

# Design and Analysis of Algorithms

## Dynamic Programming (II)

1 Chain Matrix Multiplication

2 Optimal Binary Search Tree

# Outline

1 Chain Matrix Multiplication

2 Optimal Binary Search Tree

## Chain Matrix Multiplication (矩阵链相乘)

Motivation. Suppose we want to multiply several matrices. This will involve iteratively multiplying two matrices at a time.

- Matrix multiplication is not *commutative* (in general  $A \times B \neq B \times A$ ), but it is *associative*:

$$A \times (B \times C) = (A \times B) \times C$$

- We can compute product of matrices in many different ways, depending on how we parenthesize it.

*Are some of these better than others?*

Complexity of  $C_{ik} = A_{ij} \times B_{jk}$

- Each element in  $C$  requires  $j$  multiplications, totally  $ik$  elements  $\Rightarrow$  overall complexity  $\Theta(ijk)$

## Example

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

Parenthesize	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

## Example

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

Parenthesize	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 multiplication order makes a big difference in the final complexity.

## Example

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

Parenthesize	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 multiplication order makes a big difference in the final complexity.

Natural greedy approach of always perform the cheapest matrix multiplication available may not always yield optimal solution

- see second parenthesization as a counterexample

## Brute Force Algorithm

Q. How many different parenthesization methods (add brackets) for  $A_1 A_2 \dots A_n$ ?

## Brute Force Algorithm

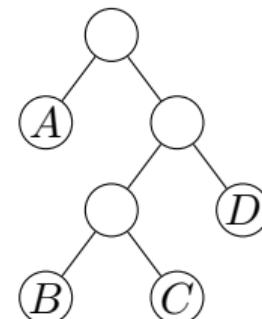
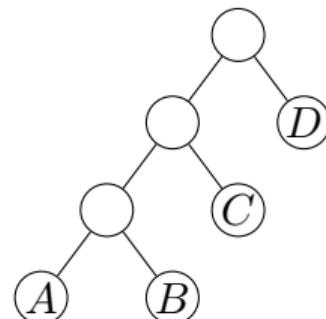
Q. How many different parenthesization methods (add brackets) for  $A_1 A_2 \dots A_n$ ?

Observation. A particular parenthesiation can be represented naturally by a *full* binary tree

- leaves nodes: individual matrices
- the root node: final product
- interior nodes: intermediate products

$$((A \times B) \times C) \times D$$

$$A \times ((B \times C) \times D)$$



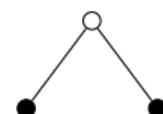
## Estimate the Number of Possible Orders

The number of possible orders correspond to various full binary trees with  $n$  leaves.

Let  $C(n)$  be the number of full binary tree with  $n + 1$  leaves, or, equivalently, with total  $n$  internal nodes:



$$C(0)$$



$$C(1)$$

$$\boxed{C(0) = 1}, C(1) = 1, C(2) = C(0)C(1) + C(1)C(0)$$

$$C(3) = C(0)C(2) + C(1)C(1) + C(2)C(0)$$

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} = \frac{1}{n+1} \binom{2n}{n}$$

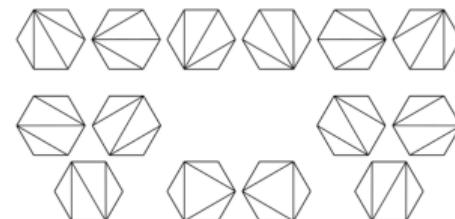
The above formula is of convolution form, can be calculated via **generating function**.

- The result is **Catalan number**, which is exponential in  $n$

## Catalan Number

Catalan number (named after the Belgian mathematician Eugène Charles Catalan).

- First discovered by Euler when counting the number of different ways of dividing a convex polygon with  $n$  sides into  $(n - 2)$  triangles.



$$\begin{aligned} C(n) &= \Omega\left(\frac{1}{n+1} \frac{(2n)!}{n!n!}\right) // \text{Stirling formula} \\ &= \Omega\left(\frac{1}{n+1} \frac{\sqrt{2\pi 2n} \left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi 2n} \left(\frac{n}{e}\right)^n \sqrt{2\pi 2n} \left(\frac{n}{e}\right)^n}\right) = \Omega(4^n / (n^{3/2} \sqrt{\pi})) \end{aligned}$$

## Brute Force Algorithm

Catalan number Occur in various counting problems (often involving recursively-defined objects)

- number of parenthesis methods
- number of full binary trees
- number of monotonic lattice paths

Since Catalan number is exponential in  $n \sim$  we certainly cannot try each tree, with brute force thus ruled out.

We turn to dynamic programming.

## Dynamic Programming

The correspondence to binary tree is suggestive: for a tree to be optimal, its subtrees must be also be optimal  $\Rightarrow$  satisfy **optimal substructure** (has somewhat locality)  $\leadsto$  do not have to try each tree from scratch

- subproblems corresponding to the subtrees: products of the form

$$A_i \times A_{i+1} \times \cdots \times A_j$$

Optimized function:

$$\begin{aligned} C(i, j) &= \text{minimum cost of multiplying } A_i \times A_{i+1} \times \cdots \times A_j \\ &\quad \text{the corresponding dimension is } m_{i-1}, m_i, \dots, m_j \end{aligned}$$

Iteration relation:

$$\underline{C(i, j)} = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{ \underline{C(i, k)} + \underline{C(k + 1, j)} + m_{i-1} m_k m_j \} & i < j \end{cases}$$

$A_i$	$\dots$	$A_k$	$A_{k+1}$	$\dots$	$A_j$
-------	---------	-------	-----------	---------	-------

$$m_{i-1} \times m_k$$

$$m_k \times m_j$$

## Some Remarks

Key points of DP

- Define subproblems
- Find iterative optimal substructure among subproblems
- Compute the subproblems in the **right order**

Sometimes the relation among subproblems may misleading. One should interpret and compute it in the right way, i.e., iterative.

## Recursive Approach (inefficient)

---

**Algorithm 1:** MatrixChain( $C, i, j$ ) // subproblem  $[i, j]$

---

```
1:  $C(i, i) = 0, C(i, j) \leftarrow \infty;$ 
2:  $s(i, j) \leftarrow \perp$  //record split position;
3: for  $k \leftarrow i$  to  $j - 1$  do
4:    $t \leftarrow \underline{\text{MatrixChain}}(C, i, k) + \underline{\text{MatrixChain}}(C, k + 1, j) + m_{i-1}m_km_j;$ 
5:   if  $t < C(i, j)$  then //find better solution
6:      $C(i, j) \leftarrow t;$ 
7:      $s(i, j) \leftarrow k;$ 
8:   end
9: end
10: return  $C(i, j);$ 
```

---

## Complexity Analysis

Recurrence relation is:

$$T(n) = \begin{cases} O(1) & n = 1 \\ \sum_{k=1}^{n-1} (T(k) + T(n-k) + \underline{O(1)}) & n > 1 \end{cases}$$

- $O(1)$ : sum and compare

$$T(n) = \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k) + O(n) = 2 \sum_{k=1}^{n-1} T(k) + O(n)$$

---

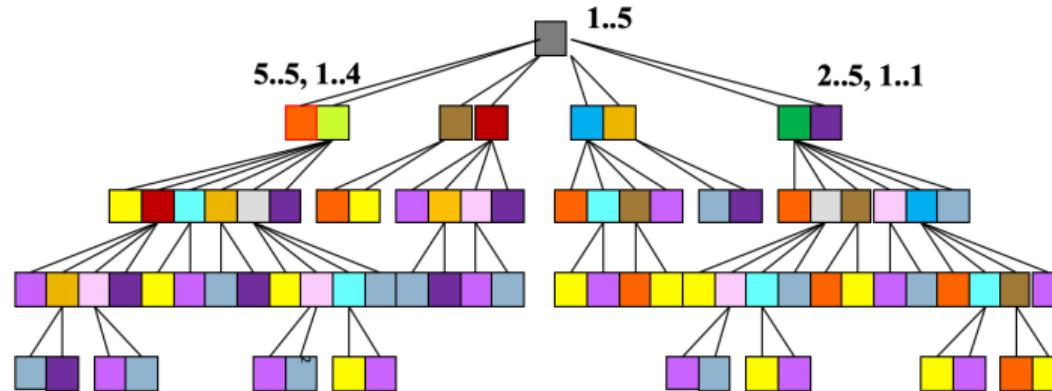
Claim.  $T(n) = \Omega(2^{n-1})$

- Induction basis:  $n = 2$ ,  $T(2) \geq c = c_1 2^{2-1}$ , let  $c_1 = c/2$ .
- Induction step:  $P(k < n) \Rightarrow P(n)$ .

$$\begin{aligned} T(n) &= O(n) + c_1 2 \sum_{k=1}^{n-1} 2^{k-1} && // \text{induction premise} \\ &\geq O(n) + c_1 2(2^{n-1} - 1) = \Omega(2^{n-1}) && // \text{geometric series} \end{aligned}$$

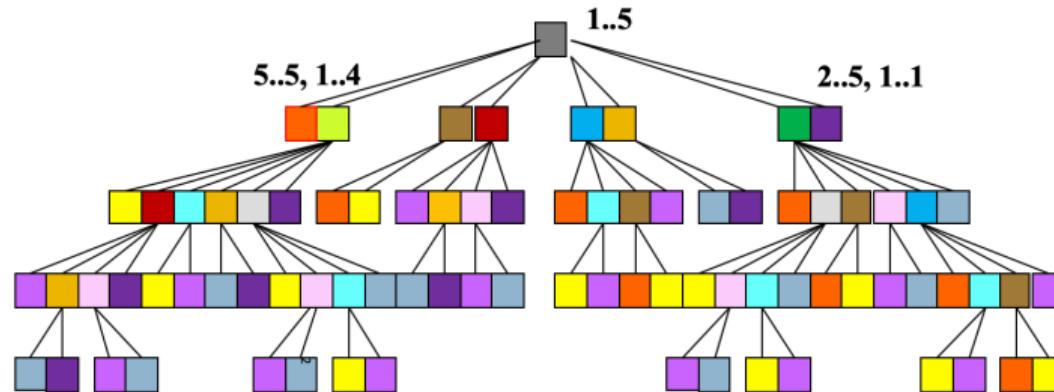
essentially same as brute force algorithm

## Root of Inefficiency (Case $n = 5$ )



15 different subproblems vs. computing 81 subproblems

## Root of Inefficiency (Case $n = 5$ )



15 different subproblems vs. computing 81 subproblems

Those who cannot remember the past are condemned to repeat it.

- Dynamic Programming



## Iterative Approach (efficient)

size = 1:  $n$  different subproblems

- $C(i, i) = 0$  for  $i \in [n]$  (no computation cost)

size = 2:  $n - 1$  different subproblems

- $C(1, 2), C(2, 3), C(3, 4), \dots, C(n - 1, n)$

...

size =  $i$ :  $n - i + 1$  different subproblems

...

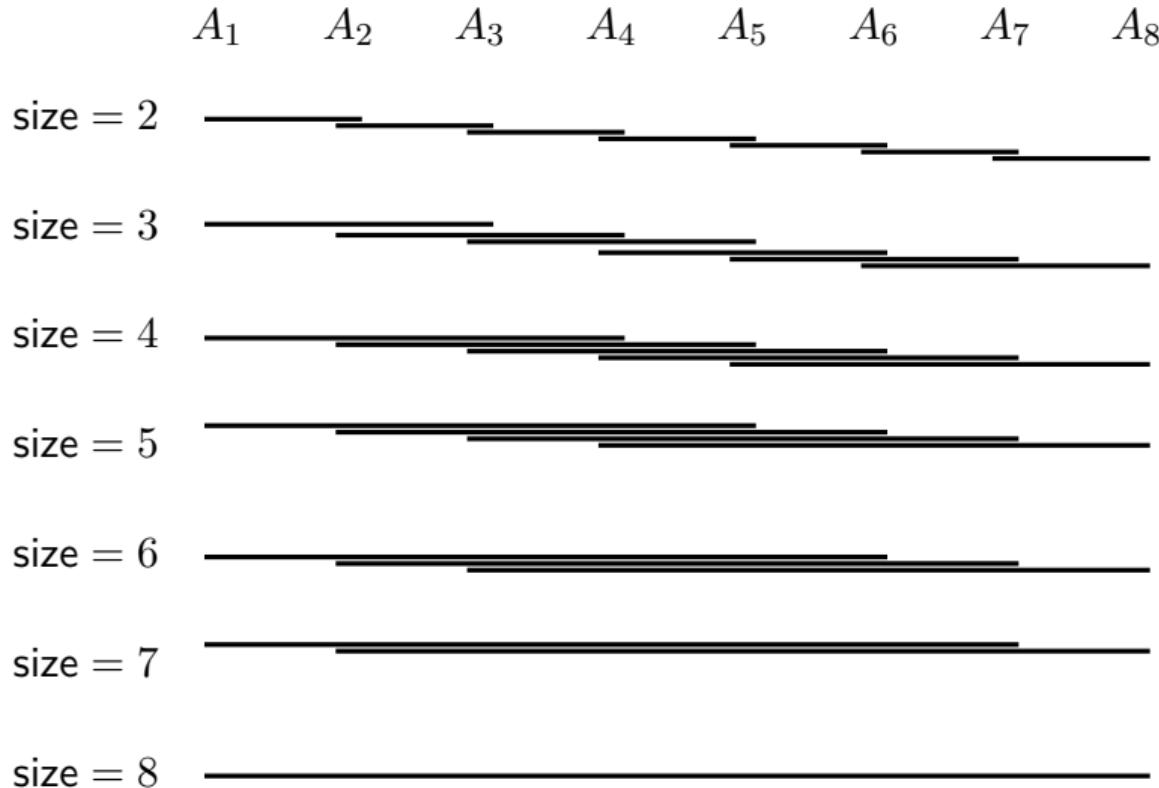
size =  $n - 1$ : 2 different subproblems

- $C(1, n - 1), C(2, n)$

size =  $n$ : original problem

- $C(1, n)$

## Demo of $n = 8$



---

**Algorithm 2:** MatrixChain( $C, n$ )

---

```
1:  $C(i, i) \leftarrow 0, C(i, j)_{i \neq j} \leftarrow +\infty;$ 
2: for  $\ell \leftarrow 2$  to  $n$  do //size of subproblem
3:   for  $i = 1$  to  $n - \ell + 1$  do //left boundary  $i$ 
4:      $j \leftarrow i + \ell - 1$  //right boundary  $j$ ;
5:     for  $k \leftarrow i$  to  $j - 1$  do //try all split position
6:        $t \leftarrow C(i, k) + C(k + 1, j) + m_{i-1}m_km_j;$ 
7:       if  $t < C(i, j)$  then
8:          $C(i, j) \leftarrow t, s(i, j) = k$  //update
9:       end
10:      end
11:    end
12: end
```

---

---

**Algorithm 3:** Trace( $s, i, j$ ) //initially  $i = 1, j = n$ 

---

```
1: if  $i=j$  then return;
2: output  $k \leftarrow s(i, j), \text{Trace}(s, i, k), \text{Trace}(s, k + 1, j);$ 
```

---

## Complexity Analysis

According to the algorithm

- line 2: subproblem size
- line 3 – 4: the boundaries of subproblem
- line 5: try all split position to find the optimal break point
- Line 2, 3 – 4, 5 constitute three-fold loop, length of each loop is  $O(n)$ ; the cost in the inner loop is  $O(1) \sim$  complexity  $O(n^3)$

## Complexity Analysis

According to the algorithm

- line 2: subproblem size
- line 3 – 4: the boundaries of subproblem
- line 5: try all split position to find the optimal break point
- Line 2, 3 – 4, 5 constitute three-fold loop, length of each loop is  $O(n)$ ; the cost in the inner loop is  $O(1) \sim$  complexity  $O(n^3)$

According to the memo

- there are totally  $n^2$  elements in the memo, to determine the value of each element, try and comparison cost is  $O(n) \sim$  complexity  $O(n^3)$

## Complexity Analysis

According to the algorithm

- line 2: subproblem size
- line 3 – 4: the boundaries of subproblem
- line 5: try all split position to find the optimal break point
- Line 2, 3 – 4, 5 constitute three-fold loop, length of each loop is  $O(n)$ ; the cost in the inner loop is  $O(1) \sim$  complexity  $O(n^3)$

According to the memo

- there are totally  $n^2$  elements in the memo, to determine the value of each element, try and comparison cost is  $O(n) \sim$  complexity  $O(n^3)$

---

Trace complexity:  $n - 1$  (number of interior nodes)

## Example

**Matrix chain.**  $A_1 A_2 A_3 A_4 A_5$ ,  $A_1 : 30 \times 35$ ,  $A_2 : 35 \times 15$ ,  $A_3 : 15 \times 5$ ,  $A_4 : 5 \times 10$ ,  $A_5 : 10 \times 20$

$\ell = 2$	$C(1, 2) = 15750$	$C(2, 3) = 2625$	$C(3, 4) = 750$	$C(4, 5) = 1000$
$\ell = 3$	$C(1, 3) = 7875$	$C(2, 4) = 4375$	$C(3, 5) = 2500$	
$\ell = 4$	$C(1, 4) = 9375$	$C(2, 5) = 7125$		
$\ell = 5$	$C(1, 5) = 11875$			

$\ell = 2$	$s(1, 2) = 1$	$s(2, 3) = 2$	$s(3, 4) = 3$	$s(4, 5) = 4$
$\ell = 3$	$s(1, 3) = 1$	$s(2, 4) = 3$	$s(3, 5) = 3$	
$\ell = 4$	$s(1, 4) = 3$	$s(2, 5) = 3$		
$\ell = 5$	$s(1, 5) = 3$			

$$s(1, 5) \Rightarrow (A_1 A_2 A_3)(A_4 A_5)$$

$$s(1, 3) \Rightarrow A_1(A_2 A_3)$$

- optimal computation order:  $(A_1(A_2 A_3))(A_4 A_5)$
- minimum multiplication:  $C(1, 5) = 11875$

## Outline

1 Chain Matrix Multiplication

2 Optimal Binary Search Tree

## Binary Search Tree

Let  $S$  be an ordered set with elements  $x_1 < x_2 < \dots < x_n$ . To admit efficient search, we store them on the nodes of a binary tree.

**Search:** If  $x \in S$ , output the index. Else, output the interval.

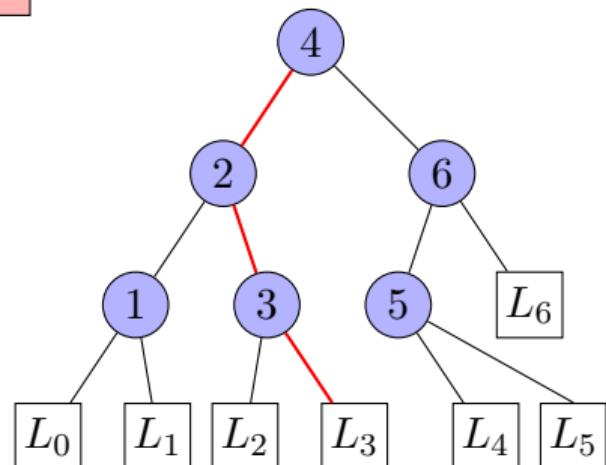
$$x = 3.5$$

1	2	3	4	5	6
---	---	---	---	---	---

$x$  vs. root

- $x <$  root, enter left subtree;
- $x >$  root, enter right subtree;
- $x =$  root, halt and output  $x$ ;

$x$  reaches leave nodes, halt, outputs  $\perp$ .



## The Distribution of Search Element

When  $x \xleftarrow{R} S \Rightarrow$  balance binary tree is optimal

What if the distribution of  $x$  is not uniform?

Let  $S = (x_1, \dots, x_n)$ . Consider intervals  $(x_0, x_1), (x_1, x_2), \dots, (x_{n-1}, x_n), (x_n, x_{n+1})$ , where  $x_0 = -\infty, x_{n+1} = +\infty$

- $\Pr[x = x_i] = b_i, \Pr[x \in (x_i, x_{i+1})] = a_i$

The distribution of  $x$  over  $S \cup \bar{S}$  is

$$P = (a_0, b_1, a_1, b_2, a_2, \dots, b_n, a_n)$$

---

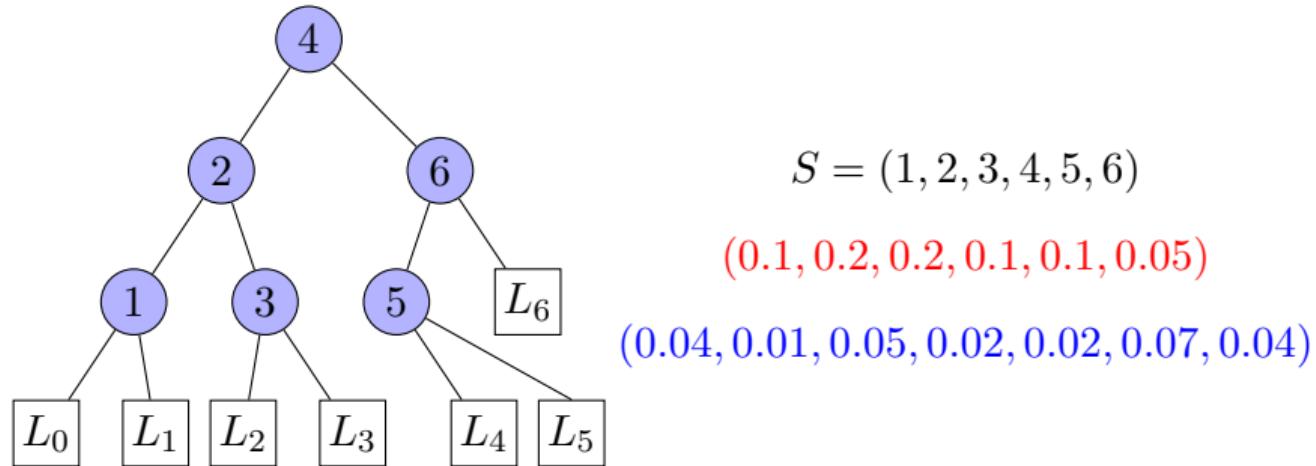
Example:  $S = (1, 2, 3, 4, 5, 6)$ . The distribution  $P$  of  $x$  is

$$(0.04, 0.1, 0.01, 0.2, 0.05, 0.2, 0.02, 0.1, 0.02, 0.1, 0.07, 0.05, 0.04)$$

$x = 1, 2, 3, 4, 5, 6$ : 0.1, 0.2, 0.2, 0.1, 0.1, 0.05

$x$  lies at interval: 0.04, 0.01, 0.05, 0.02, 0.02, 0.07, 0.04

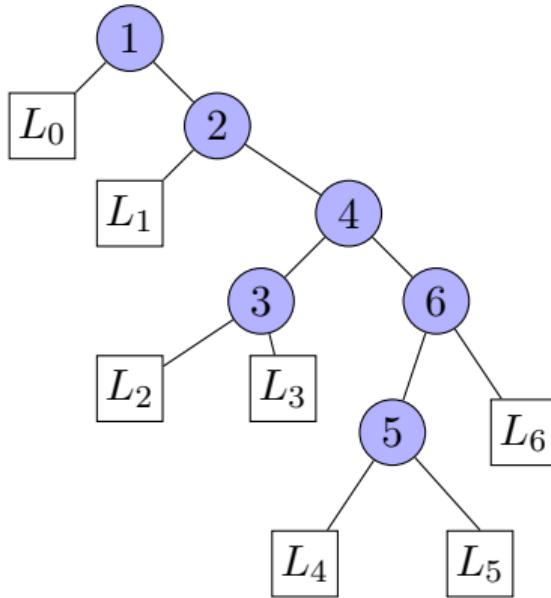
## Binary Search Tree 1



Average search times:

$$\begin{aligned} A(T_1) &= [1 \times 0.1 + 2 \times (0.2 + 0.05) + 3 \times (0.1 + 0.2 + 0.1)] \\ &\quad + [3 \times (0.04 + 0.01 + 0.05 + 0.02 + 0.02 + 0.07) \\ &\quad + 2 \times 0.04] \\ &= 1.8 + 0.71 = 2.51 \end{aligned}$$

## Binary Search Tree 2



$$S = (1, 2, 3, 4, 5, 6)$$

$$(0.1, 0.2, 0.2, 0.1, 0.1, 0.05)$$

$$(0.04, 0.01, 0.05, 0.02, 0.02, 0.07, 0.04)$$

Average search times:

$$\begin{aligned} A(T_2) &= [1 \times 0.1 + 2 \times 0.2 + 3 \times 0.1 + 4 \times (0.2 + 0.05) + 5 \times 0.1] \\ &\quad + [1 \times 0.04 + 2 \times 0.01 + 4 \times (0.05 + 0.02 + 0.04) \\ &\quad + 5 \times (0.02 + 0.07)] = 2.3 + 0.95 = 3.25 \end{aligned}$$

## Formula of Average Search Time

Set  $S = (x_1, x_2, \dots, x_n)$

Distribution  $P = (a_0, b_1, a_1, b_2, \dots, a_i, b_{i+1}, \dots, b_n, a_n)$

- the depth of  $x_i$  in  $T$  is  $d(x_i)$ ,  $i = 1, 2, \dots, n$ .
    - depth is counted from 0
    - the  $k$ -level node requires  $k + 1$  times compare
  - the depth of interval  $I_j$  is  $d(I_j)$ ,  $j = 0, 1, \dots, n$ .
- 

### Average Search Time

$$A(T) = \sum_{i=1}^n b_i(1 + d(x_i)) + \sum_{j=0}^n a_j d(I_j)$$

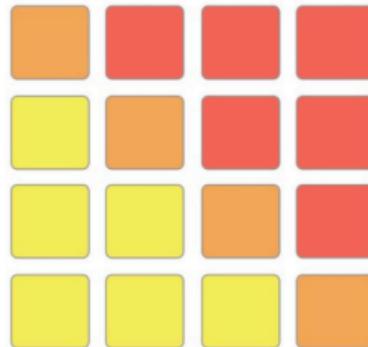
When the depth of all nodes increase by 1, the average search time increases by:

$$\sum_{i=1}^n b_i + \sum_{j=0}^n a_j$$

## Modeling of Optimal Search Tree

**Problem.** Given set  $S = (x_1, x_2, \dots, x_n)$  and distribution of search element  $P = (a_0, b_1, a_1, b_2, a_2, \dots, b_n, a_n)$ ,

**Goal.** Find an optimal binary search tree (with minimal average search times)



Dynamic  
Programming

## Dynamic Programming

**Subproblems:** defined by  $(i, j)$ ,  $i$  is the left boundary,  $j$  is the right boundary

- dataset:  $S[i, j] = (x_i, x_{i+1}, \dots, x_j)$
  - distribution:  $P[i, j] = (a_{i-1}, b_i, a_i, b_{i+1}, \dots, b_j, a_j)$
- 

**Input instance:**  $S = (A, B, C, D, E)$

$$P = (0.04, 0.1, 0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06, 0.1, 0.01)$$

Subproblem:  $(2, 4)$

- $S[2, 4] = (B, C, D)$
- $P[2, 4] = (0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06)$

## Break Up to Subproblem

Using  $x_k$  as root, break up one problem into two subproblems:

- $S[i, k - 1], P[i, k - 1]$
- $S[k + 1, j], P[k + 1, j]$

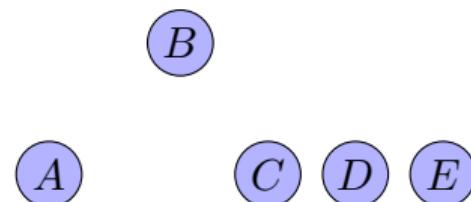
**Example.** Choose node  $B$  as root, break up the original problem into the following two subproblems:

Subproblem: (1, 1)

- $S[1, 1] = (A), P[1, 1] = (0.04, \textcolor{red}{0.1}, 0.02)$

Subproblem: (3, 5)

- $S[3, 5] = (C, D, E), P[3, 5] = (0.02, \textcolor{red}{0.1}, 0.05, \textcolor{red}{0.2}, 0.06, \textcolor{red}{0.1}, 0.01)$



## Probability Sum of Subproblem

For subproblem  $S[i, j]$  and  $P[i, j]$ , the probability sum in  $P[i, j]$  (including elements and intervals) is:

$$w[i, j] = \sum_{s=i-1}^j a_s + \sum_{t=i}^j b_t$$

---

Example of subproblem (2, 4)

- $S[2, 4] = (B, C, D)$
- $P[2, 4] = (0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06)$
- $w[2, 4] = (0.3 + 0.1 + 0.2) + (0.02 + 0.02 + 0.05 + 0.06) = 0.75$

## Optimized Function

Optimized function  $\text{OPT}(i, j)$ : the optimal average compare times of subproblem  $(i, j)$  for  $S[i, j]$ ,  $P[i, j]$ .

Parameterized optimized function.  $\text{OPT}_k(i, j)$ : optimal average compare times with  $x_k$  as root

Initial values:  $\text{OPT}(i, i - 1) = 0$  for  $i = 1, 2, \dots, n, n + 1$  corresponds to empty subproblem.

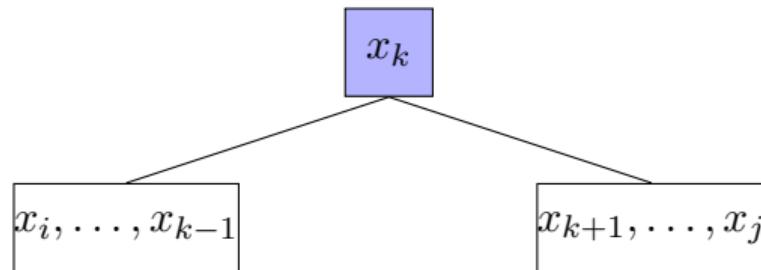
---

Example:  $S = (A, B, C, D, E)$

- ① choose  $A$  as root ( $k = 1$ ), yield subproblem  $(1, 0)$  and  $(2, 5)$ ,  $(1, 0)$  is an empty subproblem: corresponding to  $S[1, 0]$ ,  $\text{OPT}(1, 0) = 0$
- ② choose  $E$  as root ( $k = 5$ ), yield subproblem  $(1, 4)$  and  $(6, 5)$ ,  $(6, 5)$  is an empty subproblem: corresponding to  $S[6, 5]$ ,  $\text{OPT}(6, 5) = 0$

## Iterate Relation for Optimized Function

$$\begin{aligned}\text{OPT}(i, j) &= \min_{i \leq k \leq j} \{\text{OPT}_k(i, j)\}, 1 \leq i \leq j \leq n \\ &= \min_{i \leq k \leq j} \{\text{OPT}(i, k - 1) + \text{OPT}(k + 1, j) + w[i, j]\}\end{aligned}$$



- the depth of all nodes in left subtree and right subtree increase by 1

$$w[i, k - 1] + b_k + w[k + 1, j] = w[i, j]$$

## Proof of $\text{OPT}_k(i, j)$

$$\begin{aligned}\text{OPT}_k(i, j) &= (\text{OPT}(i, k - 1) + \underline{w[i, k - 1]}) + (\text{OPT}(k + 1, j) + \underline{w[k + 1, j]}) + b_k \\ &= (\text{OPT}(i, k - 1) + \text{OPT}(k + 1, j)) + (w[i, k - 1] + b_k + w[k + 1, j]) \\ &= (\text{OPT}(i, k - 1) + \text{OPT}(k + 1, j)) \\ &\quad + \left( \sum_{s=i-1}^{k-1} a_s + \sum_{t=i}^{k-1} b_t \right) + b_k + \left( \sum_{s=k}^j a_s + \sum_{t=k+1}^j b_t \right) \\ &= (\text{OPT}(i, k - 1) + \text{OPT}(k + 1, j)) + \sum_{s=i-1}^j a_s + \sum_{t=i}^j b_t \quad //\text{simplify} \\ &= \text{OPT}(i, k - 1) + \text{OPT}(k + 1, j) + w[i, j]\end{aligned}$$

## Pseudocode

Computation order: the size of subtree grows from 1 to  $n$

---

**Algorithm 4:** BinarySearchTree( $S, P, n$ )

---

```
1:  $\text{OPT}(i, i - 1) \leftarrow 0$  for all  $i \in [1, n + 1]$ ;  
2:  $\text{OPT}(i, j) \leftarrow +\infty$  for all  $i \leq j$ ;  
3: for  $\ell \leftarrow 1$  to  $n$  do //size of subproblem  
4:   for  $i = 1$  to  $n - \ell + 1$  do //left boundary  $i$   
5:      $j \leftarrow i + \ell - 1$  //right boundary  $j$ ;  
6:     for  $k \leftarrow i$  to  $j$  do //try all split position  
7:        $t \leftarrow \text{OPT}(i, k - 1) + \text{OPT}(k + 1, j) + w[i, j]$ ;  
8:       if  $t < \text{OPT}(i, j)$  then  
9:          $\text{OPT}(i, j) \leftarrow t$ ,  $s(i, j) = k$  //update  
10:        end  
11:      end  
12:    end  
13: end
```

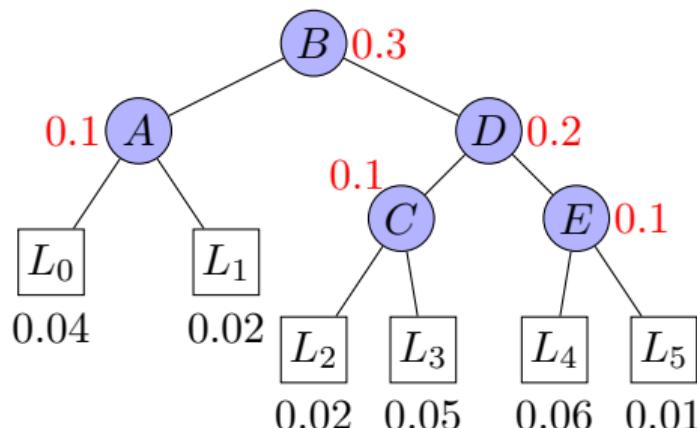
---

## Demo

$$\text{OPT}(i, j) = \min_{i \leq k \leq j} \{\text{OPT}(i, k - 1) + \text{OPT}(k + 1, j) + w[i, j]\}$$

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

$$\text{OPT}(i, i - 1) = 0, i = 1, 2, \dots, n, n + 1$$



choose  $B$  as root,  $k = 2$

$$\text{OPT}(1, 1) = 0.16$$

$$\text{OPT}(3, 5) = 0.88$$

$$\text{OPT}(3, 3) = 0.17$$

$$\text{OPT}(5, 5) = 0.17$$

$$w[3, 5] = 0.54$$

$$\text{OPT}(1, 5) = 1 + \min_{k \in [5]} \{\text{OPT}(1, k - 1) + \text{OPT}(k + 1, 5)\}$$

$$= 1 + (\text{OPT}(1, 1) + \text{OPT}(3, 5)) = 1 + (0.16 + 0.88) = 2.04$$

## Complexity Analysis

$$\text{OPT}(i, j) = \min_{i \leq k \leq j} \{\text{OPT}(i, k - 1) + \text{OPT}(k + 1, j) + w[i, j]\}$$

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

$$\text{OPT}(i, i - 1) = 0, i = 1, 2, \dots, n, n + 1$$

The number of  $(i, j)$  combination is  $O(n^2)$

For each  $\text{OPT}(i, j)$ , computation requires computing  $k$  terms and finding min. The cost of each term computation is constant time.

- Time complexity:  $T(n) = O(n^3)$
- Space complexity:  $S(n) = O(n^2)$