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 \(S\)

Example

- \([1,4], [3,5], [0,6], [5,7], [5,9], [8,10], [8,11], [8,12], [2,13], [12,14]\) -- Sorted by finish times

Why does it work?

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

- **PROOF**
  Let \(S\) be a solution that does not contain \(a_i\). Let \(a'_1\) be the activity with the earliest finish time in \(S\). Then replacing \(a'_1\) by \(a_i\) 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?
Greedy Algorithms – Huffman Coding

- Huffman Coding Problem
  Example: Release 20.1 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?
  ~2.5 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>Gin (Q)</td>
<td>3.91</td>
</tr>
<tr>
<td>Leu (L)</td>
<td>9.56</td>
</tr>
<tr>
<td>Ser (S)</td>
<td>6.98</td>
</tr>
<tr>
<td>Arg (R)</td>
<td>5.24</td>
</tr>
<tr>
<td>Glu (E)</td>
<td>6.54</td>
</tr>
<tr>
<td>Lys (K)</td>
<td>5.96</td>
</tr>
<tr>
<td>Thr (T)</td>
<td>5.52</td>
</tr>
<tr>
<td>Asn (N)</td>
<td>4.28</td>
</tr>
<tr>
<td>His (H)</td>
<td>2.26</td>
</tr>
<tr>
<td>Phe (F)</td>
<td>4.06</td>
</tr>
<tr>
<td>Tyr (Y)</td>
<td>3.13</td>
</tr>
<tr>
<td>Cys (C)</td>
<td>1.60</td>
</tr>
<tr>
<td>Ile (I)</td>
<td>5.88</td>
</tr>
<tr>
<td>Pro (P)</td>
<td>4.87</td>
</tr>
<tr>
<td>Val (V)</td>
<td>6.66</td>
</tr>
<tr>
<td>Met (M)</td>
<td>2.36</td>
</tr>
<tr>
<td>Trp (W)</td>
<td>1.18</td>
</tr>
<tr>
<td>Asp (D)</td>
<td>5.28</td>
</tr>
<tr>
<td>Gly (G)</td>
<td>6.90</td>
</tr>
<tr>
<td>Thr (M)</td>
<td>2.36</td>
</tr>
<tr>
<td>Trp (W)</td>
<td>1.18</td>
</tr>
<tr>
<td>Asp (D)</td>
<td>5.28</td>
</tr>
<tr>
<td>Gly (G)</td>
<td>6.90</td>
</tr>
<tr>
<td>Thr (M)</td>
<td>2.36</td>
</tr>
<tr>
<td>Trp (W)</td>
<td>1.18</td>
</tr>
</tbody>
</table>

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

Dynamic Programming

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

  New Approach:
  \( A_i \) = Best solution for intervals \( (a_0, \ldots, a_i) \) that includes interval \( a_i \)
  \( B_i \) = Best solution for intervals \( (a_0, \ldots, a_i) \) that does not include interval \( a_i \)

  Does it solve the problem to compute \( A_i \) and \( B_i \)?
  How to compute \( A_i \) and \( B_i \)?
Dynamic Programming

- 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.

- New Approach:
  - Observation: To solve the problem on activities $A_n = \{a_1, \ldots, a_n\}$, we notice that either
    - optimal solution does not include $a_n$ (Problem on $A_{n-1}$)
    - optimal solution includes $a_n$ (Problem on $A_{k}$, which is equal to $A_n$ without activities that overlap $a_n$, i.e., $a_k$ is the last activity that finishes before $a_n$ starts.)

An efficient implementation

- Why not solve the problem on $A_1, \ldots, A_{n-1}, A_n$?
- In what order to solve them?
- Is the problem on $A_1$ easy?
  - YES, trivial
- Can the optimal solutions to the problems on $A_1, \ldots, A_i$ 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 $a_{i+1}$ (you are left with a problem on $A_k$, which is equal to $A_i$ without activities that overlap $a_{i+1}$, i.e., $a_k$ is the last activity that finishes before $a_{i+1}$ starts.)

Dynamic Programming: Activity Selection

- Select the maximum number of non-overlapping activities from a set of n activities $A = \{a_1, \ldots, a_n\}$ (sorted by finish times).
- Identify "easier" subproblems to solve.
  - $A_1 = \{a_1\}$
  - $A_2 = \{a_1, a_2\}$
  - $A_3 = \{a_1, a_2, a_3\}, \ldots$
  - $A_n = A$
- Subproblems: Select the max number of non-overlapping activities from $A_i$
Dynamic Programming: Activity Selection

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

```
Dynamic Programming: Activity Selection

- DP-ACTIVITY-SELECTOR (s, f)
1. n = length[s]
2. N[1] = 1 // number of activities in S_1
3. F[1] = 1  // last activity in S_1
4. for i = 2 to n do
5.   let k be the last activity finished before s_i
6.   if (N[i-1] > N[k]) then // Case 1
7.     N[i] = N[i-1]
8.     F[i] = F[i-1]
9.   else // Case 2
10.    N[i] = N[k] + 1
11.    F[i] = i

How to output S_n?
Backtrack!
Time Complexity?
O(n log n)
```

Dynamic Programming Features

- Identification of subproblems
- Recurrence relation for solution of subproblems
- Overlapping subproblems (sometimes)
- Identification of a hierarchy/ordering of subproblems
- Use of table to store solutions of subproblems (MEMOIZATION)
- Optimal Substructure