

## Lecture #8:

### 0.0.1 Dynamic Programming:(Chapter 15)

We now take up the concept of dynamic programming again using examples.

#### Example 1 (Chapter 15.2) *Matrix Chain Multiplication*

INPUT: An Ordered set of Matrices  $[A_1 A_2 A_3 \dots A_n]$  with  $A_i$  of size  $p_{i-1} \times p_i$  (given vector  $p = [p_0, p_1, \dots, p_n]$ ).

OUTPUT: The product of these matrices taken in this order.

MATRIX-CHAIN-ORDER( $p$ )

$n \leftarrow \text{length}(p) - 1$

**for**  $i \leftarrow 1$  **to**  $n$

**do**  $m[i, i] \leftarrow 0$

**for**  $l \leftarrow 2$  **to**  $n$

**do for**  $i \leftarrow 1$  **to**  $n - l + 1$

**do**  $j \leftarrow i + l - 1$

$m[i, j] \leftarrow \infty$

**for**  $k \leftarrow i$  **to**  $j - 1$

**do**  $q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$

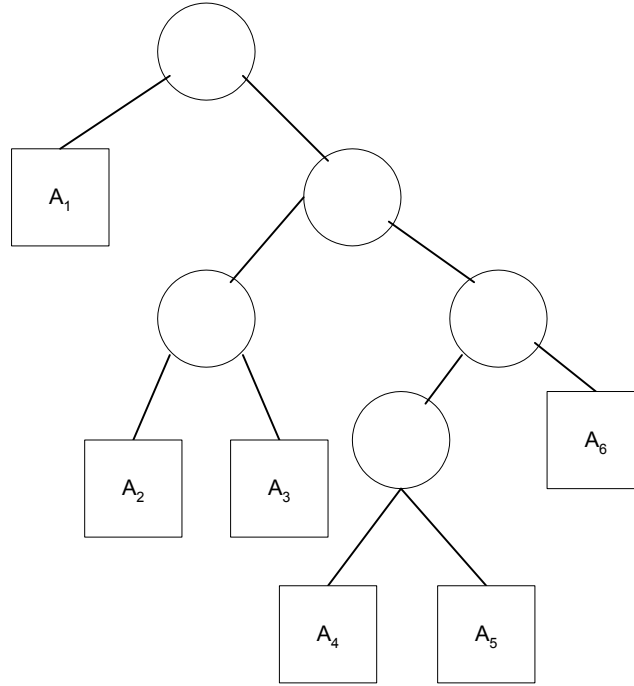
**if**  $q < m[i, j]$

**then**  $m[i, j] \leftarrow q; s[i, j] \leftarrow k$

**return**  $m$  and  $s$

Given a parenthesization for a matrix chain, we have a corresponding binary tree whose leaves correspond to the matrices and conversely. For example, if we have  $((A_1)((A_2 A_3)((A_4 A_5)(A_6))))$  [also written as  $(A_1)((A_2 A_3)((A_4 A_5)(A_6)))]$

we have the following tree:



**Example 2 (Chapter 15.4) Longest Common Subsequence**

INPUT: Two sequences  $X = \langle x_1, x_2, \dots, x_n \rangle$  and  $Y = \langle y_1, y_2, \dots, y_m \rangle$   
 OUTPUT: Matrix  $C$  and (pointer) matrix  $B$   
 LCS-Length( $X, Y$ )  
 $m \leftarrow \text{length}[X]$   
 $n \leftarrow \text{length}[Y]$   
**for**  $i \leftarrow 1$  **to**  $m$   
     **do**  $c[i, 0] \leftarrow 0$   
**for**  $j \leftarrow 1$  **to**  $n$   
     **do**  $c[0, j] \leftarrow 0$   
**for**  $i \leftarrow 1$  **to**  $m$   
     **do for**  $j \leftarrow 1$  **to**  $n$   
         **do if**  $x_i = y_j$   
             **then**  $c[i, j] \leftarrow c[i - 1, j - 1] + 1$ ;  $b[i, j] \leftarrow "$  ↖  $"$   
             **else if**  $c[i - 1, j] \geq c[i, j - 1]$   
                 **then**  $c[i, j] \leftarrow c[i - 1, j]$ ;  $b[i, j] \leftarrow "$  ↑  $"$   
                 **else**  $c[i, j] \leftarrow c[i, j - 1]$ ;  $b[i, j] \leftarrow "$  ←  $"$   
     **return**  $c$  and  $b$

**Example 3 ((First Edition)) Optimal Polygonal Triangulation**

INPUT: An ordered set of vertices of a convex polygon  $v_i; 0 \leq i \leq n$  (given vector  $v = [v_0, v_1, \dots, v_n]$ ) and  $w(\Delta v_i v_k v_j)$ .

OUTPUT: Triangulation with minimum cost.

POLYGONAL TRIANGULATION( $v$ )

$n \leftarrow \text{length}(v) - 1$

**for**  $i \leftarrow 1$  **to**  $n$

**do**  $m[i, i] \leftarrow 0$

**for**  $l \leftarrow 2$  **to**  $n$

**do for**  $i \leftarrow 1$  **to**  $n - l + 1$

**do**  $j \leftarrow i + l - 1$

$m[i, j] \leftarrow \infty$

**for**  $k \leftarrow i$  **to**  $j - 1$

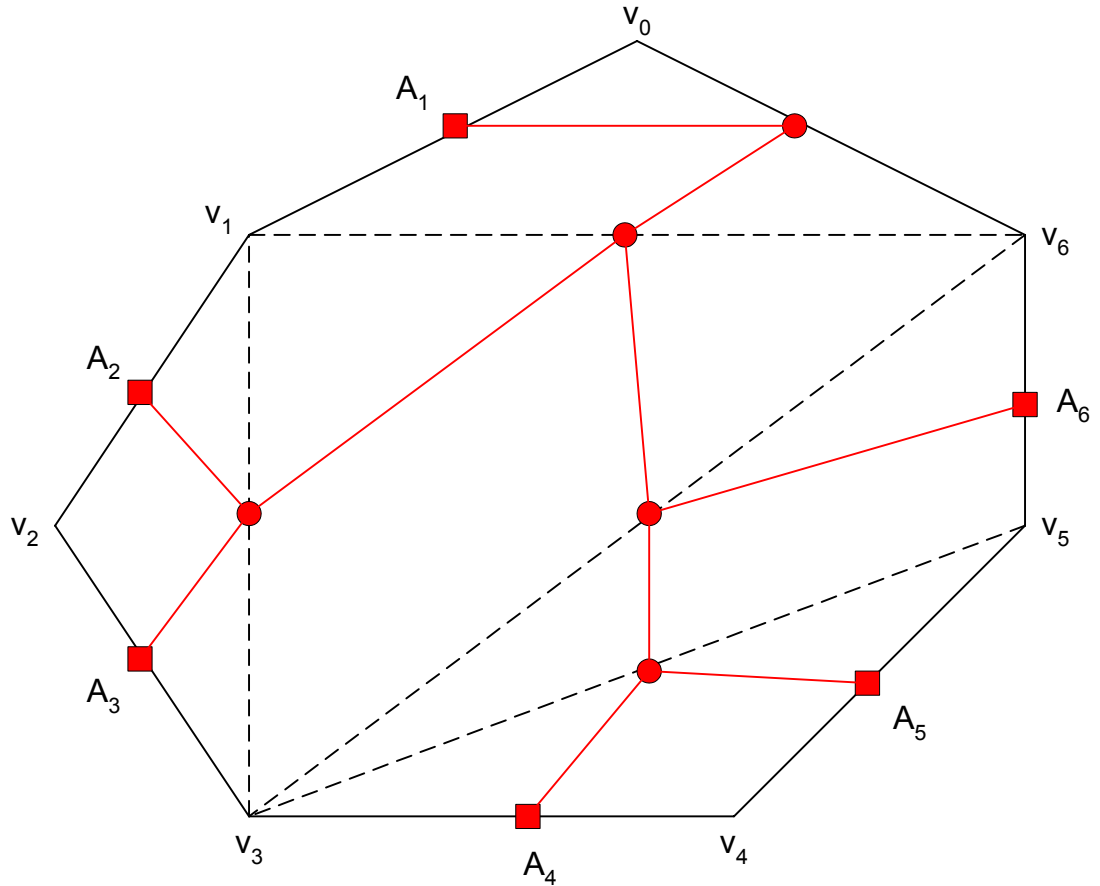
**do**  $q \leftarrow m[i, k] + m[k + 1, j] + w(\Delta v_{i-1} v_k v_j)$

**if**  $q < m[i, j]$

**then**  $m[i, j] \leftarrow q; s[i, j] \leftarrow k$

**return**  $m$  and  $s$

Consider the following convex polygon and a possible triangulation:



This triangulation corresponds to the same tree as in matrix chain multiplication. Indeed, there is a one-to-one correspondence between the triangulation and a parenthesization. It is this that is used in the above algorithm.

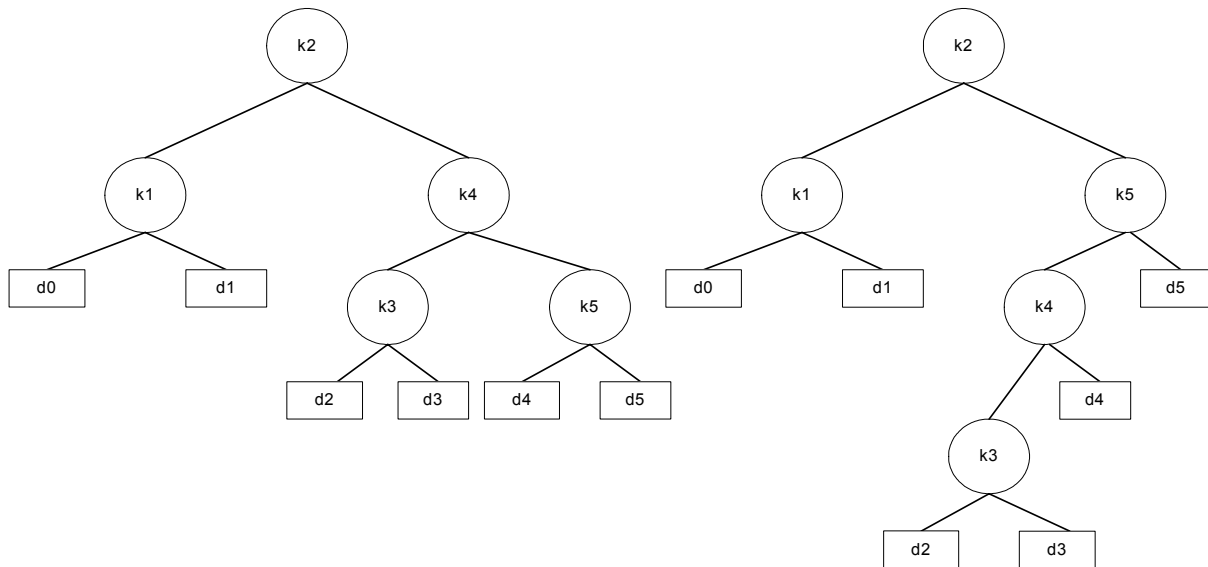
• **Example 4 (Chapter 15.5) Binary Search Trees:**

We are given an ordered sequence  $K = \{k_1, k_2, \dots, k_n\}$  of  $n$  distinct keys in sorted order so that  $k_i < k_{i+1}$ . We wish to build a binary search tree from these keys. For each key  $k_i$  we have a probability  $p_i$  that a search will be for  $k_i$ . Some searches may be for values not in  $K$ . So we have "dummy" keys  $d_0, d_1, \dots, d_n$  representing values not in  $K$ . In the sorted order these would look like:  $d_0 < k_1 < d_1 < k_2 < \dots, k_n < d_n$  (meaning that  $d_i$  represents values in between  $k_i$  and  $k_{i+1}$ ). The probability  $q_i$  associated with  $d_i$  is the probability that the search will ask for values in between  $k_i$  and  $k_{i+1}$ .  $q_0$  is the probability that values less than  $k_1$  will be asked for and  $q_n$  is the probability that values greater than  $k_n$  will be requested.

Recall, that in a binary search tree if  $y$  is a node in the left sub-tree of  $x$  and  $z$  is a node in right sub-tree of  $x$ , then  $key[y] \leq key[x] \leq key[z]$ . If the frequencies with which these keys occur is uniform, a balanced tree is the best to minimize the average search time. If, however, they are not uniform, then it may be advantageous to have tree that is not balanced in order to minimize the expected number of comparisons. For example, consider:

$i$	0	1	2	3	4	5
$p_i$		0.15	0.10	0.05	0.10	0.20
$q_i$	0.05	0.10	0.05	0.05	0.05	0.10

Two binary search trees are shown below for this example.



Expected time is given by 2.40 and 2.35.

$$\begin{aligned}
 E[T] &= \sum_{i=1}^n p_i [d_T(k_i) + 1] + \sum_{i=0}^n q_i [d_T(d_i)] \\
 &= \sum_{i=1}^n p_i + \sum_{i=1}^n p_i [d_T(k_i)] + \sum_{i=0}^n q_i [d_T(d_i)]
 \end{aligned}$$

where  $d_T(k_i)$  and  $d_T(d_i)$  denote the depth of these nodes in the tree from the root. Time here is measured by the number of queries.

We now take up the problem of designing a binary search tree by using dynamic programming.

**Problem 5** Given an ordered set  $k_1 < k_2 < k_3 < \dots < k_n$  of  $n$  distinct keys. The frequency of occurrence of requests to key  $k_i$  is  $p_i$  and the

frequency for dummy key  $d_i$  is  $q_i$ . We want a binary search tree that minimizes  $B_T$  where

$$B_T = \sum_{i=1}^n p_i + \sum_{i=1}^n p_i [d_T(k_i)] + \sum_{i=0}^n q_i [d_T(d_i)]$$

where  $d_T(k_i)$  is the depth of key  $k_i$  and that of  $d_i$  is  $d_T(d_i)$  in the tree  $T$ .

What is the difference between this and the Huffman trees?

If we select  $k_j$  for the root, then  $[d_0, k_1, \dots, k_{j-1}, d_{j-1}]$  form the left sub-tree and  $[d_j, k_{j+1}, \dots, k_n, d_n]$  form the right sub-tree. This gives the recursion for dynamic programming. Let  $c[i, j]$  denote the minimum "cost" for a search tree containing keys  $[d_{i-1}, k_i, \dots, k_j, d_j]$ . Then, for  $i \leq j$ ,

$$\begin{aligned} c[i, j] &= \min_{i \leq r \leq j} \{p_r + c[i, r-1] + \sum_{k=i}^{r-1} p_k + \sum_{k=i-1}^{r-1} q_k + c[r+1, j] + \sum_{k=r+1}^j p_k + \sum_{k=r}^j q_k\} \\ &= \sum_{k=i}^j p_k + \sum_{k=i-1}^j q_k + \min_{i \leq r \leq j} \{c[i, r-1] + c[r+1, j]\} \\ &= w(i, j) + \min_{i \leq r \leq j} \{c[i, r-1] + c[r+1, j]\} \end{aligned}$$

where  $w(i, j) = \sum_{k=i}^j p_k + \sum_{k=i-1}^j q_k$ . For  $j < i$ ,  $c[i, j] = 0$ . Please note that this is slightly different from that in the book. This is because our function  $B_T$  is different.

This can be done in  $O(n^3)$  time as shown below.

OPTIMAL-BST( $p, q, n$ )

1.     for  $i \leftarrow 1$  to  $n$
2.         do  $c[i, i] \leftarrow q_{i-1} + p_i + q_i$
3.              $w_{i,i} \leftarrow q_{i-1} + p_i + q_i$
4.     for  $l \leftarrow 1$  to  $n - 1$
5.         do for  $i \leftarrow 1$  to  $n - l$
6.             do  $j \leftarrow i + l$
7.                  $c[i, j] \leftarrow \infty$
8.                  $w_{i,j} \leftarrow w_{i,j-1} + p_j + q_j$
9.                 for  $r \leftarrow i$  to  $j$
10.                     do  $t \leftarrow c[i, r-1] + c[r+1, j] + w_{i,j}$
11.                         if  $t \leq c[i, j]$
12.                             then  $c[i, j] \leftarrow t$

13.  $m[i, j] \leftarrow r$   
 14. return  $c$  and  $m$

Actually, it can be done faster as shown below:

Let  $m[i, j]$  be the value of  $r$  that determines the minimum in the above expression for  $c[i, j]$ . Let  $M := \{m[i, j] : i < j\}$ .

$\{w_{i,j}\}$  are said to satisfy quadrangle inequality (QI) if the following holds

$$\{i \leq k \leq j \leq l\} \implies w_{i,j} + w_{k,l} \leq w_{k,j} + w_{i,l}$$

They are said to be monotone if

$$\{i \leq k \leq l \leq j\} \implies w_{k,l} \leq w_{i,j}$$

If  $w_{i,j} = \sum_{k=i}^j p_k + \sum_{k=i-1}^j q_k$  then QI holds as equality and monotonicity is obvious since the probabilities are nonnegative.

Suppose  $\{w_{i,j}\}$  satisfy quadrangle inequality and monotonicity conditions above. Consider the set of recursion equations of the form:

$$\begin{aligned} t[i, i] &= 0 \\ t[i, j] &= f_{i,j} + \min_{i < k \leq j} [t[i, k-1] + t[k, j]] \quad \text{for } i < j \end{aligned}$$

Then  $t$  can be computed in  $O(n^2)$  time. Substituting  $t[i, j+1] = c[i, j]$  and  $f_{i,j+1} = w_{i,j}$  let us transform BST to this.

**Lemma 6** *If  $f$  satisfies QI and monotonicity, then  $t$  satisfies QI.*

**Proof.** *Want to show that*

$$\{i \leq k \leq j \leq l\} \implies t[i, j] + t[k, l] \leq t[k, j] + t[i, l]$$

*We do this by induction on  $l-i$ .  $\{l-i=0\} \implies \{i=j=k=l\}$  and there is nothing to prove.  $\{l-i=1\} \implies \{i=k \text{ or } j=l\}$ . In this case also the inequality holds as an equality. Several cases to consider. ■*

**Case 1.**  $i < k = j < l$  we need to show:

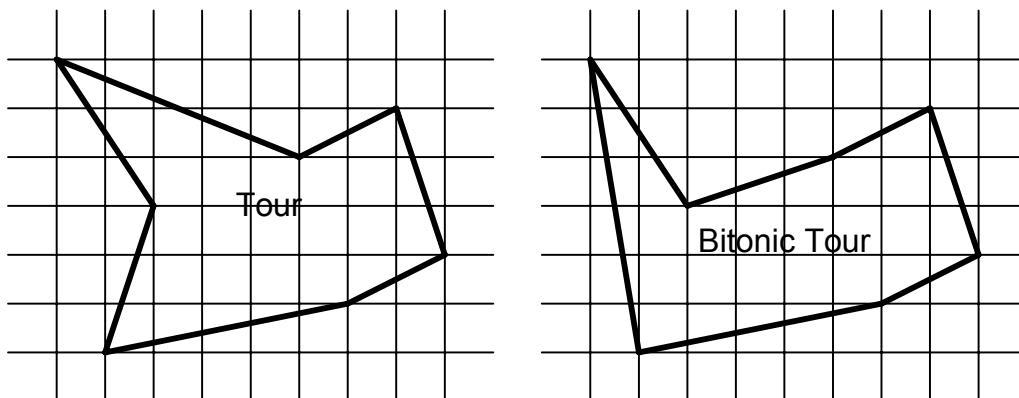
$$\{i \leq k \leq j \leq l\} \implies t[i, j] + t[j, l] \leq t[i, l]$$

*Let  $t_m[i, j] = f_{i,j} + t\{i, m-1\} + t\{m, j\}$  and let  $m[i, j] = \max\{m : t_m[i, j] = t[i, j]\}$ . If  $m[i, j] \leq j$ , then we have*

- Bitonic Tours (Problem 15-1 page 364)

The *Euclidean Traveling Salesperson Problem* (ETSP): Determine the shortest closed tour that connects a given set of  $n$  points in the plane. A

*bitonic* tour is a tour that starts at the leftmost point go left to right to the rightmost point and return right to left back to the starting point. See figures below for a tour and a bitonic tour.



The problem is to find the shortest bitonic tour by using dynamic programming. Assume that no two points have the same  $x$ -coordinate.

- Planning a Company Party (Problem 15-4)
- Edit Distance (Problem 15-3 (simplified))
- Partitioning Problem:

INPUT: An ordered set  $S[1, 2, \dots, n]$  and an integer  $p$ .

OUTPUT: Partition  $S$  into  $p$  (or fewer) subsets of consecutive elements of  $S$  so as to minimize the maximum sum over these subsets. That is,

$$\max\left\{\sum_{i=1}^{k_1} S[i], \sum_{i=k_1+1}^{k_2} S[i], \dots, \sum_{i=k_{p-1}+1}^{k_p} S[i]\right\}$$

is minimized. We are selecting the values for  $k_1 < k_2 < \dots < k_p$ .

Example:  $S = [1, 2, 3, 4, 5, 6, 7, 8, 9]$  and  $p = 3$ . The partition  $[1, 2, 3, 4, 5], [6, 7], [8, 9]$  has a cost equal to 17. We want this to be minimum.

- Calculating Binomial Coefficients:

Recall  $\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k!} = \frac{n!}{k!(n-k)!}$

We use the relation that

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

to calculate these numbers.

- Shortest Path Problems (Chapter 25 & 26)

The last set will be covered in Graph Algorithms. Since much of this material is directly from the book, there are no notes at this time.