ACMS Seminar Today

- Top Ten Algorithms of the 20th Century
- The Fast Fourier Transform
  - Speaker: Peter Blossey
- Smith 205, 3:30-4:20

CSE 417: Algorithms and Computational Complexity

Dynamic Programming, II

Autumn 2002
Paul Beame

Reading assignment

- Read sections 3.1-3.2 of The ALGORITHM Design Manual

Three Steps to Dynamic Programming

- Formulate the answer as a recurrence relation or recursive algorithm
- Show that the number of different parameters in the recursive algorithm is “small”
  - e.g., bounded by a low-degree polynomial
- Specify an order of evaluation for the recurrence so that you already have the partial results ready when you need them.

List partition problem

- Given: a sequence of n positive integers s_1,...,s_n and a positive integer k
- Find: a partition of the list into up to k blocks:
  s_1,...,s_i | s_{i+1}...s_j | s_{j+1}...s_{k-1} | s_k...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 s_i/k
- Greedy: walk along until what you have so far adds up to P then insert a divider
- Problem: it may not be exact (or correct)

100  200  400  500  900  700  600  800  600

sum is 4800 so if k-3 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
  - 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 first n #’s into k pieces.
- If best position for last divider lies between the i' st and i+1 st then
  \[ M[n,k]= \max ( M[i,k-1], \sum_{j=i+1}^{n} s_j ) \]
- In general
  \[ M[n,k]= \min_{i \in [1,n]} \max ( M[i,k-1], \sum_{j=i+1}^{n} s_j ) \]
- Base case(s)?

Time-saving - prefix sums

- Computing the costs of the blocks may be expensive and involved repeated work
- Idea: Precompute prefix sums
- Length of block
  \[ s_{i+1} + \ldots + s_j \]
  is just
  \[ p[j] - p[i] \]
  \[ p[k]=s_1, p[k+1]=s_1+s_2, \ldots, p[n]=s_1+s_2+\ldots+s_n \]
- Cost: n additions

Linear Partition Algorithm

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

Example:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>700</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>600</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Partition}(S,k): \]
\[ p[0] \leftarrow 0; \]
\[ \text{for } k \text{ to } n \text{ do } p[k] \leftarrow p[k-1]+s_k \]
\[ \text{for } j=1 \text{ to } n \text{ do } M[i,j] \leftarrow \max (M[i,j], p[i]-p[j]) \]
\[ \text{for } j=1 \text{ to } k \text{ do } M[i,j] \leftarrow s_i \]
\[ \text{for } k \text{ to } n \text{ do } \]
\[ \text{for } j=2 \text{ to } k \text{ do } \]
\[ M[i,j] \leftarrow \min_{p \in [1,\text{pos}]} \max (M[i,p], p[j]-p) \]
\[ D[i,j] \leftarrow \text{value of } pos \text{ where min is achieved} \]
Example:

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

\[ \text{Partition}(S,k): \]

- \[ p(0) = 0 \]
- for \( k \) to \( n \) do
  - \( p(i) = p(j) + 1 \) if \( x_i \leq x_j \)
  - \( p(i) = p(i) \) otherwise

Find recursive algorithm

- Solve sub-problem on \( s_{n+1} \) and then try to extend using \( s_n \)
- Two cases:
  - \( s_n \) is not used
    - answer is the same answer as on \( s_{n+1} \)
  - \( s_n \) is used
    - answer is the longest increasing subsequence in \( s_{n+1} \) that ends in a number smaller than \( s_n \)

Example:

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

\[ \text{Partition}(S,k): \]

- \[ p(0) = 0 \]
- for \( k \) to \( n \) do
  - \( p(i) = p(i) \) if \( x_i \leq x_j \)
  - \( p(i) = p(j) + 1 \) otherwise

Longest Increasing Subsequence

- Given a sequence of integers \( s_1, \ldots, s_n \), find a subsequence \( s_i < s_j < \ldots < s_k \) with \( i, j, \ldots, k \) so that \( k \) is as large as possible.
- e.g. Given 9, 5, 2, 8, 7, 3, 1, 6, 4 as input,
  - possible increasing subsequence is 5, 7
  - better is 2, 3, 6 or 2, 3, 4 (either or which would be a correct output to our problem)

Refined recursive idea (stronger notion of subproblem)

- Suppose that we knew for each \( i \in \mathbb{N} \), the longest increasing subsequence in \( s_1, \ldots, s_n \) that ends in \( s_i \).
  - \( i \in \mathbb{N}-1 \) just the \( n-1 \) size subproblem we tried before.
- Now to compute value for \( i \in \mathbb{N} \)
  - \( s_n \) preceded by the maximum over all \( i \in \mathbb{N} \) such that \( s_i \cdot s_n \) of the longest increasing subsequence ending in \( s_i \)
  - First find the best \( \text{length} \) rather than trying to actually compute the sequence itself.
Recurrence

- Let $L[i] =$ length of longest increasing subsequence in $s_1, \ldots, s_n$ that ends in $s_i$.
- $L[j] = 1 + \max \{ L[i] : i < j \text{ and } s_i < s_j \}$ (where max of an empty set is 0)
- Length of longest increasing subsequence:
  - $\max \{ L[i] : 1 \leq i \leq n \}$

Computing the actual sequence

- For each $j$, we computed $L[j] = 1 + \max \{ L[i] : l < j \text{ and } s_l < s_j \}$ (where max of an empty set is 0)
- Also maintain $P[j]$, the value of the $i$ that achieved that max
  - this will be the index of the predecessor of $s_j$ in a longest increasing subsequence that ends in $s_j$
  - by following the $P[j]$ values we can reconstruct the whole sequence in linear time

Longest Increasing Subsequence Algorithm

- for $j = 1$ to $n$
  - $L[j] = 1$
  - $P[j] = 0$
  - for $i = 1$ to $j - 1$
    - if ($s_i < s_j \text{ and } L[i] + 1 < L[j]$) then
      - $P[j] = i$
      - $L[j] = L[i] + 1$
  - endfor
  - endfor

Now find $j$ such that $L[j]$ is largest and walk backwards through $P[j]$ pointers to find the sequence

Example

For $j = 1$ to $n$
- $L[j] = 1$
- $P[j] = 0$
for $i = 1$ to $j - 1$
  - if ($s_i < s_j \text{ and } L[i] + 1 < L[j]$) then
    - $P[j] = i$
    - $L[j] = L[i] + 1$
  - endif
  - endfor

Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>90</td>
<td>50</td>
<td>20</td>
<td>80</td>
<td>70</td>
<td>30</td>
<td>10</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>$L[i]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P[i]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>