SECTION 3

Algorithm design
Is it worth doing cardio?

It looks like maybe

\[
\text{heart rate} = \frac{\text{const}}{\text{lifespan}}
\]

\[\Rightarrow \text{heartrate} \times \text{lifespan} = \text{const}\]

\[\Rightarrow \text{total lifetime heartbeats} = \text{const}\]

Equivalently,

\[
\log(\text{heart rate}) + \log(\text{lifespan}) = \text{const}
\]

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)
It looks like maybe

\[ \text{heart rate} = \frac{\text{const}}{\text{lifespan}} \]

\[ \Rightarrow \text{heartrate} \times \text{lifespan} = \text{const} \]

\[ \Rightarrow \text{total lifetime heartbeats} = \text{const} \]

Equivalently,

\[ \log(\text{heartrate}) + \log(\text{lifespan}) = \text{const} \]

and this is easier to see on a log-log plot.
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 heart rate, but it burns through our lifetime heartbeats.)
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 $[a_0, a_1, ...]$. If we're in state $x$ and we take action $a$, we gain reward $\text{reward}_{x,a}$, and move to nextstate $\text{nextstate}_{x,a}$ (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

$$v(x) = \begin{cases} \text{termreward}_x & \text{if } x \text{ is terminal} \\ \max_{a \in A} \{\text{reward}_{x,a} + v(\text{nextstate}_{x,a})\} & \text{otherwise} \end{cases}$$
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 $[a_0, a_1, ...]$. If we’re in state $x$ and we take action $a$, we gain reward $r_{x,a}$ and move to next state $x_{next}$ (unless $x$ is a terminal state, where no further actions are possible, in which case we gain term reward $r_{term,x}$). We want to find the solution.

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

$$v(x) = \begin{cases} r_{term,x} & \text{if } x \text{ is terminal} \\ \max_{a \in A} \left\{ r_{x,a} + v(x_{next}) \right\} & \text{otherwise} \end{cases}$$

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?
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 \( p_\ell \).

How should it be cut, to maximize profit?

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>£1</td>
<td>£5</td>
<td>£8</td>
<td>£9</td>
<td>£10</td>
<td>£17</td>
<td>£17</td>
<td>£20</td>
<td>£24</td>
<td>£30</td>
</tr>
</tbody>
</table>

- \( n = 10 \)
  - \( c_{\text{cuts-choice } 0} \)  
  - \( c_{\text{cuts-choice } 1} \)  
  - \( c_{\text{cuts-choice } 2} \)  
  - \( c_{\text{cuts-choice } 3} \)  

\[
\begin{align*}
\text{cuts-choice } 0 & : £30 \\
\text{cuts-choice } 1 & : £8 + £9 + £8 = £27 \\
\text{cuts-choice } 2 & : £17 + £9 = £26 \\
\text{cuts-choice } 3 & : £26
\end{align*}
\]
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 $p_\ell$.

How should it be cut, to maximize profit?

Let $v(n)$ be the maximum profit I can achieve from a rod of length $n$. Then

$$v(1) = p_1,$$

and:

$$v(n) = p_n \lor \max_{1 \leq i \leq n-1} \left\{ p_i + v(n-i) \right\} \quad \forall n \geq 2.$$ 

A (recursive) solution:

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

Sanity check:

$$v(0) = 0,$$

$$v(1) = p_1 + v(0) = p_1,$$

$$v(2) = \left( p_1 + v(1) \right) \lor \left( p_2 + v(0) \right) = (p_1 + p_1) \lor p_2.$$
Example 3.1.1  Matrix chain multiplication

The cost of multiplying two matrices depends on their dimensions:

\[
\begin{bmatrix}
\vdots & \vdots & \cdots \\
\ell & \ell & \ell \\
\end{bmatrix} \times \begin{bmatrix}
\vdots & \vdots & \cdots \\
m & m & m \\
\end{bmatrix} = \begin{bmatrix}
\vdots & \vdots & \cdots \\
\ell & \ell & \ell \\
\end{bmatrix}
\]

\(\ell mn\) multiplications + \(\ell(m - 1)n\) additions

For simplicity, let’s take the total cost to be \(\ell mn\).

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,

\[
ABCD = (AB)((CD)E) = A \left(B((CD)E)\right)
\]

Find the least-cost way to compute the product

\[
A_0 \cdot A_1 \cdot \cdots \cdot A_{n-1}
\]

\[
d_0 \times d_1 \\ d_1 \times d_2 \\ \vdots \\ d_{n-1} \times d_n
\]
Let \( v(i,j) = \min \text{ cost } \& \text{ multiplying } A_i A_{i+1} \ldots A_{j-1} \).

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

Try out some simple cases:

\[
v(0,1) = 0 \quad \text{(since } A_0 \text{ - it's right there,)}
\]

\[
v(0,2) = d_0 d_1 d_2 \quad \left( \begin{array}{c}
A_0 & A_1 \\
0 & 1 & 2
\end{array} \right) \quad \text{(cost is } d_0 d_1 d_2) \]

\[
v(0,3) = \min \left\{ d_0 d_1 d_3 + v(1,3), \ d_0 d_2 d_3 + v(0,2) \right\}
\]

\[
\begin{array}{c}
A_0 (A_1 A_2) \\
\text{costs } v(1,3) \\
\text{produces a } d_1 \times d_3 \text{ matrix}
\end{array}
\quad \begin{array}{c}
(A_0 A_1) A_2 \\
\text{costs } v(0,2) \\
\text{produces a } d_1 \times d_2 \text{ matrix}
\end{array}
\quad \begin{array}{c}
\text{extra cost } d_0 d_1 d_3
\end{array}
\]

**General case:**

\[
v(i,j) = \begin{cases} 
0 & \text{if } j = i+1, \\
d_i d_{i+1} d_{i+2} & \text{if } j = i+2, \\
\min_{i < k < j} \left\{ d_i \cdot d_k \cdot d_j + v(i,k) + v(k,j) \right\} & \text{if } j > i+2.
\end{cases}
\]

Actually this is just a special case of the third equation.

\[
\begin{array}{c}
(d_i \times d_k \text{ matrix}) (d_k \times d_j \text{ matrix})
\end{array}
\]

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

$$v(s, t) = \begin{cases} 
0 & \text{if they have no chars in common} \\
\max_{0 \leq i < \text{len}(s)} \max_{0 \leq j < \text{len}(t)} \begin{cases} 
1 + v(s[i+1:], t[j+1:]) \\
0 & s[i] = t[j] 
\end{cases} & \text{otherwise}
\end{cases}$$
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 $i$ & $j$

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

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

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

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.

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)
```

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
```

**QUESTION**

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