Lecture 10: Dynamic Programming

- Longest palindromic sequence
- Optimal binary search tree
- 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

Definition: A palindrome is a string that is unchanged when reversed.
Examples: radar, civic, t, bb, redder
Given: A string $X[1 \cdots n]$, $n \geq 1$
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 $X[i \cdots j]$ 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 $i < j$. By solving each subproblem only once. running time reduces to

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

where $\theta(n^2)$ is the number of subproblems and $\theta(1)$ is the time to solve each subproblem, given that smaller ones are solved.

Memoize $L(i, j)$, hash inputs to get output value, and lookup 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 $K_1, K_2, \cdots, K_n, K_1 < K_2 < \cdots < K_n$, WLOG $K_i = i$

weights $W_1, W_2, \cdots, W_n$

Find: BST T that minimizes:

$$\sum_{i=1}^{n} W_i \cdot (depth_T(K_i) + 1)$$

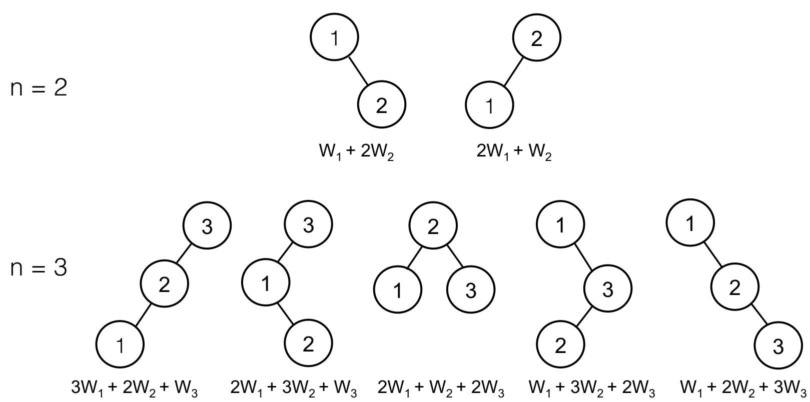
Example: $W_i = p_i =$ probability of searching for K_i

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



Strategy

$W(i,j) = W_i + W_{i+1} + \cdots + W_j$

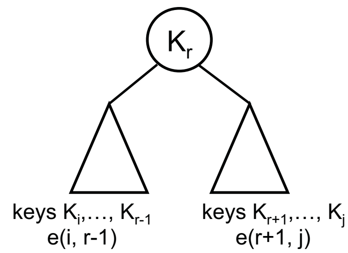
$e(i,j) =$ cost of optimal BST on $K_i, K_{i+1}, \cdots, K_j$

Want $e(1,n)$

Greedy solution?

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

greedy doesn't work, see example at the end of the notes.



DP Strategy: Guess all roots

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

$+W(i,j)$ 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 choices and takes advantage of overlapping subproblems)

Complexity: $\theta(n^2) \cdot \theta(n) = \theta(n^3)$

where $\theta(n^2)$ is the number of subproblems and $\theta(n)$ is the time per subproblem.

Alternating Coin Game

Row of n coins of values $V_1, \cdots, V_n$, n is 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

$V_1, V_2, \cdots, V_{n-1}, V_n$

1. Compare $V_1 + V_3 + \cdots + V_{n-1}$ against $V_2 + V_4 + \cdots + V_n$ 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 $V_i, \cdots, V_j$ remain.

$V(i,i)$: just pick i.
$V(i,i+1)$: pick the maximum of the two.
$V(i,i+2)$, $V(i,i+3), \cdots$

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

Solution

$V(i+1,j)$ subproblem with opponent picking
we are guaranteed $min\{V(i+1,j-1), V(i+2,j)\}$
Where $V(i+1,j-1)$ corresponds to opponent picking V_j and $V(i+2,j)$ corresponds to opponent picking V_{i+1}
We have

$$V(i,j) = max\left\{min\left\{\begin{array}{c} V(i+1,j-1), \\ V(i+2,j) \end{array}\right\} + V_i, min\left\{\begin{array}{c} V(i,j-2), \\ V(i+1,j-1) \end{array}\right\} + V_j\right\}$$

Complexity?

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

Example of Greedy Failing for Optimal BST problem

Thanks to Nick Davis!

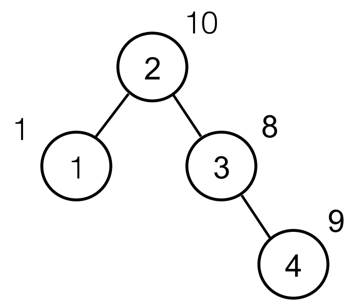


Figure 1: cost $= 1 \times 2 + 10 \times 1 + 8 \times 2 + 9 \times 3 = 55$

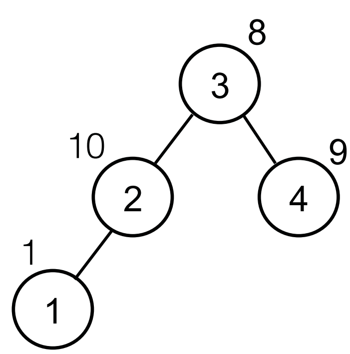


Figure 2: cost $= 1 \times 3 + 10 \times 2 + 8 \times 1 + 9 \times 2 = 49$