Dynamic Programming

- Useful when
  - same recursive sub-problems occur repeatedly
  - Can anticipate the parameters of these recursive calls
  - The solution to whole problem can be figured out with knowing the internal details of how the sub-problems are solved
- principle of optimality

List partition problem

- Given: a sequence of $n$ positive integers $s_1, \ldots, s_n$ and a positive integer $k$
- Find: a partition of the list into up to $k$ blocks:
  $s_1, \ldots, s_i | s_{i+1}, \ldots, s_j | s_{j+1}, \ldots, s_k | \ldots | s_{k-1}, \ldots, s_n$
  so that the sum of the numbers in the largest block is as small as possible.
  i.e. find spots for up to $k-1$ dividers

Greedy approach

- Ideal size would be $P = \sum_{i=1}^{n} s_i / k$
- Greedy: walk along until what you have so far adds up to $P$ then insert a divider
- Problem: it may not exact (or correct)
- sum is 4800 so size must be at least 1600.
- Greedy? Best?

Recursive solution

- Try all possible values for the position of the last divider
- For each position of this last divider
  - there are $k-2$ other dividers that must divide the list of numbers prior to the last divider as evenly as possible
    $s_1, \ldots, s_i | s_{i+1}, \ldots, s_j | s_{j+1}, \ldots, s_k | \ldots | s_{k-1}, \ldots, s_n$
  - recursive sub-problem of the same type

Recursive idea

- Let $M(n,k)$ the smallest cost (size of largest block) of any partition of the $n$ into $k$ pieces.
- If best position for last divider lies between the $j^{th}$ and $i+1^{st}$ then
  $M(n,k) = \max ( M(i,k-1), \sum_{i=1}^{j} s_i )$
- In general
  $M(n,k) = \min_{i<j} \max ( M(i,k-1), \sum_{j=1}^{n} s_j )$
- Base case(s)?
Time-saving - prefix sums

Computing the costs of the blocks may be expensive and involved repeated work.

Idea: Pre-compute prefix sums

Length of block

\[ s_{i+1} + \ldots + s_j \]

is just

\[ p[j] - p[i] \]

Cost: \( n \) additions

\[ p[1] = s_1 \]
\[ p[2] = s_1 + s_2 \]
\[ p[3] = s_1 + s_2 + s_3 \]
\[ \vdots \]
\[ p[n] = s_1 + s_2 + \ldots + s_n \]

Linear Partition Algorithm

\( \text{Partition}(S, k) : \)
\[ p[0] \leftarrow 0; \]
\[ \text{for } i = 1 \text{ to } n \text{ do } p[i] \leftarrow p[i-1] + s_i \]
\[ \text{for } i = 1 \text{ to } n \text{ do } M[i,1] \leftarrow p[i] \]
\[ \text{for } j = 1 \text{ to } k \text{ do } M[1,j] \leftarrow s_1 \]
\[ \text{for } i = 2 \text{ to } n \text{ do } \]
\[ \text{for } j = 2 \text{ to } k \text{ do } \]
\[ M[i,j] \leftarrow \infty \]
\[ \text{for } pos = 1 \text{ to } i-1 \text{ do } \]
\[ s_\text{max} \leftarrow \max(M[pos,j-1], p[i] - p[pos]) \]
\[ \text{if } M[i,j] > s_\text{max} \text{ then } \]
\[ M[i,j] \leftarrow s_\text{max}; D[i,j] \leftarrow pos \]

Example:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>700</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>2800</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>3400</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>4200</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>4800</td>
<td></td>
</tr>
</tbody>
</table>