Kuwait University

College Engineering and Petroleum
Computer Engineering Department

CpE 0612300 : DES.&ANAL.OF ALGORITHMS

Semester: Spring 2021/2022

Section No. 03A/ATTEND

Project Algorithm

Student Name: < Waheed Dheef>

Student Id: < 2191112333>

Student Name: <Yassine Nader Serrid>

Student Id: <2181156439>

Instructor Name: <Prof. Anastasios Dimitriou>

TA Name: <Eng. Maryam Aljame>

Date: <30/5/2022>

Contents

List of Figures:

Table Of Terminology And Definitions:

DP(Dynamic Programming)	An algorithmic strategy for solving an optimization issue by breaking it down into smaller subproblems and using the fact that the best solution to the overall problem is determined by the best solution to its subproblems.
1-D Arrays	The most basic type of Array, in which the items are stored linearly and may be accessed individually by supplying the index value of each element in the array.
Spaghetti Code	Spaghetti code is a pejorative term for source code that is unorganized and difficult to maintain.
Instance	A specific realization of any object

1. Introduction:

This project aims to design an efficient Dynamic programming (DP) Algorithm and come up with a recurrence that could solve the problem and prove its correctness for the required specifications of the algorithmic problem and implement it using python programming language, in advance we should take into consideration both of space requirements for the implemented program and time complexity and analyze the running time of the algorithm in the worst case scenarios both of these considerations helps us to design an algorithm that works rapidly while also conserving memory, in the process of implementing the program we are going to use a data structure that fits the requirements of the program and consumes less space. We are going to implement the code using class and object (OOP) concept instead of using a simple function implementation to avoid having a spaghetti code, also because a function represents behavior without state; a variable represents state without behavior; and a class lets you combine both and end up with something that has both state and behavior. State is what something knows or what it has, and behavior is what it can do.

1.1 Problem Summary:

In this project we are required to design a system for a new start-up that uses drones to deliver pharmaceutical products, using our knowledge of designing Dynamic Programming (DP) algorithms and python programming language, we are going to figure out an automated way that could help the start-up company to maximize the number of delivered objects which is the optimal solution that could help the start-up company to maximize its profits, considering the limitations applied on the drone. The company have a Schedule S(i) which represents the necessary number of the object that we need to deliver during hour (i), in addition drones also have a schedule R(i) which represents the number of objects that could be delivered after recharging the drone for (i) hours since the last time it was used, and we could say that the actual number of objects that will be delivered in i hours depends on R(i).

-The limitations and conditions applied on the drone:

1- The drone must be recharged regularly after each delivery.

2- The drone's battery in the beginning is discharged.

3- If the drone is operated for its first time in the kth hour, it can deliver up to min(S(k), R(k)) products.

2. Specifications:

2.1 Design:

The program design consists of a class named Drones. The class initializes an array which is called schedule and have a size of N and the other array is recharge and have of size M. We also have, two pointers were initialized pointing at the first empty index of each array. Two additional arrays of size N were initialized that will store the optimal solution, for each hour and the hours when the drone was used to achieve the optimal solution that maximizes the number of objects that will be delivered.

In the drone class, we created three methods. The first method which is name is inserts OPS and the second one is optimal_for_n. The insert method inserts an array from the user input into the schedule array or recharge array. The OPS method recalculates the optimal solution for all existing hours in the schedule array and stores the hours when the drone was used in optimal_for_n that prints the maximum number of objects the drone can deliver at hour n and the hours the drone was used for this.

2.1.1 Formulation Of Problem Using DP Recurrences:

Let P = optimal solution, i = the hour that we are trying to find an optimal solution for it, j = the hours between 0 and i which we already have the optimal solution for it, q = the maximum amount for an hour (i) to find the optimal solution for an hour (i) We set q = 0.

P(i) = max(q, P(j) + min(schedule(i) + recharge(i-j))

To find the optimal solution for any hour (i) we must solve all (j) between 0 and (i).

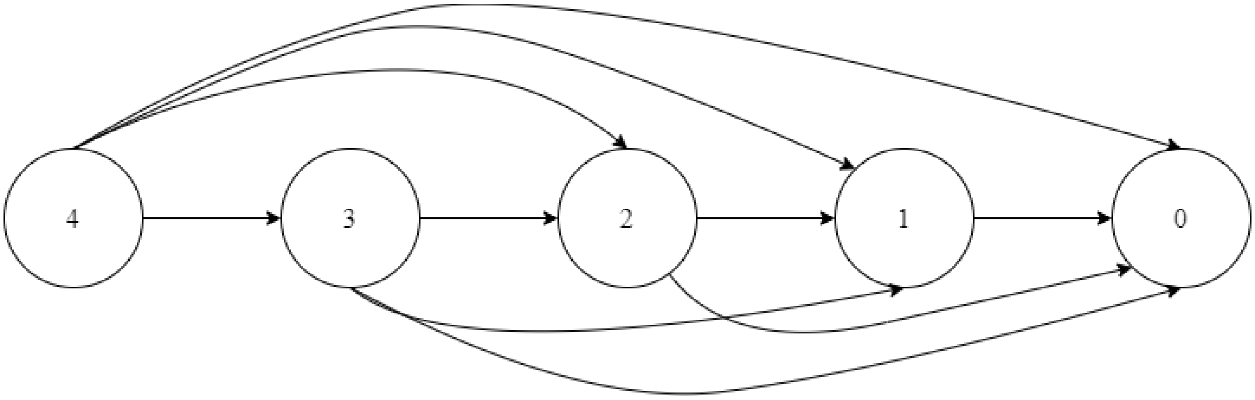


Figure 1. DAG (Directed Acyclic Graph) Of The Design.

2.1.2 Correctness Of Recurrence:

In an hour (i) the drone can only deliver the number of objects stored in array scheduled to allow even if recharge allows more. The drone can also only deliver the amount of recharge allowed even if the schedule allows more. For this, we take a minimum of the two .

To find the optimal solution for (i) we either take the minimum of both schedule [i] and recharge [i] or we send the drone at an hour (j) and let it charge (i-j) hours. We try this for all j in range 0 to (i) to find the maximum of them, which is the optimal solution for an hour (i).

schedule = [0 , 1 , 6 , 7 , 3], recharge = [0 , 1 , 2 ,5 , 6] (remember index = hour)

P = [0 , 1 , 2 , 5 , ?]

P(4) = P(0) + min(schedule[4], recharge(4-0) = 0 + min(3,6) = 0 + 3 = 3

P(4) = P(1) + min(schedule[4], recharge(4-1) = 1 + min(3,5) = 1 + 3 = 4

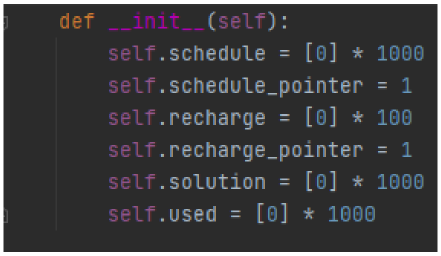
P(4) = P(2) + min(schedule[4], recharge(4-2) = 2 + min(3,2) = 2 + 2 = 4

P(4) = P(3) + min(schedule[4], recharge(4-3) = 5 + min(3,1) = 5 + 1 = 6

We take the max between these solutions which is when P(4) = 6.

2.2 Implementation:

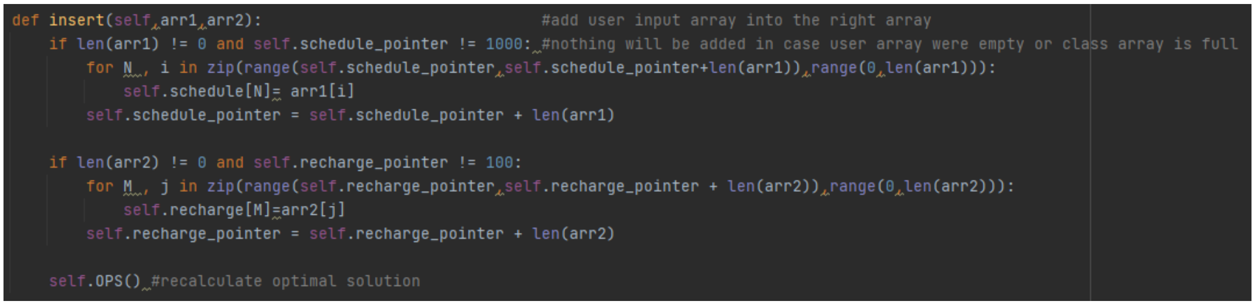
The constructer in the class as shown below in Figure 2 initializes 6 difference instances 4 as an array and 2 as an integer. Instance schedules represent the amount the drone can deliver in a given hour. The index represents the hour. Where index 0 = hour 0, index 1 = hour 1. The index value is the amount that the drone can deliver in that hour. Schedule [2] = 6, which means the drone can deliver 6 objects in the second hour. The schedule_pointer is a pointer pointing right after the last inserted hour. By default, the last inserted hour is 0 with a 0-delivery amount. The pointer will give us huge advantages that will be shown later. recharge and recharge_pointer follow the same ideology as the previous two but for the amount the drone can deliver for M hours of recharging, recharge [2] = 2 means if the drone recharges for 2 hours, it can deliver 2 objects. As for the solution instance, we will save the maximum amount the drone can deliver in each hour. While the used array will tell us what hour the drone was used before hour N to achieve the maximum amount for hour N. The initializing instance process takes constant time, giving it time complexity. O(1) Instance schedule, solution, and used arrays have a size of N while recharge has a size of M giving it a total space complexity of 3N + M or O(N+M).



```
def __init__(self):
    self.schedule = [0] * 1000
    self.schedule_pointer = 1
    self.recharge = [0] * 100
    self.recharge_pointer = 1
    self.solution = [0] * 1000
    self.used = [0] * 1000
```

Figure 2. initializing Instances.

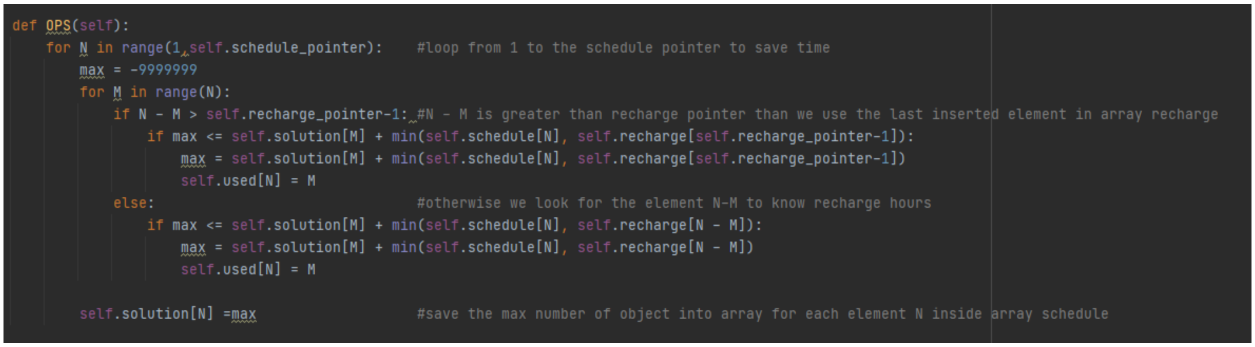
The first method in the class shown below in Figure 3 is called an insert. The method will take 2 arrays from the main input from the user. The array can be as big as N and M. arr1 is the array that represents the scheduled hour values. arr2 represents the recharge hour value. The array values will be entered starting from the pointer position. After entering them, the pointer will be moved by the length of the array. This way, the pointer gives us an advantage if we want to enter more hours in the array instead of starting over the method loop from the pointer to the size of the entered array. The average case should be small as we start looping from the pointer instead of the full array. However, if the user entered a large number for the first time, it could take a long time. This is the worst-case scenario of N + M or O(N+M). python does not make a copy array when you pass it into a function or method making the method does not consume extra space giving it space complexity of O(1). At the end, the method will call the 2nd method OPS.



```
def insert(self,arr1,arr2):                              #add user input array into the right array
    if len(arr1) != 0 and self.schedule_pointer != 1000: #nothing will be added in case user array were empty or class array is full
        for N , i in zip(range(self.schedule_pointer,self.schedule_pointer+len(arr1)),range(0,len(arr1))):
            self.schedule[N]= arr1[i]
        self.schedule_pointer = self.schedule_pointer + len(arr1)

    if len(arr2) != 0 and self.recharge_pointer != 100:
        for M , j in zip(range(self.recharge_pointer,self.recharge_pointer + len(arr2)),range(0,len(arr2))):
            self.recharge[M]=arr2[j]
        self.recharge_pointer = self.recharge_pointer + len(arr2)

    self.OPS() #recalculate optimal solution
```

Figure 3. Insert Method.

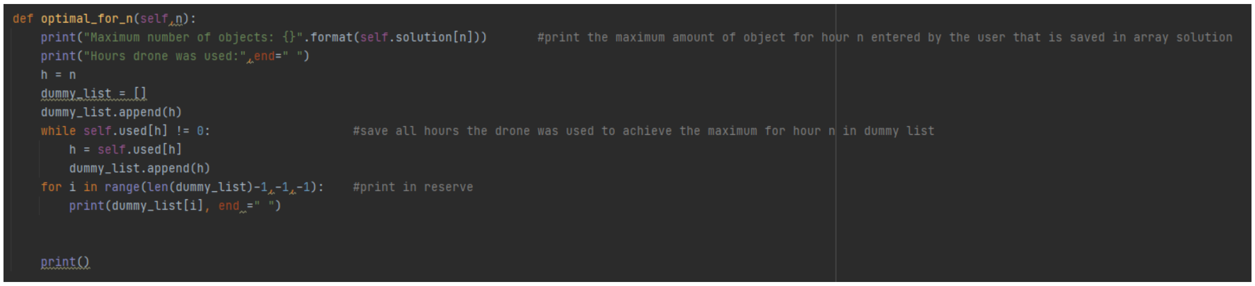
From 0 to N, we then see if the max is less than the optimal solution for M + the minimum of both schedule N and recharge N – M in a case where N-M > recharge pointer – 1 means we will exceed the inserted recharge hour when we call recharge, which may result in either an error or an inaccurate number, so we call the last inserted hour in recharge. If N – M is not > recharge – 1, then we will call recharge [N-M]. Each time the check succeeds, we will save the new max and the M, which represents the hour the drone was used to achieve the current max for N. M for the max will be saved in the array used while max will be saved in array solution both in index N which represent the current hour, we are in ,after both loops finish, we should have all optimal solution for all hours in array schedule. The method does not create any lists of arrays, so it has a space complexity of O(1). Meanwhile, the inner loop is dependent on the outer loop increasing as the outer loop increases for time $\frac{N(N-1)}{2}$, which equals O(N^2).



```
def OPS(self):
    for N in range(1,self.schedule_pointer):    #loop from 1 to the schedule pointer to save time
        max = -9999999
        for M in range(N):
            if N - M > self.recharge_pointer-1: #N - M is greater than recharge pointer than we use the last inserted element in array recharge
                if max <= self.solution[M] + min(self.schedule[N], self.recharge[self.recharge_pointer-1]):
                    max = self.solution[M] + min(self.schedule[N], self.recharge[self.recharge_pointer-1])
                    self.used[N] = M
            else:                           #otherwise we look for the element N-M to know recharge hours
                if max <= self.solution[M] + min(self.schedule[N], self.recharge[N - M]):
                    max = self.solution[M] + min(self.schedule[N], self.recharge[N - M])
                    self.used[N] = M

        self.solution[N] =max               #save the max number of object into array for each element N inside array schedule
```

Figure 4. Method OPS(Optimal Solution)

The last method in the class, as shown in Figure 5, prints the maximum number of objects from the array solution by calling solution [N]. It then saves the hour H the drone was used to achieve the optimal for N by calling used[N]. Then we find the hour when the drone was used to achieve the optimal for H by calling used [H]. And so on. When used[H] = 0, then the drone was only used for hour H to achieve optimal H. We save the used hours in a dummy array and then print them in reverse to match the test result given by the professor. The worst-case scenario (when the array schedule is full and consists of only 1), the dummy array can be as big as N, which will make its time complexity O(n). The running time in that case to find all H hours which is also take O(N).

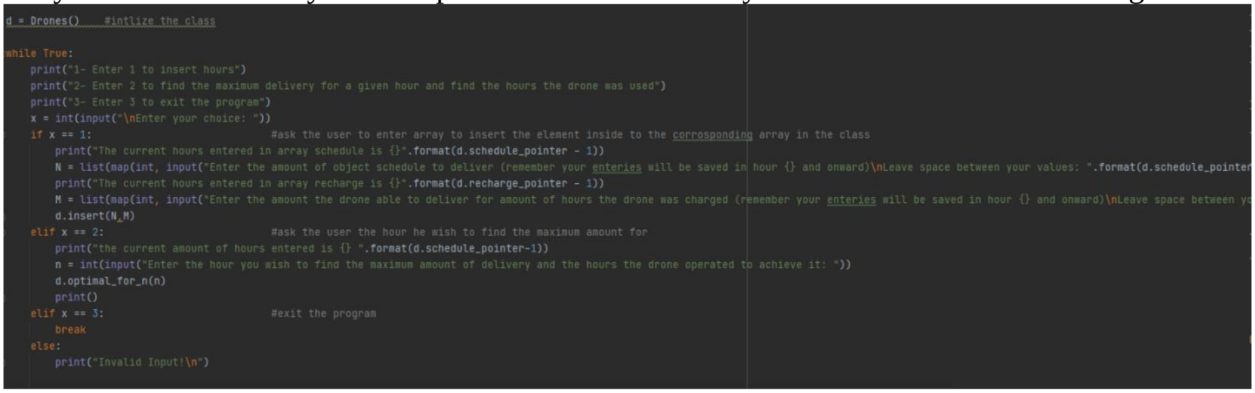


```
def optimal_for_n(self,n):
    print("Maximum number of objects: {}".format(self.solution[n]))      #print the maximum amount of object for hour n entered by the user that is saved in array solution
    print("Hours drone was used:",end=" ")
    h = n
    dummy_list = []
    dummy_list.append(h)
    while self.used[h] != 0:                    #save all hours the drone was used to achieve the maximum for hour n in dummy list
        h = self.used[h]
        dummy_list.append(h)
    for i in range(len(dummy_list)-1,-1,-1):    #print in reserve
        print(dummy_list[i], end =" ")


    print()
```

Figure 5. Printing Method.

The main program, as shown in Figure 6, will initialize the class to variable d and create an object from it. Then a while loop will ask the user to:
1-insert hours into the array schedule and recharge.
2-find the maximum number of objects as well as the hour to achieve it.
3-exit the program.

For 1, it will ask the user to input two arrays, one for schedule and the other one for recharge. 2 will ask the user to enter an hour to find the optimal solution. 3 will break the loop and exit the program. The main function will run if the user pleases, giving it no real running time. The first entry will create an array with no particular size entered by the user it can be small or big.

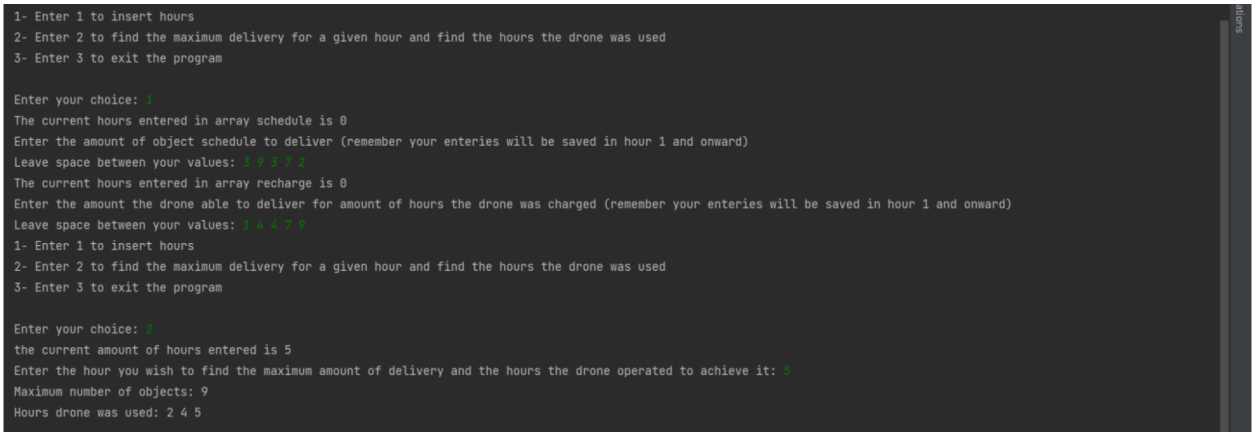


```
d = Drones()    #intlize the class

while True:
    print("1- Enter 1 to insert hours")
    print("2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used")
    print("3- Enter 3 to exit the program")
    x = int(input("\nEnter your choice: "))
    if x == 1:                              #ask the user to enter array to insert the element inside to the corrosponding array in the class
        print("The current hours entered in array schedule is {}".format(d.schedule_pointer - 1))
        N = list(map(int, input("Enter the amount of object schedule to deliver (remember your enteries will be saved in hour {} and onward)\nLeave space between your values: ".format(d.schedule_pointer
        print("The current hours entered in array recharge is {}".format(d.recharge_pointer - 1))
        M = list(map(int, input("Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour {} and onward)\nLeave space between yo
        d.insert(N,M)
    elif x == 2:                            #ask the user the hour he wish to find the maximum amount for
        print("the current amount of hours entered is {} ".format(d.schedule_pointer-1))
        n = int(input("Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: "))
        d.optimal_for_n(n)
        print()
    elif x == 3:                            #exit the program
        break
    else:
        print("Invalid Input!\n")
```

Figure 6. Main Program.

3. Testing and screenshots:

3.1. Test Case 1: N = M:

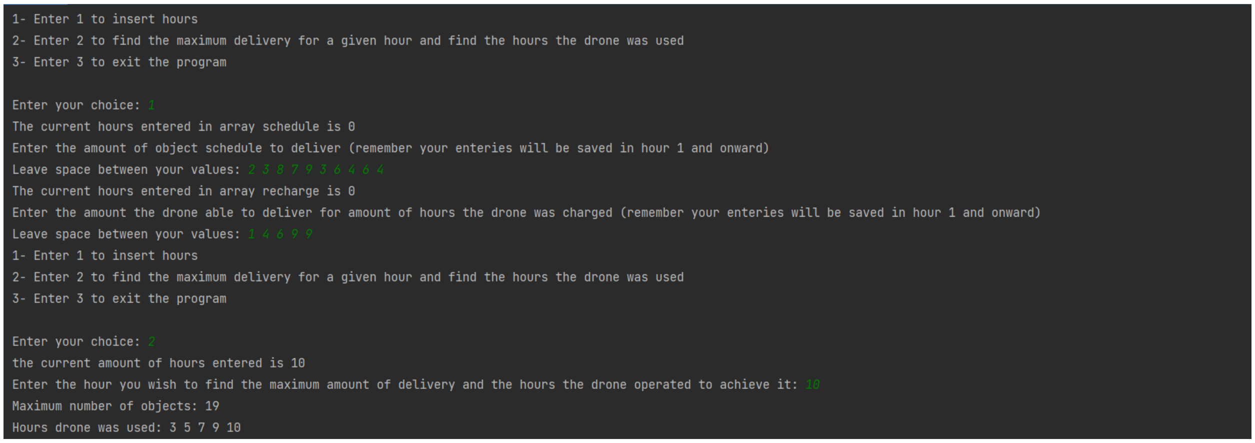


```
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 1
The current hours entered in array schedule is 0
Enter the amount of object schedule to deliver (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 3 9 3 7 2
The current hours entered in array recharge is 0
Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 1 4 4 7 9
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 2
the current amount of hours entered is 5
Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: 5
Maximum number of objects: 9
Hours drone was used: 2 4 5
```

Figure 7. Test Case 1: N = M.

3.2. Test Case 2: N > M:

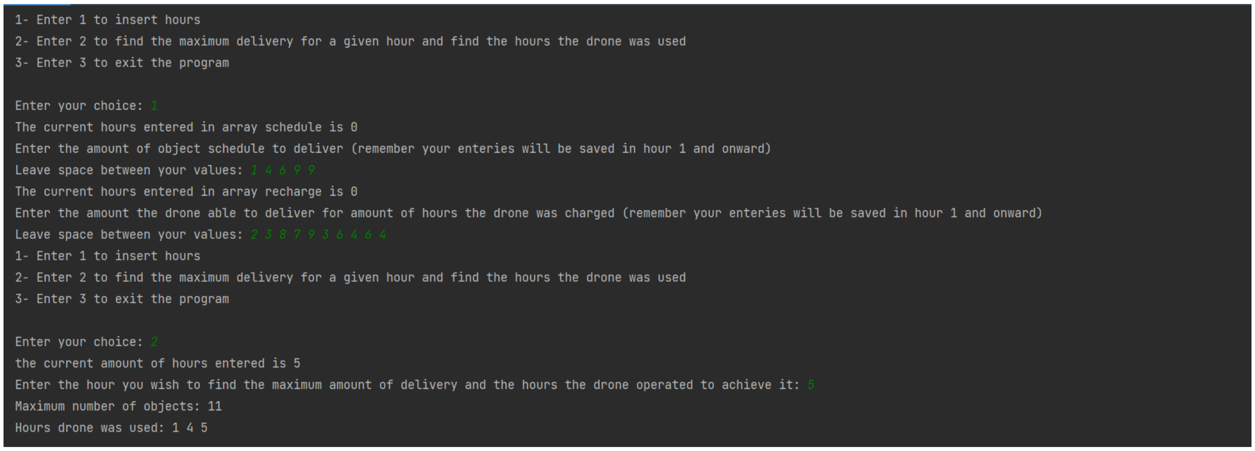
It will take the greater M.



```
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 1
The current hours entered in array schedule is 0
Enter the amount of object schedule to deliver (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 2 3 8 7 9 3 6 4 6 4
The current hours entered in array recharge is 0
Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 1 4 6 9 9
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 2
the current amount of hours entered is 10
Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: 10
Maximum number of objects: 19
Hours drone was used: 3 5 7 9 10
```

Figure 8. Test Case 2: N > M.

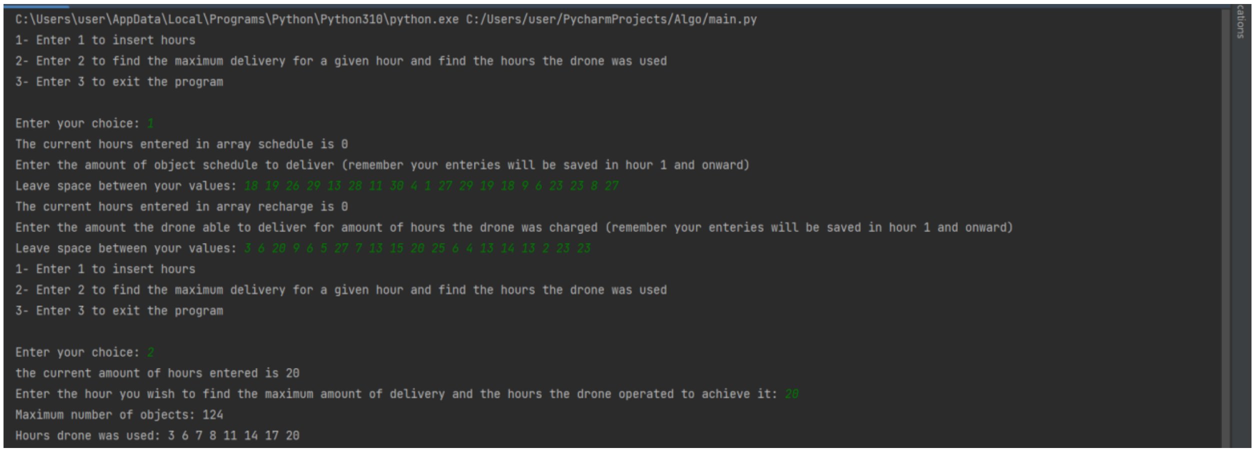
3.3. Test Case 3: N < M:



```
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 1
The current hours entered in array schedule is 0
Enter the amount of object schedule to deliver (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 1 4 6 9 9
The current hours entered in array recharge is 0
Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 2 3 8 7 9 3 6 4 6 4
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 2
the current amount of hours entered is 5
Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: 5
Maximum number of objects: 11
Hours drone was used: 1 4 5
```

Figure 9. Test Case 3: N < M.

3.4. Test Case 4: N & M Are Large:



```
C:\Users\user\AppData\Local\Programs\Python\Python310\python.exe C:/Users/user/PycharmProjects/Algo/main.py
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 1
The current hours entered in array schedule is 0
Enter the amount of object schedule to deliver (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 18 19 26 29 13 28 11 30 4 1 27 29 19 18 9 6 23 23 8 27
The current hours entered in array recharge is 0
Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 3 6 20 9 6 5 27 7 13 15 20 25 6 4 13 14 13 2 23 23
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 2
the current amount of hours entered is 20
Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: 20
Maximum number of objects: 124
Hours drone was used: 3 6 7 8 11 14 17 20
```

Figure 10. Test Case 4: N & M Are Large.

4. Conclusion:

Creating this project enabled team members to gain many skills throughout it different designing phases, it encourages the members to become more responsible and more creative, also we cannot forget the teamwork which was the main feature that help the project to be completed and we appreciate the help of Prof. Anastasios Dimitriou giving us this opportunity to deal with a real-life project. Through the project we learned how to design and analyze Dynamic Programming algorithms and finding an algorithmic solution using this concept and in the implementation phase we noticed the power of this concept in the real life and its applications in, basically this system collects all the team members effort, time, and precision, so we are thankful that the project came up with the best quality.

5. Work Distribution:

Data & Time	26/May/2022 – 4:30 PM
Meeting Held By	**Yassin Nader – Waheed Dheef**
-Figuring out how to get the array as an input from the user. **-Analyzing and reading the problem description.** **-Finding the Dynamic Programming (DP) Recurrence and test its correctness.** **-Decision and Dividing the work.**	

Data & Time	27/May/2022 – 11:30 PM
Meeting Held By	**Yassin Nader – Waheed Dheef**
-Figuring out the best options we could use to implement the program. **-Implementing the program and testing it and analyzing its time and space complexity.**	

Data & Time	28-29-30/May/2022 – 3:30 PM
Meeting Held By	**Yassin Nader – Waheed Dheef**
-Writing the report.	

6. Source Code:

Source Code

```
class Drones():
    def __init__(self):            #intilize all requested array and 
needed array and pointers
        self.schedule = [0] * 1000
        self.schedule_pointer = 1  #will always point at the last 
inserted element + 1
        self.recharge = [0] * 100
        self.recharge_pointer = 1
        self.solution = [0] * 1000
        self.used = [0] * 1000

    def insert(self,arr1,arr2):                              #add user 
input array into the right array
        if len(arr1) != 0 and self.schedule_pointer != 1000: #nothing 
will be added in case user array were empty or class array is full
            for N , i in 
zip(range(self.schedule_pointer,self.schedule_pointer+len(arr1)),range
(0,len(arr1))):
                self.schedule[N]= arr1[i]
            self.schedule_pointer = self.schedule_pointer + len(arr1)

        if len(arr2) != 0 and self.recharge_pointer != 100:
            for M , j in 
zip(range(self.recharge_pointer,self.recharge_pointer + 
len(arr2)),range(0,len(arr2))):
                self.recharge[M]=arr2[j]
            self.recharge_pointer = self.recharge_pointer + len(arr2)

        self.OPS() #recalculate optimal solution

    def OPS(self):
        for N in range(1,self.schedule_pointer):    #loop from 1 to 
the schedule pointer to save time
            max = -9999999
            for M in range(N):
                if N - M > self.recharge_pointer-1: #N - M is greater 
than recharge pointer than we use the last inserted element in array 
recharge
                    if max <= self.solution[M] + min(self.schedule[N], 
self.recharge[self.recharge_pointer 1]):
                        max = self.solution[M] + min(self.schedule[N], 
self.recharge[self.recharge_pointer 1])
                        self.used[N] = M
                else:                               #otherwise we look 
for the element N-M to know recharge hours
```

```
                    if max <= self.solution[M] + min(self.schedule[N], 
self.recharge[N - M]):
                        max = self.solution[M] + min(self.schedule[N], 
self.recharge[N - M])
                        self.used[N] = M

            self.solution[N] =max                   #save the max 
number of objects into array for each element N inside array schedule

    def optimal_for_n(self,n):
        print("Maximum number of objects: 
{}".format(self.solution[n]))       #print the maximum amount of 
object for hour n entered by the user that is saved in array solution
        print("Hours drone was used:",end=" ")
        h = n
        dummy_list = []
        dummy_list.append(h)
        while self.used[h] != 0:                    #save all hours 
the drone was used to achieve the maximum for hour n in dummy list
            h = self.used[h]
            dummy_list.append(h)
        for i in range(len(dummy_list)-1,-1,-1):    #print in reserve
            print(dummy_list[i], end =" ")


        print()

#after we finish the class, we will create the main

d = Drones()    #intlize the class

while True:
    print("1- Enter 1 to insert hours")
    print("2- Enter 2 to find the maximum delivery for a given hour 
and find the hours the drone was used")
    print("3- Enter 3 to exit the program")
    x = int(input("\nEnter your choice: "))
    if x == 1:                             #ask the user to enter 
array to insert the element inside to the corrosponding array in the 
class
        print("The current hours entered in array schedule is 
{}".format(d.schedule_pointer - 1))
        N = list(map(int, input("Enter the amount of object schedule 
to deliver (remember your enteries will be saved in hour {} and 
onward)\nLeave space between your values: 
".format(d.schedule_pointer)).strip().split())) # Use Space to enter 
your values
```

```
        print("The current hours entered in array recharge is
{}".format(d.recharge_pointer - 1))
        M = list(map(int, input("Enter the amount the drone able to
deliver for amount of hours the drone was charged (remember your
enteries will be saved in hour {} and onward)\nLeave space between
your values: ".format(d.recharge_pointer)).strip().split()))
        d.insert(N,M)
    elif x == 2:                                   #ask the user the hour he
wish to find the maximum amount for
        print("the current amount of hours entered is {}
".format(d.schedule_pointer-1))
        n = int(input("Enter the hour you wish to find the maximum
amount of delivery and the hours the drone operated to achieve it: "))
        d.optimal_for_n(n)
        print()
    elif x == 3:                                   #exit the program
        break
    else:
        print("Invalid Input!\n")
```

7. References:

1- What is Dynamic Programming? - Grokking Dynamic Programming Patterns for Coding Interviews. (2022). Retrieved 27 May 2022, from https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2G1pAq0OO0

2- One Dimensional Array in C++. (2021, May 24). Toppr-Guides. https://www.toppr.com/guides/computer-science/programming-in-c-/structured-data-type/one-dimensional-array/#:~:text=A%20One%2DDimensional%20Array%20is,element%20stored%20in%20the%20array

3- Upadhyay, S. (2021). Time and Space complexity in Data Structure | Simplilearn. Retrieved from https://www.simplilearn.com/tutorials/data-structure-tutorial/time-and-space-complexity#what_is_time_complexityA

4- (2022). Retrieved 27 May 2022, from https://www.reddit.com/r/learnpython/comments/1mc8ih/why_should_you_use_classes_instead_of_functions/

5- Spaghetti code - Wikipedia. (2022). Retrieved 27 May 2022, from https://en.wikipedia.org/wiki/Spaghetti_code

6- What is the meaning of instance?. (2022). Retrieved 28 May 2022, from https://www.techtarget.com/whatis/definition/instance#:~:text=What%20is%20an%20instance%3F,realized%20instance%20is%20called%20instantiation.