Eric Ramsey

CSCI 3102

Module 4 Assignment

Spring 2024

**Solve the following fractional knapsack problem using a greedy approach. The thief can carry a maximum of 33 Lbs. and he has the choices to select from 4 items. Each item i weights wi Lbs. and has a value of vi$.**

* **w1=5Lbs. and v1=100$**
* **w2=20Lbs. and v2=300$**
* **w3=10Lbs. and v3=300$**
* **w4=40Lbs. and v4=400$**

**Justify your choice at each step.**

To implement a greedy algorithm approach to provide a solution to the given knapsack problem, I needed to consider the steps of a greedy algorithm process and how to utilize the given information to obtain an optimal solution to the problem. The three steps of a greedy are to begin the process with sorting the given elements by worth in descending order, start with the element that contains the highest value or worth, and attempt to fill any remaining elements to the list that do not violate the requirements given (weight, maximum carry, etc.). Using the steps of a greedy algorithm I then considered the information given in the fractional knapsack problem to begin structuring the algorithm and structure the Python functions in my programming solution.

|  |  |  |  |
| --- | --- | --- | --- |
| **Item #** | **Weight** | **Value #** | **Value** |
| **w1** | 5 lbs. | **v1** | 100$ |
| **w2** | 20 lbs. | **v2** | 300$ |
| **w3** | 10 lbs. | **v3** | 300$ |
| **w4** | 40 lbs. | **v4** | 400$ |

When reviewing the initial given data or information to plan the implementation of the greedy algorithm I was able to clarify the variables I would need, the structure and ordered steps of the function, and calculation processes that would assist in the final return of the solution. Implementing a greedy algorithm will allow the ability to provide a solution, which in this case is the maximum value the thief can obtain with an item carrying capacity of 33 Lbs., by considering and choosing the action that is currently considered the most optimal regardless of future process effects of this decision.

**greedy.py:**

A screen shot of a computer program

Description automatically generated

For this assignment I chose to use my python programming knowledge and the greedy algorithm implementation recourses found below to use this technique to determine a solution for the given knapsack problem. The first file created for this approach was the ***greedy.py*** file which contains the Python function ***knapsack\_max\_value*** that uses the greedy algorithm technique to effectively return the maximum value the thief can obtain with a maximum item carrying capacity of 33 Lbs. This function accepts three argument parameters. The first parameter being the maximum ***carrying capacity*** (***carry\_capacity***), the second being ***weight*** which contains elements associated with item weight, and ***value*** which contains elements associated to an item’s currency value or worth. During the testing process I noticed that when reviewing possible test scenarios there was not implementation if the lists ***weight*** and ***value*** do not have an equal length. Therefore, the first step of this function is to determine if the lists have equal length, and if not a ValueError is raised to inform the user. The next step of this function is to initialize the variable ***n*** and set its value to the length of the ***weight*** list. This is created to use in the next step of the function’s process, which involves the calculation and determination of the list elements value to weight ratio (***value\_weight\_ratio***) and then sorting the ratio list in descending order. This action or step is key in the determination of the final solution. The value-to-weight ratio is calculated with the implementation of a one-line for loop used to iterate through each element value in the given lists ***weight*** and ***value*** while repopulating the list with ratio values determined by the result of ***value[i]/weight[i]*** and using that result to set the ***value\_weight\_ratio***. The variables ***max\_value*** and ***available\_carry\_capacity*** (used to determine the remaining weight the thief can carry for additional items to be stored) are initialized and utilized in the function’s final step, comparing the knapsack elements, updated the mentioned variables, and returning the final solution to the given problem.

***test\_greedy.py***

A screenshot of a computer program

Description automatically generated

This file contains the unit test functions implemented to determine the accuracy, flaws, and speed of the greedy algorithm implementation when providing a solution to the given knapsack problem. The first test function ***test\_greedy\_1*** contains the original data or information given in the statement of the problem and is used to test the accuracy of the ***knapsack\_max\_value*** function containing greedy algorithm implementation. The three require argument parameters for the function located in ***greedy.py*** are initialized with the corresponding values from the given knapsack problem. The test function continues the process by initializing time variables used to calculate the function’s runtime, while also initializing max\_value variable to store the result of the ***knapsack\_max\_value*** call with the newly set argument parameter variables.

A computer screen with white text

Description automatically generated

The result of the function’s determination on the maximum possible value the thief can carry then displayed in the test terminal, along with all preceding test functions. I felt it was necessary to also test for situations that could affect the programs functionality. Test functions were established for situations involving equal value-to-weight ratios, single element in a list, and situations when the lists do not meet requirements for this algorithm to function.

**knapsack\_max\_value – greedy algorithm solution process**

1. The initial step of this algorithm is to validate the input argument parameters required for the function to begin its process. The variables for this problem are set accordingly:

**carry\_capacity = 33**

**weight = [5, 20, 10, 40]**

**value = [100, 300, 300, 400]**

1. The second step of the greedy algorithm implemented in this Python function is to calculate the value-to-weight ratio of the ***value*** list elements divided by the corresponding element index in ***weight*** list. For the values given in this knapsack problem This is the order the process would calculate this information and finally sort the information (descending value – high to low):

**[(value[0] / weight[0])] = 100 / 5 = 20 $**

**[(value[1] / weight[1])] = 300 / 20 = 15 $**

**[(value[2] / weight[2])] = 300 / 10 = 30 $**

**[(value[3] / weight[3])] = 400 / 40 = 20 $**

***value\_weight\_ratio* = [30, 20, 15, 10]**

1. The third step of the greedy algorithm implemented in this Python function is to initialize the final return variable ***max\_value*** with a starting value of 0, initialize the variable ***available\_carry\_capacity*** to track how much more items can fit the ***carry\_capacity*** requirement, and beginning to compare values to determine the optimal and convenient choice. The first element is considered which the now sorted list makes this the highest element value of the list. This element and its associated value and weight are compared to the ***available­\_carry\_capacity*** value. If the elements weight value is less than the value of ***available\_carry\_capacity,*** then the corresponding value for that element is added to ***max\_value*** while updating ***available\_carry\_capacity*** by subtracted the weight amount of what was added to ***max\_value***.
2. ***available\_carry\_capacity >= weights[0]* || 33 >= 10**
   1. **weight = 10**
   2. **value = 300**
3. ***available\_carry\_capacity >= weights[1]* || 33 >= 5** 
   1. **weight = 10**
   2. **value = 100**
4. ***available\_carry\_capacity >= weights[2]* || 33 >= 20** 
   1. **weight = 20**
   2. **value = 300**
5. Currently the element ***weights[2]*** exceeds the value of ***available\_carry\_capacity*** if corresponding element  ***value***[2], which would calculate the following:
   1. ***available\_carry\_capacity* >= *weights[2]* || 8 < 20**
   2. ***8 / 20 = 0.4 lbs.*** of item ***weight[2]*** *can still be carried by the thief*
   3. **return *max\_value*** **== (*value[0] + value[1] + value[2] \* 0.4) == 670 $***

**Resources**:

* <https://medium.com/free-code-camp/how-i-used-algorithms-to-solve-the-knapsack-problem-for-my-real-life-carry-on-knapsack-5f996b0e6895>
* <https://www.tutorialspoint.com/data_structures_algorithms/fractional_knapsack_problem.htm>
* <https://www.analyticssteps.com/blogs/complete-guide-solve-knapsack-problem-using-greedy-method>