Dynamic programming

Dynamic programming

- Characterize the structure of an optimal solution. (Define subproblem.)
- Recursively define the value of an optimal solution. (Express solutions to subproblems recursively.)
- Compute the value of an optimal solution (in right order.)
- Construct an optimal solution from computed information.

Chain matrix multiplication

- Suppose that we want to multiply four matrices, $A \times B \times C \times D$, of dimensions 50×20, 20×1, 1×10, and 10×100, respectively.
- How many ways can < A, B, C, D> be fully parenthesized?
  - (A(B(C D)))
  - (A((BC)D))
  - ((AB) (CD))
  - ((A(BC))D)
  - (((AB)C)D)

Matrix multiplication

```
MATRIX-MULTIPLY(A, B)
 if columns[A] ≠ rows[B]
    then error "imcompatible dimensions"
    else for  i←1 to rows[A]
              do for j←1 to columns[B]
                      do C[i,j] ← 0
                         for k←1 to columns[A]
                              do C[i,j] ← C[i,j] +A[i,k]·B[k,j]
          return C
```

A : $p \times q$ matrix, B : $q \times r$ matrix, C = AB : $p \times r$ matrix

Time to compute C : pqr

Chain matrix multiplication

- Suppose that we want to multiply four matrices, $A \times B \times C \times D$, of dimensions 50×20, 20×1, 1×10, and 10×100, respectively.

Parenthesization	Cost computation	Cost
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	$120,200$
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	$60,200$
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	$7,000$

- The order of multiplications makes a big difference in running time.
- Greedy approach (choose the cheapest multiplication) does not work. (case 2 above)
- Problem : How do we determine the optimal order, if we want to compute $A_1 \times A_2 \times \ldots \times A_n$ where the A_i's are matrices with dimensions $m_0 \times m_1$, $m_1 \times m_2$,…, $m_{n-1} \times m_n$, respectively?

Number of parentheses

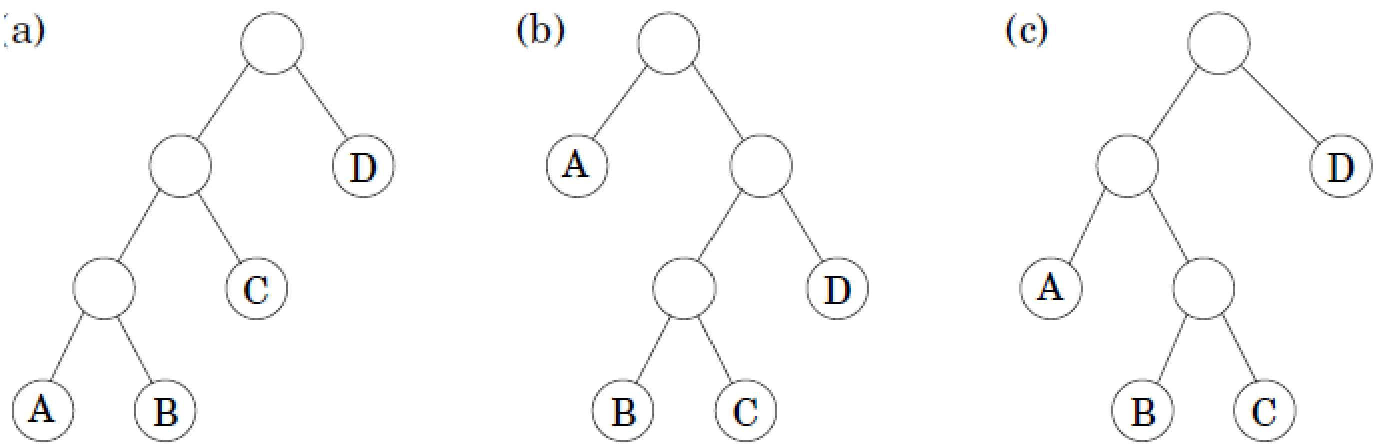
- $P(n)$ = # of alternative parenthesizations of a sequence of n matrices
- $n = 1 : 1$
- $n > 1$:
  - a fully parenthesized matrix product is the product of two fully parenthesized matrix subproducts
  - the split between the two subproducts may occur between the kth and $(k+1)$th matrices for any $k=1, 2, \ldots, n-1$.

$$P(n) = \begin{cases} 1 & \text{if } n = 1 , \\ \displaystyle\sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 . \end{cases}$$

Show that $P(n)$ is $\Omega(2^n)$ using substitution method.

- A particular parenthesization can be represented by a *binary tree*
  - the individual matrices correspond to the leaves
  - the root is the final product
  - intermediate nodes are intermediate products.
- The possible orders to do the multiplication = the various full binary trees with *n* leaves whose number is *exponential* in *n*.

(a) $((A \times B) \times C) \times D$; (b) $A \times ((B \times C) \times D)$; (c) $(A \times (B \times C)) \times D$.



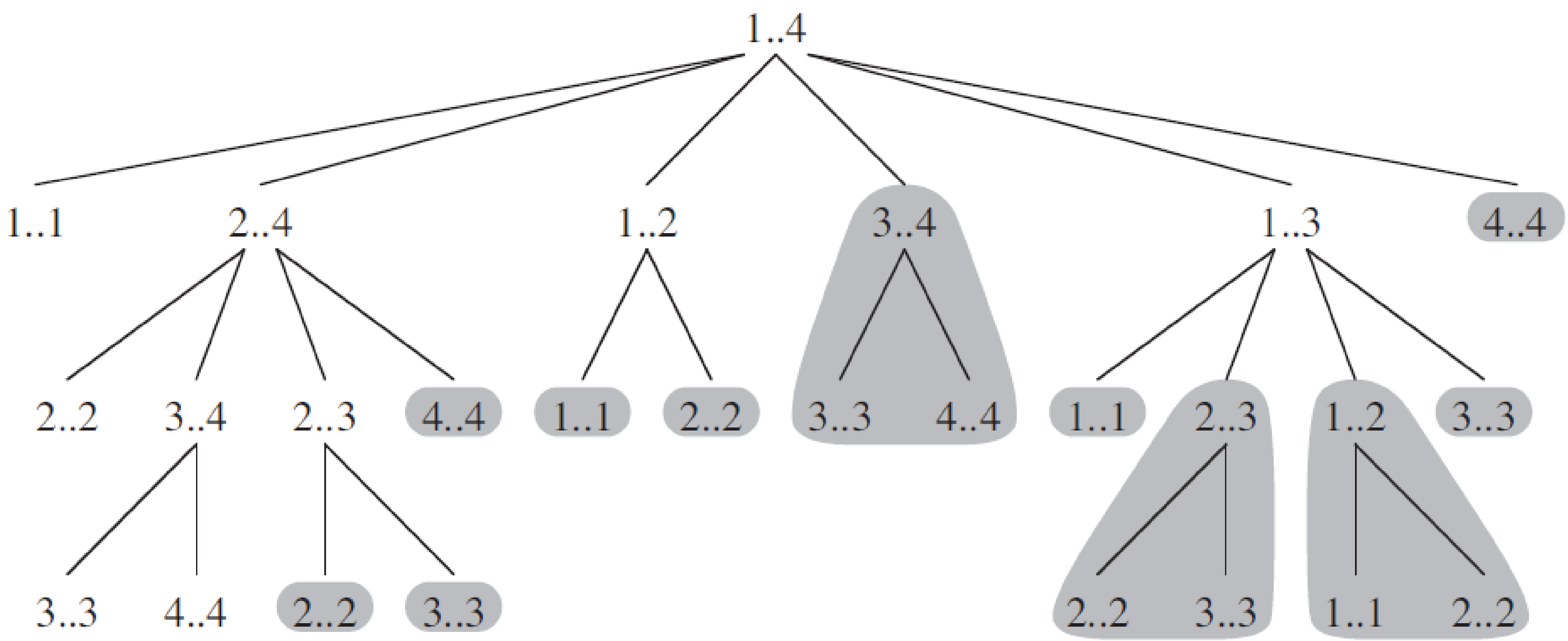
Optimal substructure

- For a tree to be optimal, its subtrees must also be optimal.
- What are the *subproblems* corresponding to the subtrees? – products of the form $A_i \times A_{i+1} \times \ldots \times A_j$
- Define $C(i, j)$ = minimum cost of multiplying $A_i \times A_{i+1} \times \ldots \times A_j$ for $1 \le i \le j \le n$.
- The size of the subproblem $C(i, j)$ = number of matrix multiplications $= j - i$
- Base case - the smallest subproblem : when $i = j$, $C(i, j) = 0$.

Optimal substructure

- For $j > i$, consider the optimal subtree for $C(i, j)$.
- Suppose that an optimal subtree of $A_i A_{i+1} \ldots A_j$ splits the product between A_k and A_{k+1}. ($i \leq k < j$)
- Then the subtrees of $A_i A_{i+1} \ldots A_k$ and $A_{k+1} A_{k+2} \ldots A_j$ within this optimal subtree of $A_i A_{i+1} \ldots A_j$ must be optimal subtrees of $A_i A_{i+1} \ldots A_k$ and $A_{k+1} A_{k+2} \ldots A_j$, respectively.
- The cost of the subtree is then the cost of these two partial products + the cost of combining them: $C(i, k) + C(k + 1, j) + m_{i-1} \cdot m_k \cdot m_j$
- Need to find the splitting point k for which this is smallest:
  $C(i, j) = \min\limits_{i \leq k < j} \{ C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j \}$

Recursion tree for C(1,4)



Algorithm

- s : problem size

$$
\begin{aligned}
&\texttt{for } i = 1 \texttt{ to } n\texttt{:} \quad C(i,i) = 0 \\
&\texttt{for } s = 1 \texttt{ to } n-1\texttt{:} \\
&\quad \texttt{for } i = 1 \texttt{ to } n-s\texttt{:} \\
&\qquad j = i + s \\
&\qquad C(i,j) = \min\{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j : i \leq k < j\} \\
&\texttt{return } C(1,n)
\end{aligned}
$$

- Two-dimensional table, each entry takes O(n) time : O(n^3) overall running time.
- How can we reconstruct the optimal parenthesization?

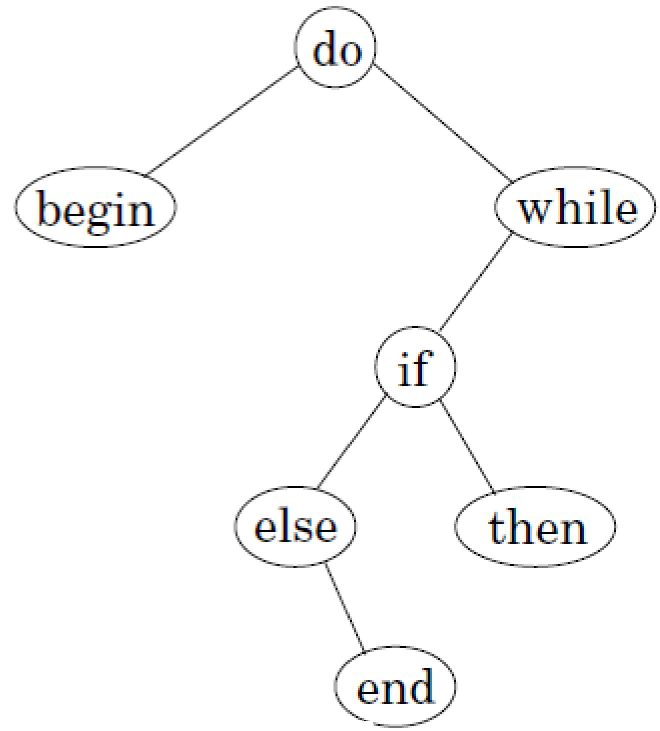
Optimal binary search trees

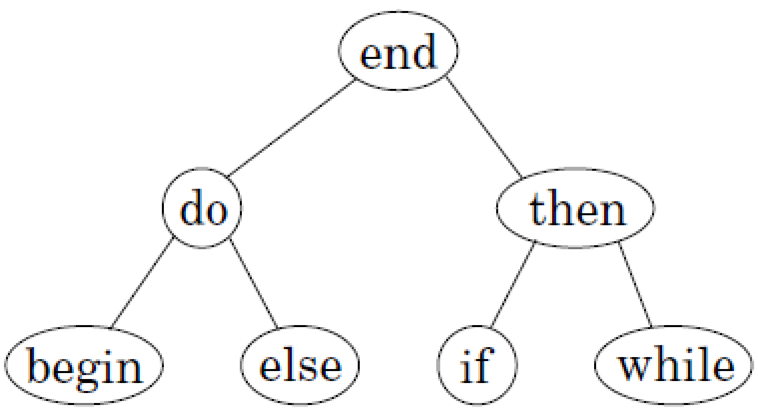
- Suppose we know the frequency with which keywords occur in programs of a certain language, for instance:

begin	5%
do	40%
else	8%
end	4%
if	10%
then	10%
while	23%

- We want to organize them in a *binary search tree*, so that the keyword in the root is alphabetically bigger than all the keywords in the left subtree and smaller than all the keywords in the right subtree.

begin	5%
do	40%
else	8%
end	4%
if	10%
then	10%
while	23%





$$\text{cost} = 1(0.04) + 2(0.40 + 0.10) + 3(0.05 + 0.08 + 0.10 + 0.23) = 2.42.$$

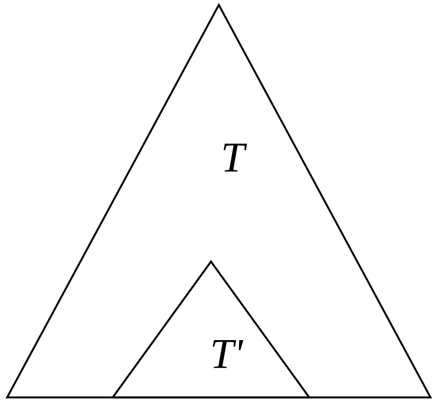
Optimal binary search trees

- Input: n keys (in sorted order); frequencies of these keys: $p_1, p_2, \ldots, p_n$.
- Output: The binary search tree of lowest cost (= the expected number of comparisons in looking up a key).

- Observations :
  - Optimal BST might not have smallest height
  - Optimal BST might not have highest-probability key at root.
- Build by exhaustive checking?
  - Construct each n-node BST
  - For each, put in keys
  - Then compute expected search cost
  - There are $\Omega(4^n / n^{3/2})$ different BSTs with n nodes.

Optimal substructure

- Consider any subtree of a BST. It contains keys in a contiguous range $k_i, \ldots, k_j$ for some $1 \leq i \leq j \leq n$.



If *T* is an optimal BST and *T* contains subtree *T'* with keys $ki, \ldots, kj$, then *T'* must be an optimal BST for $ki, \ldots, kj$.

Proof : cut and paste.

Using optimal substructure

- Given keys $k_i, \ldots, k_j$ (the problem)
- One of them, k_r ,where $i \leq r \leq j$, must be the root.
- Left subtree of k_r contains $k_i, \ldots, k_{r-1}$
- Right subtree of k_r contains $k_{r+1}, \ldots, k_j$

If we

- examine all candidate roots k_r , for $i \leq r \leq j$, and,
- determine all optimal BSTs containing $k_i, \ldots, k_{r-1}$ and containing $k_{r+1}, \ldots, k_j$

then we're guaranteed to find an optimal BST for $k_i, \ldots, k_j$

Recursive solution

Subproblem domain :

- Find optimal BST for $k_i, \ldots, k_j$, where $i \geq 1$, $i - 1 \leq j \leq n$
- When $j = i - 1$, the tree is empty.

Define $e[i,j]$ = expected search cost of optimal BST for $k_i, \ldots, k_j$

If $j = i - 1$, then $e[i,j] = 0$

If $j \geq i$,

- Select a root k_r, for some $i \leq r \leq j$
- make an optimal BST with $k_i, \ldots, k_{r-1}$ as the left subtree.
- make an optimal BST with $k_{r+1}, \ldots, k_j$ as the right subtree.

- When a subtree becomes a subtree of a node :
  - depth of every node in subtree goes up by 1.
  - expected search cost increases by $w(i, j) = \sum_{l=i}^{j} p_l$

Recursive solution

- If k_r is the root of an optimal BST for $k_i, \ldots, k_j$:
- $e[i,j] = p_r + (e[i,r-1] + w(i,r-1)) + (e[r+1,j] + w(r+1,j))$
- $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$
- $e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j)$

$$e[i,j] = \begin{cases} 0 & \text{if } j = i-1 \\ \min\limits_{i \le r \le j}\{ e[i,r-1] + e[r+1,j] + w(i,j) \} & \text{if } i \le j \end{cases}$$

Computing an optimal solution

- *Need 3 tables :*
  - $e[1..n+1, 0..n]$
  - $root[i, j]$: root of subtree with keys $k_i, \ldots, k_j$
  - $w[1..n+1, 0..n]$

```
OPTIMAL-BST(p, n)
for i=1 to n+1
      do e[i, i-1] = 0
           w[i, i-1] = 0
for l = 1 to n
      do for i = 1 to n-l+1
                 do j = i+l-1
                       e[i, j] = ∞
                       w[i, j] = w[i, j-1] + p_j
                       for r = i to j
                                do t = e[i, r-1] + e[r+1, j] + w[i, j]
                                     if t < e[i, j]
                                                then e[i, j] = t
                                                        root[i, j] = r
return e and root
```