Backtracking Algorithms Tutorial

Backtracking is a recursive problem-solving paradigm that incrementally builds candidates to the solution and abandons a candidate as soon as it is determined that it cannot lead to a valid solution.

N-Queens

The N-Queens problem asks how to place N queens on an N×N chessboard so that no two queens threaten each other. A queen in chess can attack any other piece in the same row, column, or diagonal. Therefore, the challenge is to arrange the queens so that no two share a row, column, or diagonal[1][2][3][4][5].

The standard approach uses **backtracking**:

- Place a queen in the first row, then move to the next row and try to place another queen in a safe column.
- For each placement, check if the queen is safe from attacks by previously placed queens.
- If a conflict is found, backtrack (remove the last queen) and try the next column.
- Continue this process recursively until all queens are placed or all possibilities are exhausted.

This method systematically explores all valid configurations, making it possible to find all solutions. The N-Queens problem is a classic example of constraint satisfaction and demonstrates the power of backtracking in combinatorial search problems[1][2][3][4][5].

Sudoku Solvers

A Sudoku puzzle is a 9×9 grid divided into 3×3 subgrids, where the goal is to fill the grid so that each row, column, and subgrid contains the digits 1 through 9 exactly once[6][7].

The most common algorithm for solving Sudoku is **backtracking**:

- Find an empty cell in the grid.
- Try filling it with digits 1 to 9, checking if the placement is valid (i.e., the digit does not already exist in the same row, column, or subgrid).
- If a valid digit is found, move to the next empty cell and repeat.
- If no valid digit can be placed, backtrack to the previous cell and try a different digit.

This recursive process continues until the grid is completely filled or no solution exists. More advanced solvers may use constraint propagation and logical deduction to reduce the search space, but backtracking remains the foundational technique for most Sudoku solvers[6][7].

Subset/Permutations

Subsets: The subset problem involves generating all possible subsets of a given set. For a set of size n, there are 2^n possible subsets. The standard approach uses backtracking or recursion:

- For each element, decide whether to include it in the current subset or not.
- Recursively build up all possible combinations by exploring both choices at each step89.

Permutations: The permutation problem involves generating all possible arrangements of a set's elements. For n elements, there are n! (n factorial) permutations. The approach is:

- Fix one element at a position and recursively generate permutations of the remaining elements.
- Swap elements to explore all possible orderings, backtracking after each recursive call to restore the original order8910.

Both problems are classic examples of combinatorial generation and are efficiently solved using backtracking, which systematically explores all configurations by making choices and undoing them as needed.

Word Search

The Word Search problem involves finding whether a given word exists in a 2D grid of characters. The word can be constructed from letters of sequentially adjacent cells, where adjacent means horizontally or vertically neighboring. Each cell can be used only once per word111213.
The standard algorithm is **backtracking with depth-first search (DFS)**:

- For each cell in the grid, check if it matches the first letter of the word.
- If it does, recursively search its adjacent cells for the next letter, marking the current cell as visited.
- If the entire word is found, return true. If a path fails, backtrack by unmarking the cell and trying other directions.
- The process continues until the word is found or all possibilities are exhausted.

This method efficiently prunes invalid paths and avoids revisiting cells, making it suitable for the problem's constraints111213.

Rat in a Maze

The Rat in a Maze problem is a classic pathfinding challenge. Given a maze represented as a 2D grid (with 1 as open and 0 as blocked), the goal is to find all possible paths from the top-left corner (start) to the bottom-right corner (destination)1415161718.
The typical solution uses **backtracking**:

- Start at the initial cell and explore all valid moves (usually up, down, left, right).

- Mark each cell as visited to avoid cycles.
- If the destination is reached, record the current path.
- If a dead end is encountered, backtrack by unmarking the cell and trying other directions.

Backtracking ensures that all possible paths are explored without revisiting cells, allowing the algorithm to find all solutions or determine if no path exists. This approach is widely used in robotics, AI, and navigation systems for its simplicity and effectiveness in constrained environments1415161718.

If you need further elaboration on any specific subtopic or want real-world applications, let me know!

1. N-Queens Problem

Place N queens on an N×N chessboard such that no two queens threaten each other.

C Code:

```c
#include <stdio.h>
#include <stdbool.h>
#define N 8

bool isSafe(int board[N][N], int row, int col) {
    for (int i = 0; i < col; i++)
        if (board[row][i]) return false;

    for (int i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j]) return false;

    for (int i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j]) return false;

    return true;
}

bool solveNQueensUtil(int board[N][N], int col) {
    if (col >= N) return true;

    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQueensUtil(board, col + 1)) return true;
            board[i][col] = 0;
        }
    }
    return false;
}

void solveNQueens() {
    int board[N][N] = {0};
    if (solveNQueensUtil(board, 0)) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                printf("%d ", board[i][j]);
            printf("\n");
        }
    }
}
```

---

2. Sudoku Solver

Fill a 9x9 Sudoku grid so that every row, column and 3x3 box contains digits 1 through 9.

C Code:

```
#include <stdio.h>
#include <stdbool.h>
#define UNASSIGNED 0
#define N 9

bool isSafe(int grid[N][N], int row, int col, int num) {
    for (int x = 0; x < N; x++)
        if (grid[row][x] == num || grid[x][col] == num)
            return false;

    int startRow = row - row % 3, startCol = col - col % 3;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (grid[i + startRow][j + startCol] == num)
                return false;

    return true;
}

bool solveSudoku(int grid[N][N]) {
    int row, col;
    bool empty = false;
    for (row = 0; row < N; row++) {
        for (col = 0; col < N; col++) {
            if (grid[row][col] == UNASSIGNED) {
                empty = true;
                break;
            }
        }
        if (empty) break;
    }
    if (!empty) return true;

    for (int num = 1; num <= 9; num++) {
        if (isSafe(grid, row, col, num)) {
            grid[row][col] = num;
            if (solveSudoku(grid)) return true;
            grid[row][col] = UNASSIGNED;
        }
    }
    return false;
}
```

---

3. Subset Generation / Permutations

Generate all subsets or permutations of a set.

C Code (Permutations):

```
#include <stdio.h>
void swap(char *x, char *y) {
    char temp = *x;
    *x = *y;
    *y = temp;
}

void permute(char *str, int l, int r) {
    if (l == r)
        printf("%s\n", str);
    else {
        for (int i = l; i <= r; i++) {
            swap(&str[l], &str[i]);
            permute(str, l + 1, r);
            swap(&str[l], &str[i]);
        }
    }
}
```

---

4. Word Search in Grid

Given a grid of letters and a word, check if the word exists in the grid.

C Code:

```
#include <stdio.h>
#include <stdbool.h>
```

```
#define ROW 3
#define COL 4

bool dfs(char board[ROW][COL], char* word, int x, int y, int index, bool
visited[ROW][COL]) {
    if (word[index] == '\0') return true;
    if (x < 0 || x >= ROW || y < 0 || y >= COL || visited[x][y] ||
board[x][y] != word[index])
        return false;

    visited[x][y] = true;
    bool found = dfs(board, word, x+1, y, index+1, visited) ||
                 dfs(board, word, x-1, y, index+1, visited) ||
                 dfs(board, word, x, y+1, index+1, visited) ||
                 dfs(board, word, x, y-1, index+1, visited);
    visited[x][y] = false;
    return found;
}

bool exist(char board[ROW][COL], char* word) {
    bool visited[ROW][COL] = {{false}};
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COL; j++) {
            if (dfs(board, word, i, j, 0, visited))
                return true;
        }
    }
    return false;
}
```

---

5. Rat in a Maze

Find all paths from top-left to bottom-right in a grid, only moving right or down and not through blocked cells.

C Code:

```
#include <stdio.h>
#define N 4

int isSafe(int maze[N][N], int x, int y) {
    return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
}

int solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N]) {
    if (x == N-1 && y == N-1) {
        sol[x][y] = 1;
        return 1;
    }

    if (isSafe(maze, x, y)) {
        sol[x][y] = 1;
        if (solveMazeUtil(maze, x+1, y, sol)) return 1;
        if (solveMazeUtil(maze, x, y+1, sol)) return 1;
        sol[x][y] = 0;
    }
    return 0;
}

void solveMaze(int maze[N][N]) {
    int sol[N][N] = {0};
    if (!solveMazeUtil(maze, 0, 0, sol)) {
        printf("No solution\n");
    } else {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                printf("%d ", sol[i][j]);
            printf("\n");
        }
    }
}
```

This tutorial covers core problems solvable using backtracking. Mastering these foundational patterns helps in solving complex constraint satisfaction problems effectively.

N-Queens
https://www.youtube.com/watch?v=JkP-xats3no
Sudoku Solvers
https://www.youtube.com/watch?v=VPVtlODPdPY
Subset/Permutations
(Subsets) https://www.youtube.com/watch?v=REOH22Xwdkk
(Permutations) https://www.youtube.com/watch?v=YK78FU5Ffjw
Word Search
https://www.youtube.com/watch?v=pfiQ_PS1g8E
Rat in a Maze in C
https://www.youtube.com/watch?v=bLGZhJlt4y0
https://www.youtube.com/watch?v=KpF7wq4z8yE