**Search:**

-Breadth-First Search(BFS) – Searches children nodes layer by layer.

🡪 find the shortest path, shortest transformation, minimum number of moves etc.

-Use queue, push next visit to end of queue, process from front of queue

-Depth-First Search(DFS) – Searches a single path to the very end before starting on another path. Recursive call to neighbor node or child node.

🡪 generating combinations, searching for a specific path that meets conditions

* Good for backtracking

**Backtracking:**

**Backtracking**: mark status of currently visited path so do not hit same spot again. Before returning back from current call, remove that mark so that a different path can use that spot again. Remove the mark so that a parent recursive call can go down a different path that uses that spot. 🡨 Removal requires a temporary local variable rather than directly returning the recursive call.

Word Search in a grid for a particular word:

private boolean backtrack(char[][] board, String word, int count,

boolean[][] visited, int x, int y)

{

if (x < 0 || x >= board[0].length || y < 0 || y >= board.length)

{

return false;

}

else if (word.charAt(count) != board[y][x])

{

return false;

}

else if (visited[y][x])

{

return false;

}

else if (count == word.length() - 1)

{

return word.charAt(count) == board[y][x];

}

else

{

**visited[y][x] = true;**

boolean return\_val = backtrack(board, word, count + 1, visited, x - 1, y) ||

backtrack(board, word, count + 1, visited, x + 1, y) ||

backtrack(board, word, count + 1, visited, x, y - 1) ||

backtrack(board, word, count + 1, visited, x, y + 1);

**visited[y][x] = false;**

return return\_val;

}

**Sliding Window/Two Pointer Approach:**

-Useful for arrays and strings

Left pointer and right pointer either both starting at beginning or one at end and one at beginning.

Substring with all unique chars 🡪 Two pointer moving right pointer right until a repeat, once repeat hits, move left pointer right until all unique. Repeat process.

Trapping Rainwater, longest mountain …

The window is the contents between the left/right pointer

**Trees:**

-BFS and DFS

-BST 🡪 all nodes in left subtree less than root and all nodes in right subtree greater than root.

Validate – 1) Inorder traversal is strictly ascending

2) Recurse down the tree with extra function args min and max. If left recurse, set max to curr node. If right recurse, set min to current node. If null, return true, if < min or > max, return false. Return recurse left && recurse right.

**Graphs:**

-BFS and DFS

-Detect Cycle(Undirected Graph)

Keep set of visited vertices. Iterate through the vertices and if not visited, DFS from vertex. Mark current vertex as visited. If neighbor is visited, return true, otherwise DFS to neighbor, pass in parent vertex to function so skip neighbor if it is parent.

-Detect Cycle (Directed Graph) – 2 sets/arrays size of number of vertices, unvisited, visiting

Start all unvisited as true, visited as false, and visiting as false.

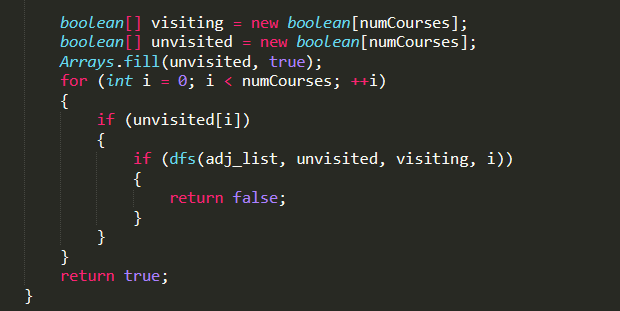
O(V + E) time

O(V + E) space

Iterate through all the vertices and DFS if it is unvisited. Mark the unvisited of the current as false visiting of current as true and iterate through all the neighbors. If a neighbor is unvisited, dfs to that neighbor and return true if it is true, otherwise if a neighbor is marked as visiting, return true for cycle.

After all neighbors have been visited, mark the current visiting as false and return false after all the neighbors have been visited signaling no cycle detected.



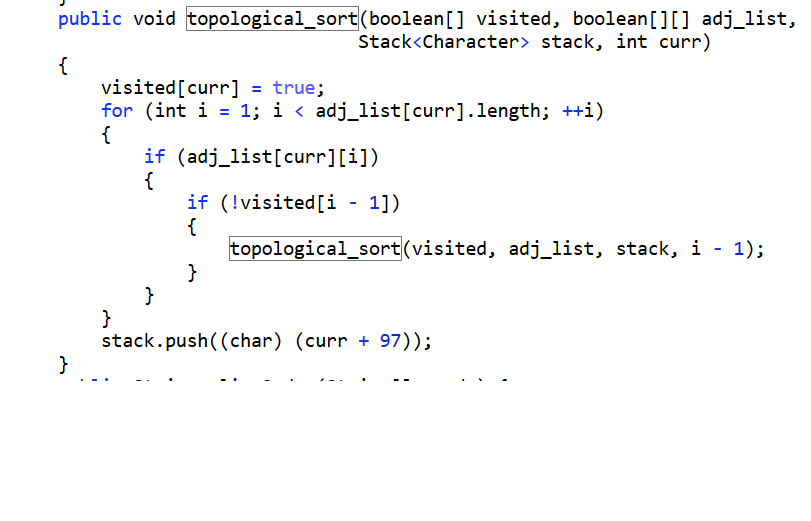


-Topological Sort

O(V + E) time

O(V) space

Set for visited vertices, stack to push vertices on. Iterate through the vertices, and if not in set, add DFS through that vertex’s neighbors if neighbor not in visited set.



Union Find –

**Dynamic Programming:**

-Tabulation(Bottom-Up) vs Memoization(Top-Down)