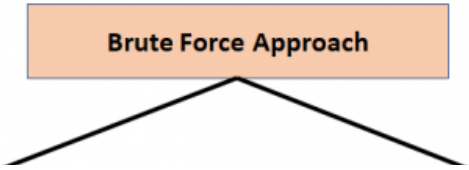
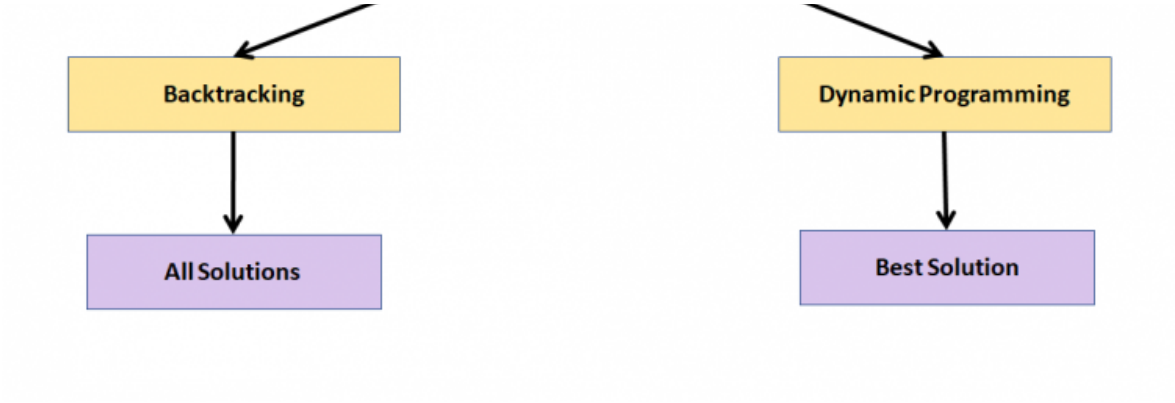
Backtracking in Python

Chapters

- Circular Doubly Linked List
- Binary Heap
- Trie
- Hashing
- Sorting Algorithms
- Searching Algorithms
- Single-Source Shortest Path
- Topological Sort
- Graphs
- Dijkstra's
- Bellman-Ford's
- All Pair Shortest Path
- Minimum Spanning Tree
- Kruskal & Prim's

A backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output. The Brute force approach tries out all the possible solutions and chooses the desired/best solutions.





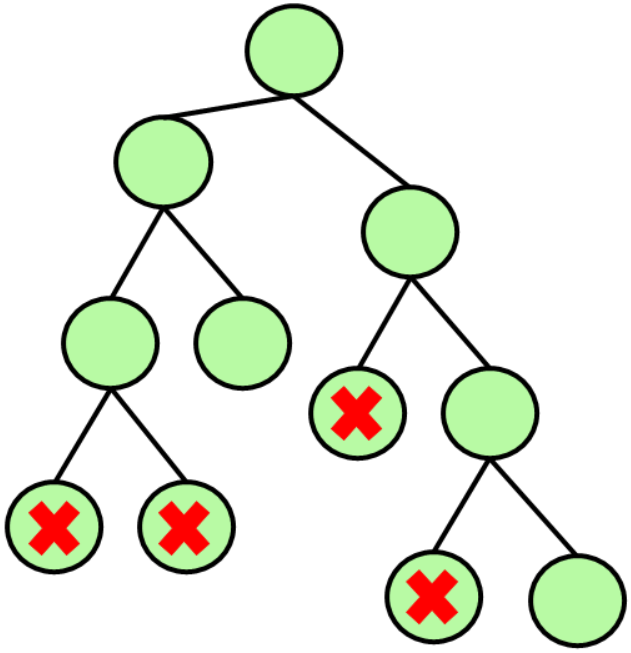
In backtracking, if the current solution is not suitable, then we backtrack and try to find other solutions. Hence, recursion is used. This approach is used to solve problems that have multiple solutions.

There are three types of problems in backtracking:

- **Decision Problem –** Search for a feasible solution
- **Optimization Problem –** Search for the best solution
- **Enumeration Problem –** Find all feasible soutions

State Space Tree – A state-space tree is a tree representing all the possible states (solution or nonsolution) of the problem from the root as an initial state to the leaf as a terminal state. Problems can be represented using the state-space

tree.



The state-space tree comprises the states of the given problem. Then, we can check if these states are the solutions or not. A backtracking algorithm traverses the tree recursively in the depth-first search manner.

Backtracking follows the below mentioned approach:

- It checks whether the given node is a valid solution or not.

- Discard several inavlid solutions with one iteration.
- Enumerate all subtree of the node to find the valid solution.

Table of Contents Show / Hide

The N-Queens Problem

Place N Queens on an NxN chessboard, in such a manner that no two queens attack each other. A queen can move horizontally, vertically, and diagonally.

Implementation of N-Queens Problem using Bactracking

The N-Queens Problem can be implemented using the Backtracking Approach with the following algorithm:

- Start in the leftmost column.
- If all the queens are placed
    - return true
- Try all rows in the current column. Do the following for every tried row.

- If the queen can be placed safely in this row, then mark this [row, column] as part of the solution and recursively check if placing the ueen here leads to a solution.
- If placing the queen in [row, column] leads to a solution, return true.
- If placing the queen in [row, column] doesn't lead to a solution, then unmark this [row, column] and go to the first step to try other rows.

- If all rows have been tried and nothing worked, return false to trigger backtracking.

```
 #Function to implement the N Queens Problem
class NQueens:
  def __init__(self, n):
    self.n = n
    self.chess_table = [[0 for i in range(n)] for j in range(n)]

  def print_queens(self):
    for i in range(self.n):
      for j in range(self.n):
        if self.chess_table[i][j] == 1:
          print(" Q ", end = '')
        else:
          print(" - ", end='')
      print("\n")


  def is_place_safe(self, row_index, col_index):
    for i in range(self.n):
      if self.chess_table[row_index][i] == 1:
```

```
        return False

    j = col_index
    for i in range(row_index, -1, -1):
      if i < 0:
        break
      if self.chess_table[i][j] == 1:
        return False
      j = j - 1

    j = col_index
    for i in range(row_index, self.n):
      if i < 0:
        break
      if self.chess_table[i][j] == 1:
        return False
      j = j - 1

    return True

  def solve(self, col_index):
    if col_index == self.n:
      return True

    for row_index in range(self.n):
      if self.is_place_safe(row_index, col_index):
        self.chess_table[row_index][col_index] = 1
        if self.solve(col_index+1):
```

```
            return True
          self.chess_table[row_index][col_index] = 0

      return False

  def solve_NQueens(self):
      if self.solve(0):
          self.print_queens()
      else:
          print("No solution exists for the problem")

queens = NQueens(8)
queens.solve_NQueens()
```

Output

Trending Posts You Might Like

Dijkstra's Algorithm in Python

File Upload / Download with Streamlit

Displaying Text, Pandas Dataframe, Table and JSON in Streamlit

Kruskal and Prim's Algorithm in Python

Matplotlib with STREAMLIT

Author : **Bhavya**

« Problem Solving in Python || Interview Questions on Trees in Python »

Search

Posts From Data Structures

Interview Questions on Graphs

Interview Questions on Trees in Python

Backtracking in Python

Problem Solving in Python

Dynamic Programming in Python

Categories

Bert

Computer Vision

Data Structures

Data Structures

Design Patterns

Kivy

Networking

NLP

Numpy

Pandas

Plotly

PyTest

Python

Streamlit

Back to top