Backtracking

- The principal idea in backtracking is to construct solutions one component at a time & evaluate such partially constructed candidates as follows:

→ If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component.

→ If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered, in this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

- It is convenient to implement this kind of processing by constructing a tree of choices being made, called the state-space tree.
- Its root represents an initial state before the search for a solution begins.
- The nodes of the first level in the tree represent the choices made for the first component of a solution, the nodes of the second level represent the choices for the second component, & so on.
- A node in a state-space tree is said to be promising if it corresponds to a partially constructed solution that may still lead to a complete solution; otherwise, it is called nonpromising.
- Leaves represent either nonpromising dead ends or complete solutions found by the algorithm.
- If the current node is promising, its child is generated by adding the first remaining legitimate option for the next component of a solution, & the processing moves to this child.
- If the current node turns out to be nonpromising, the algorithm backtracks to the node's parent to consider the next possible option

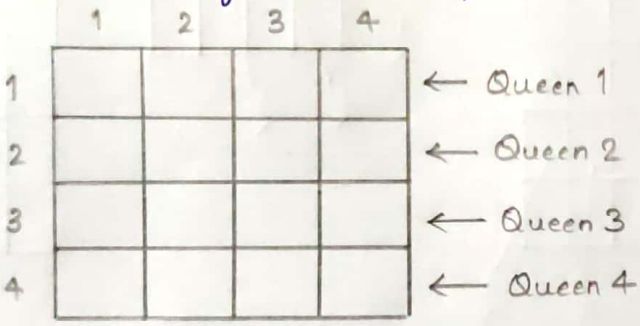
for its last component; if there is no such option, it backtracks one more level up the tree & so on.

- Finally, if the algorithm reaches a complete solution to the problem, it either stops or backtracks to continue searching for other possible solutions.

Example : n- Queens Problem

- The problem is to place n queens on an n-by-n chessboard so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

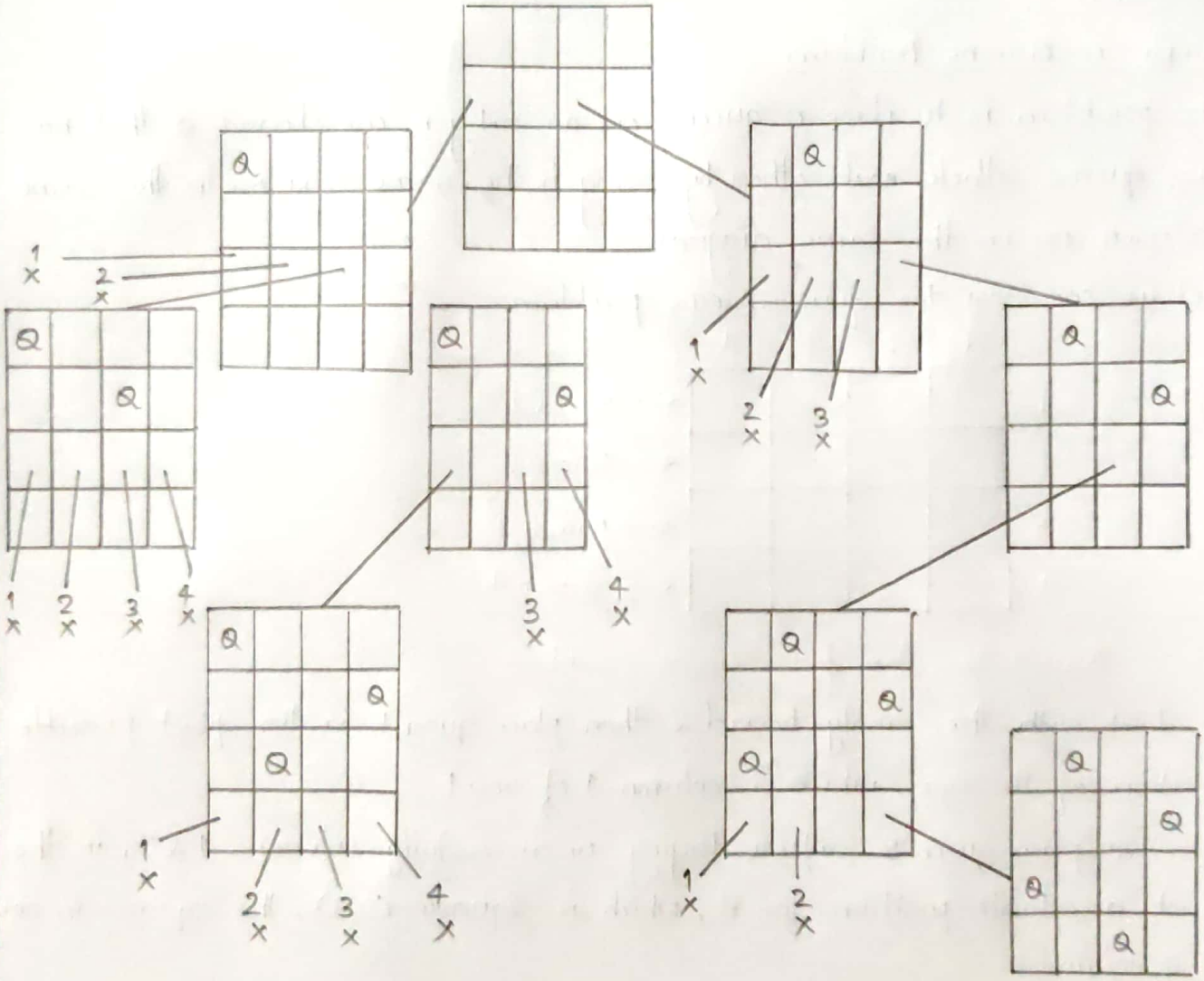
Let us consider the four-queens problem.



Board for the four queens problem

- We start with the empty board & then place queen1 in the first possible position of its row, which is column 1 of row 1.
- Then we place queen 2, after taying unsuccessfully columns 1 & 2, in the first acceptable position for it, which is square (2,3), the square in row 2 & column 3.
- This proves to be a dead end because there is no acceptable position for queen 3.
- So, the algorithm backtracks & puts queen 2 in the next possible position at (2,4).
- Then queen 3 is placed at (3,2), which proves to be another dead end.

- The algorithm then backtracks all the way to queen 1 & moves it to (1,2). Queen 2 then goes to (2,4), queen 3 to (3,1) & queen 4 to (4,3), which is a solution to the problem.
- The state space tree of this search is given below.



- An output of a backtracking algorithm can be thought of as an n-tuple $(x_1, x_2, .. x_n)$ where each coordinate x_i is an element of some finite linearly ordered set S_i.

```
Algorithm Backtrack(X[1...i])
// Gives a template of a generic backtracking algorithm
// Input: X[1...i] specifies first i promising components of a solution.
// Output: All the tuples representing the problem's solutions
If X[1..i] is a solution write X[1..i]
else
  for each element x ∈ S_{i+1} consistent with X[1..i] & the constraints do
      X[i+1] = x
      Backtrack(X[1...i+1])
```