**UMT**

**Advanced Algorithms and Programming**

**Quantifying greedy strategies’ efficiencies by statistical validation procedure.**

**Worked by: Eljon Zagradi Professor: Rene Natowicz**

*I conducted testing with 2000 instances for each algorithm in my program. This large sample size helped me analyze the performance differences between the greedy and dynamic programming approaches comprehensively.*

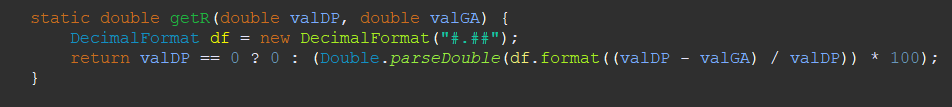
1. MaximumValueBag Algorithm

The code snippet provided below performs tests on the dynamic programming and greedy algorithms for the Maximum Value Bag problem. It measures their execution times, calculates the maximum values obtained, and visualizes the results in a histogram. This allows for a simple and efficient comparison of the two algorithms' performances in solving the problem.A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer code

Description automatically generated with low confidenceThe code snippet provided is a method called randomArray, which generates a random array of integers

The code snippet provided below is a method called getR, which calculates the relative distance between two values, valDP and valGA, which represent the results obtained from the dynamic programming and greedy algorithms.

In the dynamic aproach i used the algorithm provided by you in our sessions:A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer code

Description automatically generated with low confidence

For the greedy approach, I implemented a class called "Item" to facilitate the storage of an item's value and size.

The provided code demonstrates the implementation of the greedy algorithm for computing the maximum value in a bag. Here's a breakdown of how the algorithm works:

* The algorithm creates a list called "items" to store the items, each having a value and a size.
* It iterates over the value and size arrays to create item objects and adds them to the "items" list.
* The items in the list are sorted based on their value ratio in descending order.
* The algorithm initializes variables to keep track of the total value and remaining capacity of the bag.
* It performs a greedy selection of items by iterating over the sorted list and checking if an item can fit in the remaining capacity.
* If an item can fit, its value is added to the total value, and the item's size is subtracted from the remaining capacity.
* Finally, the algorithm returns the total value, representing the maximum value that can be achieved using

A picture containing text, screenshot, software

Description automatically generated

Also attached to this file you will find the MVB.txt file which contains the printouts of the values generated as shown in the main class. Also contains the histogram generated by java code.

1. Maximum Sum of Marks:

A picture containing text, screenshot, font

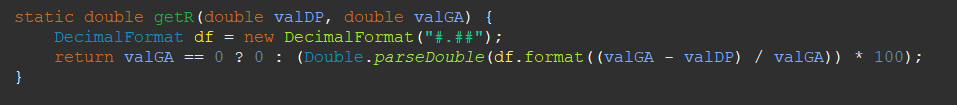
Description automatically generated

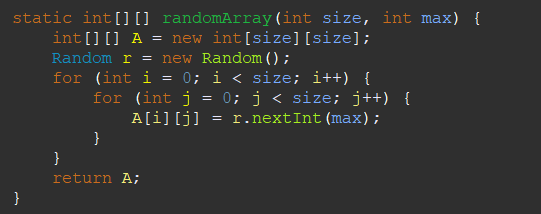
A screen shot of a computer program

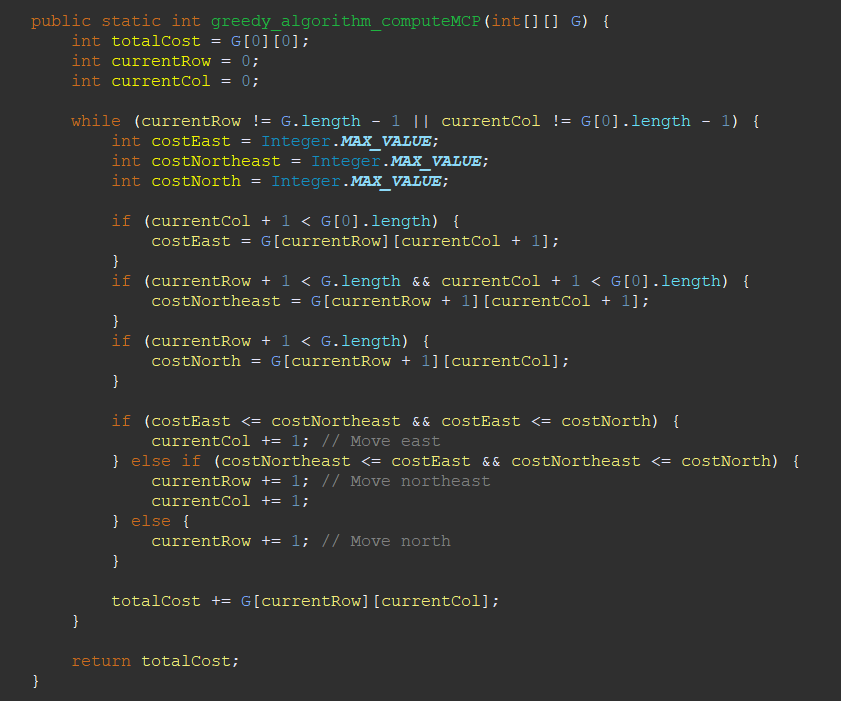
Description automatically generated with low confidence

A picture containing screenshot, text, font

Description automatically generated

1. A picture containing text, screenshot, font

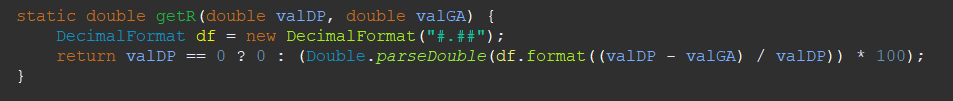
   Description automatically generatedMinimum Cost Path

A picture containing text, screenshot

Description automatically generated

1. A screen shot of a computer

   Description automatically generated with medium confidenceA screen shot of a computer program

   Description automatically generated with low confidenceTwo Bags

A picture containing text, screenshot, font

Description automatically generated

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer code

Description automatically generated with low confidence

A screen shot of a computer code

Description automatically generated with low confidence