Dynamic Programming: Piggyback, Dependency, and Space

Yufei Tao’s Teaching Team

Department of Computer Science and Engineering
Chinese University of Hong Kong
Principle of Dynamic Programming

- Remember the output of every subproblem to avoid re-computation.
- Resolve subproblems according to an appropriate order.
In the lecture, we derived for the rod cutting problem:

\[
opt(n) = \max_{i=1}^{n}(P[i] + opt(n - i)).
\]

Define \( bestSub(n) = k \) if the above maximization is obtained at \( i = k \).

**Example**

<table>
<thead>
<tr>
<th>length ( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>price ( P[i] )</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>( opt(i) )</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>( bestSub(i) )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

How to compute \( bestSub(1), bestSub(2), ..., bestSub(n) \) in \( O(n^2) \) time?
Solution

First, compute \( \text{opt}(1), \text{opt}(2), \ldots, \text{opt}(n) \) in \( O(n^2) \) time, as discussed in the lecture.

For each \( t \in [1, n] \), compute \( \text{bestSub}(t) \) as follows:

- Identify the \( k \in [1, k] \) maximizing \( P[k] + \text{opt}(t - k) \).
  - This takes \( O(t) \) time.
- Set \( \text{bestSub}(t) = k \).

Doing so for all \( t \in [1, n] \) takes \( O(n^2) \) time.

The idea of computing \( \text{bestSub}(t) \) for all \( t \in [1, n] \) is called the piggyback technique.
In the lecture, we derived for the rod cutting problem:

$$opt(n) = \max_{i=1}^{n}(P[i] + opt(n - i)).$$

Define $bestSub(n) = k$ if the above maximization is obtained at $i = k$.

Suppose that we have already computed $bestSub(1)$, $bestSub(2)$, ..., $bestSub(n)$. How do we output an optimal cutting method — namely, a sequence of lengths achieving the maximum revenue — in $O(n)$ time?
Solution

1. \( \ell \leftarrow n \)
2. while \( \ell > 0 \) do
3. output “length bestSub(\( \ell \))”
4. \( \ell \leftarrow \ell - bestSub(\ell) \)

Example

<table>
<thead>
<tr>
<th>length ( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>price ( P[i] )</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>( opt(i) )</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>( bestSub(i) )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Output:
length 2
length 2
Problem 3 (Regular List 6)

Let $A$ be an array of $n$ integers. Define function $f(a, b)$ — where $a \in [1, n]$ and $b \in [1, n]$ — as follows:

$$f(a, b) = \begin{cases} 
0 & \text{if } a \geq b \\
(\sum_{c=a}^{b} A[c]) + \min_{c=a+1}^{b-1} \{f(a, c) + f(c, b)\} & \text{otherwise}
\end{cases}$$

Design an algorithm to calculate $f(1, n)$ in $O(n^3)$ time.
List all the subproblems.

\[ f(5, 8) \]
Solution

\[ f(a, b) = 0 \text{ when } a \geq b. \]
Solution

\[ f(a, b) = \left( \sum_{c=a}^{b} A[c] \right) + \min_{\substack{c=a+1}}^{b-1} \{ f(a, c) + f(c, b) \} \text{ when } a < b. \]

Find out the dependency relationships.
Solution

\[ f(a, b) = (\sum_{c=a}^{b} A[c]) + \min_{c=a+1}^{b-1} \{f(a, c) + f(c, b)\} \text{ when } a < b. \]

Let us start with the gray cells — they correspond to \( f(a, b) \) where \( a = b - 1 \). These cells depend on no other cells.
Solution

Let us continue with the green cells — they correspond to $f(a, b)$ where $a = b - 2$. Every such cell depends on two gray cells, which have already been computed.
Let us continue with the red cells — they correspond to $f(a, b)$ where $a = b - 3$. Every such cell depends on two gray cells and two green cells, all of which have been computed.
Solution

The order can be summarized as follows.

- All cells $f(a, b)$ with $b - a = 1$, each computed in $O(1)$ time.
- All cells $f(a, b)$ with $b - a = 2$, each computed in $O(2)$ time.
- ...
- All cells $f(a, b)$ with $b - a = k$, each computed in $O(k)$ time.
- ...
- All cells $f(a, b)$ with $b - a = n - 1$, each computed in $O(n - 1)$ time.

There are $O(n^2)$ values to calculate.
Total time complexity = $O(n^3)$. 
Problem 4 (Space Consumption)

In Lecture Notes 8, our algorithm for computing $f(n, m)$ used $O(nm)$ space. Next, we will reduce the space complexity to $O(n + m)$.

Recall the dependency graph:
Solution

We can calculate the values in the row-major order, i.e., row 0 to row 3 and left to right in each row. We used $O(mn)$ space because we stored all the values. Observe, however, that only two rows need to be stored at any moment.

```
  0  1  2  3  4
  | y | B | D | C | A |
 0  x  |
 1  A  |
 2  B  |
 3  C  |
```
Solution

Same idea for the column-major order.

So the space complexity is $O(\min\{m, n\})$, in addition to the $O(n + m)$ space needed to store $x$ and $y$. 
Think: Can this trick be used to reduce the space in Problem 2?