Dynamic Programming

Longest palindromic sequence
Optimal binary search trees
Alternating coin game

DP notions

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution based on optimal solutions of subproblems
3. Compute the value of an optimal solution in bottom-up fashion (recursion & memoization).
4. Construct an optimal solution from the computed information
Longest Palindromic Sequence

Def: A palindrome is a string that is unchanged when reversed

Examples: radar, civic, t, bb, redder

Given: A string $x[1...n]$, $n \geq 1$

To find: Longest palindrome that is a subsequence

Example: Given "character"

Output: " cará "

Answer will be $\geq 1$ in length

Strategy

$L(i, j)$: length of longest palindromic subsequence of $x[i...j]$ for $i \leq j$

```python
def L(i, j):
    if i == j:
        return 1
    if x[i] == x[j]:
        if l+1 == j:
            return 2
        else:
            return 2 + L(l+1, j-1)
    else:
        return max(L(i+1, j), L(i, j-1))
```

Exercise: Compute the actual solution
**Analysis**

As written, program can run in exponential time: suppose all symbols $X[i]$ are distinct

$$T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
2T(n-1) & \text{if } n > 1 
\end{cases}$$

$$T(n) = 2^{n-1}$$

**Subproblems**

But there are only $\binom{n}{2} = \Theta(n^2)$ distinct subproblems; each is a $(i, j)$ pair with $i < j$.

Also have problems of size 1.

By solving each subproblem only once, running time reduces to

$$\Theta(n^2) \cdot \Theta(1) = \Theta(n^2)$$

*Number of subproblems* + time to solve subproblem, given that smaller ones are solved.

Memoize $L(i, j)$, hash inputs to get output value, and look up hashtable to see if the subproblem is already solved, else recurse.
Memoizing Vs. Iterating

(1) Memoizing uses a dictionary for \( L(i,j) \)
where value of \( L \) is looked up by
using \( i,j \) as a key. Could just use a
2-D array here where null entries signify
that the problem has not yet been solved.
(2) Can solve subproblems in order of increasing
\( j-i \) so smaller ones are solved first.

Optimal Binary Search Trees: CLRS 15.5

Given: keys \( k_1, k_2, \ldots, k_n \) \( k_1 < k_2 < \ldots < k_n \)
weights \( w_1, w_2, \ldots, w_n \) (search probabilities)

Find: \( \text{BST } T \) that minimizes:

\[
\sum_{i=1}^{n} w_i \cdot (\text{depth}_T(k_i) + 1)
\]

Example: \( w_i = p_i = \) probability of searching
for \( k_i \)

Then, we are minimizing expected
search cost.

(say we are representing an English \( \rightarrow \) French dictionary
and common words should have greater weight.)
Enumeration

Exponentially many trees

\[
\begin{align*}
\text{n = 2} & : 1 \rightarrow 2 \\
\quad & : w_1 + 2w_2 \\
\text{n = 3} & : 1 \rightarrow 2 \rightarrow 3 \\
\quad & : 3w_1 + 2w_2 + w_3 \\
\end{align*}
\]

Strategy

\[
\begin{align*}
w(i, j) &= w_i + w_{i+1} + \ldots + w_j \\
e(i, j) &= \text{cost of optimal BST on } K_i, K_{i+1}, \ldots, K_j \\
\text{Want } e(1, n) \\
\end{align*}
\]

Greedy solution?

Pick \( K_r \) in some greedy fashion, e.g., \( w_r \) is maximum

Greedy doesn't work

\[
\begin{align*}
\text{keys } K_i \ldots K_{r-1} & : e(1, r-1) \\
\text{keys } K_{r+1} \ldots K_j & : e(r+1, j) \\
\end{align*}
\]
DP Strategy: Guess all roots

\[ e(i,j) = \begin{cases} 
  w_i & \text{if } i = j \\
  \min_{i \leq r < j} \left( e(i, r-1) + e(r+1, j) + w(i,j) \right) & \text{else}
\end{cases} \]

+ \( w(i,j) \) accounts for the WR of root \( K_r \) as well as the increase in depth by 1 of all the other keys in the subtrees of \( K_r \).

(DP tries all ways of making local choice & takes advantage of overlapping subproblems.)

Complexity: \( \Theta(n^2) \cdot \Theta(n) = \Theta(n^3) \)

\( 1 \leq i \leq j \leq n \)

\( \binom{n}{2} \) subproblems

Taking the min from \( i \) to \( j \)
Alternating Coin Game

Row of $n$ coins of values $V_1, \ldots, V_n$ 
In each turn, a player selects either the first or last coin from the row, removes it permanently, and receives the value of the coin.

Question: Can the first player always win?

Try: 4 42 39 17 25 6

Strategy: $V_1, V_2, V_3, V_4, \ldots, V_{n-2}, V_{n-1}, V_n$

1) Compare $V_1 + V_3 + \ldots + V_{n-1}$ against $V_2 + V_4 + \ldots + V_n$
   And pick whichever is greater.
2) During the game only pick from the chosen subset (you will always be able to!)

How to maximize the amount of money won assuming you move first?
Optimal Strategy

\[ V(i, j) : \max \text{ value we can definitely win if it is our turn and only coins } v_i \ldots v_j \text{ remain} \]

\[ V(i, i) \quad V(i, i+1) \quad V(i, i+2) \quad V(i, i+3) \ldots \]

for all values of \( i \)

Just pick \( i \)

pick the maximum of the two

\[
V(i, j) = \max \left\{ \begin{array}{l}
\text{\langle range becomes } (i+1, j) \rangle \quad + \quad V_i \\
\text{pick } V_i \\
\text{\langle range becomes } (i, j-1) \rangle \quad + \quad V_j \\
\text{pick } V_j
\end{array} \right\}
\]
Solution

\( V(i+1, j) \) subproblem with opponent picking

\[ \Rightarrow \text{we are guaranteed } \min \{ V(i+1, j-1), V(i+2, j) \} \]

Opponent picks \( V_j \)

Opponent picks \( V_{i+1} \)

We have:

\[ V(i, j) = \max \left\{ \min \left\{ \frac{V(i+1, j-1)}{V(i+2, j)} \right\} + V_i, \min \left\{ \frac{V(i+2, j-2)}{V(i+1, j-1)} \right\} + V_j \right\} \]

Complexity?

\[ \Theta(n^2) \cdot \Theta(1) = \Theta(n^2) \]

\[ \frac{\text{# subproblems}}{\text{time per subproblem}} \]
Example of Greedy Failing for Optimal BST problem

Thanks to Nick Davis!

Choosing highest weight key of 2 as root doesn't work.

Cost = \(1 \times 2 + 10 \times 1 + 8 \times 2 + 9 \times 3\) = 55

Cost = \(1 \times 3 + 10 \times 2 + 8 \times 1 + 9 \times 2\) = 49