COT 6405: Analysis of Algorithms
Giri NARASIMHAN
Room Scheduling Problem

- Given a set of requests to use a room
  - [0,6], [1,4], [2,13], [3,5], [3,8], [5,7], [5,9], [6,10], [8,11], [8,12], [12,14]
- Schedule largest number of above requests in the room
- Different approaches
  - Try by hand, exhaustive search, improve an initial solution, iterative methods, divide and conquer, greedy methods, etc.

Simple Greedy Selection
- Sort by start time and pick in “greedy” fashion
- Does not work. WHY?
  - [0,6], [6,10] is the solution you will end up with.

Other greedy strategies
- Sort by length of interval
- Does not work. WHY?
Greedy Algorithms

- Given a set of activities \((s_i, f_i)\), we want to schedule the maximum number of non-overlapping activities.

**GREEDY-ACTIVITY-SELECTOR** \((s, f)\)

1. \(n = \text{length}[s]\)
2. \(S = \{a_1\}\)
3. \(i = 1\)
4. for \(m = 2\) to \(n\) do
5. if \(s_m\) is not before \(f_i\) then
6. \(S = S \cup \{a_m\}\)
7. \(i = m\)
8. return \(S\)
Why does it work?

**THEOREM**
Let $A$ be a set of activities and let $a_1$ be the activity with the earliest finish time. Then activity $a_1$ is in some maximum-sized subset of non-overlapping activities.

**PROOF**
Let $S'$ be a solution that does not contain $a_1$. Let $a'_1$ be the activity with the earliest finish time in $S'$. Then replacing $a'_1$ by $a_1$ gives a solution $S$ of the same size.

Why are we allowed to replace? Why is it of the same size?

Then apply induction! How?
First choice was a good choice. Why?
Because it can be extended to an optimal soln.

If our first choice was a good choice, then?
Then we can recursively apply correctness to the remainder.
Recursive Greedy Activity Selector

- Given a set of activities \((s_i, f_i)\), we want to schedule the maximum number of non-overlapping activities.

\textbf{GREEDY-ACTIVITY-SELECTOR} \((s, f, k) \) // Find opt sol for A\([k..n]\)

1. If \(k > n\) then return empty set
2. \(\text{First} = k+1\)
3. for \(m = k+1\) to \(n\) do
4. \(\text{if } s_m \text{ is before } f_k \text{ then discard } s_m\)
5. \(\text{if } a_m = a_{\text{First}} \text{ then } \text{First}++\)
6. return \(a_k \cup \text{GREEDY-ACTIVITY-SELECTOR} (s, f, \text{First})\)
Greedy Algorithms – Huffman Coding

- **Huffman Coding Problem**
  
  **Example**: Release 29.1 of 15-Feb-2005 of TrEMBL Protein Database contains 1,614,107 sequence entries, comprising 505,947,503 amino acids. There are 20 possible amino acids. What is the minimum number of bits to store the compressed database?

  \[ \approx 2.5 \text{ G bits or 300MB}. \]

- **How to improve this?**
  
  **Information**: Frequencies are not the same.

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ala (A)</td>
<td>7.72</td>
</tr>
<tr>
<td>Arg (R)</td>
<td>5.24</td>
</tr>
<tr>
<td>Asn (N)</td>
<td>4.28</td>
</tr>
<tr>
<td>Asp (D)</td>
<td>5.28</td>
</tr>
<tr>
<td>Cys (C)</td>
<td>1.60</td>
</tr>
<tr>
<td>Gln (Q)</td>
<td>3.91</td>
</tr>
<tr>
<td>Glu (E)</td>
<td>6.54</td>
</tr>
<tr>
<td>Gly (G)</td>
<td>6.90</td>
</tr>
<tr>
<td>His (H)</td>
<td>2.26</td>
</tr>
<tr>
<td>Ile (I)</td>
<td>5.88</td>
</tr>
<tr>
<td>Leu (L)</td>
<td>9.56</td>
</tr>
<tr>
<td>Lys (K)</td>
<td>5.96</td>
</tr>
<tr>
<td>Met (M)</td>
<td>2.36</td>
</tr>
<tr>
<td>Phe (F)</td>
<td>4.06</td>
</tr>
<tr>
<td>Pro (P)</td>
<td>4.87</td>
</tr>
<tr>
<td>Ser (S)</td>
<td>6.98</td>
</tr>
<tr>
<td>Thr (T)</td>
<td>5.52</td>
</tr>
<tr>
<td>Trp (W)</td>
<td>1.18</td>
</tr>
<tr>
<td>Tyr (Y)</td>
<td>3.13</td>
</tr>
<tr>
<td>Val (V)</td>
<td>6.66</td>
</tr>
</tbody>
</table>

- **Idea**: Use shorter codes for more frequent amino acids and longer codes for less frequent ones.
Huffman Coding

2 million characters in file.


IDEA 1: Use ASCII Code
Each need at least 8 bits,
Total = 16 M bits = 2 MB

IDEA 2: Use 4-bit Codes
Each need at least 4 bits,
Total = 8 M bits = 1 MB

IDEA 3: Use Variable Length Codes

| Character | Frequency | Code
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.22</td>
<td>11</td>
</tr>
<tr>
<td>T</td>
<td>.22</td>
<td>011</td>
</tr>
<tr>
<td>C</td>
<td>.18</td>
<td>010</td>
</tr>
<tr>
<td>G</td>
<td>.18</td>
<td>001</td>
</tr>
<tr>
<td>N</td>
<td>.10</td>
<td>00011</td>
</tr>
<tr>
<td>Y</td>
<td>.05</td>
<td>00010</td>
</tr>
<tr>
<td>R</td>
<td>.04</td>
<td>00001</td>
</tr>
<tr>
<td>S</td>
<td>.04</td>
<td>00000</td>
</tr>
<tr>
<td>M</td>
<td>.03</td>
<td></td>
</tr>
</tbody>
</table>

How to Decode?
Need Unique decoding!
Easy for Ideas 1 & 2.
What about Idea 3?

110101101110010011000000000110

110101101110010011000000000110

2 million characters in file.

Length = ?
Expected length = ?

Sum up products of frequency times the code length, i.e.,

\[(.22 \times 2 + .22 \times 2 + .18 \times 3 + .18 \times 3 + .10 \times 3 + .05 \times 5 + .04 \times 5 + .04 \times 5 + .03 \times 5) \times 2 \text{ M bits} = 3.24 \text{ M bits} = .4 \text{ MB}\]
New Room Scheduling Problem

- Room Scheduling with Attendee Numbers: Given a set of requests to use a room (with # of attendees)

- Schedule requests to maximize the total # of attendees
  - Greedy Solution will be [1,4], [5,7], [8,11], [12,14]
  - And will satisfy 4 + 15 + 5 + 6 = 30 attendees
  - Greed is not good!
Dynamic Programming

- **Old Activity Problem Revisited**: Given a set of \( n \) activities \( a_i = (s_i, f_i) \), we want to schedule the maximum number of non-overlapping activities.
- **General Approach**: Attempt a recursive solution
Recursive Solution

Observation: To solve the problem on activities $A = \{a_1, \ldots, a_n\}$, we notice that either

- optimal solution does not include $a_n$
  - then enough to solve subproblem on $A_{n-1} = \{a_1, \ldots, a_{n-1}\}$
- optimal solution includes $a_n$
  - Enough to solve subproblem on $A_k = \{a_1, \ldots, a_k\}$, the set $A$ without activities that overlap $a_n$. 
Recursive Solution

\[
\text{int Rec-ROOM-SCHEDULING (s, f, t, n)}
\]

// Here n equals length[s];
// Input: first n requests with their s & f times & # attend
// It returns optimal number of requests scheduled
1. Let k be index of last request with finish time before \( s_n \)
2. Output larger of two values:
3. \[ \{ \text{Rec-ROOM-SCHEDULING (s, f, t, n-1)}, \]
\[ \text{Rec-ROOM-SCHEDULING (s, f, t, k)} + t[n] \} \]
// \( t[n] \) is number of attendees of n-th request
Observations

- If we look at all subproblems generated by the recursive solution, and ignore repeated calls, then we see the following calls:
  - Rec-ROOM-SCHEDULING (s, f, n-1)
  - Rec-ROOM-SCHEDULING (s, f, n-2)
  - ...
  - Rec-ROOM-SCHEDULING (s, f, n')
  - ...
  - Rec-ROOM-SCHEDULING (s, f, k)
  - Rec-ROOM-SCHEDULING (s, f, k-1)
  - ...
  - Rec-ROOM-SCHEDULING (s, f, k')
  - ...

- Above list includes all subproblems $\text{Rec-ROOM-SCHEDULING} (s, f, i)$ for all values of $i$ between 1 and $n$
Let $A$ be the set of $n$ activities $A = \{a_1, \ldots, a_n\}$ (sorted by finish times).

The inputs to the subproblems are:

$A_1 = \{a_1\}$

$A_2 = \{a_1, a_2\}$

$A_3 = \{a_1, a_2, a_3\}, \ldots,$

$A_n = A$

i-th Subproblem: Select the max number of non-overlapping activities from $A_i$
An efficient implementation

- Why not solve the subproblems on $A_1$, $A_2$, …, $A_{n-1}$, $A_n$ in that order?
- Is the problem on $A_1$ easy?
- Can the optimal solutions to the problems on $A_1$, …, $A_i$ help to solve the problem on $A_{i+1}$?

  YES! Either:
  - optimal solution does not include $a_{i+1}$
  - problem on $A_i$
  - optimal solution includes $a_{i+1}$
  - problem on $A_k$ (equal to $A_i$ without activities that overlap $a_{i+1}$)
  - but this has already been solved according to our ordering.
Dynamic Prog: Room Scheduling

- Solving for $A_n$ solves the original problem.
- Solving for $A_1$ is easy.
- If you have optimal solutions $S_1$, ..., $S_{i-1}$ for subproblems on $A_1$, ..., $A_{i-1}$, how to compute $S_i$?

Recurrence Relation:
- The optimal solution for $A_i$ either
  - Case 1: does not include $a_i$ or
  - Case 2: includes $a_i$

- Case 1: $S_i = S_{i-1}$
- Case 2: $S_i = S_k \cup \{a_i\}$, for some $k < i$.
  - How to find such a $k$? We know that $a_k$ cannot overlap $a_i$. 
DP: Room Scheduling w/ Attendees

**DP-ROOM-SCHEDULING-w-ATTENDEES** \( s, f, t \)

1. \( n = \text{length}[s] \)
2. \( N[1] = t, \)  // number of attendees in \( S_1 \)
3. \( F[1] = 1 \)  // last activity in \( S_1 \)
4. for \( i = 2 \) to \( n \) do
5. \hspace{1em} let \( k \) be the last activity finished before \( s_i \)
6. \hspace{1em} if \( (N[i-1] > N[k] + t_i) \) then  // Case 1
7. \hspace{2em} \( N[i] = N[i-1] \)
8. \hspace{2em} \( F[i] = F[i-1] \)
9. \hspace{1em} else  // Case 2
10. \hspace{2em} \( N[i] = N[k] + t_i \)
11. \hspace{2em} \( F[i] = 1 \)
12. Output \( N[n] \)

How to output \( S_n \)? Backtrack!

Time Complexity? \( O(n \lg n) \)
Approach to DP Problems

- Write down a recursive solution
- Use recursive solution to identify list of subproblems to solve (there must be overlapping subproblems for effective DP)
- Decide a data structure to store solutions to subproblems (MEMOIZATION)
- Write down Recurrence relation for solutions of subproblems
- Identify a hierarchy/order for subproblems
- Write down non-recursive solution/algorithm
Longest Common Subsequence

$S_1 = \text{CORIANDER}$

$S_2 = \text{CREDITORS}$

Longest Common Subsequence($S_1[1..9], S_2[1..9]$) = $\text{CRIR}$
Recursive Solution

LCS($S_1$, $S_2$, $m$, $n$)
// $m$ is length of $S_1$ and $n$ is length of $S_2$
// Returns length of longest common subsequence
1. If ($S_1[m] == S_2[n]$), then
2. return 1 + LCS($S_1$, $S_2$, $m-1$, $n-1$)
3. Else return larger of
4. LCS($S_1$, $S_2$, $m-1$, $n$) and LCS($S_1$, $S_2$, $m$, $n-1$)

Observation:
All the recursive calls correspond to subproblems to solve and they include LCS($S_1$, $S_2$, $i$, $j$) for all $i$ between 1 and $m$, and all $j$ between 1 and $n$. 
Recurrence Relation & Memoization

- **Recurrence Relation:**
  \[ \text{LCS}[i,j] = \begin{cases} 
  \text{LCS}[i-1, j-1] + 1, & \text{if } S_1[i] = S_2[j] \\
  \max\{ \text{LCS}[i-1, j], \text{LCS}[i, j-1] \}, & \text{otherwise} 
  \end{cases} \]

- **Table (m X n table):**

- **Hierarchy of Solutions:**
  - Solve in row major order
LCS Problem

LCS_Length (X, Y )
1. m ← length[X]
2. n ← Length[Y]
3. for i = 1 to m
4. do c[i, 0] ← 0
5. for j = 1 to n
6. do c[0,j] ← 0
7. for i = 1 to m
8. do for j = 1 to n
9. do if ( xi = yj )
   10. then c[i, j] ← c[i-1, j-1] + 1
   11. b[i, j] ← “ ”
   12. else if c[i-1, j] > c[i, j-1]
   13. then c[i, j] ← c[i-1, j]
   14. b[i, j] ← “↑”
   15. else
   16. c[i, j] ← c[i, j-1]
   17. b[i, j] ← “←”
18. return c[m,n]