Activity 1.2

José Carlos Martínez Núñez, Tania Sayuri Guizado Hernández

Tecnológico de Monterrey

August 30, 2022

Contents

Algorithm and Implementation

In the `main.cpp` file, there are two main algorithms implemented:

- `greedy_algorithm()`: Receives a list of coins and a value, and returns the combination of coins needed to reach the value, with the minimum number of coins used. The algorithm is greedy, which means that it always takes the biggest coin possible, and then the next biggest coin, and so on.
- `dynamic_programming()`: Receives a list of coins and a value, and returns the combination of coins needed to reach the value, with the minimum number of coins used. The algorithm uses dynamic programming, which means that it uses a table to store the results of previous calculations, and it uses this table to avoid recalculating the same results.

These are their implementations:

```cpp
void greedy_algorithm(vector<int> &coins, int change)
{
    vector<int> coins_used;
    int total_coins = 0;

    for (int coin : coins)
    {
        while (change >= coin)
        {
            change -= coin;
            total_coins++;
            coins_used.push_back(coin);
        }
    }
    cout << "Total coins used: " << total_coins << endl;

    for (int i = 0; i < coins_used.size(); i++)
    {
        cout << coins_used[i] << endl;
    }
    cout << endl;
}
```

```cpp
void dynamic_programming(vector<int> &coins, int change)
{

    vector<int> dp(change + 1, -1);
    vector<int> coins_used(change + 1, -1);

    dp[0] = 0;
    coins_used[0] = 0;
    for (int i = 1; i <= change; i++)
    {
        for (auto coin : coins)
        {

            if (i - coin >= 0)
            {
                if (dp[i - coin] != -1)
                {
                    if (dp[i] == -1 || dp[i] > dp[i - coin] + 1)
                    {
                        dp[i] = dp[i - coin] + 1;
                        coins_used[i] = coin;
                    }
                }
            }
        }
    }
    cout << "Total coins used: " << dp[change] << endl;

    while (change > 0)
    {
        cout << coins_used[change] << endl;
        change -= coins_used[change];
    }
}
```

Big O Notation

- The `greedy_algorithm()` algorithm has a big O notation of $O(n)$, where n is the number of coins. This is the worst case scenario, if all we have is the coin with a value of 1.
- The `dynamic_programming()` algorithm has a big O notation of $O(n * m)$, where n is the number of coins and m is the amount of change needed to give.

Test Cases

- `base_test.txt`: 5 coins and a change of 266. Both algorithms return the correct combination of coins.
- `greedy_gives_completely_wrong_answer.txt`: 4 coins and a change of 10. The greedy algorithm returns the wrong combination of coins. In fact, it doesn't complete the amount of change needed.
- `greedy_gives_suboptimal_answer.txt`: 3 coins and a change of 15. The greedy algorithm returns the suboptimal combination of coins (using a total of 6). The dynamic programming algorithm returns the optimal combination of coins (using a total of 3).
- `stress_test.txt`: 16 coins and a change of 1062. The greedy algorithm uses one more coin than the dynamic programming algorithm.

Hardware

For this activity we used a MacBook Pro (13-inch, 2018, Four Thunderbolt 3 Ports) with the following specs:

- **Processor:** 2.3 GHz Quad-Core Intel Core i5
- **Memory:** 8 GB 2133 MHz LPDDR3
- **Graphics:** Intel Iris Plus Graphics 655 1536 MB