Practice 2.2: Paradigm

Artificial Intelligence

G.Guérard

Dynamic Programming: the Coin Changing problem

Suppose we need to make change for 67£. We want to do this using the fewest number of coins possible among: 1, 5, 10, 25.

If the following coins are available: 1, 5, 10, 25. It is easy to see the optimal solution 67=2*25+10+5+2*1. By repeatedly choosing the largest coin less than or equal to the remaining sum, the desired sum is obtained by a greedy algorithm.

The formal description of the Coin Changing problem is as follows: Let $D = \{d_1, \dots, d_k\}$ be a finite set of distinct coin denominations. We assume each d_i is an integer and $d_1 > d_2 > \cdots > d_k = 1$. Each denomination is available in unlimited quantity. The problem is to make change for n£ using a minimum total number of coins, d_k =1 so there is always a solution.

The greedy method does not work in any case (it does not give an optimal solution). For example, $D = \{25,10,1\}$ and $n = 30$. The greedy method would produce a solution: 25+5*1, which is not as good as 3*10.

Step 1: Characterize the sub-structure. Define $C[j]$ to be the minimum number of coins we need to make change for j£. If we knew that an optimal solution for the problem of making change for j£ used a coin of denomination d_i we would have $C[j] = 1 + C[j - d_i]$.

Step 2: Define the value of an optimal solution. We can recursively define the value of an optimal solution from the equation found in step 1.

$$C[j] = min \begin{cases} \infty & if\ j < 0, \\ 0 & if\ j = 0 \\ 1 + min_{1 \le i \le k}\{C[j - d_i]\} & if\ j \ge 1 \end{cases}.$$

Step 3: Algorithm.

```
Coin-Changing(n,d,k)
        C[0]=0;
        For j from 1 to n
                C[j]=inf;
                For I from 1 to k
                        If j ≥ d_i and 1 + C[j − d_i] < C[j] then
                                C[j] = 1 + C[j − d_i].
                                Denom[j] = d_i
        Return C
```

We use an additional array Denom, where Denom[j] is the denomination of a coin used in an optimal solution to the problem of making change for j£. If the following coins are available: 1, 5, 10, 25:

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C[j]	0	1	2	3	4	1	2	3	4	5	1	2	3	4	5	2
Denom[j]	0	1	1	1	1	5	1	1	1	1	10	1	1	1	1	5

Given some n£, find all the combinations of coins that make up the value, the following coins are available: 1, 5, 10, 25. For example, for $N = 4, D = \{1,2,3\}$, there are four solutions: $\{1,1,1,1\}, \{1,1,2\}, \{2,2\}, \{1,3\}$.

Step 1: The set of solutions for this problem, $C(N, m)$, can be partitioned into two sets:

1. There are those sets that do not contain any d_m
2. Those sets that contain at least 1 d_m

If a solution does not contain d_m, then we can solve the subproblem of N with $D = \{d_1, d_2, \dots, d_{m-1}\}$, or the solutions of $C(N, m-1)$.

If a solution does contain d_m, then we are using at least one d_m, thus we are now solving the subproblem of N- d_m , with $D = \{d_1, d_2, \dots, d_m\}$. This is C(N- d_m, m).

Step 2: Thus, we can formulate the following: C(N-m)= $C(N, m-1)$+ C(N- d_m, m) with the base cases:

1. $C(N, m) = 1, N = 0,$
2. $C(N, m) = 0, N < 0$ (negative sum of money),
3. $C(N, m) = 0, N \geq 1, m \leq 0$ (no change available).

Step 3: The algorithm is as follows:

```
count(n, m)
        for i from 0 to n
                for j from 0 to m
                        if i equals 0
                                table[i, j] = 1
                        else if j equals 0
                                table[i, j] = 1 if i%S[j] equals 0 else 0
                        else if dj. greater than i
                                table[i, j] = table[i, j - 1]
                        else
                                table[i, j] = table[i - dj, j] + table[i, j-1]
        return table[n, m]
```

n/d	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4
10	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6
25	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6