CSC 2301 INTELLIGENT SYSTEMS

GROUP PROJECT ASSIGNMENT

SECTION 2

Hasan Tanveer Mahmood	1725413
Muhammad Ismail Bin Puat	1722681

LECTURER:
Dr Amir Aatieff Bin Amir Hussin

INTRODUCTION

In this report, it is about the task given to Sentinel to hunt down mutants who have the X-gene specifically and also targeted humans with the potential to have mutant offspring. The sentinel is capable of detecting the difference between humans and mutants using their Mutant Detection Device. Our task is to create a program that can be used in order to choose the mutant that follows the requirement where the Sentinel must capture which mutant that can get the highest value group of mutant and maximize the capacity it can carry within the weight limit.

Our approach in order to solve this problem is by implementing a dynamic approach algorithm in this program. This program will help the Sentinel to carry the maximum number of items and its value. The details about the algorithm will be explained briefly inside the methodology.

BACKGROUND

We want this program to find the optimal value without any problem. This program must find the right amount of weight and the mutant value by searching through all the mutant list. Our problem is that we don't know our result will give us the optimal value. Thus by applying a dynamic programming approach the program will examine the results of the previously solved sub-problems with other sub-problem to achieve the best solution. So an optimum solution is gained.

METHODOLOGY

We used a process to break the big problem into small problems. So, in that case we evaluate our algorithm by inspiring a dynamic approach to solve a given complex problem by breaking it into subproblems. We consider the values and weight with 2d Array. Our main intention was to pick the best value with least weight. This algorithm works well for optimization of certain problems. Dynamic programming algorithm is most likely as divide and conquer method but dynamic programming combines the solution of the sub-problem and solves each of the sub-problem once. The solution is then stored to avoid re-computation of the result when needed later. This simple optimization reduces time complexities from exponential to polynomial.

So, we are following this steps:

1. Creating a matrix with dimension of Max weight + 1 by Total item +1
2. Then generate nested loops and start from the 2nd element of items until total item + 1 and 2nd elements of weight until the max weight

3. After that compare the Capacity with the current weight, if the capacity is greater than the current weight then return the Max value with current = mat[item - 1][remainingCapacity] + Max Value with current.
4. Otherwise , mat[item][capacity] = maxValWithCurr
5. However, Execution time = End time - Start Time.
6. Moreover, Accuracy = mat[totalItem][maxWeight]/expectedOptimal * 100, Where Expected optimal is 1313.

Source Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug  3 23:00:05 2020

@author: Hasan Tanveer Mahmood 1725413
muhammad ismail bin puat 1722681

"""

import time

startTime = time.time()

values =
[23,45,25,92,44,65,38,73,94,85,12,36,16,77,66,45,56,99,72,89,91,38,78,45,60,51,0,87,63,
88,62,51]
weights = [6,2,9,8,4,8,3,9,10,9,3,7,4,1,2,8,0,5,5,0,3,3,8,1,2,1,2,5,6,6,2,9]
totalItem = len(values)
maxWeight = 65
accuracy = 0
excutionTime = 0
expectedOptimal = 1313

mat = [[0 for i in range(maxWeight + 1)] for i in range(totalItem + 1)]


#Main logic
'''for item in range(1,totalItem+1):
    for capacity in range(1,maxWeight+1):
        currentMaxValue = 0
        if capacity >= weights[item-1]:
            currWeight = weights[item-1]
            currentMaxValue = values[item-1] + mat[item-1][capacity-currWeight]
        mat[item][capacity] = max(mat[item-1][capacity] , currentMaxValue)
    print(mat)
'''

for item in range(1,totalItem+1):
```

```
    for capacity in range(1,maxWeight+1):
        maxValWithoutCurr = mat[item - 1][capacity]
        maxValWithCurr = 0
        weightOfCurr = weights[item - 1]
        if capacity >= weightOfCurr:
            maxValWithCurr = values[item - 1]
            remainingCapacity = capacity - weightOfCurr
            maxValWithCurr += mat[item - 1][remainingCapacity]

        if maxValWithoutCurr > maxValWithCurr:
            mat[item][capacity] = maxValWithoutCurr
        else :
            mat[item][capacity] = maxValWithCurr

  endTime =time.time()
  excutionTime = endTime - startTime
  print("\nThe execution Time : %.3f" % excutionTime +" sec")
  accuracy = mat[totalItem][maxWeight]/expectedOptimal * 100
  print("Accuracy : ",accuracy ,"%")
  print("The optimal Value is: ", mat[totalItem][maxWeight])

  '''for x in mat:
      print(x)'''
```
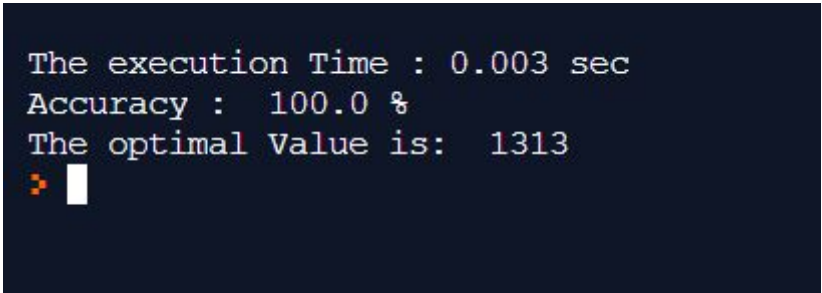
Output

EVALUATION

As from the result of the program we achieved the optimal value 1313. This optimal value is the same with our expected optimal value which is 1313. The run time for this program is faster than the brute force where this program takes only 0.003 sec to execute the program. Thus by implementing a dynamic programming approach in optimization problems it will perform better compare than brute force method.

In this case our time complexity is :

O(total item * Max weight)

Moreover, Space complexity is:

O(total item * Max weight)

Where, we use a 2d array to implement the problem.

CONCLUSION

In conclusion, we can see throughout this assignment, applying the approach of our own which is similar to dynamic approach to reduce the time complexity and improves the performance of optimization where this program only takes milliseconds to achieve the optimal solution. In contrast with the brute force approach.

Video Link:
https://www.youtube.com/watch?v=ZoJzGiyqR-w
Or
https://drive.google.com/file/d/1K-NygEzbvcPq79ND6k_god0H0HvPbwi7/view?usp=sharing