<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You lose!
long fib_rec(int n) {
    if (n < 2) {
        return n;
    } else {
        return fib_rec(n - 1) + fib_rec(n - 2);
    }
}

Count(n)

How many recursive calls to compute fib(n)?
for n = 0 or n = 1, call count(0) = count(1) = 1
n > 1, count(n) = 1 + count(n-1) + count(n-2)

\[
2^{60} \approx 10^{18.6}, n = 60 \text{ no } \sim 10 \text{ hours}
\]
\[
\text{Human} \approx 2^{50} \sim 2^{50} \sim 1 \text{ hour}
\]

For large values of n, the recursive call stack becomes very large, leading to a time complexity of \(O(2^n)\), which is exponential.
Stamps

14, 23 & 37 stamps (unlimited supply of each)

Make 50 using fewest possible stamps

\[ 50 = 37 + \frac{1 + 1 + \cdots + 1}{13} \text{ stamps} \]
\[ = 23 + \frac{1 + 1 + 1}{6} \text{ stamps} \]

best way to make 27 in postage

In general: if \( S_1, S_2, \ldots, S_k \) is shortest list with \( i \in \{1, 2, 3\} \) and \( \sum i = n \)

then \( S_1, \ldots, S_{k-1} \) is shortest list with sum \( n-S_k \)

optimal substructure

\[ \text{fewest}(n) = \text{fewest stamps to make n cents postage} \]

\[ = \begin{cases} 
0 & \text{if } n = 0 \\
\min_{v_i \in V} \left( 1 + \text{fewest}(n-v_i) \right) & \text{if opt soln for } n \text{ uses } v_i \text{ then}
\end{cases} \]

if opt soln for \( n \) uses \( v_i \) then

opt soln for \( n \) = \( 1 + \) opt soln for \( n-v_i \) (opt substructure)

but don't know which \( v_i \) opt soln for \( n \) uses

try each one and see which is best
Longest Path in a DAG

Simple

EDBCA

D: 1+1
E: 1+0
B: 1+2
C: 1+3
A: 1+4

\text{longest path \( C \rightarrow E \)}

\text{optimal substructure}

\text{longest \( A \rightarrow E \ ACBDE \)}

v_i \text{ to } v_k

= v_i \rightarrow v_x + \text{longest path } v_x \rightarrow v_k

\text{don't know what } v_x \text{ is, so try every possibility}

\text{used to order vertices so that when working on } v_i, \text{ already done with } v_j \text{ for edges } v_i \rightarrow v_j

\text{topological sort!}

0(n+m) \text{ with adj list}

\begin{align*}
\text{longest } [v] &= 0, \\
\text{for each vertex } v_i \neq \text{ to in order of topo sort} & \text{ long } = -\infty, \\
\text{for each edge } v_i \rightarrow v_j & \text{ long } \leftarrow \max(\text{ long, } 1 + \text{ longest } [v_i])
\end{align*}

\text{longest } [v_i] \leftarrow \text{ long}

\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (2,0) {B};
\node (c) at (4,0) {C};
\node (d) at (6,0) {D};
\node (e) at (8,0) {E};
\draw (a) -- (b);
\draw (b) -- (c);
\draw (c) -- (d);
\draw (d) -- (e);
\draw (e) -- (a);
\end{tikzpicture}

doesn't work if there are cycles

\text{longest path } D \rightarrow 1 \neq \text{ longest of } [D \rightarrow v_j + \text{ longest } v_j \rightarrow v_k] \text{ over all } v_j
get list of all states

record 000 as W

for each state s in list
  get list of successor states
  if all successors W
    record s as L
  else
    record s as W
1-2-3 Nim

Take 1, 2, or 3 sticks
Last stick wins

\[
\text{win}(n) = \text{whether there is a winning strategy for you if you start your turn with } n \text{ sticks}
\]

= 

\[
\begin{array}{ccccccccccccc}
\text{n=0} & \text{n=1} & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\end{array}
\]