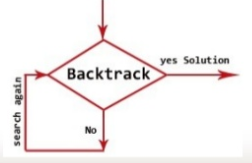
***Back Tracking:***

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

***Backtracking*** *can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.*



***The Algorithmic Approach:***

1. Backtracking systematically try and search possibilities to find the solution. Also it is an important process for solving constraint satisfaction problem like crossword, Sudoku and many other puzzles. It can be more continent technique for parsing other combinatorial optimization problem.
2. Basically the process is used when the problem has a number of option and just one solution have to be selected. After having a new option set means recursion, the procedure is repeated over and over until final stage.

|  |
| --- |
| Backtrack (v1,Vi)  If (V1,……., Vi) is a Solution Then  Return (V1,…, Vi)  For each v DO  If (V1,…….,Vi) is acceptable vector THEN  Sol = try (V1,…,Vi, V)  If sol != () Then  RETURN sol  End  End  Return ( ) |

***Advantages:***

1. Comparison with the Dynamic Programming, Backtracking Approach is more effective in some cases.
2. Backtracking Algorithm is the best option for solving tactical problem.
3. Also Backtracking is effective for constraint satisfaction problem.
4. In greedy Algorithm, getting the Global Optimal Solution is a long procedure and depends on user statements but in Backtracking It Can Easily getable.
5. Backtracking technique is simple to implement and easy to code.
6. Different states are stored into stack so that the data or Info can be usable anytime.
7. The accuracy is granted.

***Disadvantages:***

1. Backtracking Approach is not efficient for solving strategic Problem.
2. The overall runtime of Backtracking Algorithm is normally slow
3. To solve Large Problem Sometime it needs to take the help of other techniques like Branch and bound.
4. Need Large amount of memory space for storing different state function in the stack for big problem.
5. Thrashing is one of the main problem of Backtracking.
6. The Basic Approach Detects the conflicts too late.

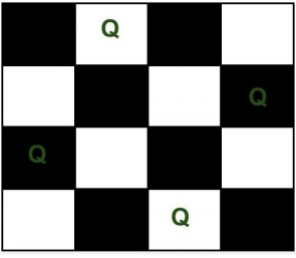
***Application of Backtracking:***

1. Optimization and tactical problems
2. Constraints Satisfaction Problem
3. Electrical Engineering
4. Robotics
5. Artificial Intelligence
6. Genetic and bioinformatics Algorithm
7. Materials Engineering
8. Network Communication
9. Solving puzzles and path

***Some Problem Solved with Backtracking Technique:***

1. N- Queens Problem
2. Sum of Subset
3. Sudoku Puzzle
4. Maze Generation
5. Hamiltonian Cycle

***N-Queen Problem:***

we have N queens and N x N chess board. The objective of this problem is such that we need to place all  N queens on N x N chess board in such a manner that no two queens in under attack to each other.

Two queens will be under attack if one of the following conditions is true:-

1. firstly, if they are in the same row.
2. secondly, if they are in the same column.
3. finally, if they are in the same diagonal.

Q1(0,1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Column 0:**  Step(0,0): check board[0][0]=safe  **Column 1:**  Step(0,1): check board[0][1]=not safe  Step(1,1): check board[1][1]=not safe  Step(2,1): check board[2][1]=safe  **Column 2:**  Step(0,2): check board[0][2]=not safe  Step(1,2): check board[1][2]=not safe  Step(2,2): check board[2][2]=not safe  Step(3,2): check board[3][2]=not safe | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 | Q1 | X | X |  | | 1 |  | X | X |  | | 2 |  | Q2 | X |  | | 3 |  |  | x |  | |

We reached last row of column 2, there is no safe position . We have to backtrack. We have to remove past placed queen. The one placed at position board[2][1].

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Column 0:**  Step(0,0): check board[0][0]=safe  **Column 1:**  Step(0,1): check board[0][1]=not safe  Step(1,1): check board[1][1]=not safe  Step(2,1): check board[2][1]= not safe  Step(3,1): check board[3][1]=safe  **Column 2:**  Step(0,2): check board[0][2]=not safe  Step(1,2): check board[1][2]=safe  **Column 3:**  Step(0,3): check board[0][3]=not safe  Step(1,3): check board[1][3]=not safe  Step(2,3): check board[2][3]= not safe  Step(3,3): check board[3][3]=not safe | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 | Q1 | X | X | X | | 1 |  | X | Q3 | X | | 2 |  | X |  | X | | 3 |  | Q2 |  | X | |

We reached last row of column 4, there is no safe position . We have to backtrack. We have to remove past placed queen. The one placed at position board[1][2].

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Column 0:**  Step(0,0): check board[0][0]=safe  **Column 1:**  Step(0,1): check board[0][1]=not safe  Step(1,1): check board[1][1]=not safe  Step(2,1): check board[2][1]= not safe  Step(3,1): check board[3][1]=safe  **Column 2:**  Step(0,2): check board[0][2]=not safe  Step(1,2): check board[1][2]= not safe  Step(2,2): check board[2][2]=not safe  Step(3,2): check board[3][2]=not safe | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 | Q1 | X | X |  | | 1 |  | X | x |  | | 2 |  | X | X |  | | 3 |  | Q2 | X |  | |

We reached last row of column 2, there is no safe position . We have to backtrack. We have to remove past placed queen. The one placed at position board[3][1] . There is no any row after 3 row in column 1. We have to backtrack and removed Queen 1 board[0][0]. Go the the next now to column 0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Column 0:**  Step(0,0): check board[0][0]=not safe  Step(1,0): check board[0][0]=safe  **Column 1:**  Step(0,1): check board[0][1]=not safe  Step(1,1): check board[1][1]=not safe  Step(2,1): check board[2][1]= not safe  Step(3,1): check board[3][1]=safe  **Column 2:**  Step(0,2): check board[0][2]=safe  **Column 2:**  Step(0,3): check board[0][3]=not safe  Step(1,3): check board[1][3]=not safe  Step(2,3): check board[2][3]=safe | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 | X | X | Q3 | X | | 1 | Q1 | X |  | X | | 2 |  | X |  | Q4 | | 3 |  | Q2 |  |  | |

.

***Rat in a Maze***

A Maze is given as N\*N binary matrix of blocks where source block is the upper left most block i.e., maze[0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1].

A rat starts from source and has to reach the destination. The rat can move only in two directions: **forward and down**.  
In the maze matrix, **0 means the** block is a dead end and **1 means the block can be used in the path from source to destination.**

Gray blocks are dead ends (value = 0).

|  |  |
| --- | --- |
|  |  |

Following is a maze with highlighted solution path.

|  |  |
| --- | --- |
|  |  |