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 "c h a r a c t e r"

output "c a r a c"

Answer will be ≥ 1 in length

Strategy

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

```
def L(i,j):
    if i == j : return 1
    if X[i] == X[j] :
        if i+1 == j : return 2
        else : return 2 + L(i+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

$T(n)$ = running time on input of length n

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

$$= 2^{n-1}$$

Subproblems

But there are only $\binom{n}{2} = \Theta(n^2)$ distinct subproblems: each is an (i,j) pair with $i<j$
→ also have problems of size 1

By solving each subproblem only once, running time reduces to

$$\underbrace{\Theta(n^2)}_{\#\text{ subproblems}} \cdot \underbrace{\Theta(1)}_{\text{time to solve subproblem, GIVEN that smaller ones are solved}} = \Theta(n^2)$$

memoize L(i,j), hash inputs to get output value, and look up hash table 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 $k_1, k_2, \ldots k_n$ $\quad k_1 < k_2 < \ldots k_n$ WLOG $k_i = i$

weights $w_1, w_2, \ldots w_n$ (search probabilities)

Find: BST T that minimizes:

$$\sum_{i=1}^{n} w_i \cdot \left(\text{depth}_T(k_i) + 1\right)$$

Example: $w_i = p_i =$ probability of searching for k_i

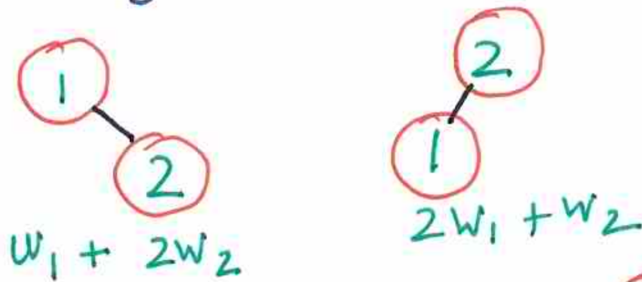
Then, we are minimizing expected search cost.

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

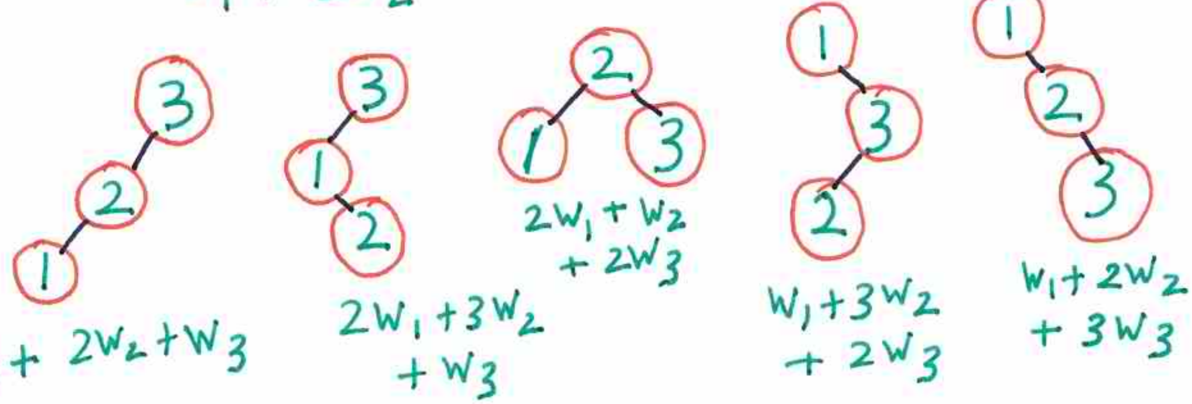
Enumeration

Exponentially many trees

n = 2



n = 3



Strategy

$$w(i,j) = w_i + w_{i+1} + \cdots w_j$$

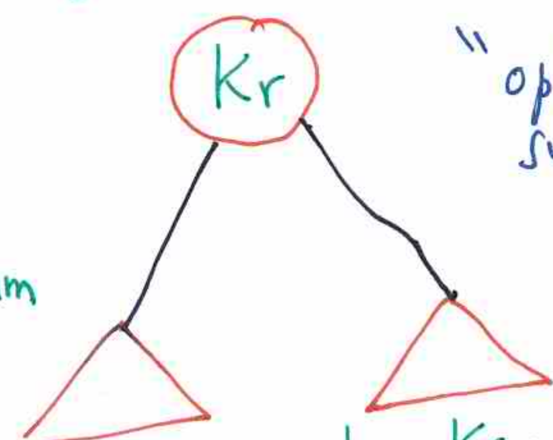
$$e(i,j) = \text{cost of optimal BST on } K_i, K_{i+1}, \ldots K_j$$

Want $e(1,n)$

Greedy solution?

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

greedy doesn't work



keys $K_i \ldots K_{r-1}$

$e(i, r-1)$

keys $K_{r+1}, \ldots K_j$

$e(r+1, j)$

"optimal substructure"

DP Strategy: Guess all roots

$$e(i,j) = \begin{cases} w_i & \text{if } i = j \\ \min\limits_{i \le r \le j} \left(e(i, r-1) + e(r+1, j) + w(i,j) \right) & \text{else} \end{cases}$$

$+\ w(i,j)$ accounts for w_r 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: $\underbrace{\Theta(n^2)}_{\#\text{ subproblems}} \cdot \underbrace{\Theta(n)}_{\text{time per subproblem}} = \Theta(n^3)$

$1 \le i \le j \le n$

$\binom{n}{2}$ subproblems

Taking the min from i to j

Alternating Coin Game

Row of n coins of values $V_1, \ldots, V_n$ n even

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:

$V_1 \quad V_2 \quad V_3 \quad V_4 \quad \ldots \quad V_{n-2} \quad V_{n-1} \quad V_n$

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

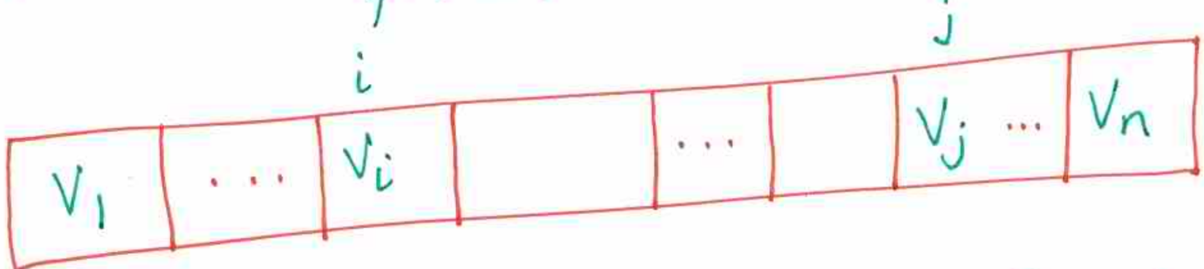
$V(i,j)$: max value we can definitely win if it is our turn and only coins $V_i \ldots V_j$ remain

$V(i,i)$ $V(i,i+1)$ $V(i,i+2)$ $V(i,i+3) \ldots$

for all values of i

$V(i,i)$ → just pick i

$V(i,i+1)$ → pick the maximum of the two



$$V(i,j) = \max \left\{ \underbrace{\langle \text{range becomes } (i+1,j) \rangle + V_i}_{\text{pick } V_i},\ \underbrace{\langle \text{range becomes } (i,j-1) \rangle + V_j}_{\text{pick } V_j} \right\}$$

Solution

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

$\Rightarrow$ we are guaranteed $\min\{\underbrace{V(i+1, j-1)}_{\text{Opponent picks } v_j}, \underbrace{V(i+2, j)}_{\text{opponent picks } v_{i+1}}\}$

We have:

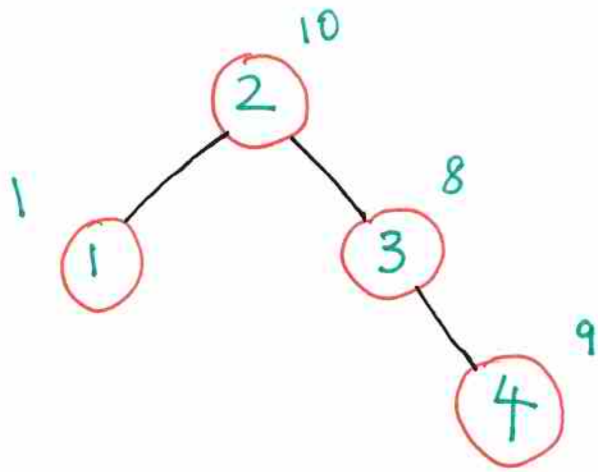
$$V(i, j) = \max\left\{\min\left\{\begin{matrix}V(i+1, j-1), \\ V(i+2, j)\end{matrix}\right\} + v_i \, , \, \min\left\{\begin{matrix}V(i, j-2), \\ V(i+1, j-1)\end{matrix}\right\} + v_j\right\}$$

Complexity?

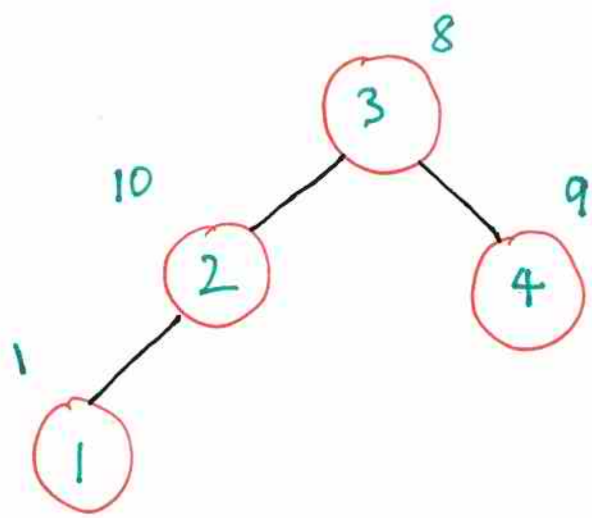
$$\underbrace{\Theta(n^2)}_{\#\text{ subproblems}} \cdot \underbrace{\Theta(1)}_{\substack{\text{time per} \\ \text{subproblem}}} = \Theta(n^2)$$

Example of Greedy Failing for Optimal BST problem

Thanks to Nick Davis!



$$\text{Cost} = \overset{w_1}{1} \times \overset{\text{depth}+1}{2} + \overset{w_2}{10} \times 1 + 8 \times 2 + 9 \times 3 = 55$$



$$\text{Cost} = \overset{w_1}{1} \times 3 + \overset{w_2}{10} \times 2 + \underset{w_3}{8} \times 1 + 9 \times 2 = 49$$

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