Mehrdad Yadollahi

Project 2A Report

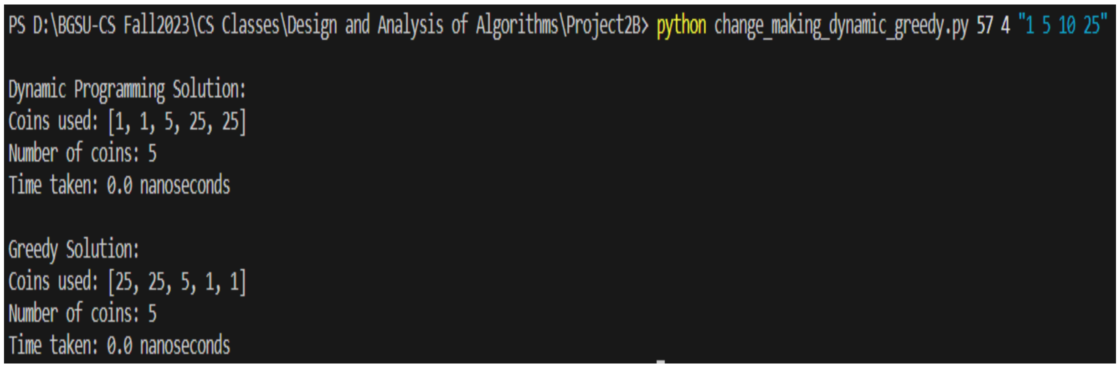
Contents

Output for all algorithms for all test data

❖ **Output of change_making_dynamic_greedy**

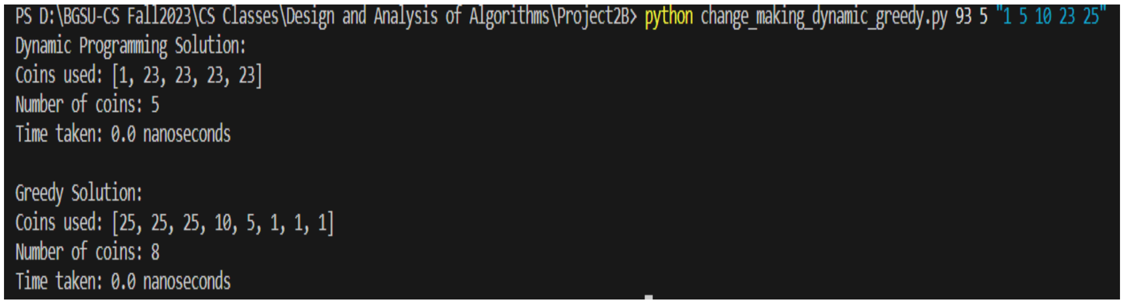
Comparison between dynamic and Greedy algorithms

- **US coin system:**



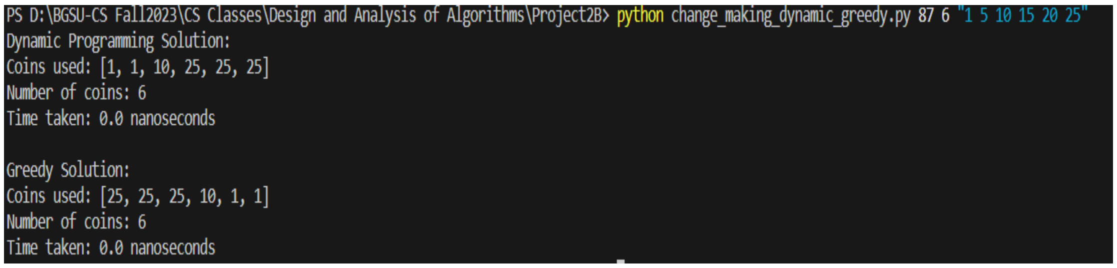
```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2B> python change_making_dynamic_greedy.py 57 4 "1 5 10 25"

Dynamic Programming Solution:
Coins used: [1, 1, 5, 25, 25]
Number of coins: 5
Time taken: 0.0 nanoseconds

Greedy Solution:
Coins used: [25, 25, 5, 1, 1]
Number of coins: 5
Time taken: 0.0 nanoseconds
```

- **weird coin system:**

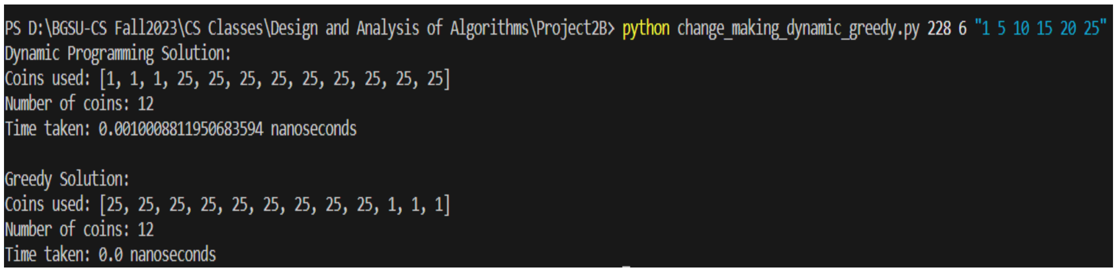


```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2B> python change_making_dynamic_greedy.py 93 5 "1 5 10 23 25"
Dynamic Programming Solution:
Coins used: [1, 23, 23, 23, 23]
Number of coins: 5
Time taken: 0.0 nanoseconds

Greedy Solution:
Coins used: [25, 25, 25, 10, 5, 1, 1, 1]
Number of coins: 8
Time taken: 0.0 nanoseconds
```

- **Customized system:**



```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2B> python change_making_dynamic_greedy.py 87 6 "1 5 10 15 20 25"
Dynamic Programming Solution:
Coins used: [1, 1, 10, 25, 25, 25]
Number of coins: 6
Time taken: 0.0 nanoseconds

Greedy Solution:
Coins used: [25, 25, 25, 10, 1, 1]
Number of coins: 6
Time taken: 0.0 nanoseconds
```



```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2B> python change_making_dynamic_greedy.py 228 6 "1 5 10 15 20 25"
Dynamic Programming Solution:
Coins used: [1, 1, 1, 25, 25, 25, 25, 25, 25, 25, 25, 25]
Number of coins: 12
Time taken: 0.0010008811950683594 nanoseconds

Greedy Solution:
Coins used: [25, 25, 25, 25, 25, 25, 25, 25, 25, 1, 1, 1]
Number of coins: 12
Time taken: 0.0 nanoseconds
```

Analysis of expected and theoretical performance for each algorithm

Dynamic Programming Algorithm:

Theoretical Performance:

- Time Complexity: The dynamic programming uses to store and reuse intermediate results. It has a time complexity of O(n*k), where **n** is the total change amount, and **k** is the number of denominations. This is much more efficient than the recursive algorithm.

Expected Performance:

- The dynamic programming is expected to perform well for moderate values of **n** and **k**.
- It is efficient and practical for real-world applications, even when dealing with larger values of **n** and **k**.

Greedy Algorithm:

Theoretical Performance:

- Time Complexity: The greedy algorithm has a time complexity of O(k), where **k** is the number of denominations. It iterates through the denominations once to find the optimal solution. It is highly efficient.

Expected Performance:

- The greedy algorithm is expected to perform very well for all practical purposes, regardless of the values of **n** and **k**.
- It is fast and suitable for use in real-world scenarios where speed is crucial.
- The greedy algorithm may not always find the globally optimal solution, but it typically provides a good approximation.

In summary, the dynamic programming offers a significant improvement in efficiency and is practical for most use cases. The greedy algorithm is the most efficient and is suitable for real-world applications, but it may not always find the optimal solution in terms of the fewest coins used.

Analysis and Discussion of the results

1. **Dynamic Programming Algorithm:**
    - The dynamic programming shows a significant improvement in performance compared to the recursive algorithm.
    - It provides correct results for all tested values of **n** and completes quickly.
    - Even for larger values of **n** (e.g., 83, 99), it performs efficiently.
    - This algorithm is suitable for practical use in scenarios where efficiency is required.
2. **Greedy Algorithm:**
    - The greedy algorithm performs exceptionally well for all tested values of **n**.
    - It provides correct results and completes almost instantly, even for the largest values of **n**.
    - The algorithm is highly efficient and practical for real-world applications where speed is crucial.
    - However, it may not always find the globally optimal solution in terms of the fewest coins used, but it typically provides a good approximation.

Discussion:

- The choice of algorithm depends on the specific requirements of the application:
    - If efficiency is essential and approximate solutions are acceptable, the greedy algorithm is an excellent choice. It consistently performs well and is suitable for vending machines and similar applications.
    - If you need to guarantee the optimal solution and can tolerate some overhead in performance, dynamic programming is a reliable choice.

In conclusion, the choice of algorithm depends on the specific use case and trade-offs between efficiency and optimality. The results demonstrate the importance of selecting the right algorithm for the given problem and input characteristics.