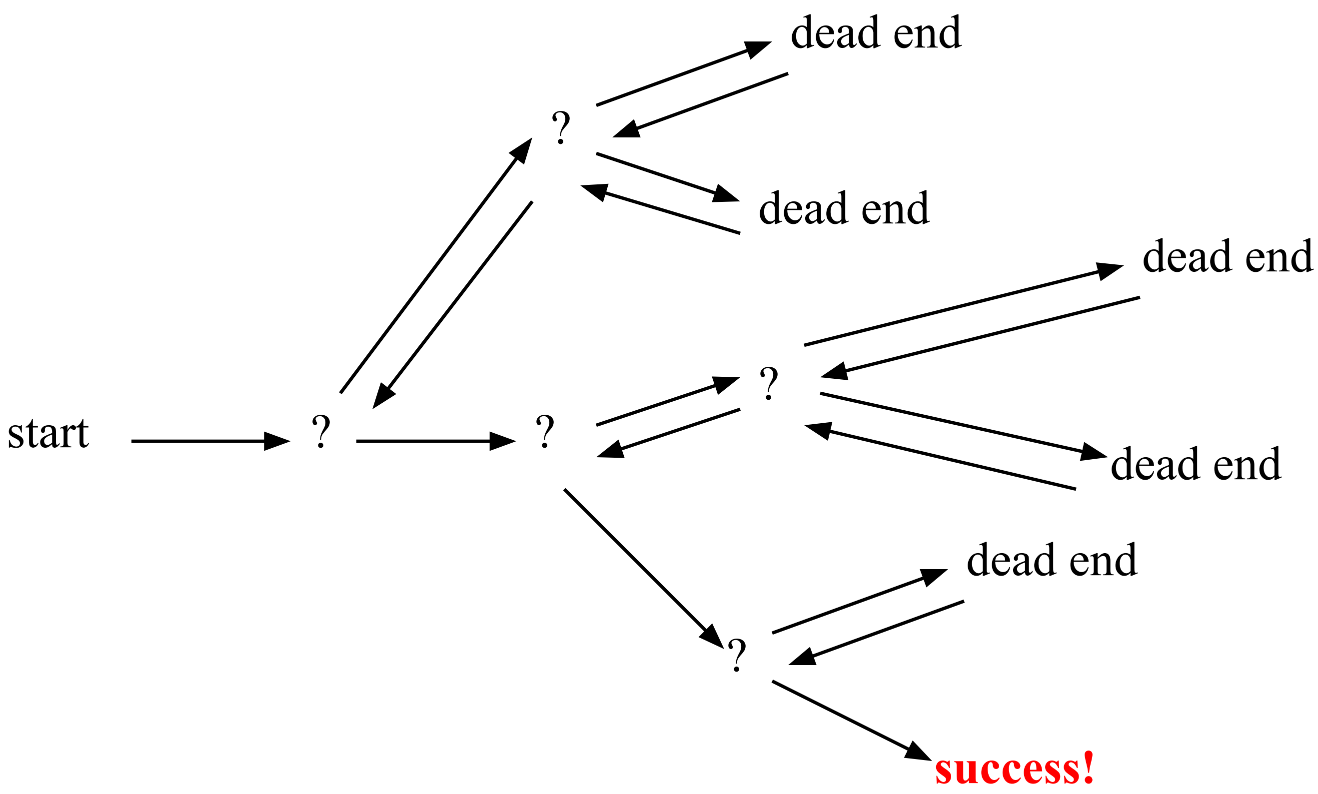
Backtracking

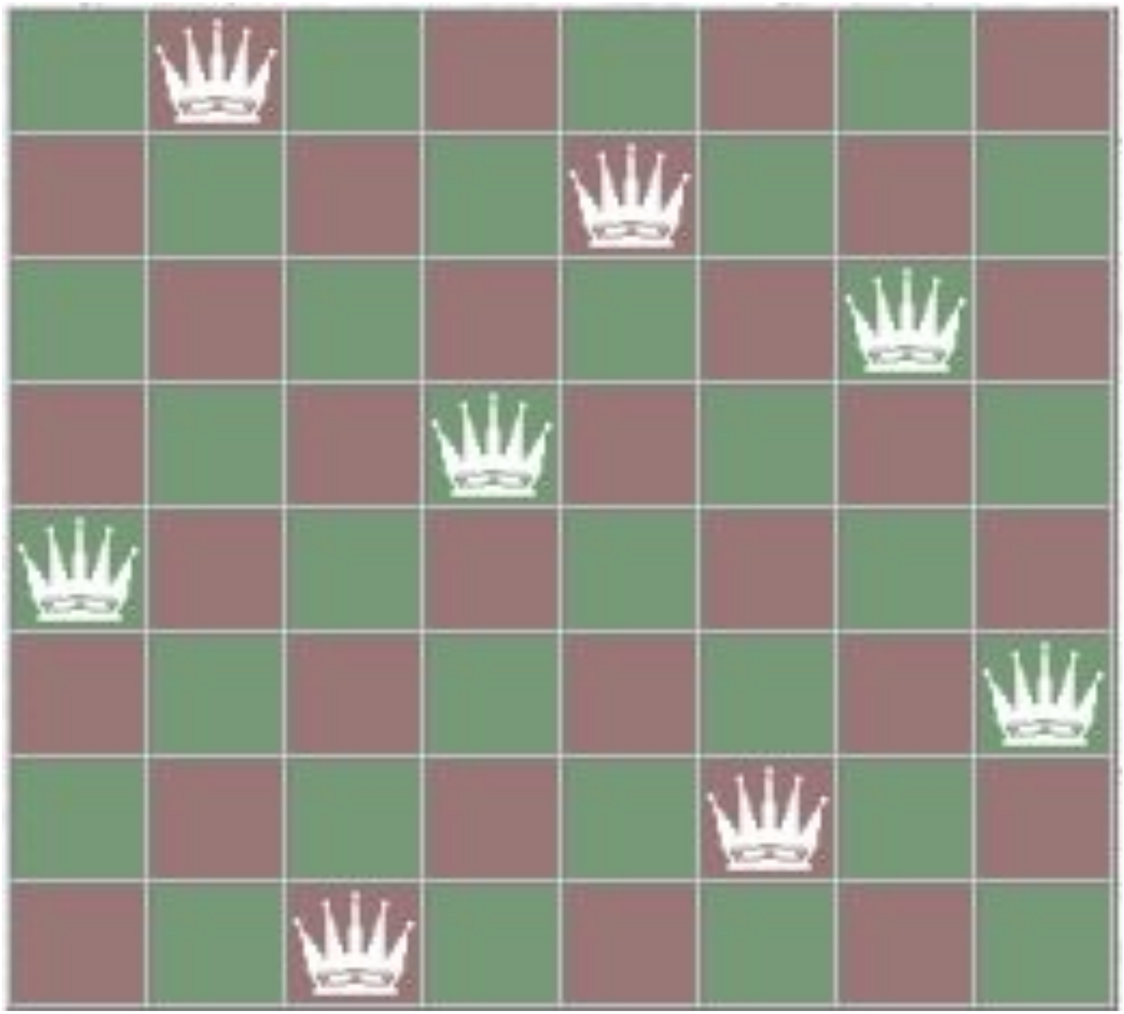
Backtracking

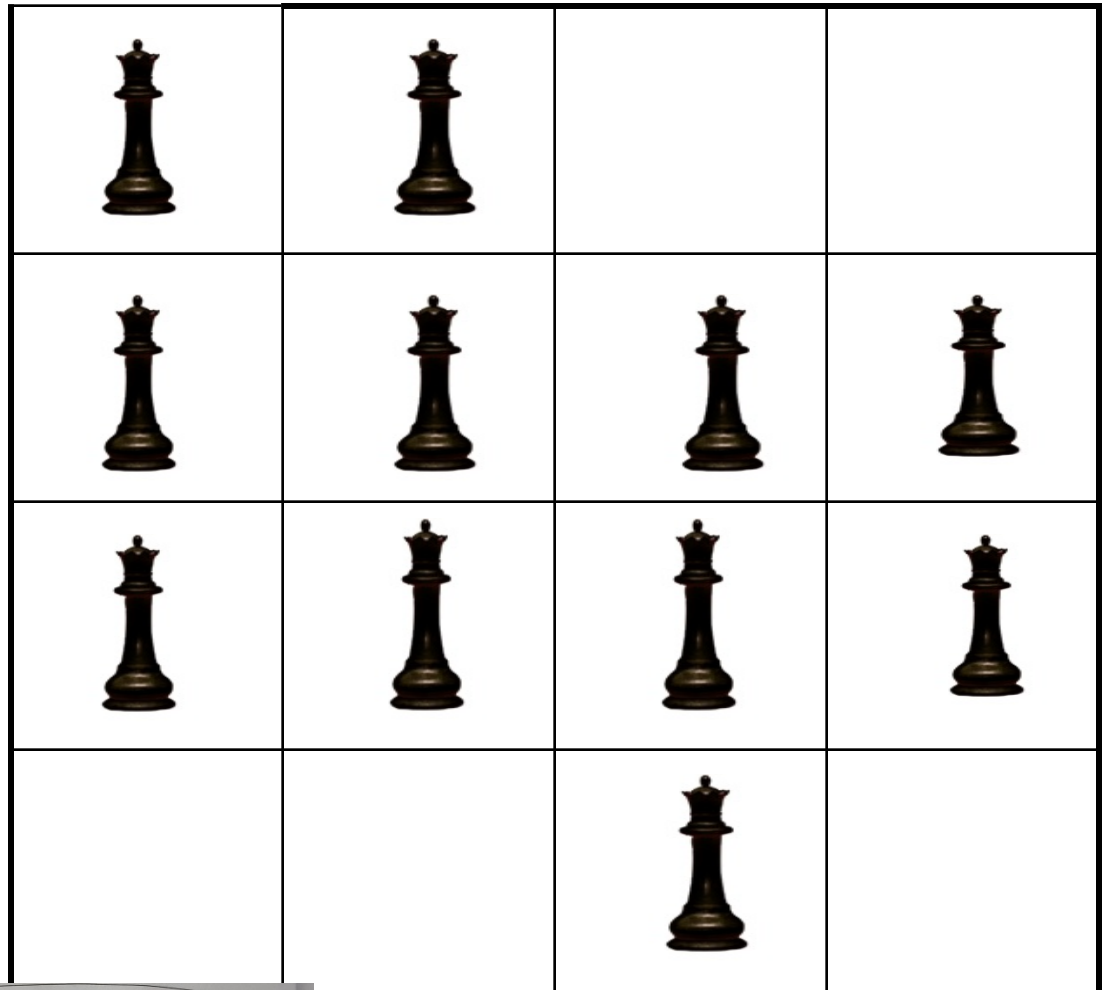
- Suppose you have to make a series of *decisions,* among various *choices,* where
  - You don't have enough information to know what to choose
  - Each decision leads to a new set of choices
  - Some sequence of choices (possibly more than one) may be a solution to your problem
- Backtracking is a methodical way of trying out various sequences of decisions, until you find one that "works"

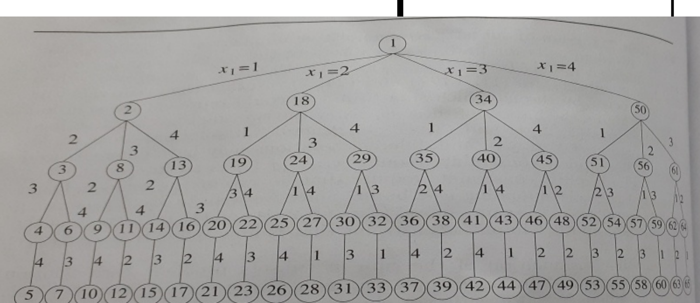
Backtracking



8 queen






x1=1
x1=2
x1=3
x1=4

Backtracking

Sum of Subsets

Sum of subsets

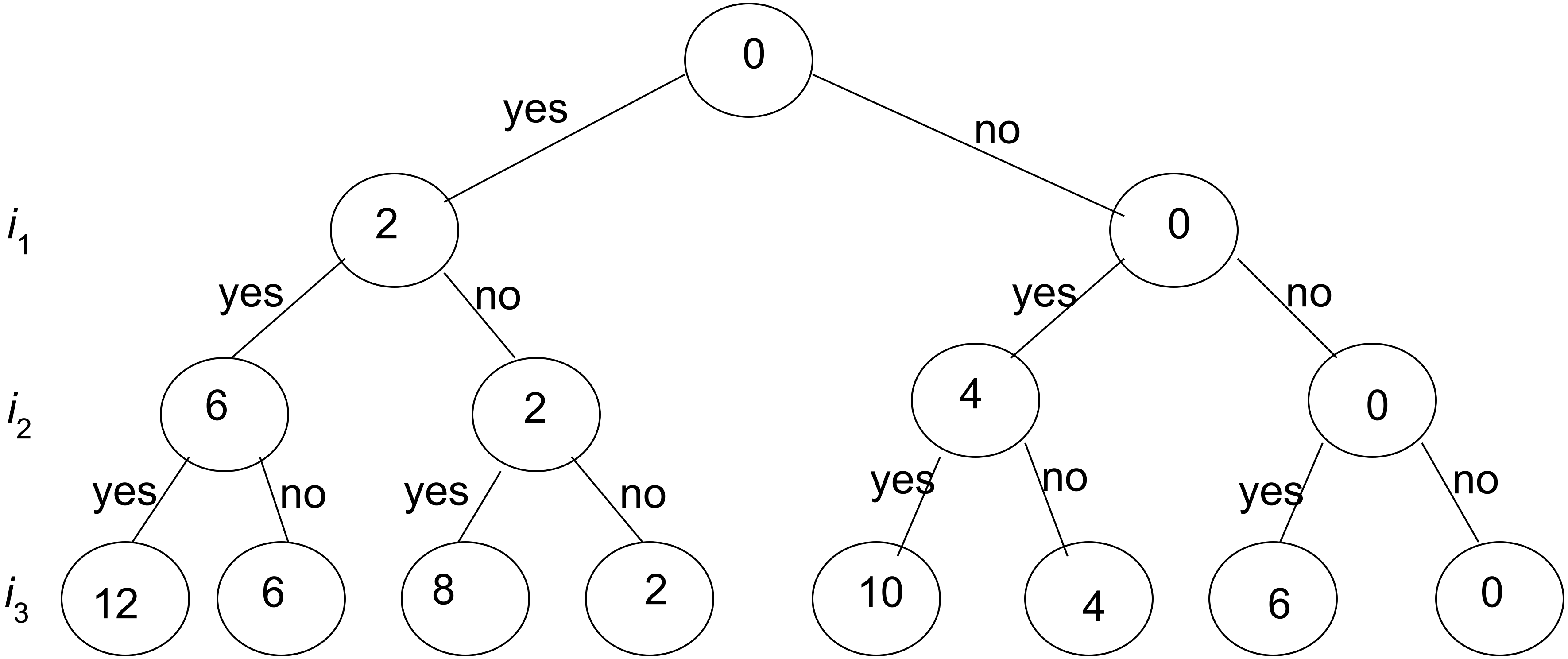
- **Problem**: Given *n* positive integers $w_{1,} \ldots w_n$ and a positive integer S. Find all subsets of $w_{1,} \ldots w_n$ that sum to S. (Can't repeat W_i)
- **Example**:
  n=3, S=6, and w_1=2, w_2=4, w_3=6
- **Solutions**:
  {2,4} and {6}*(variable length tuple)*
- {1,1,0} and {0,0,1}(*fixed length)*

Sum of subsets

- We will assume a binary state space tree.
- The nodes at depth 1 are for including (yes, no) item 1, the nodes at depth 2 are for item 2, etc.
- The left branch includes w_i, and the right branch excludes w_i.
- The nodes contain the sum of the weights included so far

Sum of subset Problem:
State SpaceTree for 3 items

$w_1 = 2,\ \ w_2 = 4,\ \ w_3 = 6$ and $S = 6$



The sum of the included integers is stored at the node.

Backtracking

- **Definition**: We call a node *nonpromising* if it cannot lead to a feasible (or optimal) solution, otherwise it is *promising*

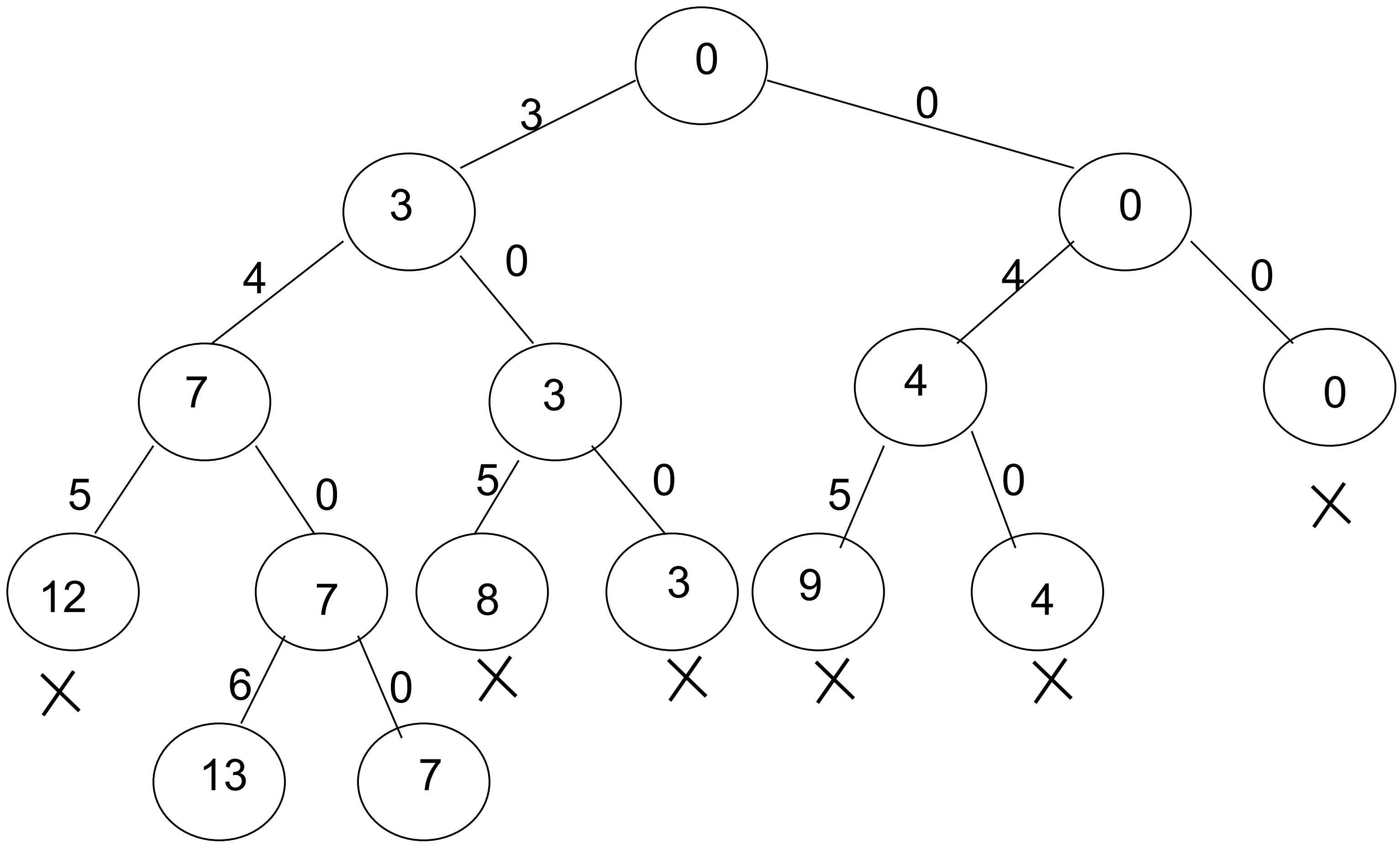
- **Main idea**: Backtracking consists of doing a DFS of the state space tree, checking whether each node is promising and if the node is nonpromising backtracking to the node's parent

Backtracking

- The state space tree consisting of expanded nodes only is called the *pruned state space tree*
- The following slide shows the pruned state space tree for the sum of subsets example
- There are only 15 nodes in the pruned state space tree
- The full state space tree has 31 nodes

A Pruned State Space Tree (find all solutions)

$w_1 = 3$, $w_2 = 4$, $w_3 = 5$, $w_4 = 6$; $S = 13$



Sum of subsets problem

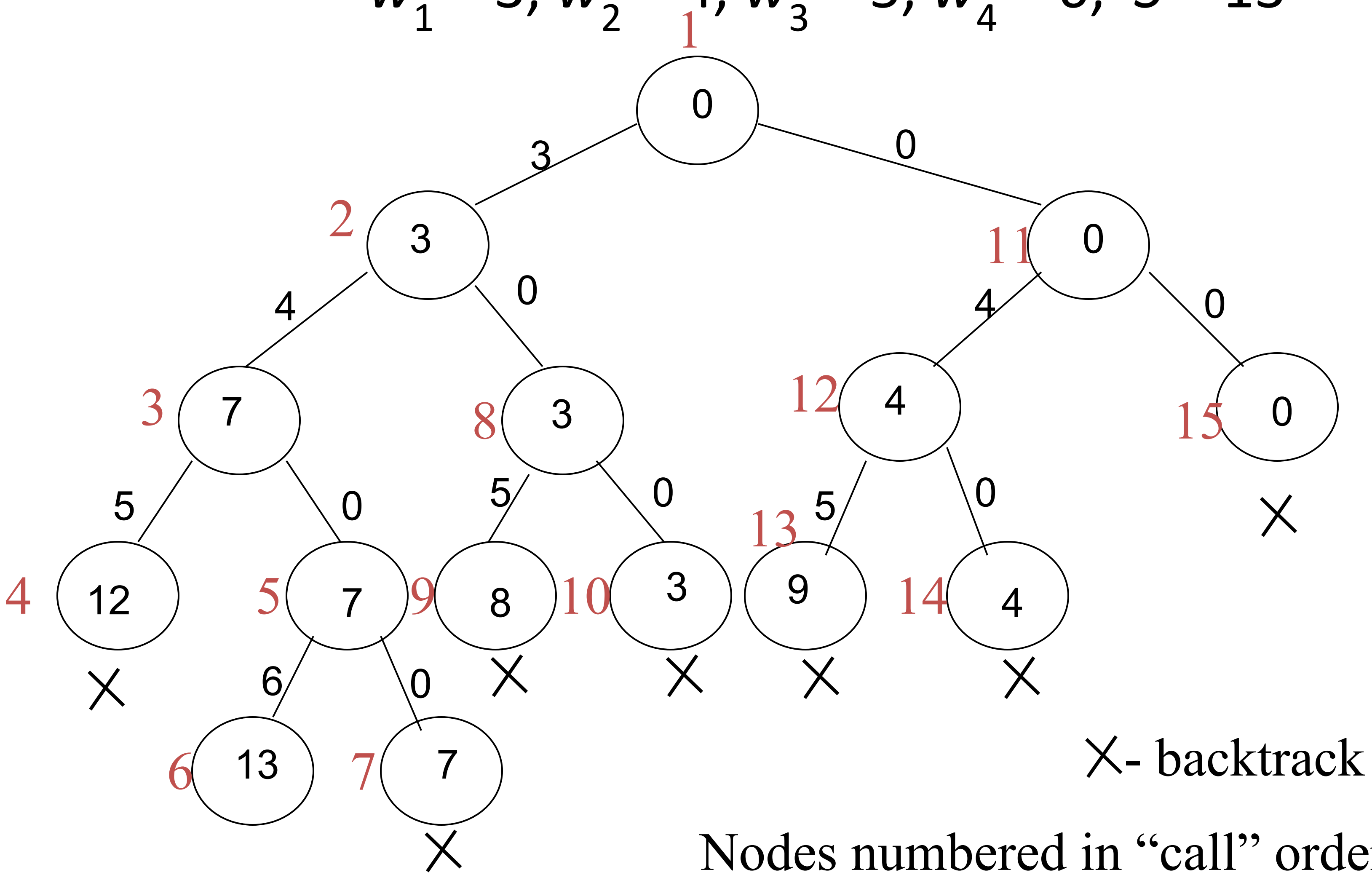
Sum of subsets – when is a node "promising"?

- Consider a node at depth i
- *weightSoFar* = weight of node, i.e., sum of numbers included in partial solution node represents

- *totalPossibleLeft* = weight of the remaining items i+1 to n (for a node at depth i)
- A node at depth i is non-promising
  if (*weightSoFar* + *totalPossibleLeft* < S)
  or (*weightSoFar* + w[i+1] > S)
- To be able to use this "promising function" the w_i must be sorted in non-decreasing order

A Pruned State Space Tree

$w_1 = 3, w_2 = 4, w_3 = 5, w_4 = 6; \ S = 13$



Nodes numbered in “call” order

```
sumOfSubsets ( i, weightSoFar, totalPossibleLeft )
    1) if (promising ( i ))                          //may lead to solution
   2) then if ( weightSoFar == S )
   3)       then print include[ 1 ] to include[ i ]      //found solution
   4)      else       //expand the node when weightSoFar < S
   5)             include [ i + 1 ] = "yes"                //try including
   6)             sumOfSubsets ( i + 1,
                          weightSoFar + w[i + 1],
                      totalPossibleLeft - w[i + 1] )
   7)             include [ i + 1 ] = "no"                 //try excluding
   8)             sumOfSubsets ( i + 1,   weightSoFar ,
                          totalPossibleLeft - w[i + 1] )

boolean promising (i )
   1) return ( weightSoFar + totalPossibleLeft ≥ S)  &&
        ( weightSoFar == S  ||  weightSoFar +  w[i + 1] ≤ S )
```

Prints all solutions!

Initial call sumOfSubsets$(0, 0, \sum_{i=1}^{n} w_i)$