Deakin University

Data Structures and Algorithms

OnTrack Submission

Programming - Coin Combinations

Submitted By:
Peter Stacey
pstacey
2020/10/05 22:19

Tutor:
Maksym Slavnenko

Outcome	**Weight**
Complexity	♦♦♦♦◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦◊◊◊◊

This task involves finding a solution to an NP-Hard problem in pseudo-polynomial time and required substantial evaluation of possible solutions to arrive at the result. This relates directly to ULO1 in terms of evaluating the memory and time complexity of approaches and to choose a recommended solution. Following selection of the solution, the task involves coding the structure and algorithm, aligned with ULO2.

October 5, 2020

```
// Student Name: Peter Stacey
// Student ID: 219011171
//
// Task 8.2HD

using System;
using System.Text;
using System.Text.RegularExpressions;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace CoinRepresentation
{
    /// <summary>
    /// Provices a static method to Solve the total number of combinations
    /// that exist in a set of coins of 2^k denominations.
    /// </summary>
    public class CoinRepresentation
    {
        private static Hashtable _cache = new Hashtable();

        /// <summary>
        /// Recursively solves the total number of combinations possible for
        /// a limited set of coins (two each) of 2^k denominations and returns
        /// the sum.
        ///
        /// Counts duplicate solutions as a single solution, so that the total
        /// returned is the total number of unique combuinations to the problem.
        ///
        /// For example, for the sum 6, the set of coins includes:
        /// 2^0, 2^1, 2^2, 2^3, 2^4, 2^5 and 2^6, or a set containing
        /// (1, 1, 2, 2, 4, 4, 8, 8, 16, 16, 32, 32, 64, 64)
        ///
        /// The solution optimises the problem space to ignore coins
        /// that exceed the total.
        ///
        /// For the example total of 6, 3 is returned, for
        /// the combinations:
        /// 4 + 1 + 1
        /// 4 + 2
        /// 2 + 2 + 1 + 1
        /// </summary>
        /// <param name="sum">The sum as a long, to find combinations for</param>
        /// <returns></returns>
        public static long Solve(long sum)
        {
            if (sum < 0) return 0; // No coins and no sum possible

            if (sum is 0) return 1; // Complete combination found

            if(!_cache.ContainsKey(sum))
            {
```

```
54                  if (sum % 2 is 0)
55                  {
56                      _cache.Add(sum, Solve(sum / 2) + Solve(sum / 2 - 1));
57                  }
58                  else
59                  {
60                      _cache.Add(sum, Solve((sum - 1) / 2));
61                  }
62              }
63              return Convert.ToInt64(_cache[sum]);
64          }
65      }
66  }
```