Introduction to Algorithm Design

Lecture Notes 9

ROAD MAP

- **Dynamic Programming**
  - The Knapsack Problem
  - All Pairs Shortest Paths
  - Optimal Binary Search Tree
  - String Editing
  - Matrix Chain Product

Dynamic Programming

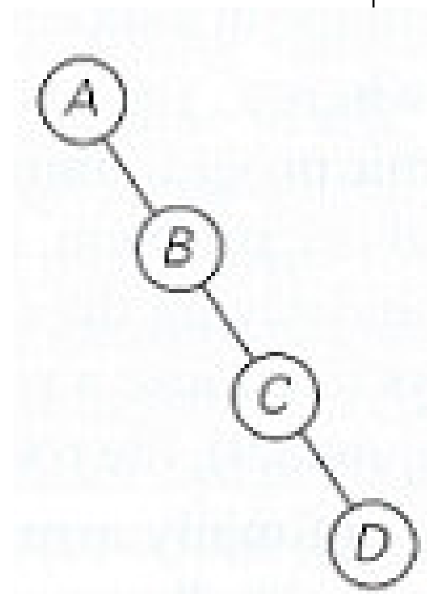
- **Definition :**
  - Dynamic programming is an interesting algorithm design technique for *optimizing multistage decision problems*
  - It is a technique for solving problems with overlapping subproblems
    - Typically arise from a recurrence relations
    - Solve subproblems once and record the results for later use
    - Bottom up solution of the recurrence
  - How to get the recurrence?
    - sequence of decisions
    - possible choices at each decision point
    - Based on principle of optimality write the recurrence relation

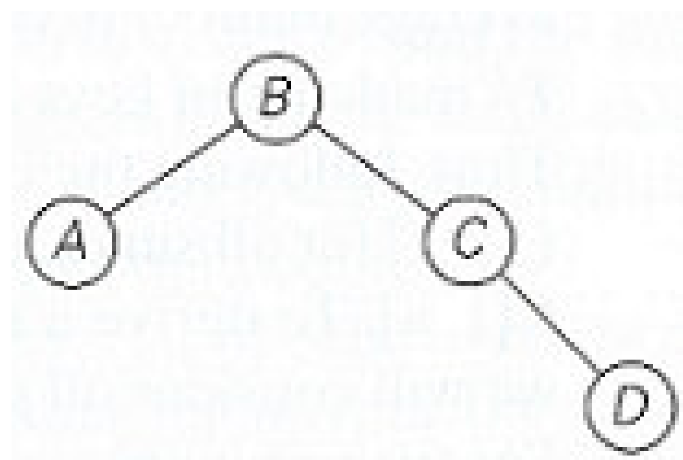
Optimal Binary Search Trees

- Definition:
  - For a given set of elements there are several BSTs possible
    - Since total number of BSTs with n nodes is given by C(2n,n)/(n+1), which grows exponentially, brute force is hopeless
  - Assume probabilities of searching for elements of a BST are known, some of them are better than the others
    - average number of comparisons is smaller

Optimal Binary Search Trees

- Example:
  - Consider 4 keys A, B, C, D to be searched with probabilities 0.1, 0.2, 0.4, 0.3
  - Two out of possible trees
  - What is the average number of comparisons in a successfull search?





Optimal Binary Search Trees

- **Given :** Set of identifiers $\{a_1, a_2, \ldots, a_n\}$

$$a_1 \leq a_2 \leq \ \ldots \ \leq a_n$$

$$p_i \ : \ \text{probability to search } a_i$$

$$\sum p_i = 1$$

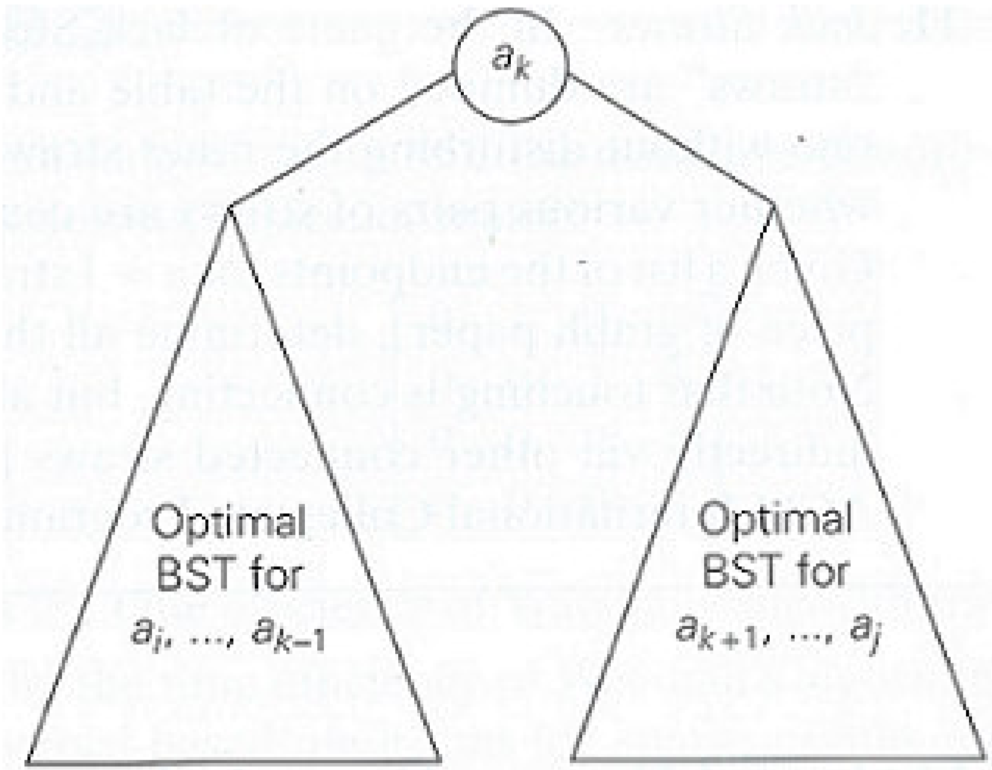
- **Find :** Optimal BST
  - Minimizing the average number of comparisons

Minimum $\sum p_i \times \text{level}(a_i)$

Optimal Binary Search Trees

- What is the sequence of decisions?
- What are possible choices at each decision point?
- What about principle of optimality…
- How to write the recurrence relation?
- How to solve the recurrence?

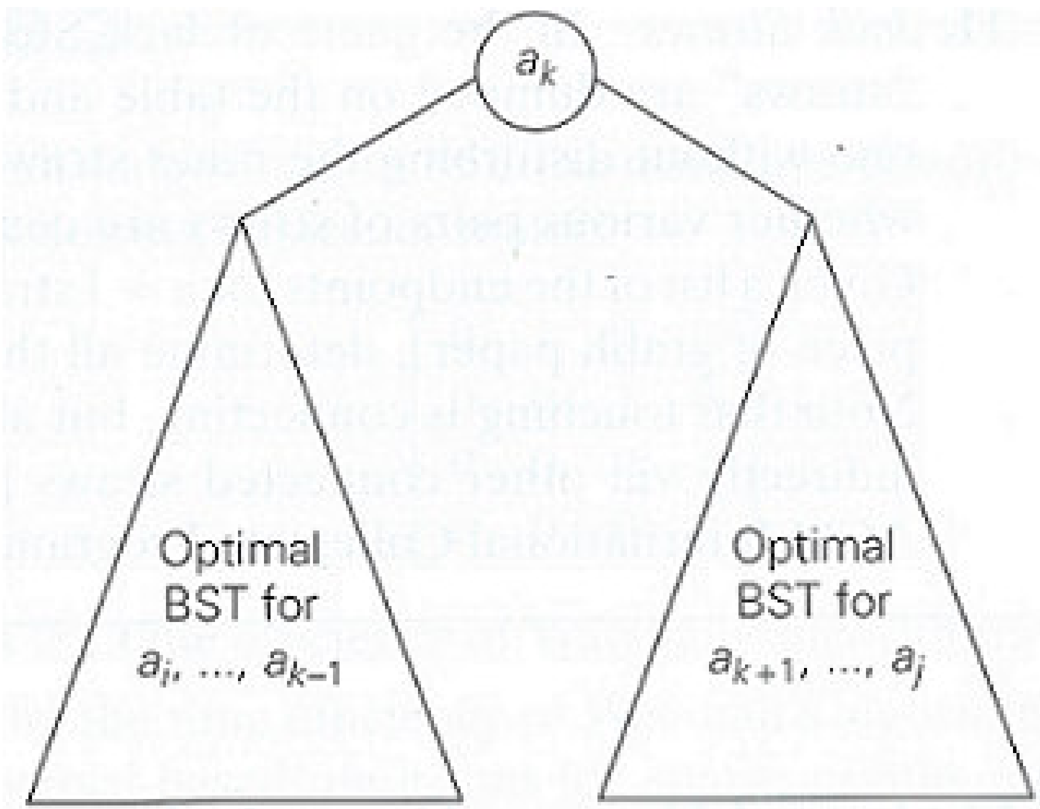
Optimal Binary Search Trees



Because of principle of optimality:

Optimal BST with root a_k and two optimal binary search subtrees

Optimal Binary Search Trees



$$\text{cost} = p_k + \text{cost}(L) + \text{cost}(R) + w(1, k-1) + w(k+1, n)$$

$$w(i, j) = \sum_{L=i}^{j} p_L$$

Optimal Binary Search Trees

Let $C[i,j]$ be minimum average number of comparisons made in T[i,j]

- T[i,j] is optimal BST for keys $a_i < \ldots < a_j$
  - where $1 \leq i \leq j \leq n$.
- Consider optimal BST among all BSTs with some a_k as their root
  - ($i \leq k \leq j$)

Optimal Binary Search Trees

$$C(1,n) = \min_{1 \le k \le n} \{ C(1,k-1) + C(k+1,n) + p_k + w(1,k\text{-}1) + w(k+1,n) \}$$

$$\mathrm{C(i,\,j)} = \min_{i \le k \le j} \left\{ c\,(i,k-1) + \mathrm{c\,(k+1,\,j)} + \underbrace{\mathrm{p_k} + \mathrm{w\,(i,k\text{-}1)} + \mathrm{w\,(k+1,\,j)}}_{\mathrm{w(i,j)}} \right\}$$

$$\mathrm{C(i,\,j)} = \min_{i \le k \le j} \{ c\,(i,k-1) + \mathrm{c\,(k+1,\,j)} + \mathrm{w\,(i,\,j)} \}$$

$$\mathrm{C(i,\,j)} = 0 \quad \text{if} \quad \mathrm{j < i}$$

Optimal Binary Search Trees

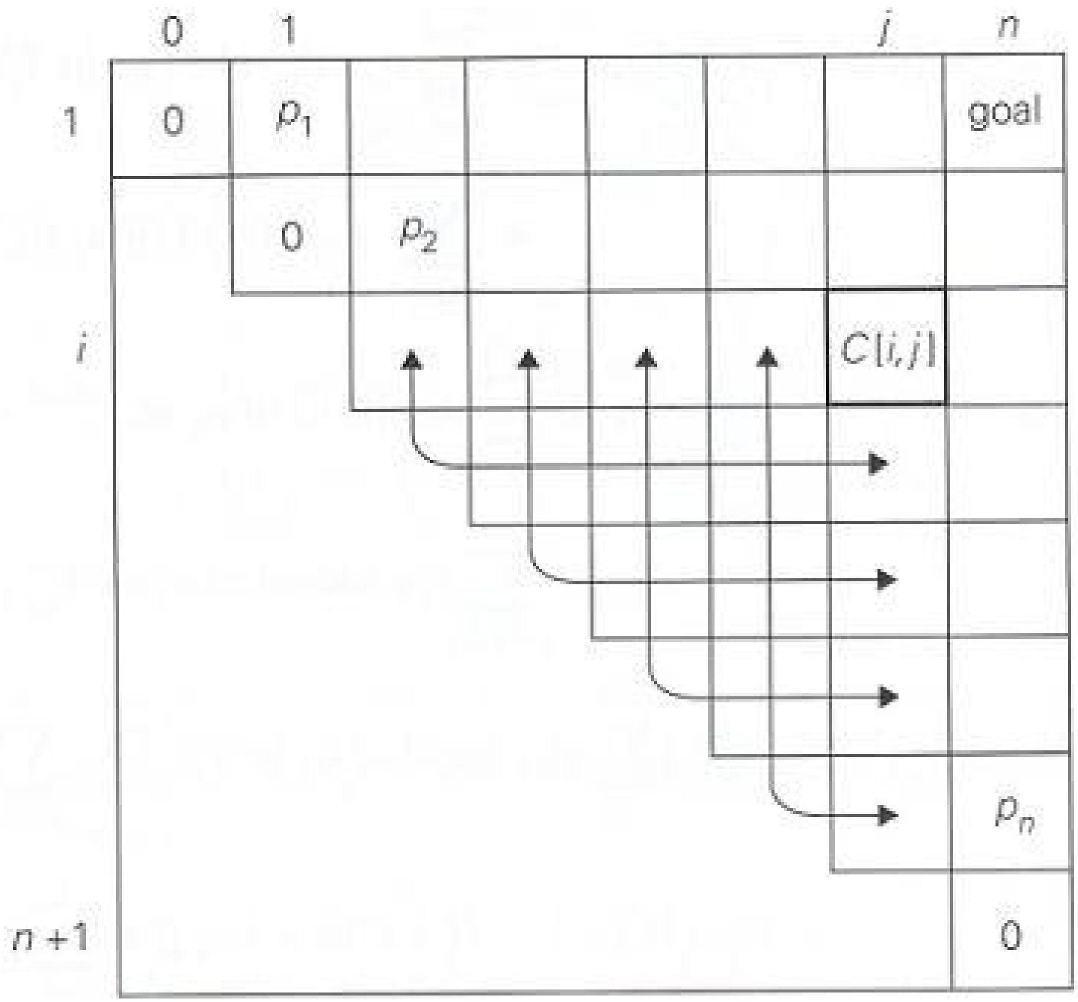


Table of dynamic programming algorithm for constructing an optimal BST

Optimal Binary Search Trees

- <u>Example :</u>
  - Four key set and probabilities are given

A → 0.1 B → 0.2 C → 0.4 D → 0.3

Initial table looks like

main table

	0	1	2	3	4
1	0	0.1			
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

root table

	0	1	2	3	4
1		1			
2			2		
3				3	
4					4
5					

Optimal Binary Search Trees

- Let us compute C[1,2] :

$$C[1,2] = \min \begin{Bmatrix} C[1,0] + C[2,2] + \sum_{s=1}^{2} p_s & \text{if } k = 1 \\ C[1,1] + C[3,2] + \sum_{s=1}^{2} p_s & \text{if } k = 2 \end{Bmatrix}$$

$$C[1,2] = \min \begin{Bmatrix} 0 + 0.2 + 0.3 = 0.5 \\ 0.1 + 0 + 0.3 = 0.4 \end{Bmatrix} = 0.4$$

- Out of two possible binary trees containing the first two keys A and B
  - the root of the optimal tree has index 2
  - average number of comparisons in a successfull search is 0.4

Optimal Binary Search Trees

- If the computations are finished, we will arrive at the following final tables

main table

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

root table

	0	1	2	3	4
1		1	2	3	3
2			2	3	3
3				3	3
4					4
5					

ALGORITHM *OptimalBST*($P[1..n]$)

//Finds an optimal binary search tree by dynamic programming
//Input: An array $P[1..n]$ of search probabilities for a sorted list of n keys
//Output: Average number of comparisons in successful searches in the
// optimal BST and table R of subtrees' roots in the optimal BST

for $i \leftarrow 1$ **to** n **do**
    $C[i, i-1] \leftarrow 0$
    $C[i, i] \leftarrow P[i]$
    $R[i, i] \leftarrow i$
$C[n+1, n] \leftarrow 0$
for $d \leftarrow 1$ **to** $n-1$ **do** //diagonal count
    for $i \leftarrow 1$ **to** $n-d$ **do**
        $j \leftarrow i + d$
        $minval \leftarrow \infty$
        for $k \leftarrow i$ **to** j **do**
            if $C[i, k-1] + C[k+1, j] < minval$
                $minval \leftarrow C[i, k-1] + C[k+1, j]$; $kmin \leftarrow k$
        $R[i, j] \leftarrow kmin$
        $sum \leftarrow P[i]$; **for** $s \leftarrow i+1$ **to** j **do** $sum \leftarrow sum + P[s]$
        $C[i, j] \leftarrow minval + sum$
return $C[1, n]$, R

Optimal Binary Search Trees

- Analysis:
  - Space efficiency is quadratic !
  - Time efficiency is cubic !
    - A more careful analysis shows that entries in the root table are always nondecreasing along each row and column
    - This limits values for $R[i,j]$ to the range $R[i,j-1], \ldots, R[i+1,j]$ and reduce the running time to $\Theta(n^2)$

String Editing

- Given :
  - $X = x_1, x_2, \ldots, x_n$
  - $Y = y_1, y_2, \ldots, y_m$
- Find :
  - A transform from X to Y using operation → insert $I(y_i)$
    → delete $D(x_j)$
    → change $C(x_i, y_j)$
- Cost Function : $\sum_{\forall \text{ operations}} \text{cost of operations}$

String Editing

- What is the sequence of decisions?
- What are possible choices at each decision point?
- What about principle of optimality…
- How to write the recurrence relation?

String Editing

$$C(i, j) = \text{optimal cost for transforming } x_1, x_2, \ldots, x_i \text{ to } y_1, y_2, \ldots, y_j$$

$$C(i,0) = C(i-1,0) + D(x_i)$$

$$C(0, j) = C(0, j-1) + I(y_j)$$

$$C(0,0) = 0$$

$$\text{if } x_i \neq y_j \Rightarrow C(i, j) = \min \begin{cases} -\text{delete } x_i & \Rightarrow c(i-1, j) + D(x_i) \\ -\text{insert } y_j & \Rightarrow c(i, j-1) + I(y_j) \\ -\text{change } x_i \text{ to } y_j & \Rightarrow c(i-1, j-1) + C(x_i, y_j) \end{cases}$$

$$\text{o/w} \qquad C(i, j) = C(i-1, j-1)$$

Matrix Chain Product

- Multiply *n* given matrices

  $M_1 \text{ x } M_2 \text{ x } \dots \text{ x } M_n$

- Goal :
  - Find the parenthesization to minimize the number of multiplications
    - M_1 , M_2 , M_3

      2x3 3x8 8x4

$$M_1 \times M_2 \times M_3 = \begin{cases} (M_1 \times M_2) \times M_3 & (2\times 3\times 8) + (2\times 8\times 4) = 112 \\ M_1 \times (M_2 \times M_3) & (3\times 8\times 4) + (2\times 3\times 4) = 120 \end{cases}$$

    - $(n-1)! = O(2^n)$ different parenthesization

Matrix Chain Product

- What is the sequence of decisions?
- What are possible choices at each decision point?
- What about principle of optimality…
- How to write the recurrence relation?

Matrix Chain Product

$$C(1,n): \quad \text{cost for 1 to n}$$

$$C(1,n) \;:\; \min_{1 \le i \le n\text{-}1} \left\{\, C(1,i) \;+\; C(i+1,n) \;+\; r_1 c_i c_n \right.$$

In general

$$\mathrm{C(i,j)}: \;\; \min_{\mathrm{i} \le k < j} \left\{ \mathrm{C(i,k)} \;+\; \mathrm{C(k+1,j)} \;+ \mathrm{r_i c_k c_j} \right.$$

$$C(i,i) = 0 \qquad C(1,n) \;= \text{optimal solution}$$

- # of C(i,j)'s = $O(n^2)$
- For each C(i,j) we check (j-i) expressions
- Overall complexity is $O(n^3)$