Department of Computer Science and Engineering (Data Science)

NAME: ALISTAIR SALDANHA
SAPID: 60009200024
BATCH: K1

Experiment 6

(Dynamic Programming)

Aim: Implementation of coin change problem using dynamic programming.

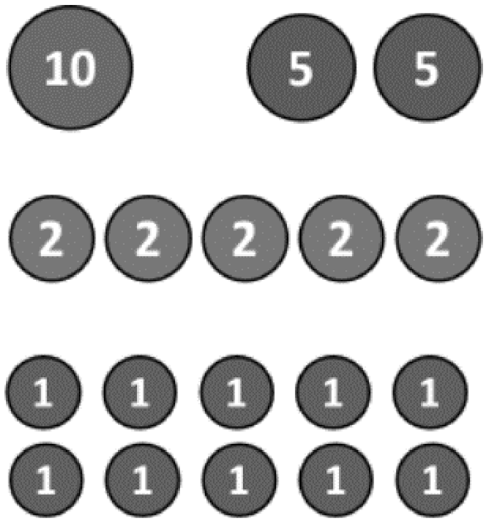
Theory:

Making Change problem is to find change for a given amount using a minimum number of coins from a set of denominations.

Explanation: If we are given a set of denominations D = {d0, d1, d2, …, dn} and if we want to change for some amount N, many combinations are possible. Suppose {d1, d2, d5, d8}, {d0, d2, d4}, {d0, d5, d7} all feasible solutions.

The aim of making a change is to find a solution with a minimum number of coins / denominations. Clearly, this is an optimization problem.

This problem can also be solved by using a greedy algorithm. However, greedy does not ensure the minimum number of denominations.



Various denominations for amount 10

General assumption is that infinite coins are available for each denomination. We can select any denomination any number of times.

Mathematical Formulation:

$$C[i, j] = \begin{cases} 1 + C[1, j - d_1], & \text{, if } i = j \\ C[i - 1, j], & \text{if } j < d_i \\ \min(C[i - 1, j], 1 + C[i, j - d_i]), & \text{otherwise} \end{cases}$$

Department of Computer Science and Engineering (Data Science)

Pseudocode:

```
Algorithm MAKE_A_CHANGE(d,N)
// d[1…n] = [d1,d2,…,dn] is array of n denominations
// C[1…n, 0…N]  is n x N array to hold the solution of sub problems
// N is the problem size, i.e. amount for which change is required

for  i ← 1 to n do
  C[i, 0] ← 0
end

for  i ← 1 to n do
  for  j ← 1 to N do
    if i = = 1 & & j < d [i]  then
            C[i, j]  ←  ∞
    else if i == 1 then
            C[i, j]  ← 1 + C[1, j – d[1])
    else if j < d [i]  then
            C[i, j]  ←  C[I – 1, j]
    else
            C[i, j]  ← min (C[i – 1, j] ,1 + C[i, j – d[i])
    end
  end
end
return C[n, N]

Algorithm TRACE_MAKE_A_CHANGE(C)
// When table C is filled up, i = n and j = N

Solution = { }
while ( j > 0 ) do
  if (C[i, j] == C[i – 1, j] then
    i ← i – 1
  else
    j ← j – di
    Solution = Solution ∪  {d[i] }
  end
end
```

Complexity:

Best Case Time Complexity: O(n)