Dynamic Programming

Andreas Klappenecker

[partially based on slides by Prof. Welch]

Dynamic Programming

- Optimal substructure
  - An optimal solution to the problem contains within it optimal solutions to subproblems.
- Overlapping subproblems
  - The space of subproblem is "small" so that the recursive algorithm has to solve the same problems over and over.

Giving Optimal Change

Motivation

We have discussed a greedy algorithm for giving change. However, the greedy algorithm is not optimal for all denominations.

Can we design an algorithm that will give the minimum number of coins as change for any given amount?

Answer: Yes, using dynamic programming.

Dynamic Programming Task

For dynamic programming, we have to find some subproblems that might help in solving the coin-change problem.

Ideas:

- Vary amount
- Restrict the available coins

Initial Set Up

Suppose we want to compute the minimum number of coins with values

$v[1]>v[2]>\ldots>v[n]=1$

to give change for an amount C.

Let us call the (i,j)-problem the problem of computing minimum number of coins with values

$v[i]>v[i+1]>\ldots>v[n]=1$

to give change for an amount $1<= j <= C$.

The original problem is the $(1,C)$-problem.

Tabulation

Let m[i][j] denote the solution to the (i,j)-problem.

Thus, m[i][j] denotes the minimum number of coins to make change for the amount j using coins with values v[i],...,v[n].

Tabulation Example

Denomination v[1]=10, v[2]=6, v[3]=1

Table of m[i][j] values:

		j											
		0	1	2	3	4	5	6	7	8	9	10	11	12
	1	0	1	2	3	4	5	1	2	3	4	1	2	2
i	2	0	1	2	3	4	5	1	2	3	4	5	6	2
	3	0	1	2	3	4	5	6	7	8	9	10	11	12

A Simple Observation

In calculating m[i][j], notice that:

a) Suppose we do not use the coin with value v[i] in the solution of the (i,j)-problem, then m[i][j] = m[i+1][j]

b) Suppose we use the coin with value v[i] in the solution of the (i,j)-problem, then m[i][j] = 1 + m[i][j-v[i]]

Tabulation Example

Denomination v[1]=10, v[2]=6, v[3]=1

Table of m[i][j] values:

		j											
		0	1	2	3	4	5	6	7	8	9	10	11	12
	1	0	1	2	3	4	5	1	2	3	4	1	2	2
i	2	0	1	2	3	4	5	1	2	3	4	5	6	2
	3	0	1	2	3	4	5	6	7	8	9	10	11	12

Recurrence

We either use a coin with value v[i] in the solution or we don't.

$$m[i][j] = \begin{cases} m[i+1][j] & \text{if } v[i] > j \\ \min\{\, m[i+1][j],\ 1+m[i][\ j-v[i]\ ]\ \} & \text{otherwise} \end{cases}$$

```
Dynamic_Coin_Change(C,v,n)

    allocate array m[1..n][0..C];

    for(i = 0; i<=C, i++)

       m[n][i] = i;  // make change for amount i using coins of value v[n]=1.

    for(i=n-1; i=>1; i--) { // successively allow a larger number coin values

       for(j=0; j<=C; j++) { // calc values of the array.

          if( v[i]>j || m[i+1][j]<1+m[i][j - v[i]] )

             m[i][j] = m[i+1][j]; // large coin does not help

          else m[i][j] = 1+m[i][j -v[i]]; //

       }

     }

    return &m;
```

Question

The previous algorithm allows us to find the minimum number of coins.

How can you modify the algorithm to actually compute the change (i.e., the multiplicities of the coins)?