Greedy Algorithm

Activities Selection: \( a_i = \{s_i, f_i\}, \{s_2, f_2\}, \ldots, \{s_n, f_n\} \)

1. Select the largest set of non-overlapping activities.
2. Observation: OK to start with activity ending first.
3. Then, take the next activity whose starting time is closest to the finishing time of first activity.

Recursive (already sorted with finishing time)

\[
\text{Rec - } A S(s, f, i) \rightarrow \text{Initially } o
\]
\[
m \leftarrow i + 1
\]

while \( m \leq n \) & \( s_m < f_i \)
\[
m \leftarrow m + 1
\]

if \( m \leq n \)
\[
\text{Return } \{s_m\} \cup \text{Rec - } A S(s, f, m)
\]
else: Return \( o \)

Iterative

Top-down: AS(s, f)

\[
i \leftarrow 1
\]

for \( m = 2 \) to \( n \)
\[
\text{if } s_m \geq f_i
\]
\[
A_i = A_i \cup \{s_m\}
\]
\[
A_i = m
\]

Return \( A_i \)

Greedy is often Dynamic Programming with 1 known subproblem.

Iterative version of Greedy Algorithm is top-down.
Greedy stays ahead. Show (by induction) \( V_i \) the first \( i \) "steps" greedy algorithm are "better" than the first \( i \) "steps" of any other algorithm (including optimal)

(a) do it by induction,
(b) since true for \( i = 1 \) last step \(\Rightarrow\) greedy is opt-
(c) trick part define "step", "better".

As: \( V \uparrow \forall \) schedule \( Z \), define:

\[
F_i(Z) = \text{finish time of } i^{th} \text{ scheduled activity in } Z
\]

Inductive Claim: \( \forall i, Z, F_i(\text{greedy}) \leq F_i(Z) \)

Proof: Induction on \( i \), \( i = 1 \) (just the definition of greedy algo).

Assume true up to \( i = m \),

**Greedy**: \( \#1 \cdots \#m+1 \)

**Z**: \( \#1 \cdots \#m+1 \)

\[ f_m \leq f_m(Z) \leq s_{m+1}(Z) \]

So, \( f_{m+1} \leq f_{m+1}(Z) \)

**Corollary**: \( |\text{Greedy}| \geq |Z| \)
**Local Search**: Argue always safe to include (one greedy choice in opt in optimal.

(*) Take \( v \in \text{opt} \), construct \( \text{opt}' \) not worse than \( \text{opt} \) which has one greedy choice (and reduces to the same subproblem as greedy.)

**Key Step**: Argue subproblem at \( \text{opt}' \) = subproblem greedy.

Essentially, induction on size of activity selection.

---

**Huffman Codes**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>fixed</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>110</td>
<td>101</td>
</tr>
<tr>
<td>opt</td>
<td>0</td>
<td>101</td>
<td>100</td>
<td>111</td>
<td>110</td>
<td>1100</td>
</tr>
</tbody>
</table>

**Prefix-Free Encoding**: \( \forall c_1, c_2 \), then, encoding of \( c_1 \) named \( E(c_1) \) is not prefix of \( E(c_2) \)

\[
\text{Cost}(E(c)) = \sum f(c) |E(c)|
\]

\[
\text{Cost (fixed)} = 3 \quad ; \quad \text{Cost opt} = 2.24
\]
Claim 1: Opt. never has nodes of degree 1 (one child)

Claim 2: Let $T_1, T_2$ be two solid trees which are identical except 2 frequencies swap.

$T_1$: $l_1 / l_2$

$T_2$: $l_1 / l_2$

$l_1 > l_2$ & $x < y$

$\text{cost}(T_2) \leq \text{cost}(T_1)$

Corollary: If we sort frequencies, then optimal lengths