Adam Smith (1723 – 1790), an economist and philosopher of the Scottish Enlightenment.

He argued that if individuals act greedily in their own self-interest then the outcome will be beneficial for society.

“[The individual who acts for his own gain] is led by an invisible hand to promote an end which was no part of his intention.”
Several different university societies have all requested to book the sports hall, request $k$ having start time $u_k \in \mathbb{R}$ and end time $v_k \in \mathbb{R}$. The hall can only fit one activity at a time. What is the maximum number of requests that can be satisfied without overlap?

Let $f(X)$ be the maximum number of requests in a set $X$ that can be simultaneously satisfied. Then

$$f(X) = \begin{cases} \max_{k \in X} \{1 + f(\text{events in } X \text{ that end before } k \text{ starts}) + f(\text{events in } X \text{ that start after } k \text{ ends}) \} & \text{if } X \neq \emptyset \\ 0 & \text{if } X = \emptyset \end{cases}$$

**QUESTION**

Can we find a different way to set up this task so that the states aren’t sets?
Several different university societies have all requested to book the sports hall, request $k$ having start time $u_k \in \mathbb{R}$ and end time $v_k \in \mathbb{R}$. The hall can only fit one activity at a time. What is the maximum number of requests that can be satisfied without overlap?

Let's make this problem a bit more algorithm-friendly by making it discrete. Instead of using real numbers $(u_k, v_k) \in \mathbb{R} \times \mathbb{R}$ for start and end times, let's use integer time indexes $(\bar{u}(k), \bar{v}(k)) \in \mathbb{N} \times \mathbb{N}$, indexes into a list of "interesting" timepoints $t_0 < t_1 < \ldots < t_n \in \mathbb{R}$.

“All problems in computer science can be solved by adding a layer of indirection.”  
“Adding a layer of indirection creates more problems than it solves.”
Example 3.2.1 Resource allocation

Several different university societies have all requested to book the sports hall, request $k$ having start time $u_k \in \mathbb{R}$ and end time $v_k \in \mathbb{R}$. The hall can only fit one activity at a time. What is the maximum number of requests that can be satisfied without overlap?

Let $f(i,j) = \max \# \text{ requests that can be satisfied in } [t_i, t_j]$, for $i \leq j$. We want $f(0,n)$ (where the "interesting" time points are $t_0 < t_1 < \cdots < t_n$).

The Bellman equation is

$$f(i,j) = \begin{cases} 0 & \text{if } i = j \\ \max_{k \in X(i,j)} \left\{ 1 + f(i, \bar{u}(k)) + f(\bar{v}(k), j) \right\} & \text{otherwise} \end{cases}$$

Where $X(i,j) = \{ \text{requests that fit in } [t_i, t_j] \} = \{ l \in X : \bar{u}(l) \geq i \text{ and } \bar{v}(l) \leq j \}$. 
3.2 Greedy algorithms

To compute the best action from state $x$ using the Bellman recursion, we need to evaluate $v(\cdot)$ for all of $x$’s children in the dependency graph.

What if instead we use a simple heuristic to choose the next action?

The greedy strategy, with heuristic function $h$, is to pick action

$$\arg \max_{a \in A} h(x, a)$$

Heuristics are fast, but typically don’t give an optimal solution to the overall problem.

However, in some cases, if we set the problem up carefully, we can show that a greedy strategy is optimal.
Heuristic 1: always pick the shortest available activity

Heuristic 2: always pick the available activity with the fewest overlaps

Heuristic 3: pick the available activity with the earliest end-time
Theorem: We should always pick the activity with the earliest end-time.

What does this even mean? How can we express it as a proposition that's amenable to proof? Let's at least start by making some definitions.

Let $X$ be a set of requests.

Let $EE(X) \subseteq X$ be the set of events in $X$ with the earliest end-time. [There may be more than one.]

Let $Y \subseteq X$ be a maximal overlap-free subset.

Second attempt

Theorem: $\exists k \in EE(X)$ such that $k \in Y$.

This isn't even true! Consider $Y'$. Then neither of the events $k \in EE(X)$ come in $Y$.

Also, it isn't helpful: this says $\exists k \in EE(X)$, which is unhelpful, because it doesn't tell us which $k \in EE(X)$ we should pick.

What we really want to say: "we don't lose anything by picking some arbitrary $k \in EE(X)$".

Final attempt

Theorem: $\forall k \in EE(X)$ $\exists Y' \subseteq X$ such that $k \in Y'$ and $Y'$ is overlap-free and $|Y'| = |Y|$.

Example: EE($X$) $\rightarrow$ $\exists Y'$ such that $k \in Y'$ and $Y'$ is optimal $\rightarrow$ an optimal $Y'$. The theorem asserts that we can pick either of the $EE(X)$ requests, and find some optimal $Y'$ containing it.
Theorem: \( \forall k \in \text{EE}(x) \exists Y' \leq x \) such that \( k \in Y' \) and \( Y' \) is overlap-free and \( |Y'| = |Y| \).

Proof: Pick an arbitrary solution \( Y \) to the resource allocation problem, and an arbitrary \( k \in \text{EE}(x) \).

Either \( k \in Y \), in which case set \( Y' = Y \), and we are done.

Or \( k \notin Y \), in which case let's pick some \( l \in \text{EE}(Y) \) and construct \( Y' = Y \cup \{k\} \setminus \{l\} \).

\[ \text{CLAIM: } k \in Y' \text{ and } Y' \text{ is overlap-free and } |Y'| = |Y| \]

1. \( \text{CLAIM: } Y' \text{ is overlap-free.} \)
   - Suppose not; i.e., suppose there is some \( a, b \in Y', a \neq b \), that overlap.
     - Either \( a + k \) and \( b + k \) in which case \( a, b \in Y \), but \( Y \) is overlap-free.
     - Or \( a = k \) or \( b = k \), wlog \( a = k \). Since \( b \in Y' \) and \( b + k \), \( b \in Y \).
       - Since \( b \in Y \) and \( l \in Y \), \( b \) doesn't overlap with \( l \).
       - So Either \( b \) starts after \( l \) ends — but \( k \in \text{EE}(x) \) so \( \text{end}(k) \geq \text{end}(l) \), so \( k \) can't overlap \( b \).
       - Or \( b \) ends before \( l \) starts — impossible, by choice of \( l \) to be \( \in \text{EE}(Y) \).
     - We conclude that \( \) is false, i.e. \( \) is true. This proves \( \).
I call this a “might as well” proof.

We “might as well” pick some arbitrary $k \in EE(X)$, and it won’t hurt us i.e. won’t prevent us from achieving an optimal allocation.

The proof structure is:
1. Take an optimal solution $Y$
2. Propose a tweaked version $Y'$ that satisfies the property we want
3. Show that $Y'$ is also an optimal solution

To be able to prove (3), we need to choose a very cunning tweak for (2)!

Theorem $\forall k \in EE(x) \exists Y' \leq X$ such that $k \in Y'$ and $Y'$ is overlap free and $|Y'| = |Y|$.
Example 3.1.2 Longest common subsequence

A subsequence of a string $s$ is any string obtained by dropping zero or more characters from $s$. Given two strings $s$ and $t$, what’s the longest subsequence they have in common?

**Bellman equation:** Let $v_{i,j}$ be the length of the LCS between $s[0:i]$ and $t[0:j]$. Then

$$v_{i,j} = \begin{cases} 
  v_{i,j-1} \lor v_{i-1,j-1} \lor (1 + v_{i-1,j-1}) & \text{if } i = 0 \text{ or } j = 0 \\
  v_{i,j-1} \lor v_{i-1,j} & \text{if } i > 0 \text{ and } j > 0 \text{ and } s[i-1] = t[j-1] \\
  v_{i-1,j} \lor v_{i,j-1} & \text{if } i > 0 \text{ and } j > 0 \text{ and } s[i-1] \neq t[j-1]
\end{cases}$$

Claim: we might as well pick the $m$ action.

Proof sketch. Let $y$ be an optimal action sequence (yielding a LCS). Either $y$ uses this $m$, or there’s a $y'$ just as good that does.
3.2.2 Huffman codes

We have a string that we’d like to compress a string into a sequence of bits. We want a code that says how each character is to be encoded, e.g.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>11101</td>
<td>01010</td>
<td>11011</td>
<td>001</td>
<td>110101</td>
<td>110100</td>
<td>0001</td>
<td>0111</td>
<td></td>
</tr>
</tbody>
</table>

Our code has to produce uniquely decodable bit sequences. We can ensure this by insisting on a code that takes the form of a tree, called a **prefix-free code**.

0001001101111011110001011001010

**Problem statement.** Find a prefix-free code that minimizes the average codelength $L = \sum_i p_i \ell_i$, where $p_i$ is the frequency of letter $i$ and $\ell_i$ is the length of its codeword.

**Beautiful greedy algorithm.** Left as an optional tick.