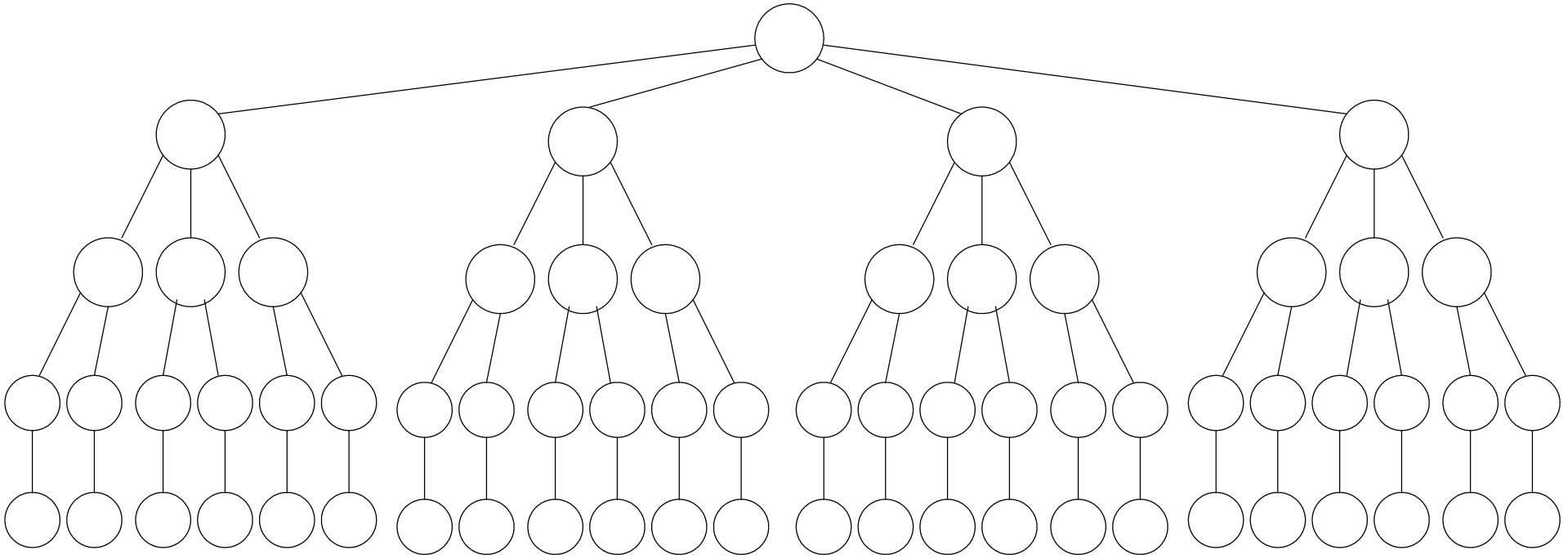
1. Backtracking searched the tree in depth-first order from "left" to "right." In the "general" state-space tree below number the tree nodes with the Backtracking order.



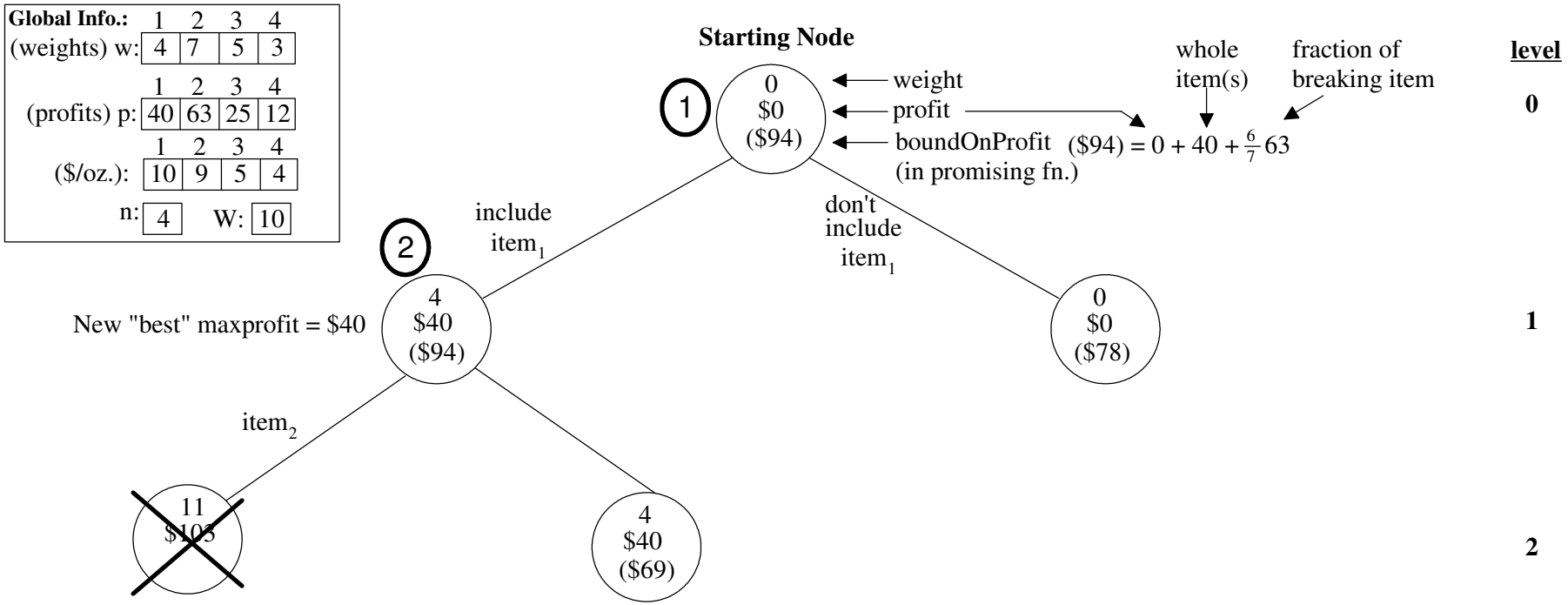
2. Why might we do better to first follow branchs of the tree that we think might lead to the best solution?

3. In the *Best-First search with Branch-and-Bound* (chapter 6) approach:

- does not limit us to any particular search pattern in the state-space tree
- calculates a "bound" estimate for each node that indicates the "best" possible solution that could be obtained from any node in the subtree rooted at that node, i.e., how "promising" following that node might be
- expands the most promising node first by visiting its children

For example, the 0-1 Knapsack problem can use the Fractional Knapsack calculation to "bound" the best possible solution in the subtree.

a) Complete the state-space tree for the following 0-1 Knapsack problem instance with four items and a knapsack weight limit of W=10 oz. **Indicate the order in which the nodes are expanded.**



b) "Problems" with *Best-First search with Branch-and-Bound* (chapter 6) approach:

- we must store information about each node waiting to be expanded (backtracking kept a single, global state)
- we need to find the "best" node to expand

What data structure would aid in finding the "best" node to expand?

4. In the *Best-First search with Branch-and-Bound* (chapter 6) approach:
- does not limit us to any particular search pattern in the state-space tree
- calculates a "bound" estimate for each node that indicates the "best" possible solution that could be obtained from any node in the subtree rooted at that node, i.e., how "promising" following that node might be
- expands the most promising node first by visiting its children

Traveling Salesperson Problem (TSP) -- Find an optimal (i.e., minimum length) tour when at least one tour exists. A *tour* (or *Hamiltonian circuit*) is a path from a vertex back to itself that passes through each of the other vertices exactly once. (Since a tour visits every vertice, it does not matter where you start.)
a) For this graph, what are the optimal tour and its length?

To solve a problem using branch-and-bound or backtracking, you need to answer the following questions:
b) What should the state-space tree look like? (i.e.,What would the "for each child c" loop iterate over?)

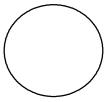
Global Problem-Instance Information

W:

From \ To	1	2	3	4
1	0	2	9	∞
2	1	0	6	4
3	∞	7	0	8
4	6	3	∞	0

n: 4

Starting Node



c) What state information is needed at each node?

d) How can we determine a bound to guide our best-first search of the state-space tree? Hints: We want:

- a lower-bound to allow pruning in a minimization problem like TSP
- consider what your greedy TSP algorithm used for its greedy criteria

e) What information is needed by our bound calculation function?

f) Customize the Breadth-first-branch-and-bound optimization template (p. 248) for the TSP problem.

```
void bread-first-branch-and-bound( tree T, number & best) {

    priorityQueue Q;
    treeNode u, v;

    initialize(Q)
    v = root of T
    enqueue(Q, v)
    best = value(v)
    while (not emtpy(Q)) do
        dequeue(Q, v)
        for (each child u of v) do
            if  value of (u) is better than best solution
                 best = value(u)                          # remember as the best solution
            end if
            if bound(u) is better than best then          # u could lead to a better solution
                enqueue(Q, u)
            end if
        end for
} // end checknode
```