Admin:
- Quiz 1 on 10/18 (one "cheat sheet" allowed)
- extension policy for psets
- new location for office hours (see web site)

Today: Dynamic Programming
- Longest palindromic subsequence
- Optimal binary search trees
- Parsing for context-free grammar
- Parsing for probabilistic grammars (if time)

These problems deal with strings (sequences)
where natural subproblem structure involves substrings
Longest Palindromic Subsequence

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 “carac”

Note: answer has length \( \geq 1 \) always.

[Ref: Problem 15-2 pg 405 of CLRS]
Idea:

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

$$
def \text{L}(i, j):
\begin{cases}
    \text{if } i == j : \text{ return } 1 \\
    \text{if } X[i] \neq X[j] : \\
        \text{return max}(L(i+1, j), L(i, j-1)) \\
        \text{if } i + 1 == j : \text{ return } 2 \\
    \text{return } 2 + L(i+1, j-1)
\end{cases}
$$

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

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

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

$$= 2^{n-1}$$
But there are only $\binom{n}{2} = \Theta(n^2)$ distinct subproblems: each is an \((i, j)\) pair with \(i \leq j\).

By solving each subproblem only once, running time reduces to

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

Can either:

1. **Memoize** implementation above

   - E.g., hash input/output for \(L, \& \)

   before solving subproblem, check to see if it is already solved, with soln in hash table.

2. Solve subproblems in order of increasing “size” (\(j-i\)), so smaller ones solved first. (“bottom up”)

Common features to most DP problems:

- recursive soln is exponential, but only a polynomial \# of distinct subproblems
- memoization, or bottomup approach, gives polynomial running time, assuming each subproblem easy to solve given soln to smaller subproblems.
Optimal Binary Search Trees

Given: keys $K_1, K_2, ..., K_n$ ($K_i < K_{i+1}$) (wlog $K_i = i$)
Weights $w_1, w_2, ..., w_n$ (e.g. search probabilities)

Find: BST that minimizes "cost"

$$\sum_{i=1}^{n} w_i \cdot (\text{depth}_{T_i}(K_i) + 1)$$

(equivalently $\sum_{i=1}^{n} w_i \cdot \text{depth}_{T_i}(K_i)$, but we’ll use $\sum$)

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

Then we are minimizing expected search cost.

[Ref: §15.5 of CLRS]
Let \( b_n = \# \) binary trees on \( n \) keys

\[
\frac{1}{n+1} \binom{2n}{n} \quad \text{"Catalan numbers"}
\]

\[
\approx \Theta \left( \frac{4^n}{n^{3/2}} \right) \quad \text{exponential!}
\]

(see Prob. 12-4 CLRS pg 306)

Too many to enumerate unless \( n \) is small.

\[
\begin{align*}
\text{\( n = 2 \)} & \quad \begin{array}{c}
\text{1} \\
\text{2}
\end{array} & \quad \begin{array}{c}
\text{1} \\
\text{2}
\end{array} & \quad b_2 = 2 \\
\text{\( w_1 + 2w_2 \)} & \quad \text{\( 2w_1 + w_2 \)} & \quad \text{cost}
\end{align*}
\]

\[
\begin{align*}
\text{\( n = 3 \)} & \quad \begin{array}{c}
\text{1} \\
\text{2} \\
\text{3}
\end{array} & \quad \begin{array}{c}
\text{1} \\
\text{2} \\
\text{3}
\end{array} & \quad \begin{array}{c}
\text{1} \\
\text{2} \\
\text{3}
\end{array} & \quad b_3 = 5 \\
\text{\( 3w_1 + 2w_2 + w_3 \)} & \quad \text{\( 2w_1 + 3w_2 + w_3 \)} & \quad \text{\( w_1 + 3w_2 + 2w_3 \)} & \quad \text{\( w_1 + 2w_2 + 3w_3 \)}
\end{align*}
\]
Let $w(i, j) = w_i + w_{i+1} + \ldots + w_j \quad i \leq j$

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

(so $e(1, n)$ is what we are after)

Suppose we try to solve problem in greedy manner

1. Pick $K_r$ in some "greedy" manner (e.g. key with max weight $w_r$ ???)

2. Solve induced subproblems $e(i, r-1)$ & $e(r+1, j)$ optimally ("optimal substructure property")

But :-( we don't know how to do 1 !
Claim:
\[ e(i,j) = \begin{cases} 
  w_i & \text{if } i = j \\
  \min_{i \neq r \neq j} \left( e(i,r-1) + e(r+1,j) + w(i,j) \right) & \text{else}
\end{cases} \]

(The "\(+ w(i,j)\)" accounts for both searching for root \(K_r\), and fact that all other elements "pushed down a level" in overall problem, compared to subproblems—they all have now to compare against root \(K_r\).)

Greedy, when it works, knows how to make local choice in globally optimal manner. (i.e., pick \(K_r\).

Doesn't work here.

Dyn Prog tries all ways of making local choice, & takes advantage of "overlapping subproblems" via memoization to get poly-time algorithm.

Above recurrence can be solved in time \(\Theta(n^3)\):
\[
\frac{\Theta(n^2)}{\# \text{subproblems}} \cdot \frac{\Theta(n)}{\text{time per subproblem}} = \Theta(n^3)
\]
Parsing a string given context-free grammar (CFG):

terminal symbols: \( \Sigma = \{ a, b, c, \ldots \} \) letters, digits, special

non-terminal symbols: \( N = \{ A, B, C, \ldots \} \)

rules describing structure of strings in language

non-terminal \( \rightarrow \) given sequence of terminals & nonterminals

grammar = set of such rules
\( S = \Sigma \cup N \) (all symbols)

Example:

(Toy)
\[
\begin{align*}
A & \rightarrow b \\
A & \rightarrow aB \\
B & \rightarrow Bb \\
B & \rightarrow CC \\
C & \rightarrow AB \\
C & \rightarrow a \\
\end{align*}
\]

\( A \rightarrow aB \rightarrow aCC \rightarrow \ldots \rightarrow abaaa \)

parse tree of string abaaa of nonterminals showing how it can be produced, starting from non terminal A.

\( A \rightarrow abaa \)

"A derives abaaa"
Given: string $X[1..n]$ of symbols from $\Sigma$

CFG with nonterminals $N = \{A_1, \ldots, A_m\}$
and $r$ rules

$R = \Sigma \cup N$ symbols

Determine: if $A_i \rightarrow^* X[1..n]$

Let $P(i,j,v) =$

True if $(v = X[i] \land i = j)$ or

$(v \in N \land v \rightarrow^* X[i..j])$

$1 \leq i \leq j \leq n$

$v \in S$

$(\frac{n}{2}) \times |S|$ subproblems, we want $P(1,n,A_i)$
Solve recursively:

\[
\text{def } P(i, j, V) : \\
\begin{cases} \\
\text{if } V \in \Sigma : & \text{return } (i = j \& \& X[i] = V) \\
\text{for each rule } V \rightarrow Y & \text{return True} \\
\text{for each rule } V \rightarrow YZ : & \\
\text{for each } k \in \{i, \ldots, j-1\} & \text{if } P(i, k, Y) \& P(k+1, j, Z) : \text{return True} \\
\text{return False} \\
\end{cases}
\]

# subproblems = \( (\frac{n}{2}) \cdot |S| \)

time to solve each is \( \Theta(\frac{n^2}{2} \cdot r) \)

Total time is \( \Theta(n^3 \cdot |S| \cdot r) \)

(Note: exercise to massage grammar first so there are no loops of form \( A_1 \Rightarrow A_2 \Rightarrow A_3 \Rightarrow \cdots \Rightarrow A_7 \Rightarrow A_1 \Rightarrow \cdots \) (indeed, no RHS of rule with only single nonterminal)}
Probabilistic Grammars

- Suppose now each rule has a probability $p_i$.

- Probability of derivation = product of probs of steps

\[
\begin{align*}
A & \rightarrow b \quad \frac{1}{2} \\
A & \rightarrow aB \quad \frac{1}{2} \quad \rightarrow aB \quad \frac{1}{2} \\
B & \rightarrow Bb \quad \frac{3}{4} \quad \rightarrow aCC \quad \frac{1}{4} \\
B & \rightarrow CC \quad \frac{1}{4} \quad \rightarrow aABC \quad \frac{5}{8} \\
C & \rightarrow AB \quad \frac{5}{8} \quad \rightarrow abBC \quad \frac{1}{2} \\
C & \rightarrow a \quad \frac{3}{8} \quad \rightarrow abCC \quad \frac{1}{4} \\
& \quad \rightarrow abCC \quad \frac{3}{8} \quad \rightarrow a\ b\ a\ C \\
& \quad \rightarrow a\ b\ a\ a \\
\prod p_i &= 5.15 \times 10^{-4}
\end{align*}
\]

- What is derivation of maximum probability?

- What is total probability of all derivations?

Call desired function \( f \) for
Only minor changes to previous program needed. replace predicate $P(i, j, V)$ by $f(i, j, V)$ adjust program to take max, or sum, over rules. E.g. for total probability of all derivations:

```python
def f(i, j, V):
    sum = 0
    if V in Σ:
        if i == j & X[i] == V:
            return 1
        else:
            return 0
    for each rule $V \Rightarrow Y$:
        sum = sum + f(i, j, Y)
    for each rule $V \Rightarrow YZ$:
        for each $k \in \{i, \ldots, j-1\}$:
            sum = sum + f(i, k, Y) * f(k+1, j, Z)
    return sum
```

Note: special case is Viterbi alg., for regular languages: all rules have form $A \Rightarrow a$ or $A \Rightarrow aB$