Mehrdad Yadollahi

Project 2A Report

Contents

Output for all algorithms for all test data

- **Output of change_making_recursive_greedy**

Comparison between Straight forward recursive and Greedy algorithms

- **US coin system:**

```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2A> python change_making_dynamic_greedy.py 1233 4 "1 5 10 2
5"
Dynamic Programming Result:
[1, 1, 1, 5, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 2
5, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25]
Time taken (nanoseconds): 0.0019915103912353516

Greedy Algorithm Result:
[25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 2
5, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 5, 1, 1, 1]
Time taken (nanoseconds): 0.0
```

- **weird coin system:**

```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2A> python change_making_dynamic_greedy.py 83 5 "1 5 10 23
25"
Dynamic Programming Result:
[10, 23, 25, 25]
Time taken (nanoseconds): 0.0

Greedy Algorithm Result:
[25, 25, 25, 5, 1, 1, 1]
Time taken (nanoseconds): 0.0
```

```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2A> python change_making_dynamic_greedy.py 1923 6 "1 5 10 1
5 23 25"
Dynamic Programming Result:
[23, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 2
5, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
 25, 25, 25, 25, 25, 25, 25, 25, 25, 25]
Time taken (nanoseconds): 0.0037674903869628906

Greedy Algorithm Result:
[25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 2
5, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
 25, 25, 25, 25, 25, 25, 25, 25, 25, 23]
Time taken (nanoseconds): 0.0
```

❖ 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\Project2A> python change_making_dynamic_greedy.py 83 4 "1 5 10 25"

Dynamic Programming Result:
[1, 1, 1, 5, 25, 25, 25]
Time taken (nanoseconds): 0.0

Greedy Algorithm Result:
[25, 25, 25, 5, 1, 1, 1]
Time taken (nanoseconds): 0.0010018348693847656
```

- **weird coin system:**

```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2A> python change_making_dynamic_greedy.py 83 5 "1 5 10 23
25"
Dynamic Programming Result:
[10, 23, 25, 25]
Time taken (nanoseconds): 0.0

Greedy Algorithm Result:
[25, 25, 25, 5, 1, 1, 1]
Time taken (nanoseconds): 0.0
```

```
PS D:\BGSU-CS Fall2023\CS Classes\Design and Analysis of Algorithms\Project2A> python change_making_dynamic_greedy.py 1234 5 "1 5 10 1
5 23 25"
Dynamic Programming Result:
[15, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 2
3, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23]
Time taken (nanoseconds): 0.0025963783264160156

Greedy Algorithm Result:
[23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 2
3, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 15]
Time taken (nanoseconds): 0.0
```

Analysis of expected and theoretical performance for each algorithm

Straightforward Recursive Algorithm:
Theoretical Performance:

- Time Complexity: The straightforward recursive algorithm explores all possible combinations of coins. In the worst case, it has an exponential time complexity of O(k.n), where **k** is the number of denominations and **n** is the total change amount. This is highly inefficient for larger values of **n** and **k**.

Expected Performance:

- The recursive algorithm is expected to be very slow for large values of **n** and **k** due to its exponential time complexity.
- It is suitable for educational purposes and small inputs but not for practical use when efficiency is required.

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 recursive algorithm has poor theoretical and expected performance for larger inputs. 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. **Straightforward Recursive Algorithm:**
    - The straightforward recursive algorithm is very slow for larger values of **n** due to its exponential time complexity.
    - For small values of **n** (e.g., 11, 23), it provides correct results but takes a considerable amount of time.
    - For larger values of **n** (e.g., 83, 99), the algorithm becomes impractical and takes an excessively long time to complete.
    - This algorithm is not suitable for real-world applications where efficiency is crucial.
2. **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.
3. **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.
    - The straightforward recursive algorithm should generally be avoided for real-world applications due to its poor performance.

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.