General Framework for a Backtracking Algorithm

- let us assume that the solution takes the form

$$x[1], x[2], \ldots, x[n]$$

where $x[i] \in S$

Observation: in the n queens problem $S = \{1, 2, 3, \ldots, n\}$

```
backtrack(n)
rbacktrack(1,n)

rbacktrack(k,n)
for each x[k] ∈ S
   if bound(k) == true
      if k==n
         output a solution; stop here if only one solution is desired
      else //k<n
         rbacktrack(k+1,n)

bound(k)
//give pseudocode implementation
```

→

$x[1], x[2], \ldots, x[k], \ldots, x[n]$

- function bound(k)
  - assume that $x[1], x[2], \ldots, x[k-1]$ is a partial feasible solution and that $x[k]$ has been assigned some value
  - return
    - true if $x[1], x[2], \ldots, x[k]$ is a partial feasible solution
    - false otherwise

- goal: design an efficient bound() function to eliminate many potential nodes (i.e. non promising nodes) from the search tree.