**Refer to Deepseek**

**You mention about backtracking, what's the relationship between DFS and backtracking ? or let's say what's the difference between using DFS with backtracking and using DFS without backtracking ? when must we use DFS with backtracking ?**

Great question! The relationship between **DFS (Depth-First Search)** and **Backtracking** is often misunderstood, but they are closely related concepts. Let’s break it down:

**DFS vs. Backtracking**

**1. DFS (Depth-First Search):**

DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking.

It is used to traverse or search through a graph or tree structure.

**In DFS, you visit a node, then recursively visit its unvisited neighbors until you reach a dead end. Then, you backtrack to explore other branches.**

Key Characteristics:

Explores one path at a time.

Uses a stack (implicitly via recursion or explicitly with a data structure).

**Does not necessarily involve "undoing" actions (backtracking).**

**2. Backtracking:**

**Backtracking is a general algorithmic technique for solving problems by exploring all possible solutions incrementally and abandoning a solution ("backtracking") as soon as it determines that the solution cannot be completed.**

**It is often implemented using DFS because DFS naturally explores one path at a time and can easily "undo" decisions when it backtracks.**

Key Characteristics:

Explores all possible solutions.

Involves making a choice, exploring further, and then undoing the choice if it doesn’t lead to a solution.

Often used in problems like permutations, combinations, and pathfinding.

**Relationship Between DFS and Backtracking:**

**Backtracking is a refinement of DFS:**

**Backtracking uses DFS to explore all possible solutions, but it adds the ability to "undo" decisions when a path doesn’t lead to a valid solution.**

**In other words, backtracking is DFS with pruning (eliminating invalid paths early).**

DFS without backtracking:

In standard DFS, you don’t explicitly "undo" decisions. You simply explore all reachable nodes from the current node.

For example, in a tree traversal, you don’t need to backtrack because you don’t revisit nodes.

DFS with backtracking:

In backtracking, you explicitly "undo" decisions (e.g., remove the last element from a path) after exploring a branch.

This is necessary when you need to explore multiple paths that share common nodes (e.g., finding all paths in a graph).

**When to Use DFS with Backtracking:**

You must use DFS with backtracking in the following scenarios:

Finding All Possible Solutions:

When the problem requires exploring all possible solutions (e.g., all paths, all permutations, all combinations).

Example: LeetCode 797 (All Paths From Source to Target), LeetCode 46 (Permutations).

Pruning Invalid Paths Early:

When you can determine that a partial solution cannot lead to a valid solution, so you abandon it early.

Example: LeetCode 39 (Combination Sum), LeetCode 51 (N-Queens).

Exploring Multiple Paths That Share Nodes:

When multiple paths can share the same nodes, and you need to explore all of them.

Example: Finding all paths in a graph.

Problems with Constraints:

When the problem has constraints that require you to explore all valid configurations.

Example: Sudoku solver, subset problems.

**When to Use DFS Without Backtracking:**

You can use DFS without backtracking in the following scenarios:

Traversing a Tree or Graph:

When you only need to visit each node once (e.g., tree traversal, finding connected components).

Example: LeetCode 104 (Maximum Depth of Binary Tree), LeetCode 200 (Number of Islands).

Finding a Single Solution:

When you only need to find one valid solution (e.g., a path, a configuration).

Example: Finding a path from source to target (if only one path is needed).

No Need to Undo Decisions:

When the problem does not require revisiting nodes or undoing decisions.

Example: Topological sorting, cycle detection.

**Example: DFS with Backtracking vs. DFS Without Backtracking**

**Problem: Find All Paths from Source to Target (LeetCode 797)**

DFS with Backtracking:

You explore all paths from the source to the target.

After exploring a path, you backtrack to explore other paths.

Example:

private void dfs(int[][] graph, int node, List<Integer> path, List<List<Integer>> result) {

if (node == graph.length - 1) {

result.add(new ArrayList<>(path)); // Add a copy of the current path

return;

}

for (int neighbor : graph[node]) {

path.add(neighbor); // Make a choice

dfs(graph, neighbor, path, result); // Explore further

path.remove(path.size() - 1); // Backtrack (undo the choice)

}

}

DFS Without Backtracking:

**You would only find one path from the source to the target.**

**You don’t undo decisions, so you can’t explore multiple paths.**

Example:

private boolean dfs(int[][] graph, int node, List<Integer> path, List<List<Integer>> result) {

if (node == graph.length - 1) {

result.add(new ArrayList<>(path)); // Add a copy of the current path

return true; // Stop after finding one path

}

for (int neighbor : graph[node]) {

path.add(neighbor); // Make a choice

if (dfs(graph, neighbor, path, result)) {

return true; // Stop after finding one path

}

// No backtracking here

}

return false;

}

**Key Differences:**

|  |  |  |
| --- | --- | --- |
| Feature | DFS Without Backtracking | DFS With Backtracking |
| **Purpose** | Traverse or find a single solution | Explore all possible solutions |
| **Undo Decisions** | No | Yes |
| **Use Case** | Tree/graph traversal, single path | All paths, permutations, pruning |
| **Example Problems** | Tree depth, connected components | All paths, N-Queens, Sudoku |

**Summary:**

DFS without backtracking is used for traversal or finding a single solution.

DFS with backtracking is used when you need to explore all possible solutions, undo decisions, or prune invalid paths early.

Use backtracking when the problem requires exploring multiple paths or configurations, especially in combinatorial problems.