Optimal Binary Search Trees

Contrast with optimal static ordering for lists.

1. Assume access probabilities are known:

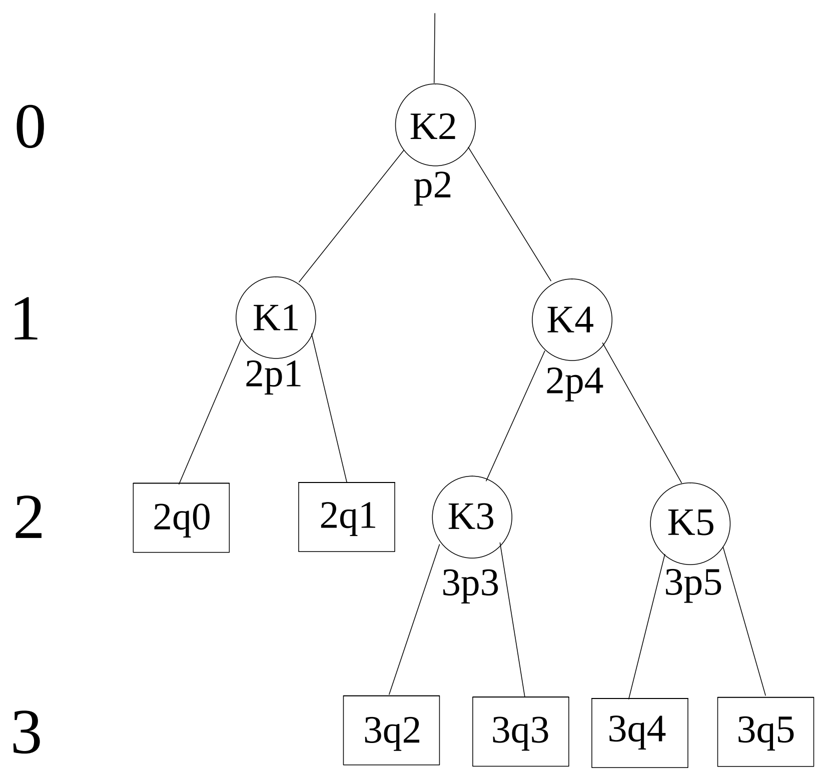
   keys are $K_1 < K_2 < . . . < K_n$

   p_i = probability of request for K_i
   q_i = probability of request with K_i < request < K_{i+1}
   q_0 = probability of request < K_1
   q_n = probability of request > K_n

2. Assume that levels are numbered with root at level 0. Minimize:

$$\sum_{1 \le j \le n} p_j(\text{Internal}_j + 1) + \sum_{0 \le k \le n} q_k(\text{External}_k)$$

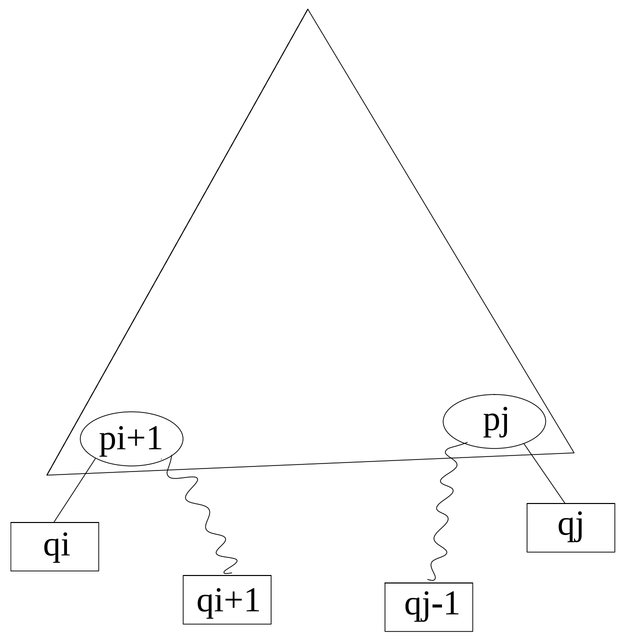
3. Example tree:



4. Solution is by dynamic programming:

   Principle of optimality - solution is not optimal unless the subtrees are optimal.

   Base case - empty tree, costs nothing to search.



   $c(i,j)$ - cost of subtree with keys $K_{i+1}, \ldots K_j$

   $c(i,j)$ always includes exactly $p_{i+1}, \ldots, p_j$ and $q_i, \ldots, q_j$

   $c(i,i) = 0$ - Base case, no keys, just misses for q_i (request between K_i and K_{i+1})

Recurrence for finding optimal subtrees:

$$c(i,j) = w(i,j) + \min_{i < k \leq j} (c(i,k-1) + c(k,j))$$

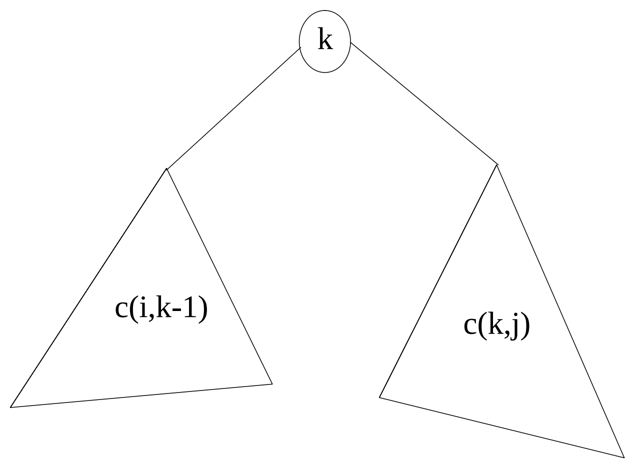
tries every possible root (‘‘k’’) for the subtree with keys $K_{i+1}, \ldots K_j$

$w(i,j) = p_{i+1} + \ldots + p_j + q_i + \ldots + q_j$ accounts for adding another probe for all keys:

Left: $p_{i+1} + \ldots + p_{k-1} + q_i + \ldots + q_{k-1}$

Right: $p_{k+1} + \ldots + p_j + q_k + \ldots + q_j$

Root: p_k



5. Implementation: A k-family is all cases for c(i,i+k). k-families are computed in ascending order from 1 to n.

Complexity: $O(n^2)$ space is obvious. $O(n^3)$ time from:

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

where k is the number of roots for each c(i,i+k) and n + 1 - k is the number of c(i,i+k) cases in family k.

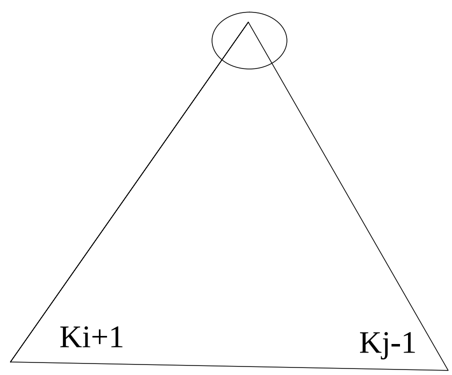
6. Traceback - besides having the minimum value for each c(i,j), it it necessary to save the subscript for the optimal root for c(i,j) as r[i][j].

   This also leads to Knuth's improvement:

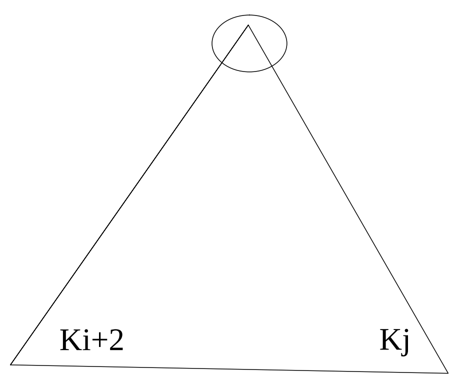
   Theorem: The root for the optimal tree c(i,j) must have a key with subscript no less than the key subscript for the root of the optimal tree for c(i,j-1) and no greater than the key subscript for the root of optimal tree c(i+1,j). (These roots are computed in the preceding family.)

   Proof:

   1. Consider adding p_j and q_j to tree for c(i,j-1). Optimal tree for c(i,j) must keep the same key at the root or use one further to the right.



   2. Consider adding p_{i+1} and q_i to tree for c(i+1,j). Optimal tree for c(i,j) must keep the same key at the root or use one further to the left.



7. Analysis of Knuth’s improvement.

Each c(i,j) case for k-family will vary in the number of roots to try, but overall time is reduced to $O(n^2)$ by using a telescoping sum:

$$\sum_{k=2}^{n} \sum_{i=0}^{n-k} (r[i+1][i+k] - r[i][i+k-1] + 1) = \sum_{k=2}^{n} \begin{pmatrix} r[1][k] - r[0][k-1] + 1 \\ + \\ r[2][1+k] - r[1][k] + 1 \\ + \\ r[3][2+k] - r[2][1+k] + 1 \\ + \ldots + \\ r[n-k+1][n] - r[n-k][n-1] + 1 \end{pmatrix}$$

$$= \sum_{k=2}^{n} (r[n-k+1][n] - r[0][k-1] + n - k + 1)$$

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