DP Example: Matrix-Chain Multiplication

- If A is a p **x** q matrix and B a q **x** r matrix, then $C = AB$ is a p **x** r matrix

$$C[i, j] = \sum_{k=1}^{q} A[i, k]\, B[k, j]$$

time complexity: $O(pqr)$.

```
Matrix-Multiply(A, B)
1.  if A.columns ≠ B.rows
2.        error "incompatible dimensions"
3.  else let C be a new A.rows * B.columns matrix
4.        for i = 1 to A.rows
5.              for j = 1  to B.columns
6.                    c_ij = 0
7.                    for k = 1 to A.columns
8.                          c_ij = c_ij + a_ik b_kj
9.        return C
```

DP Example: Matrix-Chain Multiplication

- **The matrix-chain multiplication problem**
  - Input: Given a chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, matrix $\boldsymbol{A_i}$ has dimension $\boldsymbol{p_{i-1} \times p_i}$
  - Objective: Parenthesize the product $A_1 A_2 \ldots A_n$ to minimize the number of scalar multiplications

- **Exp:** dimensions: A_1: 4 **x** 2; A_2: 2 **x** 5; A_3: 5 **x** 1

  $(A_1A_2)A_3$: total multiplications = 4 **x** 2 **x** 5 + 4 **x** 5 **x** 1 = 60

  $A_1(A_2A_3)$: total multiplications = 2 **x** 5 **x** 1 + 4 **x** 2 **x** 1 = 18

- So the order of multiplications can make a big difference!

Matrix-Chain Multiplication: Brute Force

- $A = A_1 A_2 \ldots A_n$: How to evaluate A using the minimum number of multiplications?
- Brute force: check all possible orders?
  - $P(n)$: number of ways to multiply n matrices.

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

  - $P(n) = \Omega\left(\frac{4^n}{n^{3/2}}\right)$, **exponential** in n.
- Any efficient solution?
  - The matrix chain **is linearly ordered and cannot be rearranged!!**
  - Smell Dynamic programming?

Matrix-Chain Multiplication

- $m[i, j]$: minimum number of multiplications to compute matrix $A_{i..j} = A_i A_{i+1} \cdots A_j$, $1 \le i \le j \le n$.
  - $m[1, n]$: the cheapest cost to compute $A_{1..n}$.

$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j. \end{cases}$$

matrix A_i has dimension $p_{i-1} \times p_i$

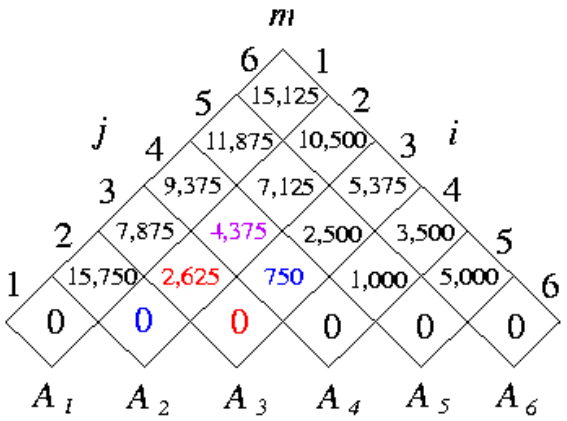
- Applicability of dynamic programming
  - **Optimal substructure:** an optimal solution contains within its optimal solutions to subproblems.
  - **Overlapping subproblem:** a recursive algorithm revisits the same problem over and over again; only $\theta(n^2)$ subproblems.

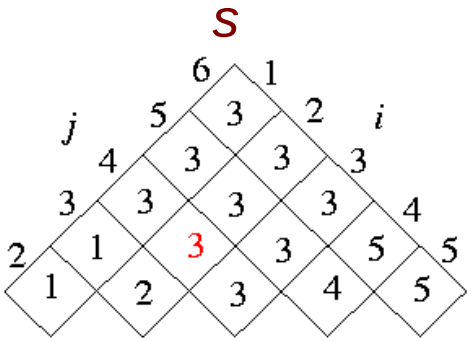
Bottom-Up DP Matrix-Chain Order

```
Matrix-Chain-Order(p)
1.  n = p.length – 1
2.  Let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
3.  for i = 1 to n
4.        m[i, i] = 0
5.  for  l = 2 to n       // l is the chain length
6.        for i = 1 to n – l +1
7.              j = i + l -1
8.              m[i, j] = ∞
9.              for k = i  to j -1
10.                  q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
11.                  if q < m[i, j]
12.                        m[i, j] = q
13.                        s[i, j] = k
14. return m and s
```

A_i dimension $p_{i-1} \times p_i$

matrix	dimension
A_1	30 * 35
A_2	35 * 15
A_3	15 * 5
A_4	5 * 10
A_5	10 * 20
A_6	20 * 25





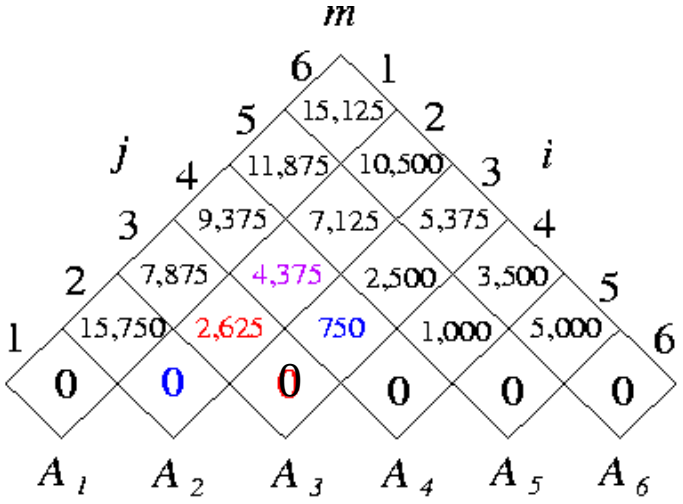
$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + p_1 p_2 p_4 = 0 + 750 + 35 \times 15 \times 10 = 6000. \\ m[2,3] + m[4,4] + p_1 p_3 p_4 = 2625 + 0 + 35 \times 5 \times 10 = 4375. \end{cases}$$

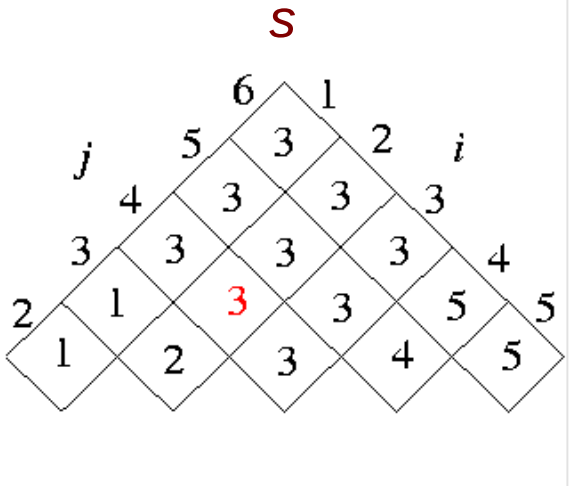
Constructing an Optimal Solution

- $s[i, j]$: value of k such that the optimal parenthesization of $A_i A_{i+1} \ldots A_j$ splits between A_k and A_{k+1}
- Optimal matrix $A_{1..n}$ multiplication: $A_{1..s[1, n]} A_{s[1, n] + 1..n}$
- **Exp:** call Print-Optimal-Parens(s, 1, 6): $((A_1 (A_2 A_3))((A_4 A_5) A_6))$

```
Print-Optimal-Parens(s, i, j)
1. if i == j
2.       print "A"_i
3. else print "("
4.       Print-Optimal-Parens(s, i, s[i, j])
5.       Print-Optimal-Parens(s, s[i, j] + 1, j)
6.       print ")"
```

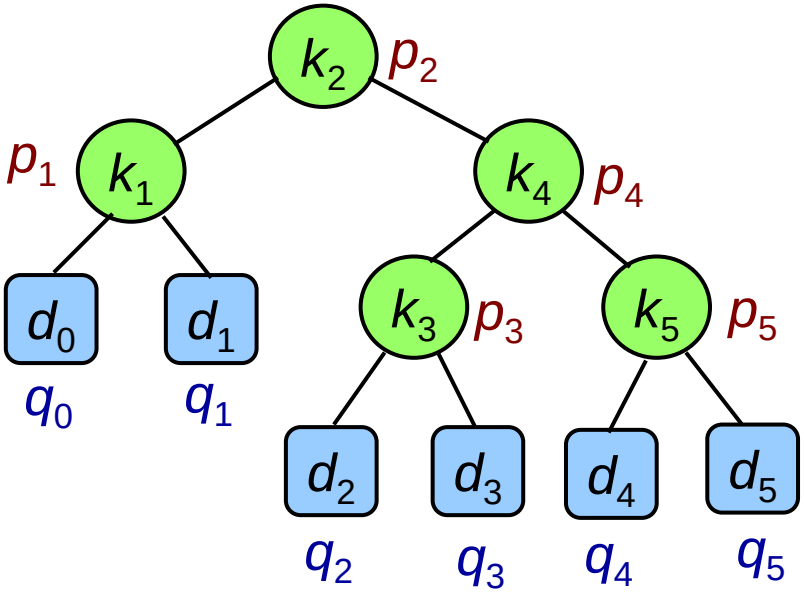
matrix	dimension
A_1	30 * 35
A_2	35 * 15
A_3	15 * 5
A_4	5 * 10
A_5	10 * 20
A_6	20 * 25

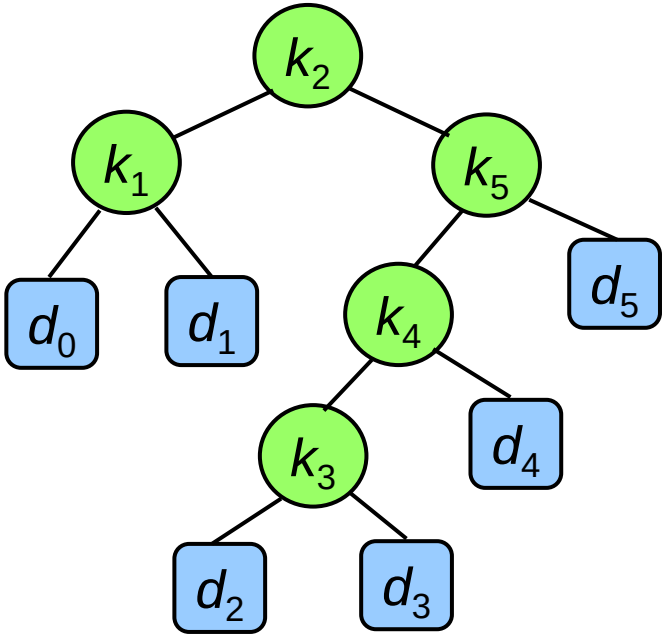




Optimal Binary Search Tree

- Given a sequence $K = <k_1, k_2, \ldots, k_n>$ of n distinct keys in sorted order ($k_1 < k_2 < \ldots < k_n$) and a set of probabilities $P = <p_1, p_2, \ldots, p_n>$ for searching the keys in K and $Q = <q_0, q_1, q_2, \ldots, q_n>$ for unsuccessful searches (corresponding to $D = <d_0, d_1, d_2, \ldots, d_n>$ of ***n+1*** distinct dummy keys with d_i representing all values between k_i and k_{i+1}), construct a binary search tree whose expected search cost is smallest.

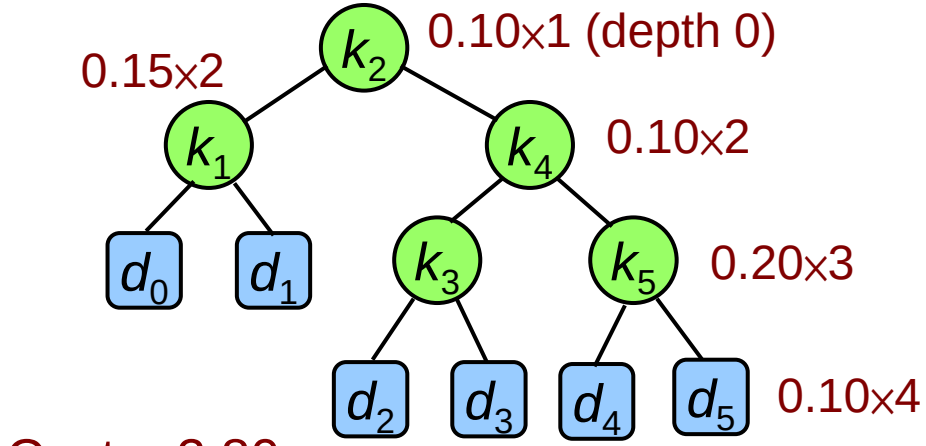




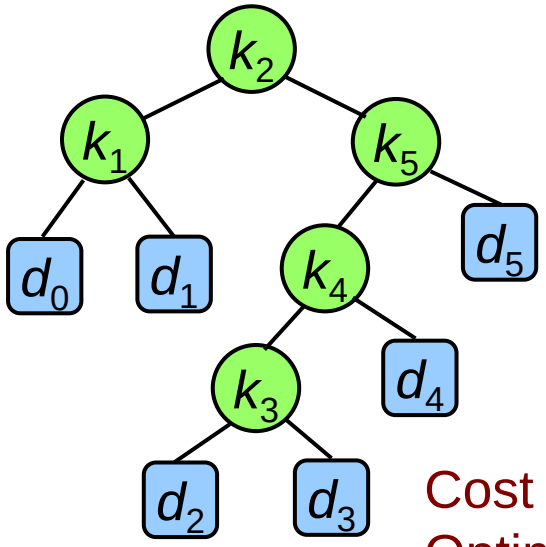
An Example

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

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$



Cost = 2.80



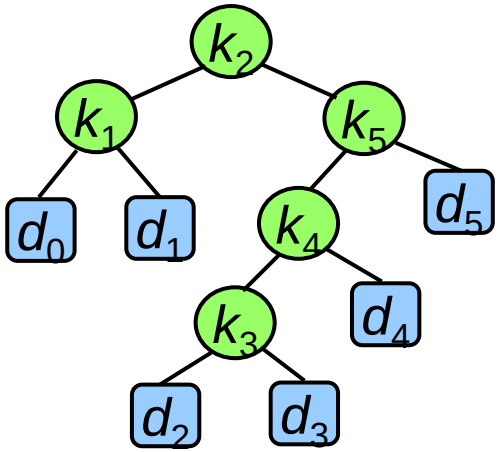
Cost = 2.75
Optimal!!

$$E[\text{search cost in } T] = \sum_{i=1}^{n} (depth_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n} (depth_T(d_i)+1)\cdot q_i$$

$$= 1 + \sum_{i=1}^{n} depth_T(k_i)\cdot p_i + \sum_{i=0}^{n} depth_T(d_i)\cdot q_i$$

Optimal Substructure

- If an optimal binary search tree T has a subtree T' containing keys $k_i, \ldots, k_j$, then this subtree T' must be optimal as well for the subproblem with keys $k_i, \ldots, k_j$ and dummy keys $d_{i-1}, \ldots, d_j$.
  - Given keys $k_i, \ldots, k_j$ with k_r ($i \le r \le j$) as the root, the left subtree contains the keys $k_i, \ldots, k_{r-1}$ (and dummy keys $d_{i-1}, \ldots, d_{r-1}$) and the right subtree contains the keys $k_{r+1}, \ldots, k_j$ (and dummy keys $d_r, \ldots, d_j$).
  - For the subtree with keys $k_i, \ldots, k_j$ with root k_i, the left subtree contains keys **$k_i, .., k_{i-1}$ (no key)** with the dummy key d_{i-1}.



Overlapping Subproblem: Recurrence

- $e[i, j]$: expected cost of searching an optimal binary search tree containing the keys $k_i, \ldots, k_j$.
  - Want to find $e[1, n]$.
  - $e[i, i-1] = q_{i-1}$ (only the dummy key d_{i-1}).
- If k_r ($i \le r \le j$) is the root of an optimal subtree containing keys $k_i, \ldots, k_j$ and let $w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$ then

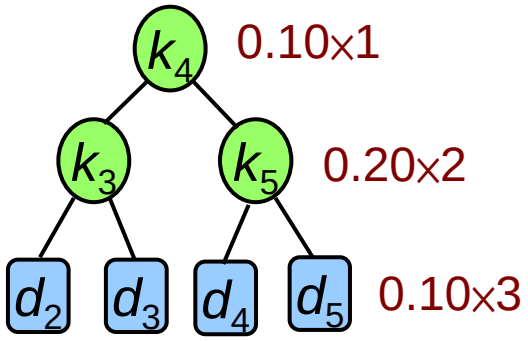
  $e[i, j] = \mathbf{p_r} + (e[i, r-1] + \mathbf{w(i, r-1)}) + (e[r+1, j] + \mathbf{w(r+1, j)})$

  $= e[i, r-1] + e[r+1, j] + \mathbf{w(i, j)}$

Node depths increase by 1 after merging two subtrees, and so do the costs

- Recurrence:

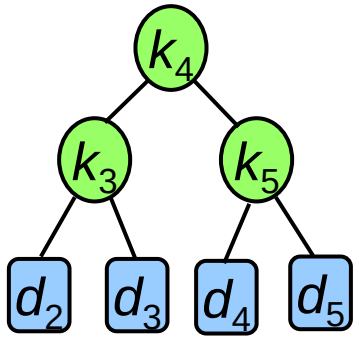
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$



Computing the Optimal Cost

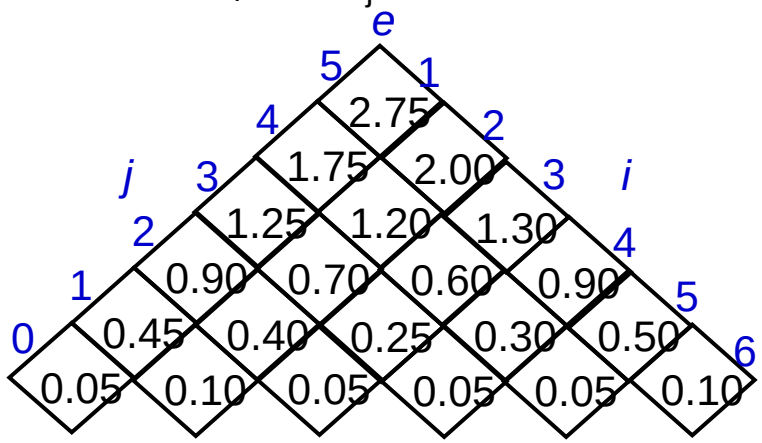
- Need a table $e[1..n+1, 0..n]$ for $e[i, j]$ (why $e[1, 0]$ and $e[n+1, n]$?)
- Apply the recurrence to compute $w(i, j)$ (why?)

$$w[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ w[i,j-1] + p_j + q_j & \text{if } i \le j \end{cases}$$




```
Optimal-BST(p, q, n)
1.  let e[1..n+1, 0..n], w[1..n+1, 0..n],
    and root[1..n, 1..n] be new tables
2.  for i = 1 to n + 1
3.      e[i, i-1] = q_{i-1}
4.      w[i, i-1] = q_{i-1}
5.  for l = 1 to n
6.      for i = 1 to n – l + 1
7.          j = i + l -1
8.          e[i, j] = ∞
9.          w[i, j] = w[i, j-1] + p_j + q_j
10.         for r = i  to j
11.             t = e[i, r-1] + e[r+1, j] + w[i, j]
12.             if t < e[i, j]
13.                 e[i, j] = t
14.                 root[i, j] = r
15. return e and root
```

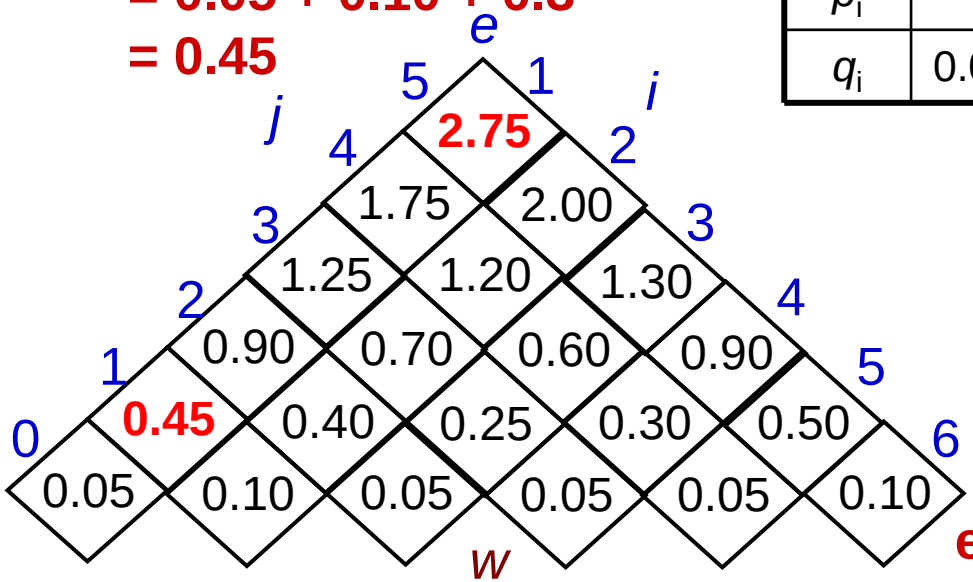
- *root*$[i, j]$: index r for which k_r is the root of an optimal search tree containing keys $k_i, \ldots, k_j$.

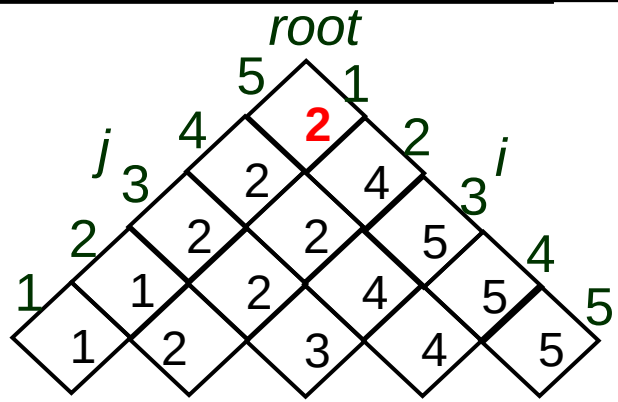


Example

e[1, 1] = e[1, 0] + e[2, 1] + w(1,1)
= 0.05 + 0.10 + 0.3
= 0.45

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





e[1, 5] = e[1, 1] + e[3, 5] + w(1,5)
= 0.45 + 1.30 + 1.00 = 2.75