7. Modify algorithm BellmanFord so that it obtains the shortest paths, in addition to the lengths of these paths. What is the computing time of your algorithm?

5.5 OPTIMAL BINARY SEARCH TREES (*)

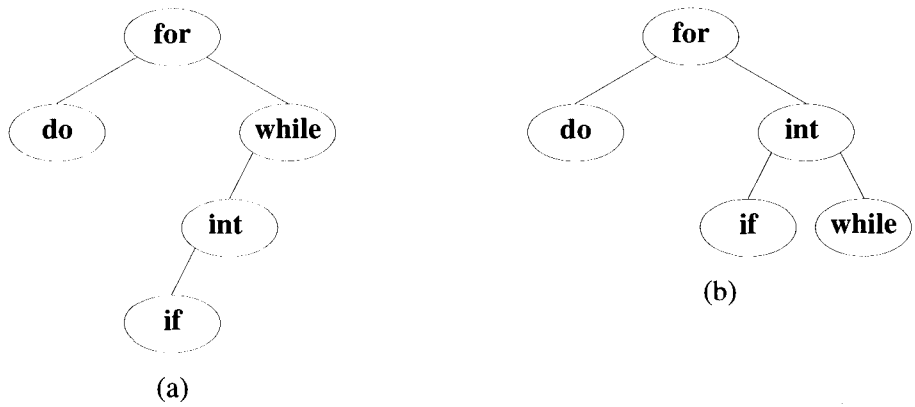

Figure 5.12 Two possible binary search trees

Given a fixed set of identifiers, we wish to create a binary search tree (see Section 2.3) organization. We may expect different binary search trees for the same identifier set to have different performance characteristics. The tree of Figure 5.12(a), in the worst case, requires four comparisons to find an identifier, whereas the tree of Figure 5.12(b) requires only three. On the average the two trees need 12/5 and 11/5 comparisons, respectively. For example, in the case of tree (a), it takes $1, 2, 2, 3$, and 4 comparisons, respectively, to find the identifiers **for, do, while, int,** and **if**. Thus the average number of comparisons is $\frac{1+2+2+3+4}{5} = \frac{12}{5}$. This calculation assumes that each identifier is searched for with equal probability and that no unsuccessful searches (i.e., searches for identifiers not in the tree) are made.

In a general situation, we can expect different identifiers to be searched for with different frequencies (or probabilities). In addition, we can expect unsuccessful searches also to be made. Let us assume that the given set of identifiers is $\{a_1, a_2, \ldots, a_n\}$ with $a_1 < a_2 < \cdots < a_n$. Let $p(i)$ be the probability with which we search for a_i. Let $q(i)$ be the probability that the identifier x being searched for is such that $a_i < x < a_{i+1}$, $0 \leq i \leq n$ (assume $a_0 = -\infty$ and $a_{n+1} = +\infty$). Then, $\sum_{0 \leq i \leq n} q(i)$ is the probability of

an unsuccessful search. Clearly, $\sum_{1 \le i \le n} p(i) + \sum_{0 \le i \le n} q(i) = 1$. Given this data, we wish to construct an optimal binary search tree for $\{a_1, a_2, \ldots, a_n\}$. First, of course, we must be precise about what we mean by an optimal binary search tree.

In obtaining a cost function for binary search trees, it is useful to add a fictitious node in place of every empty subtree in the search tree. Such nodes, called external nodes, are drawn square in Figure 5.13. All other nodes are internal nodes. If a binary search tree represents n identifiers, then there will be exactly n internal nodes and $n+1$ (fictitious) external nodes. Every internal node represents a point where a successful search may terminate. Every external node represents a point where an unsuccessful search may terminate.

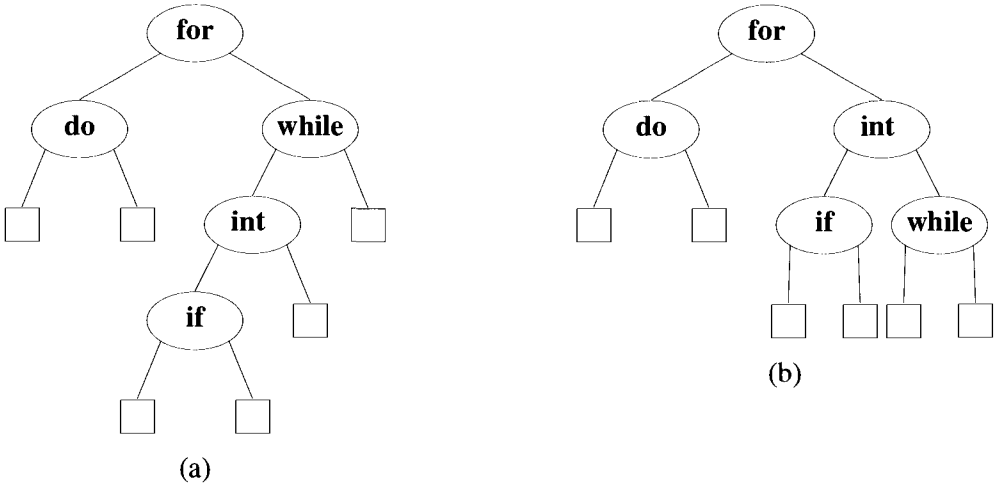

Figure 5.13 Binary search trees of Figure 5.12 with external nodes added

If a successful search terminates at an internal node at level l, then l iterations of the **while** loop of Algorithm 2.5 are needed. Hence, the expected cost contribution from the internal node for a_i is $p(i) * \text{level}(a_i)$.

Unsuccessful searches terminate with $t = 0$ (i.e., at an external node) in algorithm ISearch (Algorithm 2.5). The identifiers not in the binary search tree can be partitioned into $n+1$ equivalence classes $E_i, 0 \le i \le n$. The class E_0 contains all identifiers x such that $x < a_1$. The class E_i contains all identifiers x such that $a_i < x < a_{i+1}$, $1 \le i < n$. The class E_n contains all identifiers x, $x > a_n$. It is easy to see that for all identifiers in the same class E_i, the search terminates at the same external node. For identifiers in different E_i the search terminates at different external nodes. If the failure

node for E_i is at level l, then only $l-1$ iterations of the **while** loop are made. Hence, the cost contribution of this node is $q(i) * (level(E_i) - 1)$.

The preceding discussion leads to the following formula for the expected cost of a binary search tree:

$$\sum_{1 \le i \le n} p(i) * level(a_i) + \sum_{0 \le i \le n} q(i) * (level(E_i) - 1) \qquad (5.9)$$

We define an optimal binary search tree for the identifier set $\{a_1, a_2, \ldots, a_n\}$ to be a binary search tree for which (5.9) is minimum.

Example 5.17 The possible binary search trees for the identifier set (a_1, a_2, a_3) = (**do, if, while**) are given if Figure 5.14. With equal probabilities $p(i) = q(i) = 1/7$ for all i, we have

$$\begin{array}{llllll} cost(\text{tree a}) & = & 15/7 & cost(\text{tree b}) & = & 13/7 \\ cost(\text{tree c}) & = & 15/7 & cost(\text{tree d}) & = & 15/7 \\ cost(\text{tree e}) & = & 15/7 & & & \end{array}$$

As expected, tree b is optimal. With $p(1) = .5$, $p(2) = .1$, $p(3) = .05$, $q(0) = .15$, $q(1) = .1$, $q(2) = .05$ and $q(3) = .05$ we have

$$\begin{array}{llllll} cost(\text{tree a}) & = & 2.65 & cost(\text{tree b}) & = & 1.9 \\ cost(\text{tree c}) & = & 1.5 & cost(\text{tree d}) & = & 2.05 \\ cost(\text{tree e}) & = & 1.6 & & & \end{array}$$

For instance, $cost$(tree a) can be computed as follows. The contribution from successful searches is $3*0.5+2*0.1+0.05 = 1.75$ and the contribution from unsuccessful searches is $3 * 0.15 + 3 * 0.1 + 2 * 0.05 + 0.05 = 0.90$. All the other costs can also be calculated in a similar manner. Tree c is optimal with this assignment of p's and q's. □

To apply dynamic programming to the problem of obtaining an optimal binary search tree, we need to view the construction of such a tree as the result of a sequence of decisions and then observe that the principle of optimality holds when applied to the problem state resulting from a decision. A possible approach to this would be to make a decision as to which of the a_i's should be assigned to the root node of the tree. If we choose a_k, then it is clear that the internal nodes for $a_1, a_2, \ldots, a_{k-1}$ as well as the external nodes for the classes $E_0, E_1, \ldots, E_{k-1}$ will lie in the left subtree l of the root. The remaining nodes will be in the right subtree r. Define

$$cost(l) = \sum_{1 \le i < k} p(i) * \text{level}(a_i) + \sum_{0 \le i < k} q(i) * (\text{level}(E_i) - 1)$$

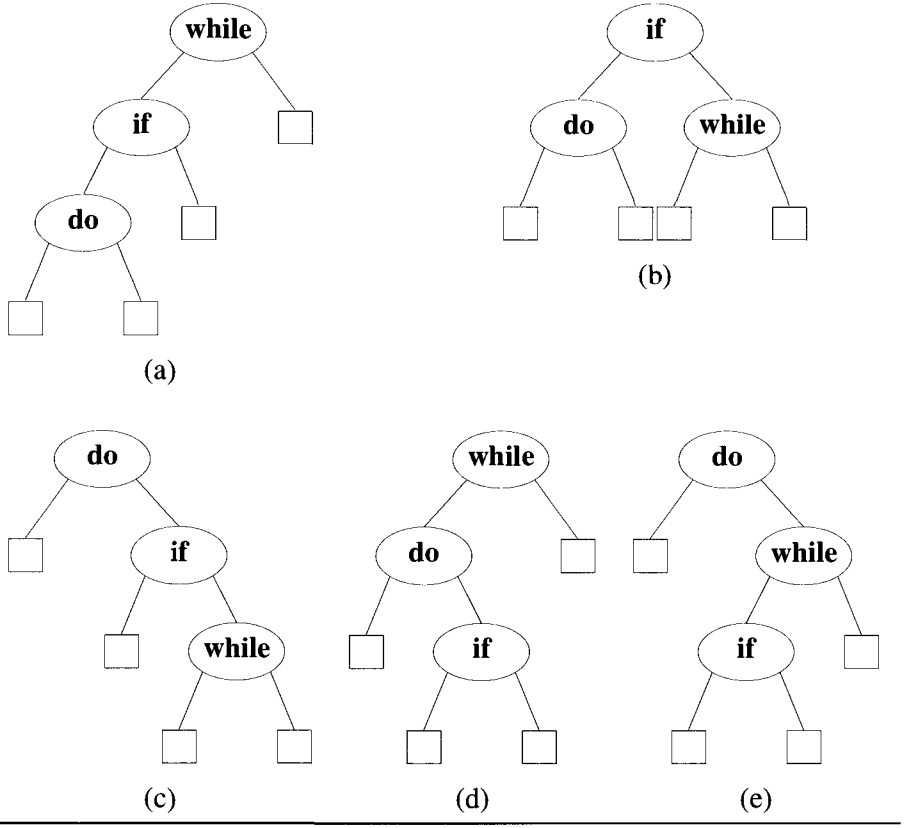

Figure 5.14 Possible binary search trees for the identifier set {**do, if, while**}

and

$$cost(r) = \sum_{k<i\leq n} p(i) * \text{level}(a_i) + \sum_{k<i\leq n} q(i) * (\text{level}(E_i) - 1)$$

In both cases the level is measured by regarding the root of the respective subtree to be at level 1.

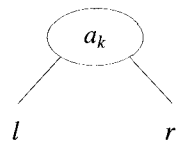

Figure 5.15 An optimal binary search tree with root a_k

Using $w(i,j)$ to represent the sum $q(i) + \sum_{l=i+1}^{j}(q(l) + p(l))$, we obtain the following as the expected cost of the search tree (Figure 5.15):

$$p(k) + cost(l) + cost(r) + w(0, k-1) + w(k, n) \tag{5.10}$$

If the tree is optimal, then (5.10) must be minimum. Hence, $cost(l)$ must be minimum over all binary search trees containing $a_1, a_2, \ldots, a_{k-1}$ and $E_0, E_1, \ldots, E_{k-1}$. Similarly $cost(r)$ must be minimum. If we use $c(i,j)$ to represent the cost of an optimal binary search tree t_{ij} containing $a_{i+1}, \ldots, a_j$ and $E_i, \ldots, E_j$, then for the tree to be optimal, we must have $cost(l) = c(0, k-1)$ and $cost(r) = c(k, n)$. In addition, k must be chosen such that

$$p(k) + c(0, k-1) + c(k, n) + w(0, k-1) + w(k, n)$$

is minimum. Hence, for $c(0,n)$ we obtain

$$c(0,n) = \min_{1\leq k\leq n} \{c(0, k-1) + c(k, n) + p(k) + w(0, k-1) + w(k, n)\} \tag{5.11}$$

We can generalize (5.11) to obtain for any $c(i,j)$

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

$$c(i,j) = \min_{i<k\leq j}\{c(i,k-1)+c(k,j)\}+w(i,j) \tag{5.12}$$

Equation 5.12 can be solved for $c(0,n)$ by first computing all $c(i,j)$ such that $j-i=1$ (note $c(i,i)=0$ and $w(i,i)=q(i)$, $0\leq i\leq n$). Next we can compute all $c(i,j)$ such that $j-i=2$, then all $c(i,j)$ with $j-i=3$, and so on. If during this computation we record the root $r(i,j)$ of each tree t_{ij}, then an optimal binary search tree can be constructed from these $r(i,j)$. Note that $r(i,j)$ is the value of k that minimizes (5.12).

Example 5.18 Let $n=4$ and $(a_1,a_2,a_3,a_4)=$ (**do, if, int, while**). Let $p(1:4)=(3,3,1,1)$ and $q(0:4)=(2,3,1,1,1)$. The p's and q's have been multiplied by 16 for convenience. Initially, we have $w(i,i)=q(i), c(i,i)=0$ and $r(i,i)=0$, $0\leq i\leq 4$. Using Equation 5.12 and the observation $w(i,j)=p(j)+q(j)+w(i,j-1)$, we get

$$\begin{aligned}
w(0,1) &= p(1)+q(1)+w(0,0)=8\\
c(0,1) &= w(0,1)+\min\{c(0,0)+c(1,1)\} = 8\\
r(0,1) &= 1\\
w(1,2) &= p(2)+q(2)+w(1,1) = 7\\
c(1,2) &= w(1,2)+\min\{c(1,1)+c(2,2)\} = 7\\
r(0,2) &= 2\\
w(2,3) &= p(3)+q(3)+w(2,2) = 3\\
c(2,3) &= w(2,3)+\min\{c(2,2)+c(3,3)\} = 3\\
r(2,3) &= 3\\
w(3,4) &= p(4)+q(4)+w(3,3) = 3\\
c(3,4) &= w(3,4)+\min\{c(3,3)+c(4,4)\} = 3\\
r(3,4) &= 4
\end{aligned}$$

Knowing $w(i,i+1)$ and $c(i,i+1)$, $0\leq i<4$, we can again use Equation 5.12 to compute $w(i,i+2)$, $c(i,i+2)$, and $r(i,i+2)$, $0\leq i<3$. This process can be repeated until $w(0,4)$, $c(0,4)$, and $r(0,4)$ are obtained. The table of Figure 5.16 shows the results of this computation. The box in row i and column j shows the values of $w(j,j+i)$, $c(j,j+i)$ and $r(j,j+i)$ respectively. The computation is carried out by row from row 0 to row 4. From the table we see that $c(0,4)=32$ is the minimum cost of a binary search tree for (a_1,a_2,a_3,a_4). The root of tree t_{04} is a_2. Hence, the left subtree is t_{01} and the right subtree t_{24}. Tree t_{01} has root a_1 and subtrees t_{00} and t_{11}. Tree t_{24} has root a_3; its left subtree is t_{22} and its right subtree t_{34}. Thus, with the data in the table it is possible to reconstruct t_{04}. Figure 5.17 shows t_{04}. □

	0	1	2	3	4
0	$w_{00} = 2$ $c_{00} = 0$ $r_{00} = 0$	$w_{11} = 3$ $c_{11} = 0$ $r_{11} = 0$	$w_{22} = 1$ $c_{22} = 0$ $r_{22} = 0$	$w_{33} = 1$ $c_{33} = 0$ $r_{33} = 0$	$w_{44} = 1$ $c_{44} = 0$ $r_{44} = 0$
1	$w_{01} = 8$ $c_{01} = 8$ $r_{01} = 1$	$w_{12} = 7$ $c_{12} = 7$ $r_{12} = 2$	$w_{23} = 3$ $c_{23} = 3$ $r_{23} = 3$	$w_{34} = 3$ $c_{34} = 3$ $r_{34} = 4$	
2	$w_{02} = 12$ $c_{02} = 19$ $r_{02} = 1$	$w_{13} = 9$ $c_{13} = 12$ $r_{13} = 2$	$w_{24} = 5$ $c_{24} = 8$ $r_{24} = 3$		
3	$w_{03} = 14$ $c_{03} = 25$ $r_{03} = 2$	$w_{14} = 11$ $c_{14} = 19$ $r_{14} = 2$			
4	$w_{04} = 16$ $c_{04} = 32$ $r_{04} = 2$				

Figure 5.16 Computation of $c(0,4)$, $w(0,4)$, and $r(0,4)$

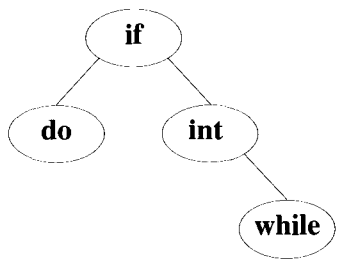

Figure 5.17 Optimal search tree for Example 5.18

The above example illustrates how Equation 5.12 can be used to determine the c's and r's and also how to reconstruct t_{0n} knowing the r's. Let us examine the complexity of this procedure to evaluate the c's and r's. The evaluation procedure described in the above example requires us to compute $c(i,j)$ for $(j-i) = 1, 2, \ldots, n$ in that order. When $j - i = m$, there are $n - m + 1$ $c(i,j)$'s to compute. The computation of each of these $c(i,j)$'s requires us to find the minimum of m quantities (see Equation 5.12). Hence, each such $c(i,j)$ can be computed in time $O(m)$. The total time for all $c(i,j)$'s with $j - i = m$ is therefore $O(nm - m^2)$. The total time to evaluate all the $c(i,j)$'s and $r(i,j)$'s is therefore

$$\sum_{1 \le m \le n} (nm - m^2) = O(n^3)$$

We can do better than this using a result due to D. E. Knuth which shows that the optimal k in Equation 5.12 can be found by limiting the search to the range $r(i, j-1) \le k \le r(i+1, j)$. In this case the computing time becomes $O(n^2)$ (see the exercises). The function OBST (Algorithm 5.5) uses this result to obtain the values of $w(i,j)$, $r(i,j)$, and $c(i,j)$, $0 \le i \le j \le n$, in $O(n^2)$ time. The tree t_{0n} can be constructed from the values of $r(i,j)$ in $O(n)$ time. The algorithm for this is left as an exercise.

EXERCISES

1. Use function OBST (Algorithm 5.5) to compute $w(i,j)$, $r(i,j)$, and $c(i,j)$, $0 \le i < j \le 4$, for the identifier set $(a_1, a_2, a_3, a_4) =$ (**cout, float, if, while**) with $p(1) = 1/20$, $p(2) = 1/5$, $p(3) = 1/10$, $p(4) = 1/20$, $q(0) = 1/5$, $q(1) = 1/10$, $q(2) = 1/5$, $q(3) = 1/20$, and $q(4) = 1/20$. Using the $r(i,j)$'s, construct the optimal binary search tree.

2. (a) Show that the computing time of function OBST (Algorithm 5.5) is $O(n^2)$.

   (b) Write an algorithm to construct the optimal binary search tree given the roots $r(i,j), 0 \le i < j \le n$. Show that this can be done in time $O(n)$.

3. Since often only the approximate values of the p's and q's are known, it is perhaps just as meaningful to find a binary search tree that is nearly optimal. That is, its cost, Equation 5.9, is almost minimal for the given p's and q's. This exercise explores an $O(n \log n)$ algorithm that results in nearly optimal binary search trees. The search tree heuristic we use is

```
1   Algorithm OBST(p, q, n)
2   // Given n distinct identifiers a_1 < a_2 < ··· < a_n and probabilities
3   // p[i], 1 ≤ i ≤ n, and q[i], 0 ≤ i ≤ n, this algorithm computes
4   // the cost c[i, j] of optimal binary search trees t_ij for identifiers
5   // a_{i+1}, ..., a_j. It also computes r[i, j], the root of t_ij.
6   // w[i, j] is the weight of t_ij.
7   {
8       for i := 0 to n − 1 do
9       {
10          // Initialize.
11          w[i, i] := q[i]; r[i, i] := 0; c[i, i] := 0.0;
12          // Optimal trees with one node
13          w[i, i + 1] := q[i] + q[i + 1] + p[i + 1];
14          r[i, i + 1] := i + 1;
15          c[i, i + 1] := q[i] + q[i + 1] + p[i + 1];
16      }
17      w[n, n] := q[n]; r[n, n] := 0; c[n, n] := 0.0;
18      for m := 2 to n do  // Find optimal trees with m nodes.
19          for i := 0 to n − m do
20          {
21              j := i + m;
22              w[i, j] := w[i, j − 1] + p[j] + q[j];
23              // Solve 5.12 using Knuth's result.
24              k := Find(c, r, i, j);
25                  // A value of l in the range r[i, j − 1] ≤ l
26                  // ≤ r[i + 1, j] that minimizes c[i, l − 1] + c[l, j];
27              c[i, j] := w[i, j] + c[i, k − 1] + c[k, j];
28              r[i, j] := k;
29          }
30      write (c[0, n], w[0, n], r[0, n]);
31  }
```

```
1   Algorithm Find(c, r, i, j)
2   {
3       min := ∞;
4       for m := r[i, j − 1] to r[i + 1, j] do
5           if (c[i, m − 1] + c[m, j]) < min then
6           {
7               min := c[i, m − 1] + c[m, j]; l := m;
8           }
9       return l;
10  }
```

Algorithm 5.5 Finding a minimum-cost binary search tree