Dynamic Programming

Refinement of Divide & Conquer for situations with overlapping subproblems

1) solve subproblems, often bottom up
2) memorize their solutions (in a table), to avoid solving them repeatedly

D&C vs. Dynamic Programming

Example How many shortest rectilinear paths from square (0,0) to square (n,n)?

Denote this number by $L_{n,n}$

Divide-and-Conquer solution:

func $L(r,s: \text{int})$ returns int:

if $r = 0$ or $s = 0$ return 1;
else return $L(r-1,s) + L(r,s-1)$;

Recursion tree for $L(3,3)$:

A tabulating (DP) solution:

for $j := 0$ to $n$ do $L[0,j] := 1$;
for $i := 1$ to $n$ do
$L[i,0] := 1$;
for $j := 1$ to $n$ do
$L[i,j] := L[i-1,j] + L[i,j-1]$;
return $L[n,n]$;

A tabulating (DP) solution:

for $j := 0$ to $n$ do $L[0,j] := 1$;
for $i := 1$ to $n$ do
$L[i,0] := 1$;
for $j := 1$ to $n$ do
$L[i,j] := L[i-1,j] + L[i,j-1]$;
return $L[n,n]$;
Example Compute the product of $n$ matrices

\[ M = M_1 \times M_2 \times \ldots \times M_n \]

Exponentially many orders – Which is optimal?

Assume dimensions $|M_1| = 10 \times 20$, $|M_2| = 20 \times 50$, $|M_3| = 50 \times 1$ and $|M_4| = 1 \times 100$. Order

\[ M = M_1(M_2(M_3M_4)) \]

requires 125000 MUL operations. The order

\[ M = (M_1(M_2M_3))M_4 \]

requires 2200 MUL ops only

Input: Dimensions of $M_i$ as pairs $r_{i-1} \times r_i$

What is the smallest number $m_{ij}$ of MUL operations for computing $M_i \times \cdots \times M_j$?

\[ i = j; \]

\[ i < j; \]

Compute the required numbers of MUL ops $m_{ij}$ in increasing order of length $l = j - i$:

\[
\text{for } i := 1 \text{ to } n \text{ do } m_{ii} := 0;
\]

\[
\text{for } l := 1 \text{ to } n-1 \text{ do }
\]

\[
\text{for } i := 1 \text{ to } n-l \text{ do }
\]

\[
\quad j := i + l;
\]

\[
\quad m_{ij} := \min_{i \leq k < j} \{m_{ik} + m_{k+1,j} + r_{i-1} \times r_k \times r_j\};
\]

\[
\text{return } m_{1,n};
\]

Complexity is $O(n^3)$

Optimization with matrices $[10 \times 20] \times [20 \times 50] \times [50 \times 1] \times [1 \times 100]$:

\[
m_{12} = m_{11} + m_{22} + r_{0} \times r_{1} \times r_{2},
\]

\[
m_{13} = \min\{m_{11} + m_{23} + r_{0} \times r_{1} \times r_{3},
\]

\[
m_{12} + m_{33} + r_{0} \times r_{2} \times r_{3}\}
\]

\[
m_{14} = \min\{m_{11} + m_{24} + r_{0} \times r_{1} \times r_{4},
\]

\[
m_{12} + m_{34} + r_{0} \times r_{2} \times r_{4},
\]

\[
m_{13} + m_{44} + r_{0} \times r_{3} \times r_{4}\}
Why did we tabulate?

1. The problem is separable: the solution is a combination of sub-solutions
2. Without memorizing the same subproblems would be solved repeatedly, leading to exponential complexity
3. There are not too many subproblems
   \(|\{m_{ij} | 1 \leq i \leq j \leq n\}| = \Theta(n^2)\)
   → it is possible to store their solutions

Example (Floyd-Warshall algorithm)

DP computation of all shortest paths in a weighted graph

Let \(d_{ij}\) be the weight of the edge \((v_i, v_j)\)
\((1 \leq i, j \leq n)\)

```plaintext
procedure shortestPaths(
    d: array[1..n; 1..n] of real
)
returns array of real
for i := 1 to n do
    for j := 1 to n do
        \(m_{ij} := d_{ij};\)
    for k := 1 to n do
        for j := 1 to n do
            \(m_{ij} := \min\{m_{ij}, m_{ik} + m_{kj}\};\)
return \(m_{ij};\)
```

Idea of correctness

Invariant: \(m_{ij}\) is the min length for paths from \(v_i\) to \(v_j\) using nodes \(\{v_1, \ldots, v_k\}\) only

\[
\begin{array}{c|ccc}
    & 1 & 2 & 3 \\
\hline
   1 & \infty & 5 & 1 \\
   2 & 6 & \infty & \infty \\
   3 & \infty & 2 & \infty \\
\end{array}
\]

\[
\begin{array}{c|ccc}
    & 1 & 2 & 3 \\
\hline
   1 & 1 \\
   2 & 2 \\
   3 & 3 \\
\end{array}
\]

\[
\begin{array}{c|ccc}
    & 1 & 2 & 3 \\
\hline
   1 & 1 \\
   2 & 2 \\
   3 & 3 \\
\end{array}
\]

\[
\begin{array}{c|ccc}
    & 1 & 2 & 3 \\
\hline
   1 & 1 \\
   2 & 2 \\
   3 & 3 \\
\end{array}
\]
Example Knapsack with DP

Pack, in volume $T$, an optimal load of items with volume $t_i$ and value $a_i$

Tabulate solutions for subproblems of items \{1, \ldots, i\}, $i = 0, \ldots, n$, and volumes $t = 0, \ldots, T$:

$A_{i,t} =$ max value for a selection of items \{1, \ldots, i\} that fits in volume $t$

Value of an optimal load: $A_{n,T}$

Clearly $A_{0,t} = A_{i,0} = 0$

If $t_i > t$, the best load from items \{1, \ldots, i\} is the same as from items \{1, \ldots, i - 1\} → $A_{i,t} = A_{i-1,t}$

If $t_i \leq t$, the optimal load is the better one of those that either include item $i$ or do not:

$A_{i,t} = \max \{ A_{i-1,t-t_i} + a_i, A_{i-1,t} \}$

```
for i := 0 to n do A_{i,0} := 0;
for t := 1 to T do A_{0,t} := 0;
for i := 1 to n do
  for t := 1 to T do
    A_{i,t} := A_{i-1,t};
    if t ≥ t_i and
      A_{i-1,t-t_i} + a_i > A_{i,t} then
      A_{i,t} := A_{i-1,t-t_i} + a_i;
```

$(T = 5)$

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_i$</td>
<td>$a_i$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$t_1$</td>
<td>2</td>
<td>$a_1$</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>$a_2$</td>
<td>10</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>3</td>
<td>$a_3$</td>
<td>20</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$t_4$</td>
<td>2</td>
<td>$a_4$</td>
<td>15</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>