03.04.2020 IV. Backtracking

- The principal idea in backtracking is to construct solutions one component at a time and 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 1st 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 <u>state-space tree</u>
- Its root represents an initial state before the search for a solution begins.
- The nodes of the ith level in the tree represent the choice made for the ith component of of a solution.

- A node in a state-space tree is said to be <u>promising</u> if it corresponds to a partially constructed solution that may still lead to a complete solution; otherwise, it is called <u>non-promising</u>.
- Leaves represent either non-promising dead ends or complete solutions found by the algorithm.

- If the current node is promising, its child is generated by adding the 1st remaining legitimate option for the next component of the solution, and the processing moves to this child.
- If the current node turns out to be non-promising, the algorithm backtracks to the current node's parent to consider the next possible option for its last component.

If there is no such option, it backtracks 1 more level of the tree, and 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.

eg: n-Queens problem.

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

Algorithm Backtrack (X[1..i]) // gives a template of generic backtracking algo.

Input: X[1..i] specifies 1st i promising components of a solution

Output: all tuples representing the problem's solutions.

```
if X[1..i] is a solution, write X[1..i]
else:
    for each element $x \in S_{i+1}$ consistent with X[1..i] and the constraints:
        X[i+1] = x
        Backtrack(X[1..i+1])
```