Dynamic Programming 3: Dependency

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
Recall: Principle of dynamic programming

Resolve subproblems according to a certain order. Remember the output of every subproblem to avoid re-computation.

Sometimes, the order may not be immediately obvious. To figure out a working order, we need to look at the subproblems’ dependency.
Define a **string** as a sequence of characters.

A string $s$ with length $\ell$ can be stored in an array of length $\ell$.

We use $s[i]$ to represent the $i$-th char of $s$, for $i \in [1, \ell]$.

Given $i, j$ s.t. $1 \leq i \leq j \leq \ell$, we use $s[i : j]$ to represent the sequence $s[i]s[i + 1]...s[j]$, which is called a **substring** of $s$.

**Example:** If $s = \text{ABCD}$, then $s[2] = \text{B}$ and $s[2 : 4] = \text{BCD}$. 
**Problem**

$x = \text{a string of length } n$

$y = \text{a string of length } m$

Consider function $f(i, j)$ defined for any $i \in [0, n]$ and $j \in [0, m]$:

$$f(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
 f(i - 1, j - 1) + 1 & \text{if } i, j > 0 \text{ and } x[i] = y[j] \\
 \max\{f(i, j - 1), f(i - 1, j)\} & \text{if } i, j > 0 \text{ and } x[i] \neq y[j]
\end{cases}$$

**Goal:** Compute $f(n, m)$.

**Example:** Let $x = \text{ABC}$ and $y = \text{BDCA}$.
Then $f(2, 1) = f(2, 2) = f(2, 3) = 1$ and $f(3, 3) = f(3, 4) = 2$. 
A subproblem **depends** on another if the output of the latter is needed to solve the former.

\[
f(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
 0 & \text{if } i, j > 0 \text{ and } x[i] = y[j] \\
 0 & \text{if } i, j > 0 \text{ and } x[i] \neq y[j] \\
\end{cases}
\]

Subproblem \( f(i, j) \) may depend on

- one subproblem \( f(i - 1, j - 1) \) or
- two subproblems \( f(i, j - 1) \) and \( f(i - 1, j) \).

Which case it is depends on whether \( x[i] = y[j] \).
**Example:** \( x = \text{ABC} \) and \( y = \text{BDCA} \).

The dependency graph:

The cell at row \( i \in [0, 3] \) and column \( j \in [0, 4] \) represents subproblem \( f(i, j) \).

For example, \( f(3, 4) \) depends on \( f(3, 3) \) and \( f(2, 4) \), while \( f(3, 3) \) depends only on \( f(2, 2) \). Each arrow direction indicates the intended computation order, e.g., both \( f(3, 3) \) and \( f(2, 4) \) must be tackled before \( f(3, 4) \).
To solve all subproblems with dynamic programming, we need to find a **topological order**.

This is an ordering of all subproblems satisfying the following condition: if subproblem $A$ depends on subproblem $B$, then $A$ must appear after $B$ in the ordering.

Any topological order can be deployed for dynamic programming.
For function

\[
f(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
 f(i - 1, j - 1) + 1 & \text{if } i, j > 0 \text{ and } x[i] = y[j] \\
 \max\{f(i, j - 1), f(i - 1, j)\} & \text{if } i, j > 0 \text{ and } x[i] \neq y[j]
\end{cases}
\]

we can use either the row-major order or the column-major order.

**Example:** \(x = ABC\) and \(y = BDCA\).

\[
\begin{array}{c|cccc}
0 & 1 & 2 & 3 & 4 \\
\hline 
y & B & D & C & A \\
0 & x & & & & \\
1 & A & & & & \\
2 & B & & & & \\
3 & C & & & &
\end{array}
\]

Row-major: \(F(0, 0), F(0, 1), \ldots, F(0, 4), F(1, 0), \ldots, F(1, 4), \ldots\)

Col-major: \(F(0, 0), F(1, 0), \ldots, F(3, 0), F(0, 1), \ldots, F(3, 1), \ldots\)
For any \( f(i,j) \), we can compute it in \( O(1) \) time, \textbf{given} the outputs of the subproblems it depends on.

We can therefore compute \( f(n,m) \) in \( O(nm) \) time (the number of subproblems is \( O(nm) \)).