Section 4.5: Backtracking

Backtracking (as a general technique)

- finds all (or some) solutions to a computational problem
- typically builds candidate solutions incrementally by trying each possible value
- to conceptually keep track of candidate solutions, each one is represented as a node in a tree (called the **search tree**)
- the algorithm traverses the tree in depth first order
- a process called **pruning** is used to eliminate branches or subtrees of the search tree when a partial candidate (and all choices thereafter) will not result in a solution

Suppose we need to solve a problem where a solution is of the form x[1], x[2], …, x[n] and the values of x[i] are elements of a set S of size n. (The elements of S could be any type and depend on the problem being solved.)

The general form for the backtracking algorithm is as follows:

```
// assume the scope of the array x is global
backtrack(n){
  rbacktrack(1, n)
}

rbacktrack(k, n){           // fills position k of array x
  for each  x[k]  in  S     // systematically try each value of S
    if (bound(k))           // pruning step
      if (k == n){
        // output solution
        // stop if only one solution is desired
      }
      else
        rbacktrack(k+1, n) // recursively fill position k+1
}

bound(k){
  // a function that assumes that x[1], x[2],..., x[k-1]  is a
  // partial solution and x[k] has been assigned a value to try
  // returns true if  x[1], x[2],..., x[k] is a partial solution
  // returns false otherwise
}
```