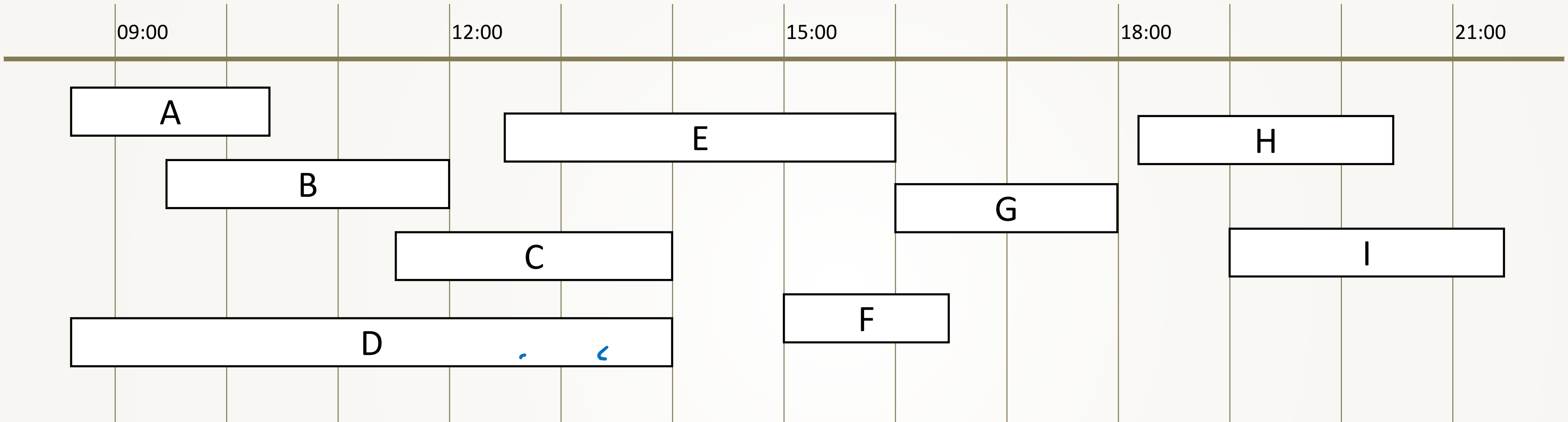
Adam Smith (1723 – 1790), an economist and philosopher of the Scottish Englightenment.

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.”

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

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

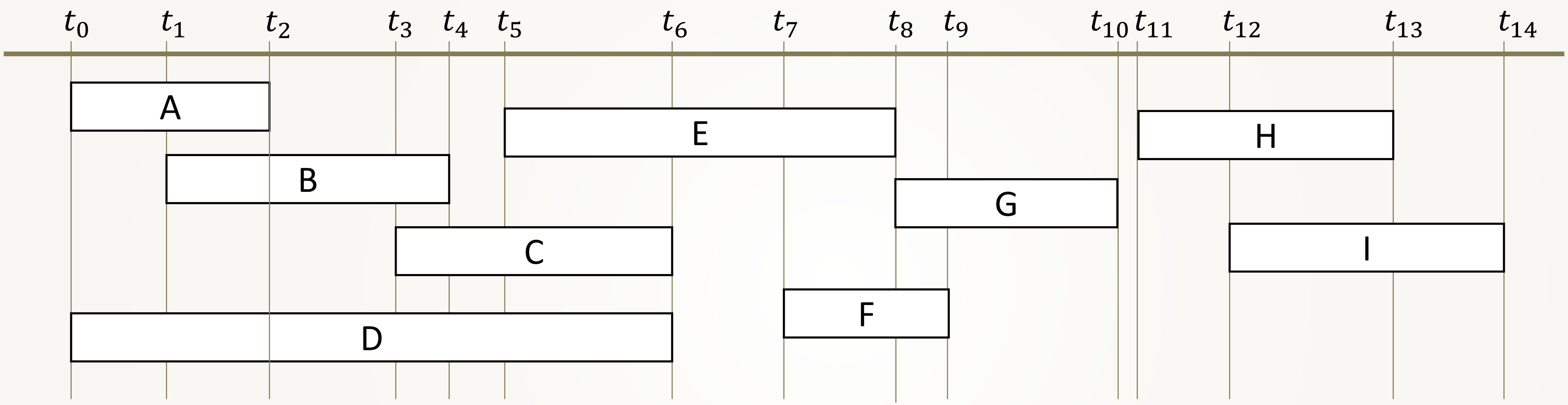
QUESTION
Can we find a different way to set up this task so that the states aren't sets?

# Example 3.2.1 Resource allocation

$t_{\bar{u}(k)}$, $\bar{u}(k) \in \mathbb{N}$

$t_{\bar{v}(k)}$, $\bar{v}(k) \in \mathbb{N}$.

Several different university societies have all requested to book the sports hall, request $k$ having start time $\cancel{u_k \in} \mathbb{R}$ and end time $\cancel{v_k \in \mathbb{R}}$. 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 $t_0 < t_1 < \cdots < t_n \in \mathbb{R}$.

"All problems in computer science can be solved by adding a layer of indirection."

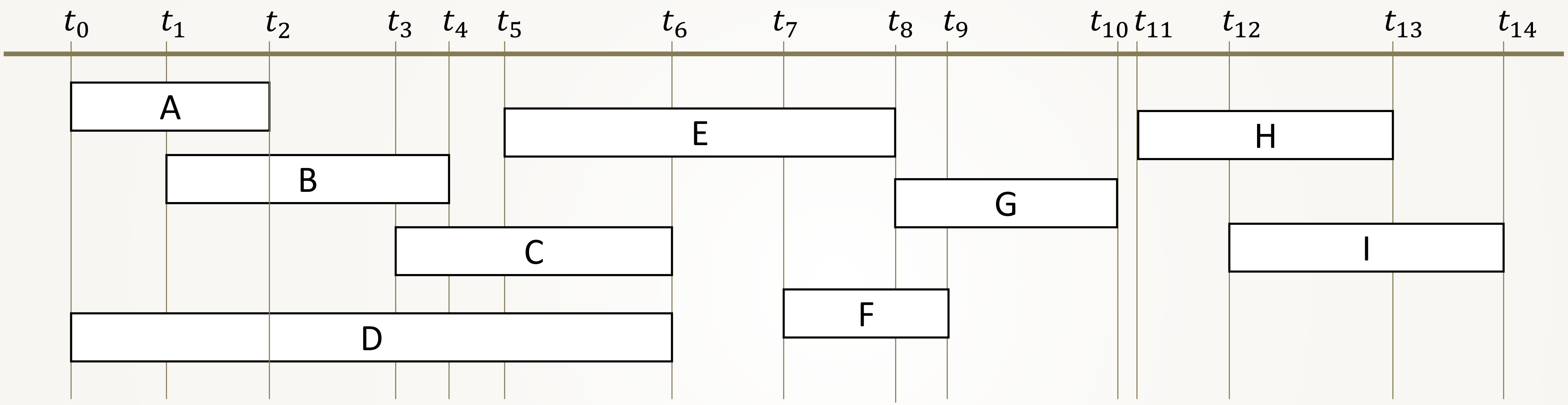
"Adding a layer of indirection creates more problems than it solves."

# Example 3.2.1 Resource allocation

$t_{\bar{u}(k)},\ \bar{u}(k) \in \mathbb{N}$

$t_{\bar{v}(k)},\ \bar{v}(k) \in \mathbb{N}.$

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 # requests that can be satisfied in $[t_i, t_j]$, for $i \le j$.

We want $f(0,n)$ (where the "interesting" time points are $t_0 < t_1 < \cdots < t_n$).

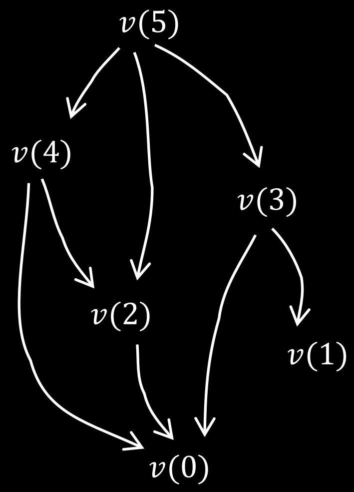
The Bellman equation is

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

where $X(i,j) = \{\text{requests that fit in } [t_i, t_j]\} = \{\ell \in X : \bar{u}(\ell) \ge i \text{ and } \bar{v}(\ell) \le j\}.$

# 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.



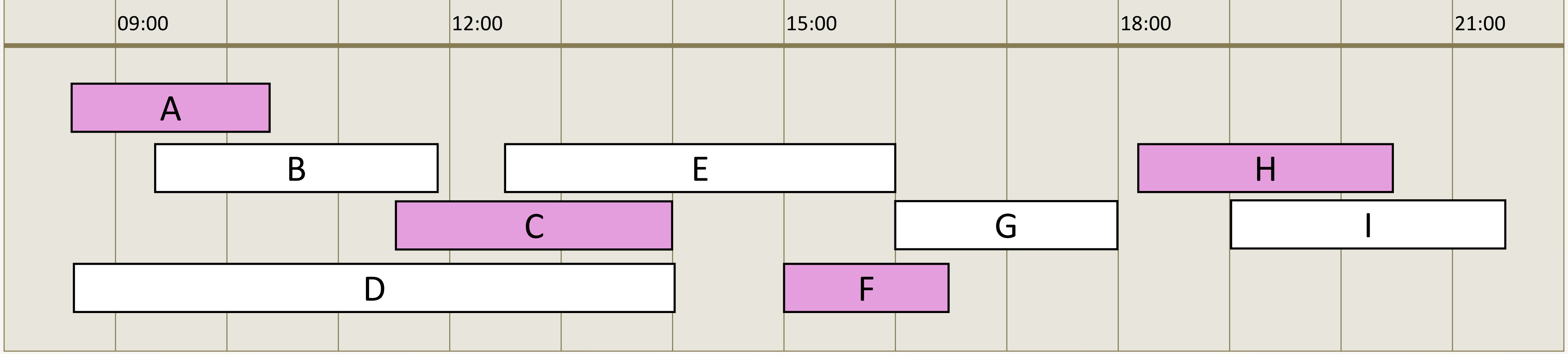

What if instead we use a simple heuristic to choose the next action?

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

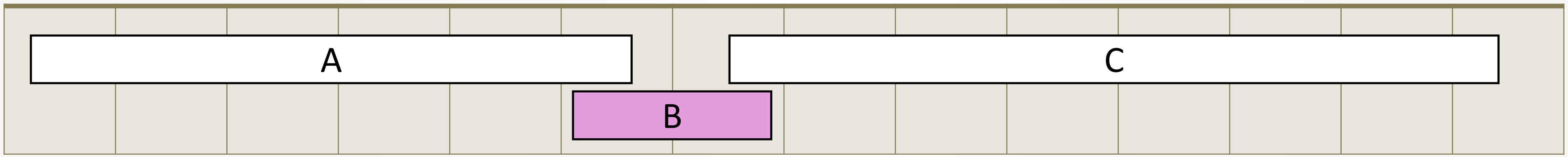
$$\arg\max_{a \in A} h(x, a)$$

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

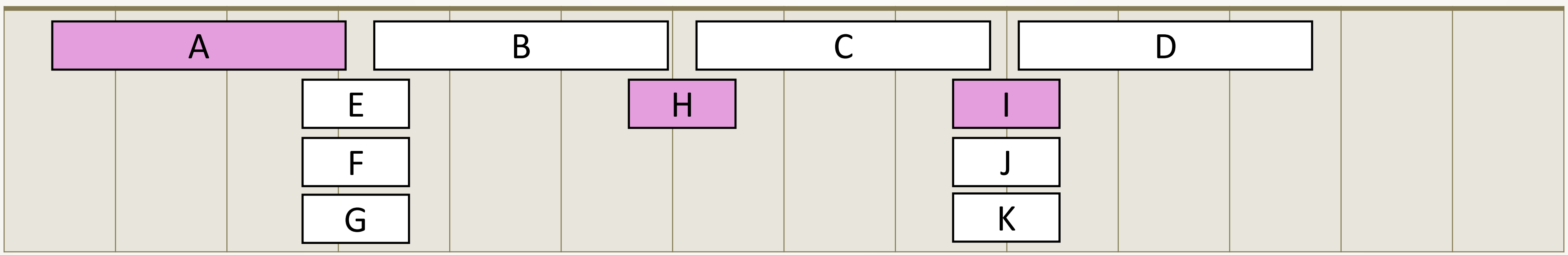
However, in some cases, if we set the problem up carefully, we can show that a greedy strategy is optimal.




Heuristic 1: always pick the shortest available activity ✗ Doesn't always work




Heuristic 2: always pick the available activity with the fewest overlaps ✗ Doesn't always work




Heuristic 3: pick the available activity with the earliest end-time

first attempt

**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.

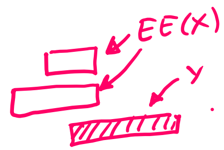
Let $EE(X) \subseteq X$ be the set of events in $X$ with the earliest end-time. [ There may be more than one.

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




Then neither of the events $k \in EE(X)$ are in $Y$.

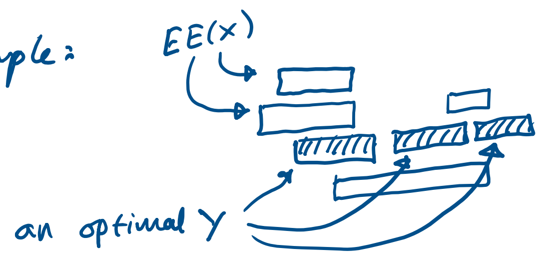
Also, it isn't helpful: this says $\exists\, k \in EE(X)$, which is unhelpful, because it doesn't tell us <u>which</u> $k \in EE(X)$ we should pick.

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

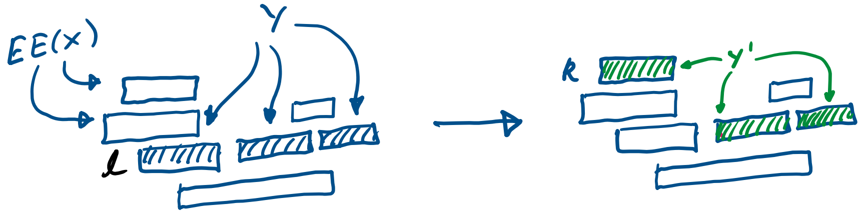
Final attempt

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

Example:




The theorem asserts that we can pick either of the $EE(X)$ requests, and find some optimal $Y'$ containing it.




**Theorem** $\forall\, k \in EE(X) \quad \exists\, Y' \subseteq X$ 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 $k \in EE(X)$.

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 $l \in EE(Y)$ and construct $Y' = Y \cup \{k\} \setminus \{l\}$.

③ CLAIM: $k \in Y'$ and $Y'$ is overlap-free and $|Y'| = |Y|$

we explicitly built $Y'$ to include $k$

Remains To Prove.

we added one item, removed one, so $|Y'| = |Y|$.

② CLAIM: $Y'$ is overlap-free.

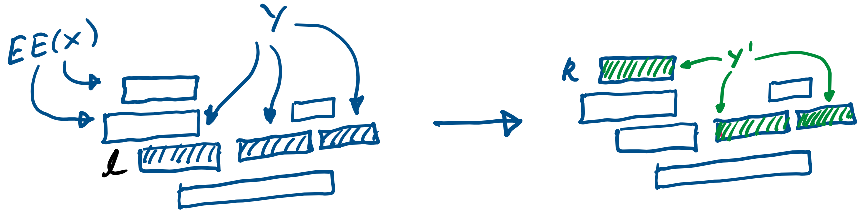
① Suppose not, ie suppose there is some $a, b \in Y'$, $a \neq b$, that overlap.

Either $a \neq k$ and $b \neq k$ 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 $b \neq k$, $b \in Y$.
Since $b \in Y$ and $l \in Y$, $b$ doesn't overlap with $l$.
So Either $b$ starts after $l$ ends — but $k \in EE(X)$ so $\text{end}(k) \leq \text{end}(l)$, so $k$ can't overlap $b$. $\#$
Or $b$ ends before $l$ starts — impossible, by choice of $l$ to be $\in EE(Y)$. $\#$



We conclude that ① is false, ie ② is true. This proves ③.




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

**I call this a "might as well" proof.**

We "might as well" pick some arbitrary $k \in \mathrm{EE}(X)$, and it won't hurt us i.e. won't prevent us from achieving an optimal allocation.

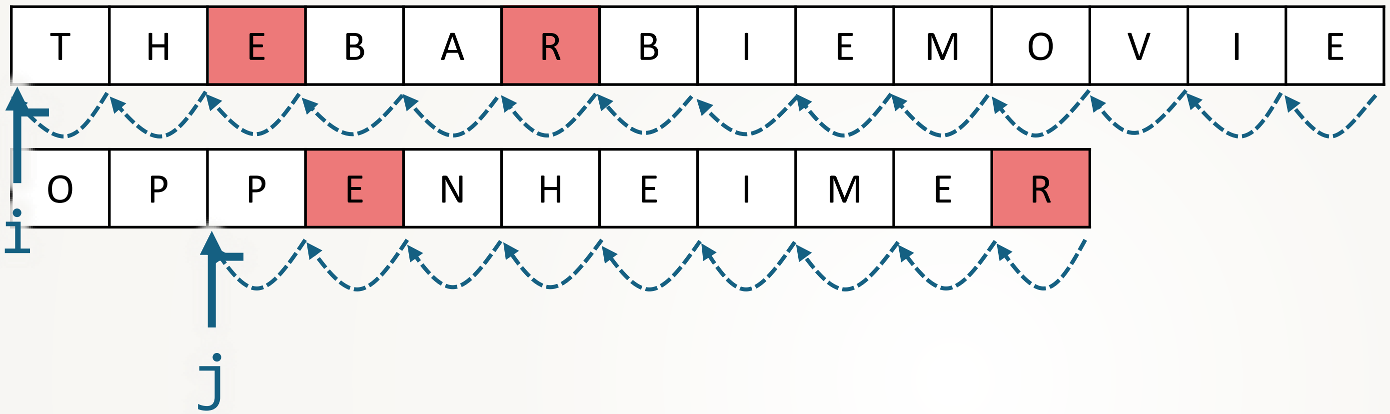
The proof structure is:

1. Take an optimal solution $Y$
2. Propose a tweaked version $Y'$ that satisfies the property we want In this case, the property $k \in Y'$
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)!

# 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?




Let's frame the task as choosing a sequence from these actions:

- $i$ decrement i
- $j$ decrement j
- $m$ match a character and decrement i & j

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

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

Three actions available.

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

Proof structure. Let $y$ be an optimal action sequence (yielding a LCS).

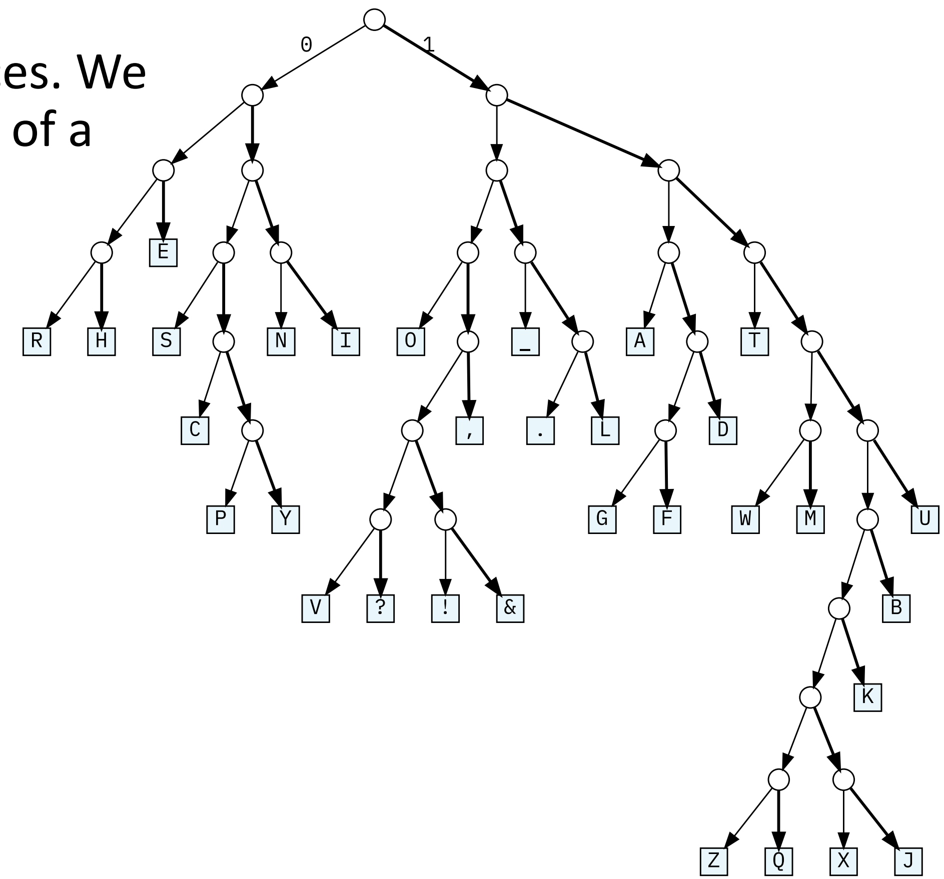
Either $y$ uses this $m$, or there's a $y'$ just as good that does.

# 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.

| A | B | C | D | E | F | G | H | I | … |
|---|---|---|---|---|---|---|---|---|---|
| 1100 | 111101 | 01010 | 11011 | 001 | 110101 | 110100 | 0001 | 0111 | |

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.*




```
000100110111101111000101100101O
```

**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.