

**CS 430 Project**

**TASKS: Most profitable purchase.**

**BY: SHREERAM VENKATESH**

## **ANALYSIS OF ALGORITHM DESIGN:**

The algorithm implemented in the code is a recursive depth-first search algorithm. It aims at obtaining the optimal cost or minimal cost for completing a purchase using the given or latest promotions.

The algorithm uses dynamic programming as it utilizes recursion to break down the entire problem into small sub-problems. Each sub-problem is solved individually and then their solutions are combined with the solutions of the remaining sub-problems and then we get the optimum solution. It uses a top-down approach where the solution to main problem is calculated by recursively solving the sub-problems and storing their solutions.

At first, we read the input file, to read the purchase intentions or purchase details. We get all the data in the input file as mentioned in the Problem description, i.e., The contents in line 1 i.e., the number of merchandises to be bought is read, the further number of lines depends on the number mentioned in line 1. The contents in line 2, i.e., the identifier of merchandise, amount and price are all also read in first sub-function. The purchase intention denoted as “purchase\_details” are stored as dictionary.

Then in the second subfunction we read the second input file that is promotions file. This file contains the information about the offered promotions. The line 1 contains the number of promotions in effect currently, the line 2 and other lines contains the number of merchandise types, ID's, amounts and price. All these lines are read and the information's are stored. The promotions are stored as a list of dictionaries, where each of them represents the promotions.

The most important function used in the algorithm design is “calculatecost”. It uses the purchase\_details (purchase intentions) and promos (promotions) as input. It returns the minimum payment cost that is required for the purchase. It searches through all possible combinations of promotions that can be applied for purchase details.

The functions at first loops over the purchase\_details and it adds the cost of that item to the “total\_cost” if it has a valid price. Then it loops over the promotions list to check if the current promo can be availed for current purchase\_details by creating a copy of that dictionary and applying promotion to that copy. If the promotion is applicable then the cost of that promotion is added to the total cost. The “calculatecost” function is called recursively with the latest remaining purchase details and promotions list. The total cost in the end is updated as the

minimum of current total\_cost and the sum of promotions and remaining cost. The function will return the optimum cost / minimum cost for purchasing the items. The algorithm checks all possible combinations of promotions by recursively checking until any promotion can be availed that will give us the minimum cost.

For creating a Graphic user interface, the python library “tkinter” is used and the command of “filedialog” is imported from that library. For a GUI we create some main window where we add or create input file selection box and its button to browse files, create promotion file selection box and its button to browse files, output file selection box and its button to browse files, create calculate box and result label which will print the result using the libraries. The example of the testcase output and UI is given below in further pages.

### **RUNNING ON SOME TEST-CASES:**

The given input.txt file will not work as the format of input files does not match the format of Task description.

That is the file does not contain the number of merchandises to buy or ticket price from line 2 onwards. Also, the ““number”” of items cannot be float values.

Therefore, the following Inputs were tested against,

#### **Testcase 1:**

##### **input.txt**

```
2
7 3 2
8 2 5
```

##### **promotion.txt**

```
2
1 7 3 5
```

2 7 1 8 2 10

**output.txt**

14

**Testcase 2:**

**input.txt**

2

9 4 6

1 5 2.5

**promotion.txt**

15

1	1	4	8		
2	1	2	2	1	15
1	4	2	15		
1	7	3	5		
2	5	1	7	2	8
2	3	2	4	1	12
1	9	2	10		
1	22	2	13		
2	13	1	3	2	20
1	15	2	8		
1	10	3	10		
1	11	2	20		
1	22	4	25		
2	7	1	8	2	10
2	1	2	11	1	15

**output.txt**

{1: 4.0, 'price': 18.0, 9: 2.0}

30.5

**Testcase 3:**

### **input.txt**

```
2
13 1 19
3 4 3
```

### **promotion.txt**

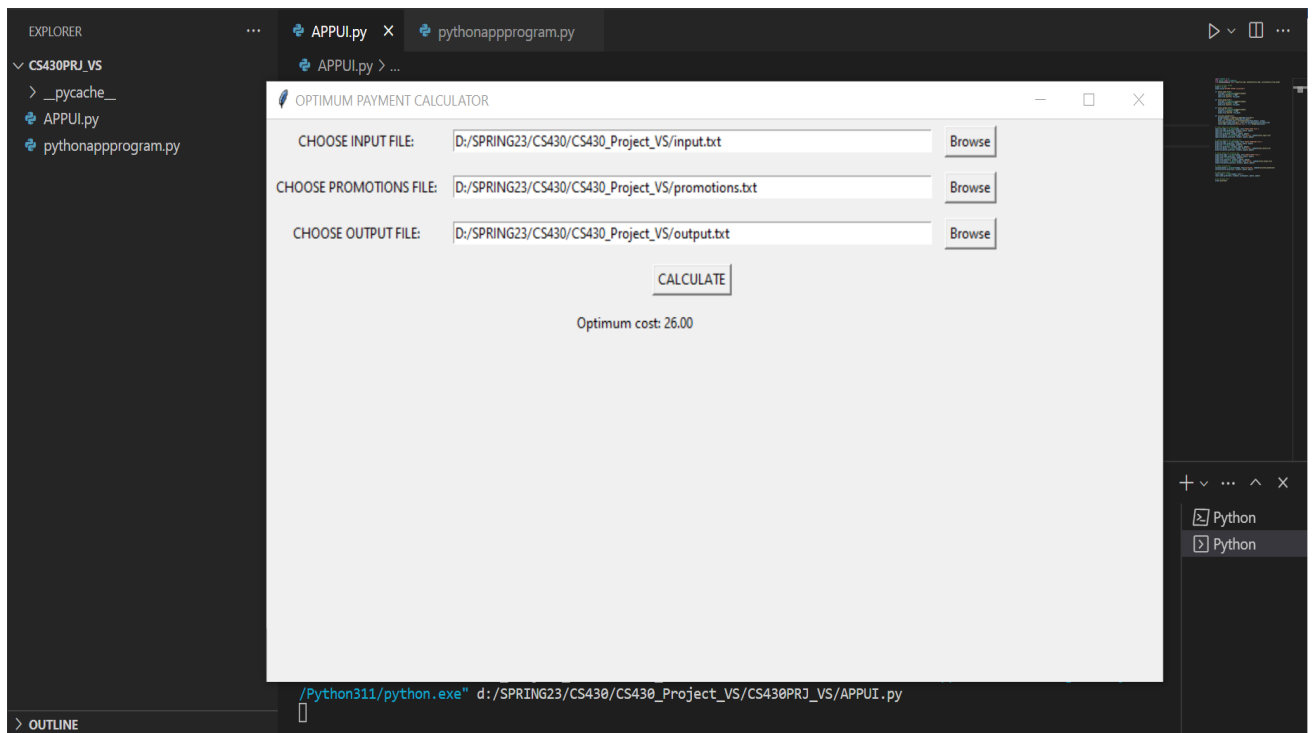
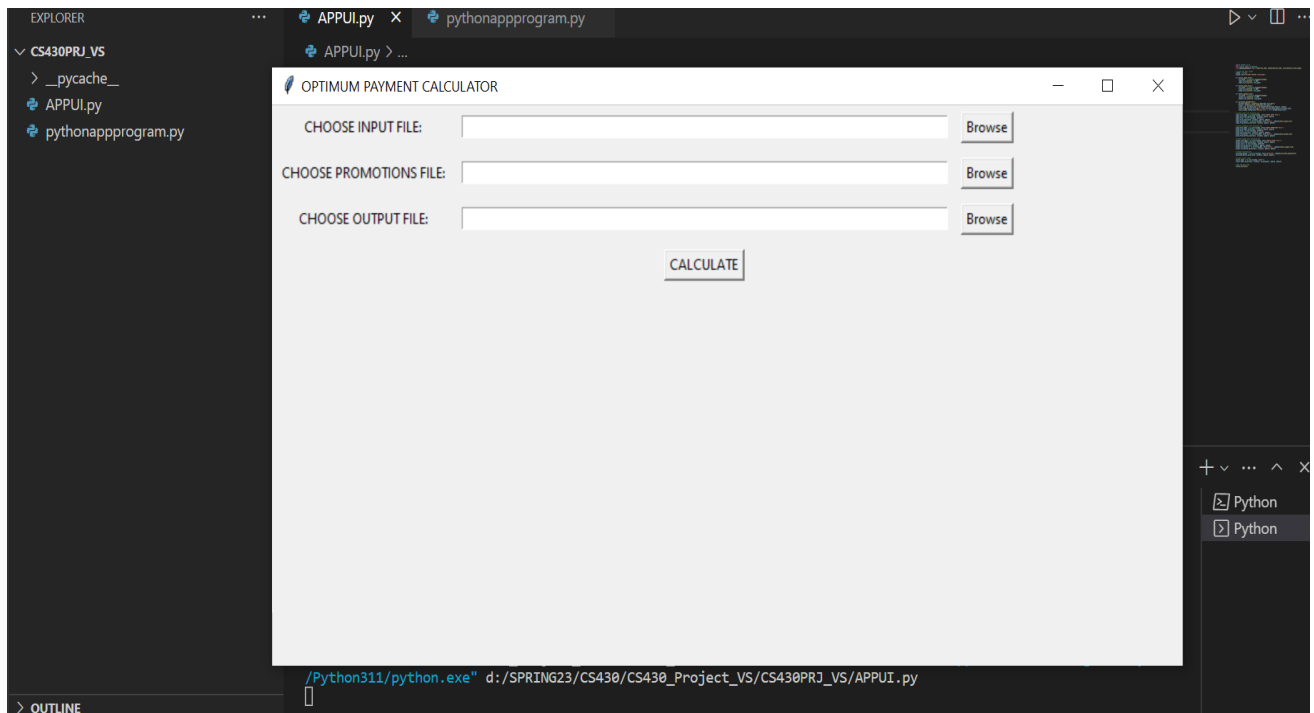
```
15
1    1    4    8
2    1    2    2    1    15
1    4    2    15
1    7    3    5
2    5    1    7    2    8
2    3    2    4    1    12
1    9    2    10
1    22   2    13
2    13   1    3    2    20
1    15   2    8
1    10   3    10
1    11   2    20
1    22   4    25
2    7    1    8    2    10
2    1    2    11   1    15
```

### **output.txt**


```
{3: 2.0, 'price': 20.0, 13: 1.0}
26.0
```

## Output-Example (TestCase-3):

When you run APPUI.py, this is the GUI that pops up. User needs to provide the path for Input.txt, Promotions.txt and output.txt.



The final output is in the output.txt file in the file location given above.

 output - Notepad

File Edit Format View Help

```
{3: 2.0, 13: 1.0, 'price': 20.0}  
26.0
```