Lab Report-10

(Coin Change)

CSE-2212 (Design and Analysis of Algorithms Lab)

Submitted By:

Name: Eyasir Ahamed
Exam Roll: 413
Class Roll: 15
Registration No: 202004017

Submitted To:

Sharad Hasan
Ex. Lecturer
Dept. of CSE
Sheikh Hasina University, Netrokona

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SHEIKH HASINA UNIVERSITY
NETROKONA, BANGLADESH

#Coin Change

#10.1_Counting number of ways

Problem Definition: Given a set of coins with different denominations and a target sum, the problem is to find the total number of ways to make change for the target sum using the coins. Each coin can be used an unlimited number of times.

Formal Statement of Algorithm (Counting Ways to Make Change using Dynamic Programming):

- Define a function countWaysToMakeChange that takes a vector of coin denominations (arr), the number of coins n, and the target sum T as parameters.
- Initialize a 2D DP table dp with dimensions n x T+1, where dp[i][j] represents the total number of ways to make change for the target sum j using the first i coins.
- Base Condition: Initialize the first row of the DP table such that dp[0][j] is 1 if j is divisible by the denomination of the first coin (arr[0]), otherwise initialize it as 0.
- Iterate over the remaining coins (ind = 1 to n-1) and the target sum (target = 0 to T).
- For each coin and target sum, calculate the total number of ways to make change by either:
    - Not taking the current coin (notTaken = dp[ind - 1][target]).

    - Taking the current coin if its denomination is less than or equal to the current target sum (taken = dp[ind][target - arr[ind]]).
- Update dp[ind][target] with the sum of notTaken and taken.
- The final result is in dp[n-1][T], representing the total number of ways to make change for the target sum using all the coins.

Complexity Analysis:

- Time Complexity: The algorithm fills in a 2D DP table of size n x T, so the time complexity is O(nT), where n is the number of coins and T is the target sum.
- Space Complexity: The space complexity is also O(nT) because of the DP table.

Actual Code and Output

```cpp
#include <bits/stdc++.h>
using namespace std;

long countWaysToMakeChange(vector<int>& arr, int n, int T) {
    vector<vector<long>> dp(n, vector<long>(T + 1, 0));

    for (int i = 0; i <= T; i++) {
        if (i % arr[0] == 0)
            dp[0][i] = 1;
    }

    for (int ind = 1; ind < n; ind++) {
        for (int target = 0; target <= T; target++) {
            long notTaken = dp[ind - 1][target];
            long taken = 0;
            if (arr[ind] <= target)
                taken = dp[ind][target - arr[ind]];

            dp[ind][target] = notTaken + taken;
        }
    }

    return dp[n - 1][T];
}

int main() {
    vector<int> arr = {1, 2, 3};
    int target = 4;
    int n = arr.size();

    cout << "The total number of ways is " << countWaysToMakeChange(arr, n, target) << endl;

    return 0;
}
```

```
The total number of ways is 4
[Finished in 1.2s]
```

#10.2_Minimum number of coins

Problem Definition: Given an array of denominations and a target sum, the problem is to find the minimum number of elements needed to form the target sum. Each element (coin) can be used an unlimited number of times.

Formal Statement of Algorithm (Minimum Number of Elements to Form Target Sum using Dynamic Programming):

- Define a function minimumElements that takes a vector of coin denominations (arr) and the target sum (T) as parameters.
- Initialize a 2D DP table dp with dimensions n x T+1, where dp[i][j] represents the minimum number of elements needed to form the target sum j using the first i coins.
- Initialize the first row of the DP table such that dp[0][j] is equal to j/arr[0] if j is divisible by the denomination of the first coin (arr[0]), otherwise set it to a very large value (1e9).
- Iterate over the remaining coins (ind = 1 to n-1) and the target sum (target = 0 to T).
- For each coin and target sum, calculate the minimum elements needed by either:
    - Not taking the current coin (notTake = dp[ind - 1][target]).
    - Taking the current coin if its denomination is less than or equal to the current target sum (take = 1 + dp[ind][target - arr[ind]]).
- Update dp[ind][target] with the minimum of notTake and take.
- The final result is in dp[n-1][T], representing the minimum number of elements needed to form the target sum using all the coins.

Complexity Analysis:

- Time Complexity: The algorithm fills in a 2D DP table of size n x T, so the time complexity is O(nT), where n is the number of denominations and T is the target sum.
- Space Complexity: The space complexity is also O(nT) because of the DP table.

Actual Code and Output

```cpp
#include <bits/stdc++.h>
using namespace std;

int minimumElements(vector<int>& arr, int T) {
    int n = arr.size();
    vector<vector<int>> dp(n, vector<int>(T + 1, 0));

    for (int i = 0; i <= T; i++) {
        if (i % arr[0] == 0)
            dp[0][i] = i / arr[0];
        else
            dp[0][i] = 1e9;
    }

    for (int ind = 1; ind < n; ind++) {
        for (int target = 0; target <= T; target++) {
            int notTake = dp[ind - 1][target];
            int take = 1e9;
            if (arr[ind] <= target)
                take = 1 + dp[ind][target - arr[ind]];
            dp[ind][target] = min(notTake, take);
        }
    }

    int ans = dp[n - 1][T];
    if (ans >= 1e9)
        return -1;
    return ans;
}

int main() {
    vector<int> arr = {1, 2, 3};
    int T = 7;
    int result = minimumElements(arr, T);
    cout << "The minimum number of coins required to form the target sum is " << result << endl
    return 0;
}

```

```
The minimum number of coins required to form the target sum is 3
[Finished in 1.2s]
```