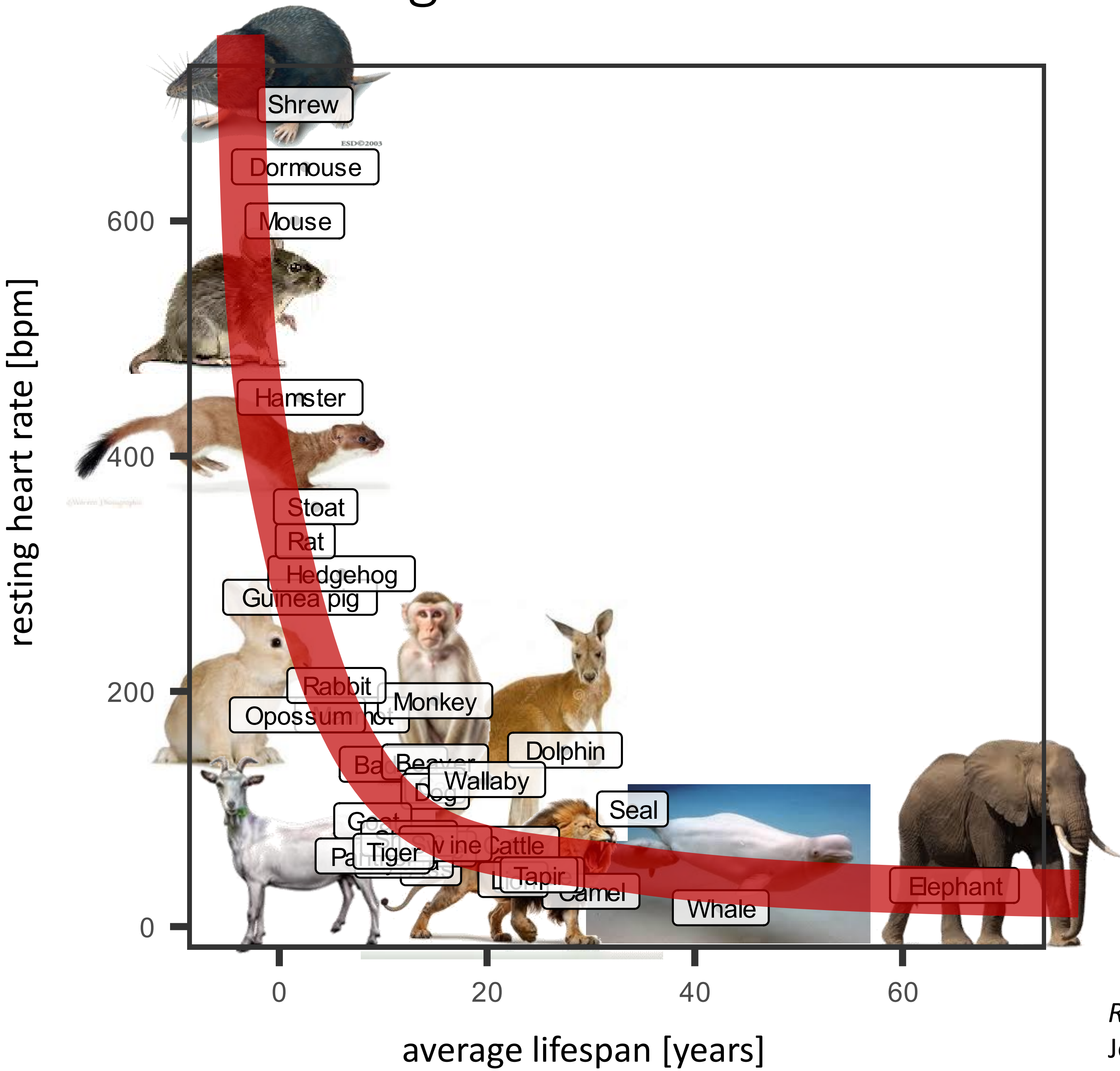
SECTION 3

Algorithm design

Is it worth doing cardio?



It looks like maybe

$$\text{heart rate} = \frac{\text{const}}{\text{lifespan}}$$

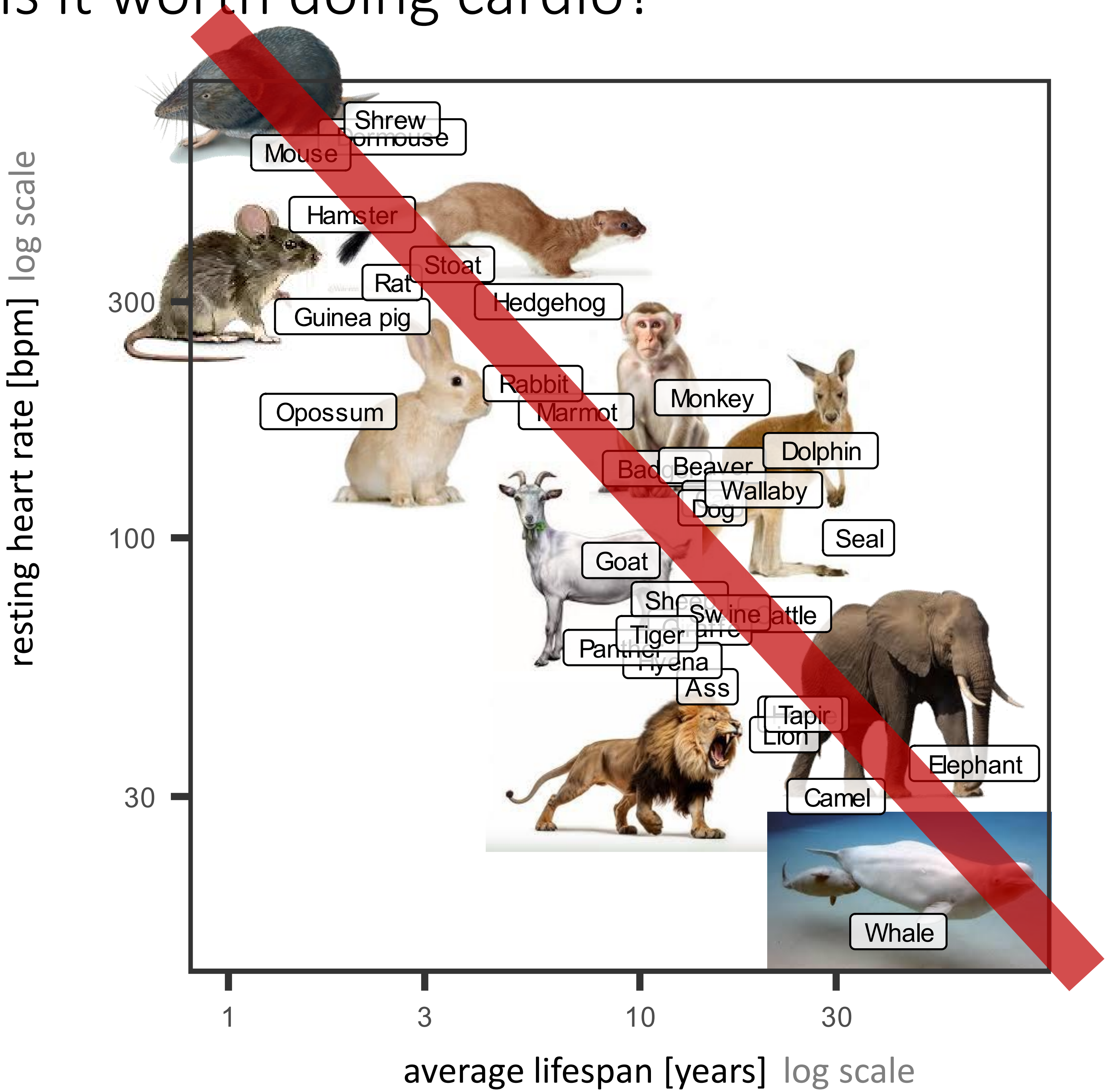
$\Longrightarrow$ $\text{heartrate} \times \text{lifespan} = \text{const}$

$\Longrightarrow$ $\text{total lifetime heartbeats} = \text{const}$

Equivalently,
$\log(\text{heartrate}) + \log(\text{lifespan}) = \text{const}$
and this is easier to see on a log-log plot.

Rest Heart Rate and Life Expectancy, Herbert J. Levine, Journal of the American College of Cardiology (1997)

Is it worth doing cardio?



It looks like maybe

$$\text{heart rate} = \frac{\text{const}}{\text{lifespan}}$$

$$\implies \text{heartrate} \times \text{lifespan} = \text{const}$$

$$\implies \text{total lifetime heartbeats} = \text{const}$$

Equivalently,
$\log(\text{heartrate}) + \log(\text{lifespan}) = \text{const}$
and this is easier to see on a log-log plot.

Rest Heart Rate and Life Expectancy, Herbert J. Levine, Journal of the American College of Cardiology (1997)

Let’s suppose we have a fixed number of total lifetime heartbeats.
If we want to live as long as possible, how much should we exercise?

(Exercise decreases resting heartrate 👍, but it burns through our lifetime heartbeats 👎.)

let $x = (r, b)$ be the current state, r = resting heartrate and b = # heartbeats remaining as of today

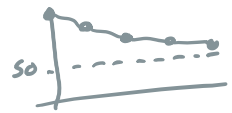
let's invent a model:

If we exercise:

$$r \leftarrow r - \lambda(r - 50)$$

$$b \leftarrow b - (23 \times 60 \times r - 60 \times 155)$$

this has the effect that, if we exercise each day, r will be pulled down to a floor of 50 bpm



23 hours at heart rate r, 1 hour at heart rate 155 bpm

If we don't exercise:

$$r \leftarrow r + \lambda(90 - r)$$

$$b \leftarrow b - 24 \times 60 \times r$$

if we never exercise, r will creep up to a ceiling of 90 bpm

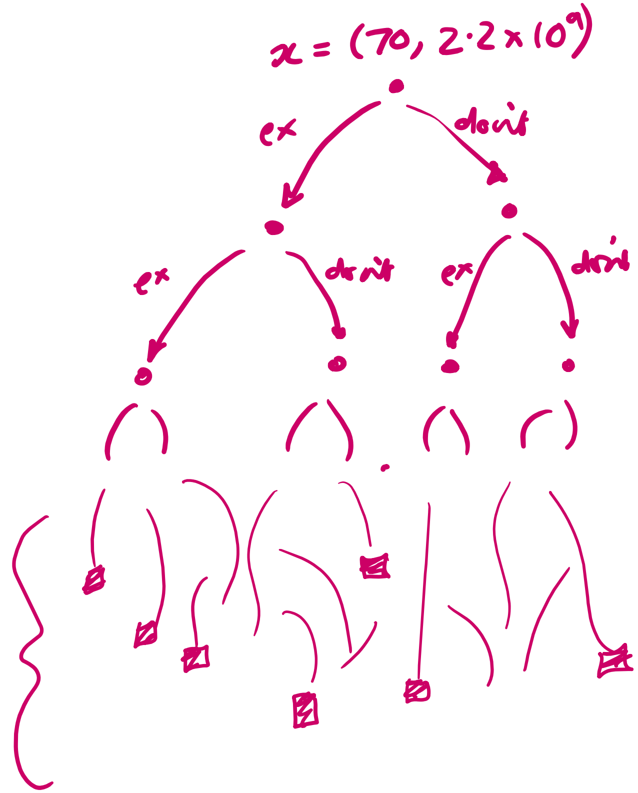
let $v(x)$ = max possible remaining lifespan starting from state $x = (r, b)$.

If $b \leq 0$: $v(x) = 0$

If $b > 0$: $v(x) = \max_{a \in \{\text{exercise, don't exercise}\}} \{ 1 + v(\text{next}_{x,a}) \}$

we live for one day

$\text{next}_{x,a}$ is shorthand for the four equations above specifying how x evolves



terminal states (death)

3.1 The Bellman equation and dynamic programming

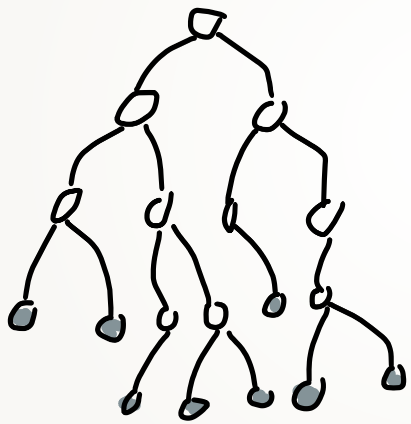
Problem statement

programme

We're given an initial state x_0, and we wish to choose a sequence of actions $[a_0, a_1, \dots]$.

If we're in state x and we take action a, we gain $\text{reward}_{x,a}$, and move to $\text{nextstate}_{x,a}$ (unless x is a terminal state, where no further actions are possible, in which case we gain termreward_x).

We want to find the maximum possible total reward starting from x_0.



Bellman recursion

Let $v(x)$ be the total reward that can be gained starting in state x. Then

$$v(x) = \begin{cases} \text{termreward}_x & \text{if } x \text{ is terminal} \\ \max_{a \in A} \left\{ \text{reward}_{x,a} + v(\text{nextstate}_{x,a}) \right\} & \text{otherwise} \end{cases}$$

3.1 The Bellman equation and dynamic programming

Problem statement

We're given an initial state x_0, and we wish to choose a sequence of actions $[a_0, a_1, \dots]$.

If we're in state x a ... $\text{te}_{x,a}$
(unless x is a termi ... we gain
termreward_x).

We want to find the

How can I frame my task as "find an optimal sequence of actions"?

- What are the actions?
- What is the value/cost that I'm optimizing?

Bellman recursion

Let $v(x)$ be the total reward that can be gained starting in state x. Then

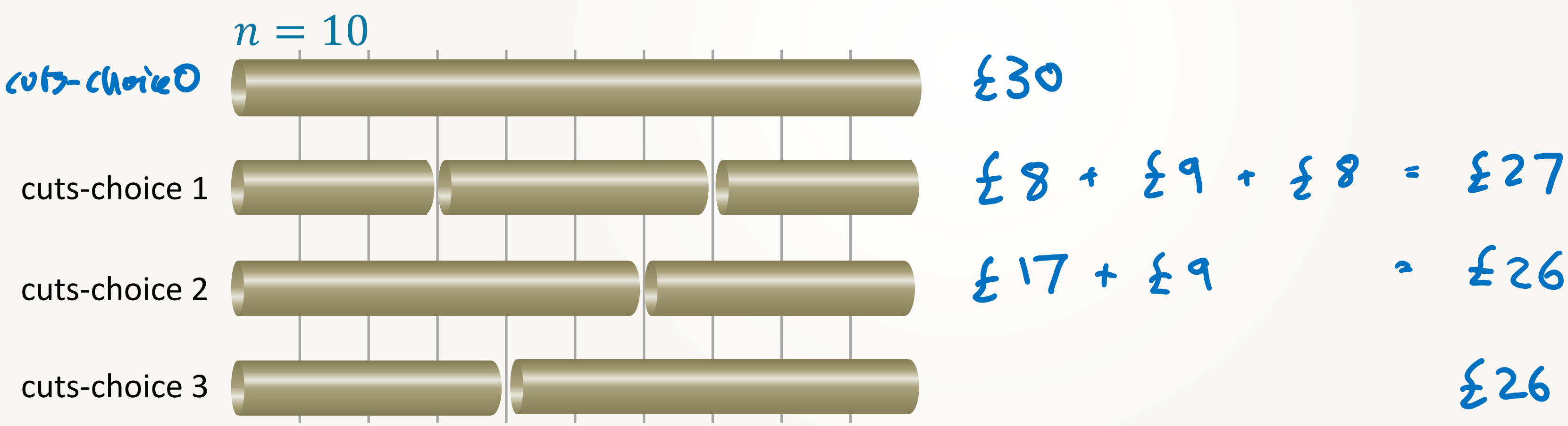
$$v(x) = \begin{cases} \text{termreward}_x & \text{if } x \text{ is terminal} \\ \max_{a \in A} \{\text{reward}_{x,a} + v(\text{nextstate}_{x,a})\} & \text{otherwise} \end{cases}$$

Example: rod cutting

A DIY supplier has a steel rod of length $n \in \mathbb{N}$, and a machine that can cut it into smaller pieces. Different lengths can be sold for different prices; a piece of length $\ell \in \mathbb{N}$ fetches p_ℓ.

How should it be cut, to maximize profit?

length	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
price	£1	£5	£8	£9	£10	£17	£17	£20	£24	£30



Example: rod cutting

A DIY supplier has a steel rod of length $n \in \mathbb{N}$, and a machine that can cut it into smaller pieces.
Different lengths can be sold for different prices; a piece of length $\ell \in \mathbb{N}$ fetches p_ℓ.

How should it be cut, to maximize profit?

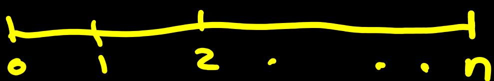
let $v(n)$ = max profit I can achieve from a rod of length n. Then

$$v(1) = p_1$$

$n \geq 2$: $$v(n) = p_n \vee \max_{1 \leq i \leq n-1} \{ p_i + v(n-i) \}$$

$\vee \equiv \max$

$\wedge \equiv \min$



Alternatively:

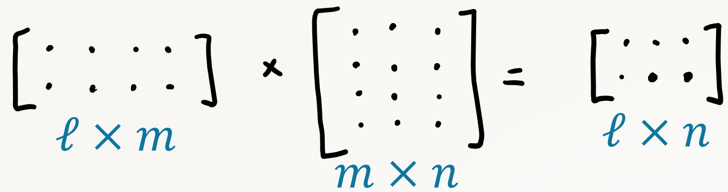
$$v(n) = \begin{cases} 0 & \text{if } n = 0 \\ \max_{1 \leq i \leq n} \{ p_i + v(n-i) \} & \text{if } n > 0. \end{cases}$$

Sanity check:

$$v(0) = 0$$
$$v(1) = p_1 + v(0) = p_1$$
$$v(2) = (p_1 + v(1)) \vee (p_2 + v(0)) = (p_1 + p_1) \vee p_2$$

Example 3.1.1 Matrix chain multiplication

The cost of multiplying two matrices depends on their dimensions:



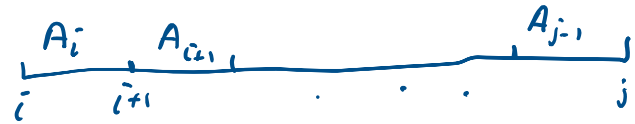
ℓmn multiplications $+ \ell(m-1)n$ additions
For simplicity, let's take the total cost to be ℓmn.

If we want to compute the product of several matrices, we have a choice about the order of multiplication (because matrix multiplication is associative). For example,

$$ABCDE = (AB)\big((CD)E\big) = A\Big(B\big((CD)E\big)\Big)$$

Find the least-cost way to compute the product $\underset{d_0 \times d_1}{A_0} \cdot \underset{d_1 \times d_2}{A_1} \cdot \cdots \cdot \underset{d_{n-1} \times d_n}{A_{n-1}}$

let $v(i,j)$ = min cost of multiplying $A_i A_{i+1} \cdots A_{j-1}$.

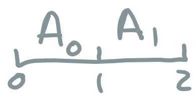


We want to find $v(0,n)$.

Try out some simple cases:

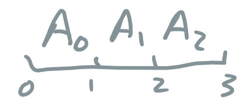
$v(0,1) = 0$ (since we don't need to do any work to compute A_0 — it's right there.)

$v(0,2) = d_0 d_1 d_2$ (



cost is $d_0 d_1 d_2$)

$d_0 \times d_1$ $\quad d_1 \times d_2$



$$v(0,3) = \min \{ d_0 d_1 d_3 + v(1,3),\ d_0 d_2 d_3 + v(0,2) \}$$

$A_0 (A_1 A_2)$
costs $v(1,3)$
produces a $d_1 \times d_3$ matrix
extra cost $d_0 d_1 d_3$

$(A_0 A_1) A_2$
costs $v(0,2)$
produces a $d_0 \times d_2$ matrix
extra cost $d_0 d_2 d_3$

General case:

$$v(i,j) = \begin{cases} \text{if } j = i+1: & 0 \\ \text{if } j = i+2: & d_i d_{i+1} d_{i+2} \\ \text{if } j \geq i+2: & \min\limits_{i<k<j} \{ d_i d_k d_j + v(i,k) + v(k,j) \} \end{cases}$$

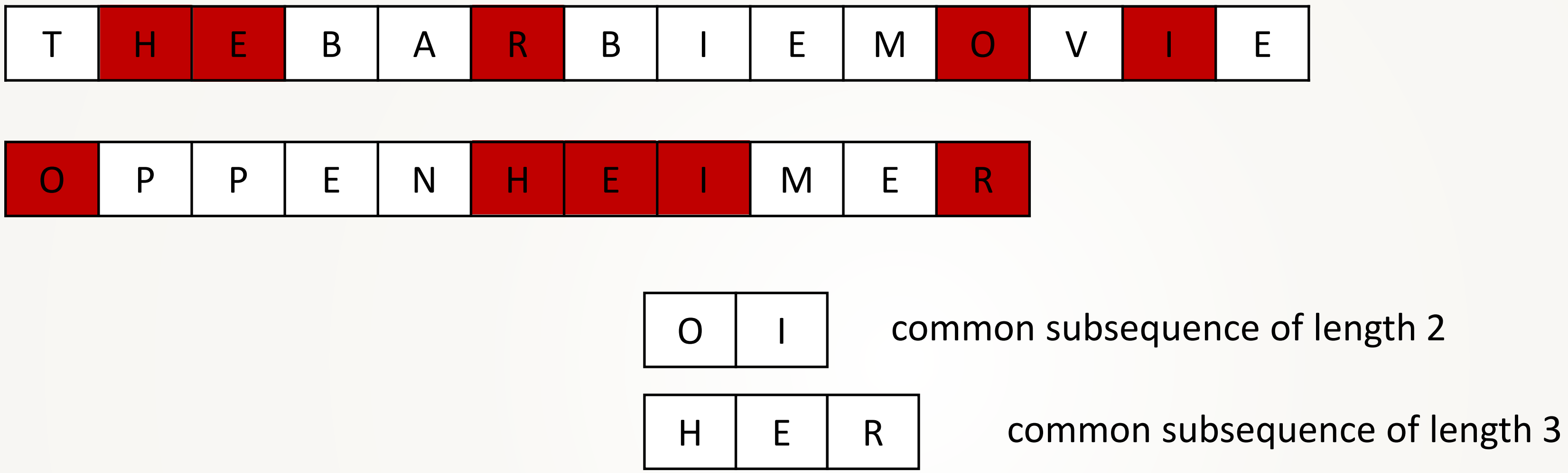
~~if $j=i+2$: $d_i d_{i+1} d_{i+2}$~~ Actually this is just a special case of the third equation.

($d_i \times d_k$ matrix)($d_k \times d_j$ matrix)

and we can choose any bracketing point $k \in \{i+1, \dots, j-1\}$.

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?

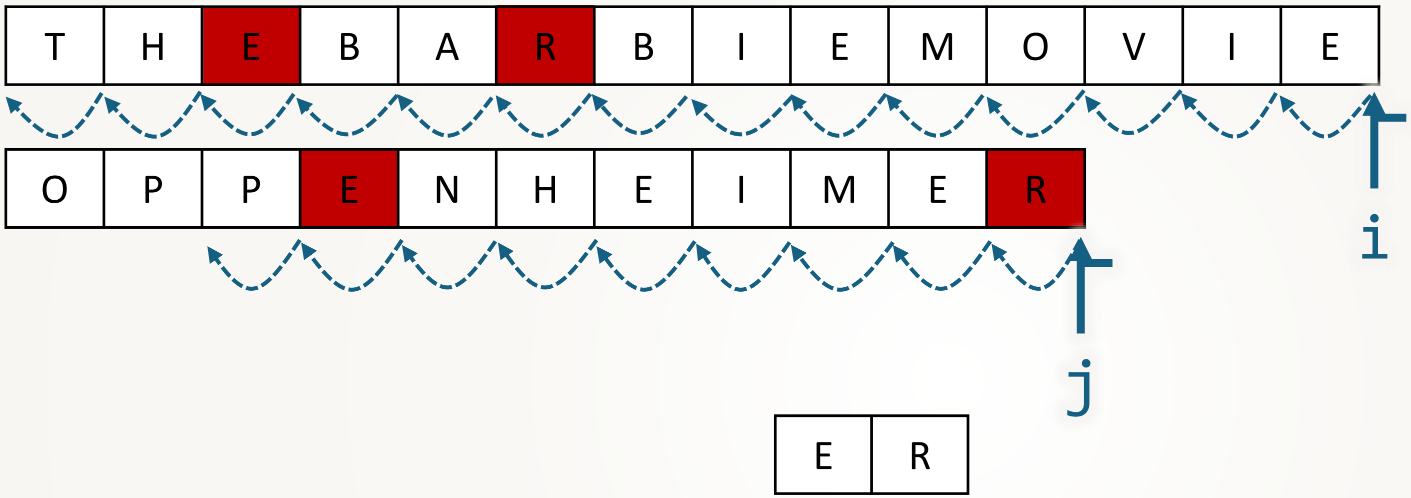


let $v(s,t)$ = length of longest common subsequence between s and t.

$$v(s,t) = \begin{cases} 0 & \text{if they have no chars in common} \\ \max\limits_{\substack{0 \le i < \text{len}(s) \\ 0 \le j < \text{len}(t) \,:\, s[i]=t[j]}} \left\{ 1 + v\left(s[i+1:], t[j+1:] \right) \right\} \end{cases}$$

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

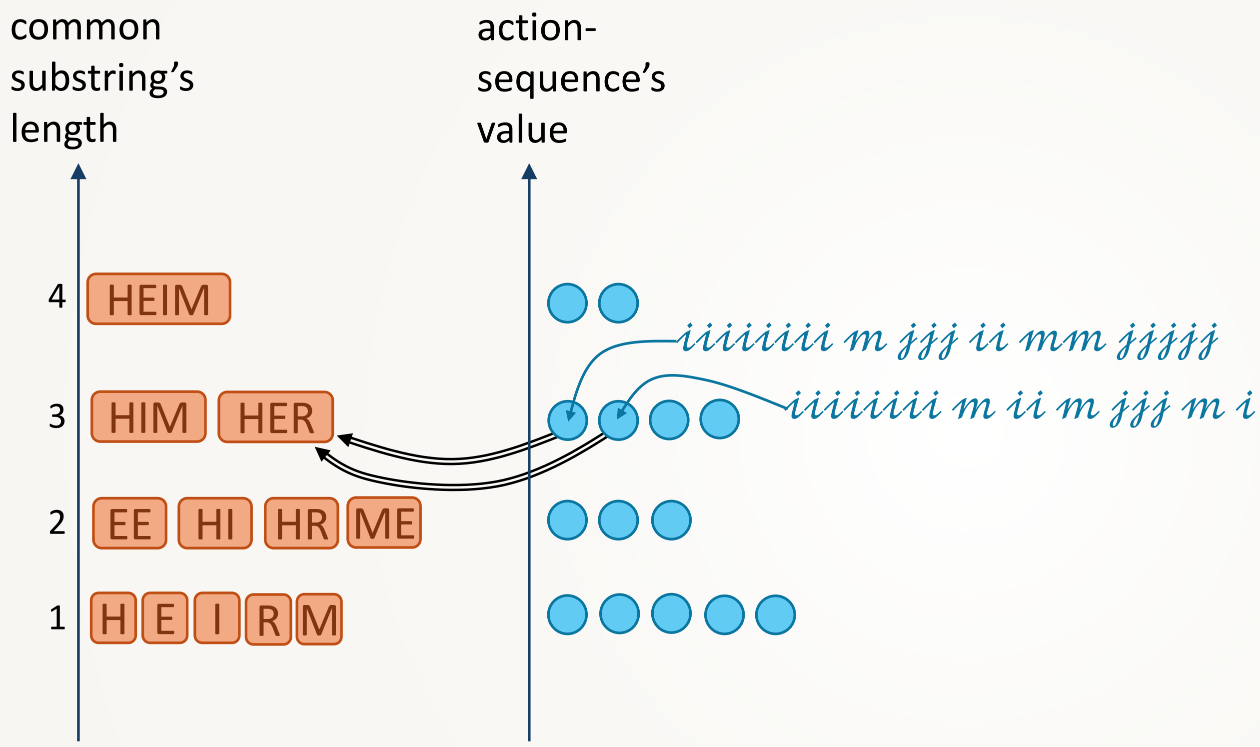
$v_{i,j}$ = length of LCS in $s[0:i]$ and $t[0:j]$

we want to find $v(\text{len}(s), \text{len}(t))$

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

The Translation strategy for designing algorithms

It's up to us how to translate the problem into "choose a sequence of actions".
How can we make sure that our translation is legitimate?



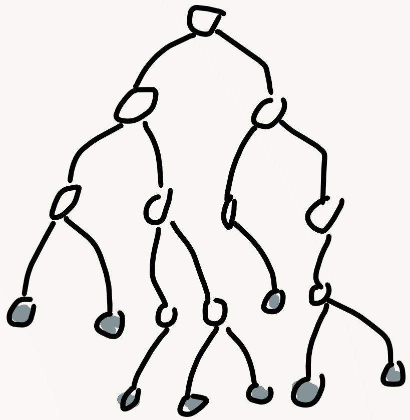
Searching for the highest-value action sequence will find us the longest common substring, when ...

1. Every common substring can be achieved through some valid action sequence, and every action sequence produces a common substring.
2. The higher the value of the action sequence, the longer its corresponding common substring.

Extracting the programme

We've seen how to compute the *value* $v(x)$ of the optimal programme (i.e. action sequence) starting from state x:

```
# The Bellman recursion
def v(x):
  if is_terminal(x):
    return terminal_reward(x)
  else:
    return max(reward(x,a) + v(nextstate(x,a)) for a in ACTIONS)
```



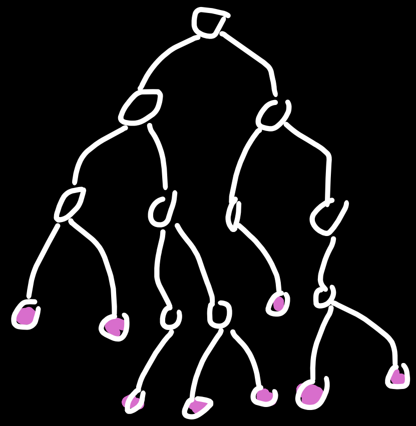
QUESTION
What's the extra memory cost of extracting the programme, for a tree of height h?

If we also want to *extract* an optimal programme,

```
def vp(x):
  # return a pair with optimal (value, programme)
  if is_terminal(x):
    return terminal_reward(x), []
  else:
    children = [vp(nextstate(x,a)) for a in ACTIONS]
    vals,progs = zip(*children)
    vals = [reward(x,a) + v for a,v in zip(ACTIONS,vals)]
    imax = index of max item in vals
    return vals[imax], progs[imax] with ACTIONS[imax] prepended
```

What can go wrong?

The running time of naïve recursion is typically exponential in the size of the problem, making it impractical for all but the smallest problems.



Workarounds

- IA Algorithms: look at problems with a special "overlapping" structure that permits efficient solution.
- IB Artificial Intelligence: branch-and-bound, backtracking
- MPhil: Reinforcement learning, to approximate the value function
- Maths tripos: analytical methods for approximating large problems