**DFS:**

1. Min number of parenthesis to be removed such that the given string is balanced.
   1. Eg. ((a)))(b))
   2. Output should be a vector of such bal strings with min brackets removed
2. Number or connected components in a graph
3. Single source shortest path to reach to all the vertex from a source, output should be the distances of all the vertices from the given source
4. Biju and its gf- biju has to choose a gf who lives closest to his place, and if multiple options possible he would give priority to lesser state ID girl
   1. An application of SSCC
   2. Output -> ID of the girl he would date
5. Tree or not – given a graph check whether it represents a tree or not
   1. E=V-1
   2. No cycle allowed
   3. Connected graph
6. Bipartite – check if a given graph can be bipartite or not, it’s solve by coloring
   1. Bipartite: no odd len cycle is allowed
   2. Start coloring the nodes, with -1 and 1 representing the only colors allowed, uncoloured means 0
   3. Color the children with the negation of parent’’s color
   4. If at some moment the adjacent node is already colored and it’s not the negation of the given node’s color, that means the graph has odd len cycle – return false
7. Opposite genders – an application of bipartite – given a graph (may or may not be connected, every edge should have 1 vertex M and 1 F, check if the opposite gender rule is violating – again no odd cycles are possible
8. Detect Cycles – a basic application of dfs, if you hit an already connected vertex again, that means there is a cycle ( exclude hitting again for backtracked vertices)
   1. This can be easily done by using and int returned type dfs function
   2. Dfs should have 2 parameters, current node and parent
   3. if(vis[c]==0) {if(dfs(c,s)==true) return true; }
   4. else{if(c!=par) return true; }
   5. if both these fail, return false
9. In-out time – in-time and out-time of each node in the graph
   1. Maintain two arrays I and out, while entering increase in time, while exiting, out time
   2. For each recurrence, increase time
10. Fire Escape routes – maths + graphs – an application of connected components
11. Longest path – a smart problem for finding the longest possible path in a graph in O(n) time
    1. Brute force would be O(n^2) time, to find the longest path from each vertex
    2. Efficient solution: Choose any vertex, go to the farthest node it can visit, then choose that as the source node and do the dfs traversal, this traversal gives the longest possible path in the graph
    3. Can be proved using mathematical induction! There would be two case, depending on whether the root node should be incl or not
12. Subtree size – output the subtree size of each vertex of the graph
    1. This problem requires a deep thought about the graph concepts
    2. Not a difficult problem to do, we just have to maintain a subree array in which for each tree we find that is traversed after a particular source node, we would be inc the subtree size.

**BFS:**

1. Search the last node – basic application of BFS where we have to store the distances and display the dist of the last node from the first node to the last node
2. Prime path – two 4 digit numbers are given, you have to convert the first number into the second. The condition is that in one step you can only change one digit of the number and the resulting number should also be a prime
   1. Store all the prime numbers, fill the graph such that there are edges only between the adjacent vertices, by adj it means that only one char should be diff the two numbers!
   2. BFs search and find the shortest dist from given prime 1 to prime 2.