('----------------------------------------')
            else:
                time+=task[3] #adding the duration of the task to the total time
                task[6]='COMPLETED' #setting the status to COMPLETED
                print(task[2]) #printing the statement of the action
                print('Time taken:',task[3])
                print('Total time:',time)
                print('----------------------------------------')
```

```
/Users/kubawarmuz/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:45: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/kubawarmuz/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:64: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/kubawarmuz/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:54: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/kubawarmuz/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:61: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html


Preparing a healthy breakfast
Multitasking with Listen to my favorite podcast for 20 minutes
```

```
Time taken: 30
Total time: 30
----------------------------------------
Going to the gym
Time taken: 120
Total time: 150
----------------------------------------
Reflecting on the day
Time taken: 15
Total time: 165
----------------------------------------
Taking a shower
Time taken: 15
Total time: 180
----------------------------------------
Morning meditation
Time taken: 20
Total time: 200
----------------------------------------
Setting my goals for the day
Time taken: 10
Total time: 210
----------------------------------------
Do the readings for CS110
Multitasking with Watch some youtube for 20 minutes
Time taken: 80
Total time: 290
----------------------------------------
AH111 Readings
Multitasking with Listen to my favorite podcast for 20 minutes
Time taken: 120
Total time: 410
----------------------------------------
Do the readings for CS111A
Time taken: 60
Total time: 470
----------------------------------------
Pre-class work for CS110
Time taken: 90
Total time: 560
----------------------------------------
CS111A pre-class
Time taken: 45
Total time: 605
----------------------------------------
AH111 Pre-class
Time taken: 100
Total time: 705
```

```
----------------------------------------
Take the CS110 class
Time taken: 90
Total time: 795
----------------------------------------
CS111A class
Time taken: 90
Total time: 885
----------------------------------------
Hang out with friends
Time taken: 120
Total time: 1005
----------------------------------------
Grab a dinner with roommates
Time taken: 35
Total time: 1040
----------------------------------------
Meet my civic partner
Time taken: 120
Total time: 1160
----------------------------------------
Talk to my parents
Time taken: 15
Total time: 1175
----------------------------------------


/Users/kubawarmuz/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:70: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

There are multiple problems with the algorithm I used for the LBA assignment. They will be named and discussed below: - Strong dependency of the algorithm on the input: Between the lines 169 and 207, I had a part of the algorithm that used the names from the input to create heaps. When the input changed, the function was useless and the part of the code had to be updated accordingly to the input. In my next iteration of the scheduler, I will avoid such dependency on the input, which will make the algorithm more robust. - No time-constraints: This algorithm was primarily designed to schedule the long-term activities. My purpose was to schedule the tasks I wanted to accomplish in the 4 months period in Seoul. Because of that, the algorithm assumed unlimited time and only ordered the tasks based on their priority. However, such implementation will not yield correct results if we wanted to use it as a daily tasks scheduler. This flaws of the LBA's algorithm can be reflected by the amount of time the scheduler advices to use in a day. In case of this input, it is 1175 minutes = 19.5 hours. This is an unlikely schedule (and definitely not a favorable and sustainable to live by in a long-term one) since we have only 5.5 hours of sleep left. In our updated scheduler, we will want to use the time constraint to prevent from such a thing happening and automatically leave some time for rest. We can therefore imagine a potential

solution. One of them could be to constrain each of the activities to be less than 16 hours. Also, we will need to ensure that there are no activities that are both non-overlapping, and happen towards the end of our 16hr period, since that would result in the algorithm crossing this threshold. - Poorly organized schedule: In this scheduler, I have such nonsense output pieces, as reflecting on the day as only a third task within a day (meaning that after 2.5 hours after waking up we would need to reflect on the day given we followed the scheduler). Also, the algorithm advises to meet the Civic Partner in a late night - in 18th hour of the day. In the updated scheduler, I will try to avoid such absurd schedules. It will mainly be done by implementing the start-times that were used in the first part of this assignment - if the algorithm decides not to include a specific task that needs to happen at the beginning of the day, it will simply not be included at all in the schedule, rather than being pushed at the end. - Huge space inefficiency - In this implementation, I was storing my data in a DataFrame. Even though this data structure can be very useful for specific purposes (manipulating the data, analyzing it, etc.), in our case we don't use its functionality, while suffering from its disadvantages. Changing this datastructure to a different one will yield a better space efficiency. In my case, even a terribly poor use of memory will not constitute a problem because the data is relatively small. However, if we wanted to implement our algorithm on a larger scale (e.g. because we want to develop a scheduler app and put it on the market for public use), we would be highly concerned with how much memory we need to serve our customers.

4 3). Improved Scheduler

Improved Scheduler Info

- I assume that all of the tasks within an activity have to be done. If the scheduler decides to execute an activity, all of its tasks will be executed. For that reason, it does not matter what dependencies the tasks have on one another - the ones that have a dependency will be executed before the ones without dependencies. The more dependencies the task has, the lower in the priority queue it ought to be
- I will combine two solutions from the previous part. I will use the algorithm from the first question to decide which activities will be executed and the priority queue to schedule the order in which the tasks within this activity will be executed.
- To avoid making irrelevant computations, I will firstly get the schedule of all the activities that our algorithm decides to execute, and then compute the priority queue for tasks within these activites. By doing it in this order, I avoid situation in which I calculate the priority queue for the activity that ends up not being executed.

In [334]:
```
#importing necessary heapq library
import heapq

class Activity(object):
    def __init__(self, name, start_time, profit, tasks):
        self.name = name
        self.start_time = start_time
        #For the purpose of scheduling activites, we will need the end time of
        #the activity, which basically is the start time + sum of durations
        #of all the tasks within the activity
        #For now, we will initialize this to start time and later we will
```

```
        #use a function to calculate these end times for each activity
        self.end_time = start_time
        self.profit = profit
        self.tasks = tasks

class Task(object):
    def __init__(self, ID, description, duration, dependencies,
                 multitasking, priority):
        self.ID = ID
        self.description = description
        self.duration = duration
        self.multitasking = multitasking
        self.dependencies = dependencies
        self.priority = priority
        self.status = 'Not Yet Started'


def get_end_time(activity):
    duration_sum=0
    for i in range(len(activity.tasks)):
        if activity.tasks[i].status=='Not Yet Started':
            duration_sum+=activity.tasks[i].duration
    activity.end_time+=duration_sum



def schedule_activities(list_of_activities):
    #loop through all the activities
    for x in range(len(list_of_activities)):
        duration_sum=0
        #loop through all the tasks within the activity
        activity_tasks=list_of_activities[x].tasks
        for i in range(len(activity_tasks)):
            #if the task has any dependencies, its priority will be increased
            #it is because python's heapq library supports min heaps
            #and I want my first element in the heap to be the task without
            #any dependencies
            activity_tasks[i].priority=len(activity_tasks[i].dependencies)*5
            if activity_tasks[i].multitasking==True:

                for n in range(i+1, len(activity_tasks)):
                    if activity_tasks[n].multitasking==True:
                        #setting up the new ID - it will be the ID of the first
                        #of the tasks
                        new_ID=activity_tasks[i].ID
                        #setting up the name for the new multitasked task
                        new_description=activity_tasks[i].description+(' multitasked
                        #We have 3 options : the task i is longer than task n,
```

```
                        #task n is longer than task i, or the tasks' durations
                        #are equal. In either case, the duration time of having
                        #these executed simultaneously will not exceed the
                        #maximum duration from among these 2
                        new_duration=max(activity_tasks[i].duration,
                                         activity_tasks[n].duration)
                        #setting new dependencies to be the sum of 2 tasks'
                        #dependencies
                        new_dependencies=[]
                        new_dependencies.append(activity_tasks[i].dependencies)
                        new_dependencies.append(activity_tasks[n].dependencies)
                        #setting new priority
                        new_priority=len(new_dependencies)*3
                        #We mark the used tasks so that we can recognize them
                        #later on. We also set our multitasking attribute to
                        #false since we don't want to
                        #merge more than two tasks into one multitask
                        activity_tasks[n].status='Delete'
                        activity_tasks[n].multitasking=False
                        activity_tasks[i].status='Delete'
                        activity_tasks[i].multitasking=False
                        #Creating the task instance
                        activity_tasks.append(Task(ID=new_ID,
                                                   description=new_description,
                                                   duration=new_duration,
                                                   dependencies=new_dependencies,
                                                   multitasking=False,
                                                   priority=new_priority))

    #calling the function to obtain the end times of all the activities
    get_end_time(list_of_activities[x])
#after resolving for all the cases with multitasking, we choose the optimal
#activities, create the priority queues for tasks within them, and print the
#output
schedule=schedule_weighted_activities(list_of_activities)
for i in range(len(schedule)):
    #changing the font attributes so that the output looks more visually
    #appealing
    print('\033[1m'+'Execute the activity: '+'\033[0m')
    print('\033[91m'+schedule[i].name+'\033[0m')
    print('\033[1m'+'Tasks to execute: '+'\033[0m')
    #using python's heapq library to create the priority queue
    tasks_heap=[]
    heapq.heapify(tasks_heap)
    tasks_list=schedule[i].tasks
    #running a loop that adds all the tasks within an activity to the
    #priority queue
    for x in range(len(tasks_list)):
```

```
                    if tasks_list[x].status=='Not Yet Started':
                        heapq.heappush(tasks_heap, (tasks_list[x].priority,
                                                    tasks_list[x].description))

                #printing the tasks
                for h in range(len(tasks_heap)):
                    print(heapq.heappop(tasks_heap)[1])
                print('----------------------')
                print('')
```

In [335]:
```
#creating the input for the function
morning_routine_tasks = [Task(1,'Driniking a glass of water', 1,
                              dependencies=[], multitasking=False, priority=0),
                         Task(2,'Meditate', 15, dependencies=[],
                              multitasking=False, priority=0),
                         Task(3,'Get some coffee', 3, dependencies=[],
                              multitasking=False, priority=0)]
Morning_Routine = Activity('Morning Routine', 0, 5, morning_routine_tasks)

wash_tasks = [Task(4,'Taking a shower', 15, dependencies=[],
                   multitasking=False, priority=0),
              Task(5,'Brushing my teeth', 3, dependencies=[4],
                   multitasking=True, priority=0),
              Task(12,'Listening to a podcast', 15, dependencies=[],
                   multitasking=True, priority=0)]
Wash = Activity('Washing up at the beginning of the day', 20, 20, wash_tasks)

breakfast_tasks = [Task(14,'Prepare the meal', 15, dependencies=[],
                        multitasking=False, priority=0),
                   Task(16,'Eat', 15, dependencies=[14],
                        multitasking=True, priority=0),
                   Task(17,'Watch some Youtube', 15, dependencies=[14],
                        multitasking=True, priority=0)]
Breakfast = Activity('Starting the day with a healthy meal', 120, 10, breakfast_tasks

CS110_tasks = [Task(6,'CS110 readings', 90, dependencies=[],
                    multitasking=False, priority=0),
               Task(7,'CS110 pre-class work', 120, dependencies=[6],
                    multitasking=True, priority=0),
               Task(8,'CS110 class', 90, dependencies=[6,7],
                    multitasking=False, priority=0),
               Task(13,'Listening to music to get more focused', 100,
                    dependencies=[], multitasking=True, priority=0)]
CS110 = Activity('Preparing for and going to the class of CS110', 150, 50, CS110_tasl

CS111A_tasks = [Task(9,'CS111A readings', 60, dependencies=[],
                     multitasking=False, priority=0),
                Task(10,'CS111A pre-class work', 20, dependencies=[9],
```

```
                    multitasking=False, priority=0),
                Task(11,'CS111A class', 90, dependencies=[9,10],
                    multitasking=False, priority=0)]
CS111A = Activity('Preparing for and going to the class of CS111A', 450, 45, CS111A_t

work_tasks = [Task(18,'Work on the model for the internship', 60,
                   dependencies=[], multitasking=False, priority=0),
              Task(19,'Work-study', 45, dependencies=[],
                   multitasking=True, priority=0),
              Task(20,'Chat to a co-worker', 10, dependencies=[19],
                   multitasking=True, priority=0)]
Work = Activity('Get some work for internship and work-study done', 800, 30, work_tas

gym_tasks = [Task(21,'Warm up', 5, dependencies=[], multitasking=False, priority=0),
             Task(22,'Bench press', 15, dependencies=[21],
                  multitasking=False, priority=0),
             Task(23,'Shoulder press', 10, dependencies=[21, 22],
                  multitasking=False, priority=0),
             Task(24,'Incline dumbell press', 10, dependencies=[21,22,23],
                  multitasking=False, priority=0),
             Task(25,'Lateral raises', 10, dependencies=[21,22,23,24],
                  multitasking=False, priority=0),
             Task(26,'Triceps extensions', 10, dependencies=[21,22,23,24,25],
                  multitasking=False, priority=0)]
Gym = Activity('Gym session: Chest day', 60,20, gym_tasks)

recovery_tasks = [Task(27,'Stretch', 10, dependencies=[],
                       multitasking=False, priority=0),
             Task(28,'Massage on a foam roller', 7, dependencies=[],
                  multitasking=False, priority=0),
             Task(29,'Breathing exercise', 6, dependencies=[],
                  multitasking=False, priority=0)]
Recovery = Activity('Post-workout recovery', 120,5, recovery_tasks)

explore_tasks = [Task(30,'Go to a location from your bucket list', 45,
                      dependencies=[], multitasking=False, priority=0),
             Task(31,'Taste local food', 30, dependencies=[30],
                  multitasking=True, priority=0),
             Task(32,'Meet a local', 15, dependencies=[],
                  multitasking=True, priority=0),
             Task(33,'Take some pics for the Gram', 15, dependencies=[30],
                  multitasking=False, priority=0)]
Explore = Activity('Explore the rotation city', 500,10, explore_tasks)

dinner_tasks = [Task(34,'Decide on the food', 15, dependencies=[],
                    multitasking=False, priority=0),
             Task(35,'Go to the restaurant', 30, dependencies=[34],
                  multitasking=True, priority=0),
```

```
                    Task(36,'Enjoy the meal!', 30, dependencies=[34, 35],
                         multitasking=False, priority=0),
                    Task(37,'Catch up with a friend', 30, dependencies=[],
                         multitasking=True, priority=0)]
        Dinner = Activity('Get a dinner with a friend', 900,10, dinner_tasks)

        entertainment_tasks = [Task(38,'Watch an episode of The Office', 35,
                                    dependencies=[], multitasking=False, priority=0),
                    Task(39,'Play a video game', 30, dependencies=[],
                         multitasking=False, priority=0),
                    Task(40,'Play table tennis with a friend', 20,
                         dependencies=[], multitasking=False, priority=0)]
        Entertainment = Activity('Unwind and have some fun', 850,10, entertainment_tasks)

        #turning the activities into a list
        list_of_activities=[Morning_Routine, Wash, CS110, CS111A, Work, Breakfast,
                            Gym, Recovery, Explore, Dinner,Entertainment]


        #calling the function to get the results
        schedule_activities(list_of_activities)
```

```
Execute the activity:
Morning Routine
Tasks to execute:
Driniking a glass of water
Get some coffee
Meditate
---------------------

Execute the activity:
Washing up at the beginning of the day
Tasks to execute:
Taking a shower
Brushing my teeth multitasked with Listening to a podcast
---------------------

Execute the activity:
Gym session: Chest day
Tasks to execute:
Warm up
Bench press
Shoulder press
Incline dumbell press
Lateral raises
Triceps extensions
---------------------
```

```
Execute the activity:
Starting the day with a healthy meal
Tasks to execute:
Prepare the meal
Eat multitasked with Watch some Youtube
---------------------

Execute the activity:
Preparing for and going to the class of CS110
Tasks to execute:
CS110 readings
CS110 pre-class work multitasked with Listening to music to get more focused
CS110 class
---------------------

Execute the activity:
Preparing for and going to the class of CS111A
Tasks to execute:
CS111A readings
CS111A pre-class work
CS111A class
---------------------

Execute the activity:
Get some work for internship and work-study done
Tasks to execute:
Work on the model for the internship
Work-study multitasked with Chat to a co-worker
---------------------
```

4.1 Improved scheduler complexity analysis

As discussed in the first question, the time complexity of the activities' scheduler is $O(nlog(n))$. To get the time complexity of the Improved Scheduler, we need to analyze the tasks that the *scheduleactivities* function executes before calling the main activites scheduler function. - First, we are resolving the tasks that can be multitasked. In this part, we run three nested loops, suggesting the $O(n^3)$ time complexity. However, in reality, one of the loops runs through activities, while the other two run through the tasks. Hence, if we consider our n to be the number of tasks across all the activities, the time complexity of this part of the algorithm is $O(n^2)$. - Next, the *getendtime* function runs through all the tasks within each activity, resulting in the linear $O(n)$. - Next, we call the *scheduleweightedactivities* function, which runs in $O(nlog(n))$. - Next, we run a nested loop with the heapifying inside of one of the loops. However, similarly to what was happening when we had a triple nested loop, one of the loops runs through activities, while the other over tasks, resulting in $O(n)$. Within the loop running through activities, we create a heap for all the tasks of an activity by pushing each of the tasks to the heap. This operation has an average time complexity of $O(1)$, and the worst-case of $O(log(n))$. Hence, our time complexity of creating the

heap is $O(n) * O(log(n)) = O(nlog(n))$. - Finally, we run a loop through all the tasks within the selected activity and heappop the first task in a priority queue, which takes $O(n)$ for the loop multiplied by $O(log(n))$ from the heappop function, resulting in the final $O(nlog(n))$. As we can see, the biggest time complexity is taken by the part of function that resolves the multitasking. Overall, the algorithm has $O(n^2)$ time complexity. From that, we can infer that it is not the best possible algorithm and some improvements in the way the multitasks are handled could reduce the complexity to logarithmic. However, we can still se the improvement from the function used in the LBA assignment, which was running in $O(n^2log(n))$ (because double nested loop with heap opertions within the inner one).

When it comes to the space complexity, the use of heap data structure for priority queue helps us save the space, since heaps operate in $O(n)$ space complexity (are in-place). Also, storing the tasks within activities with the use of a class gives us an improvement from the LBA assignment, where the pandas DataFrames were used.

4.1.1 Discussion of the Improved Scheduler

Firstly, it is worth noting that the Improved Scheduler is more advanced than the Activity Scheduler from the first part of this assignment since it incorporates not only the activities but also the tasks within them. The lack of time constraint that was present in the LBA scheduler is solved by both proper input format, and the fact that the Improved Scheduler 'drops' the tasks that are overlapping rather than just scheduling them one by one. There are, however, a couple of flaws of the new algorithm. One of them is that the multitasking is resolved in a way that any of the multitasking tasks can not be executed over multiple other multitasking tasks. In other words, the multitasking can only happen between two tasks and the longer one can't be divided and spread across a couple of tasks with multitasking enabled. That yields to some time inefficiency in the scheduler. If we had an optimal solution, it could be the case that the activities scheduler would find the time to execute at least one more activity within a day. Also, there are a couple of multitasking tasks that could've been executed across multiple other activities. An example of such could be listening to a podcast - I could start listening to it at the gym, continue during the shower, and finish while preparing a breakfast. Incorporating that in the algorithm could also save us some time and make the scheduler more optimal. Also, it might be the case, that a poorly designed input could yield an incorrect priority queue of the tasks. It could happen if the dependencies and priorities were assigned such as the tasks that have a certain dependency are executed before the tasks that they are dependent on. It is because the algorithm does not create a network of dependencies, which would make it 'smart' in a way that it knows which tasks have to be executed before which other ones. Instead, currently, this problem is resolved by assigning certain priorities to the tasks based on their dependencies.

4.2 HCs

#optimization: In this assignment, we wanted to optimize the profit derived from performing a certain number of tasks. We used dynamic programming to achieve the globally optimal set of tasks that yields the biggest profit possible

#levelsofanalysis: In the questions 1 and 3, we firstly analyzed the subproblems and found all local optima of our problem. From that, we could derive what is our global maximum for profit. In other words, using the Dynamic Programming, we achieved the globally optimal solution by analyzing smaller subsets of the problem.

#algorithms: Using python coding, we were able to create an algorithm that, given the set of activities with specified start time, duration, and profit, outputs the optimal set of tasks that maximizes the profit. Dynamic Programming approach was used to achieve a better time complexity of the algorithm.

4.3 LOs

#DynamicProgramming: In the question 1, I have defined why the activity scheduler reflects the optimal substructure and why there exist overlapping subproblems that we can avoid solving multiple times. Later in this question, I have developed the Python 3 code that uses this dynamic programming approach and applied it in the third question as well.

#GreedyAlgorithms: In the question 1, I have explained why choosing the solution with maximum profit at each subproblem yields to arriving at a globally optimal solution. I have implemented that in my Python 3 code and used it throughout questions 1 and 3.

#ComputationalSolutions: I have broken the question 3 down into tractable subproblems (handling multitasking, prioritizing the tasks, finding optimal subset of activities, printing the output in a concise format), and used Python programming to solve these subproblems.

#ComputationalCritique: Throughout the assignment, I was discussing the approaches I am taking to solving the tasks, its advantages and disadvantages. In question 3, I worked on improving the code from parts 1 and 2 based on my #ComputationalCritique