Combinatorial Optimization
Dynamic programming

Industrial Informatics Research Center
http://industrialinformatics.fel.cvut.cz/

April 14, 2020

Overview of the tutorial

- Introduction to Dynamic Programming (10 minutes)
- Exercise: Coin-exchange problem (30 minutes)
- Exercise: Optimal coin system (20 minutes)

Part 1: Revision

Dynamic programming (DP) is a solution method for solving complex problems by breaking them into a collection of simpler subproblems, solving them once and storing their solutions. If the subproblems are nested recursively inside a larger problem, dynamic programming is applicable.

Very simple example: Imagine that you want to sum the squares of all numbers between 1 and n, i.e., $f(n) = \sum\limits_{i=1}^{n} i^2$. If you spent some effort and calculated the resulting number $f(k)$ for some input k (e.g., $f(9) = 285$), then it should be simple to calculate the result of $f(k+1)$. You could simply say it was $f(k)$ (that you already knew) plus $(k+1)^2 \rightarrow f(10) = 285 + 10^2 = 385$. You memorized the result of simpler problem and solved the the original problem for $f(k+1)$ by using a simple decomposition. That is the very basic principle of dynamic programming.

More formally: Typically, the problem is represented by a recursive function

$$f(n) = g(f(0), f(1), \ldots, f(n-1)), \tag{1}$$

where f is the problem we want to solve, n is the size of the problem, and g is some aggregate function.

Cliché example: Sometimes, the situation is not as simple as for the first example, where the value of $f(n)$ depended only on $f(n-1)$ and n itself. Remember the Fibonacci's sequence:

$$f(n) = \begin{cases} 0, \text{ if } n = 0, \\ 1, \text{ if } n = 1, \\ f(n-1) + f(n-2), \text{ if } n > 1. \end{cases} \tag{2}$$

You see that the subproblems repeat.

$$f(n) = \underbrace{f(n-1)}_{f(n-2)+f(n-3)} + f(n-2).$$

Exercise: Coin-exchange problem

Motivation: We have a vending machine with unlimited supply of coins of denominations $D = \{d_1, d_2, \dots, d_n\}$ and we want to make change for value V, i.e., we want to get the value V using only the denominations in D. Note that for some instances of the problem, the solution might not exist, e.g., denominations 5 and 10 can make change for 35, but not for 12.

For now, we solve the decision variant of the problem. Later on, we will complicate the formulation a little bit.

Exercise: Try to formulate and solve the decision variant of coin-exchange problem using DP. You are given the value V – the task is to decide whether the value V can be composed by the denominations from D only.

How to solve it in general? We can define function

$$f : \{0, \dots, n\} \times \{0, \dots, V\} \to \{0, 1\}$$

with meaning '$f(i, v) = 1$ iff it is possible to change value v by coins with denominations $(d_1, \dots, d_i)$'.

Then we can write the recursive formula

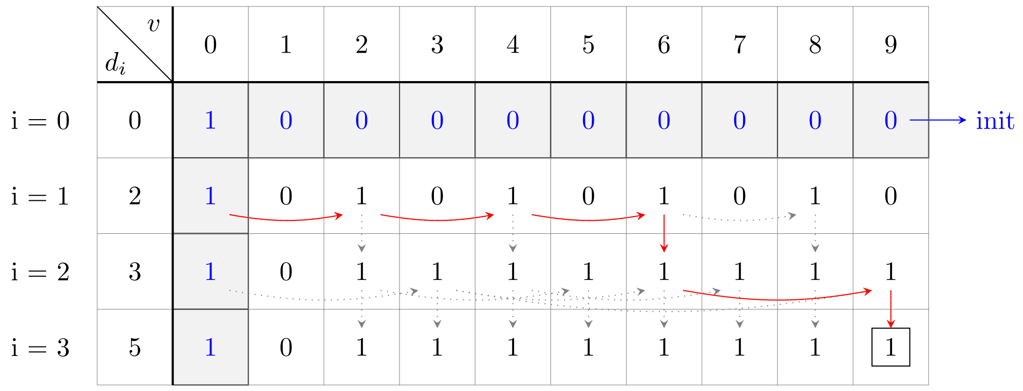
$$f(i, v) = \underbrace{f(i-1, v)}_{\text{done by } d_{i,\dots,d_{i-1}}} \quad \text{OR} \quad \underbrace{f(i, v - d_i)}_{\text{done using } d_i}. \tag{3}$$

The initialization is done as follows

$$f(0, v) = 0,\ \forall v \in \{1, \dots, V\}, \tag{4}$$
$$f(i, 0) = 1,\ \forall i \in \{0, \dots, n\}, \tag{5}$$
$$f(0, 0) = 1. \tag{6}$$

Then, we simply fill the table – left to right, top to bottom (i.e., we calculate the values of $f(i, v)$ using formula (3)). One can see, that the complexity is $\mathcal{O}(|D| \cdot V)$, which is pseudopolynomial. The solution can be read from the bottom-right cell. If we wanted to see, which coins contributed to the solution, we could remember the back-links.

Example: Let $D = \{2, 3, 5\}$ and $V = 9$.



	d_i \ v	0	1	2	3	4	5	6	7	8	9
i = 0	0	1	0	0	0	0	0	0	0	0	0
i = 1	2	1	0	1	0	1	0	1	0	1	0
i = 2	3	1	0	1	1	1	1	1	1	1	1
i = 3	5	1	0	1	1	1	1	1	1	1	$\boxed{1}$

Figure 1: Unlimited coins exchange (many lines are missing for better readability)

We see that value $V = 9$ can be composed from denominations $\{2, 3, 5\}$, e.g., by $2 + 2 + 2 + 3$ (as shown by red lines).

Optimization version: Let's slightly change the problem. Now, we would like to know, what is the minimal number of coins, which sum up to V (∞ if there is no solution).

Exercise: Try to solve the optimization version of coin-exchange problem by DP.

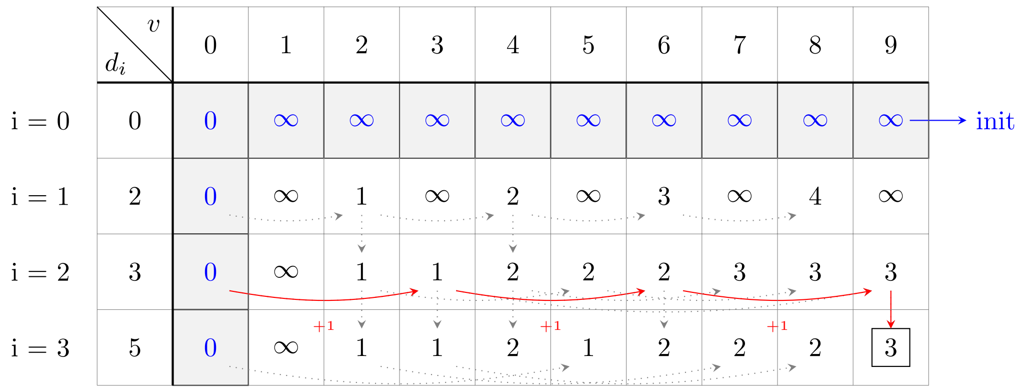
Solution: We need to change our function. Now, we have

$$f(i,v) = \min\{f(i-1,v), 1 + f(i, v-d_i)\}, \tag{7}$$

where $f(i,v)$ gives the minimal number of coins that sum up to v using only values $\{d_1,\dots,d_i\}$. Initialization is also slightly different:

$$f(0,v) = \infty,\ \forall v \in \{1,\dots,V\}, \tag{8}$$
$$f(i,0) = 0,\ \forall i \in \{0,\dots,n\}, \tag{9}$$
$$f(0,0) = 0. \tag{10}$$

Continuing with the previous example: $D = \{2,3,5\}$ and $V = 9$. We, again, fill the table. Finally, we see that 9 can be obtained from $3+3+3$ (using only 3 coins) – this solution is shown by read lines in the following figure. Alternatively, we could also get the solution $5+2+2$, which is also optimal.



	$d_i \backslash v$	0	1	2	3	4	5	6	7	8	9
i = 0	0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
i = 1	2	0	∞	1	∞	2	∞	3	∞	4	∞
i = 2	3	0	∞	1	1	2	2	2	3	3	3
i = 3	5	0	∞	1	1	2	1	2	2	2	$\boxed{3}$

Figure 2: Optimal version of coin exchange problem (many lines are missing for better readability)

Question – each coin only once: What would change, if each denomination should be used only once?

For decision problem, we would need to redefine function f:

$$f(i,v) = f(i-1,v) \text{ OR } f(i-1, v-d_i). \tag{11}$$

Question – limited number of coins: What would change, if the number of coins was limited? Let us assume, that we have maximally u_i coins of denomination d_i.

We could simply change D to

$$D = \{\underbrace{d_1, d_1, \dots, d_1}_{u_1 \text{ times}}, \underbrace{d_2, \dots, d_2}_{u_2 \text{ times}}, \dots, d_n\}, \tag{12}$$

then we can solve the decision problem with maximally one coin per denomination.

For example, for $D = \{5,2\}$, $u_1 = 3, u_2 = 2$, we would construct new $\tilde{D} = \{5,5,5,2,2\}$ and solve the problem with function (11).

Exercise: Optimal coin system

A dictator made a revolution in some country. He wants to optimize his country and so he remakes everything from scratch. This time, he is concerned with the old coin system. He decided that new coins and banknotes will be distributed. Their denominations will be such that the number of exchanged pieces will be minimized for some often used values.

Advisers found a set of often used values V. It was decided, that number of banknotes and coins will be at most u. We want to find a system D, $|D| < u$, of coins and banknotes such that

$$q(D) = \sum_{v \in V} \min \left\{ |S| \;\middle|\; \forall s_i \in S : s_i \in D, \sum_{i \in S} i = v \right\} \tag{13}$$

is minimized. Formally,

$$\min_{D} q(D) \tag{14}$$

$$\text{such that } |D| \leq k. \tag{15}$$

Exercise: Let $u = 3$ and $V = \{1, 2, \ldots, 100\}$.

Now, try to find a system of new banknotes and coins D, such that is contains at most 3 coins or banknotes, and all the values in V can be composed by system D efficiently (i.e., if you composed all values in V by denominations in D, you would have the minimal number of coins and banknotes). Write the DP and solve the problem.

Note that system $D = \{1\}$ is feasible, but surely not optimal. Value 1 can be composed by 1 coin, value 2 by 2 coins, etc. Finally, value 100 can be composed by 100 coins. It is easy to verify that the sum of all coins needed to represent values in V is $q(\{1\}) = 5050$ $((100 + 1) \cdot 100/2)$. Can you find some better system? Before solving the problem, try to guess the optimal system. The solution is shown on the following page.

Solution: The optimal solution is $D = \{1, 12, 19\}$ with objective value 521.

Some other systems (sorted by their evaluations):

```
System            Evaluation
(1, 12, 19)       521
(1, 7, 23)        522
(1, 8, 19)        525
(1, 9, 22)        526
(1, 13, 18)       526
...
(1, 98)           4759
(1, 99, 100)      4853
(1, 99)           4854
(1, 100)          4951
(1,)              5050
```

Summary

You should understand the concept of dynamic programming, and you should be able to solve some simple examples, such as small instances of knapsack, by hand. Also, you should be able to write the code to solve some bigger instances automatically (try. e.g., the *optimal coin system* problem).