**APPLICATIONS OF BFS:-**

1. Find the shortest path from root to any other vertex.
2. Finding all the connected components in the undirected graph in O(V+E) time. To do this run the BFS algo from each vertex seperately if that node is not visited. If we run the BFS first time through any source S then after traversing only the nodes that are not reachable from S will be having visited[x]=false. So we have to call BFS through them also.
3. Finding the solution to a GAME or the PROBLEM with the least number of moves if each state of graph can be represented as the vertex of the graph and the transition from one state to the other state is the edge of the graph.
4. Finding the shortest path in the graph with weights 0 or 1. This requires just a little modification to normal breadth-first search: Instead of maintaining array used[], we will now check if the distance to vertex is shorter than current found distance, then if the current edge is of zero weight, we add it to the front of the queue else we add it to the back of the queue.This modification is explained in more detail in the article 0-1 BFS.
5. Finding the shortest cycle in a directed unweighted graph: Start a breadth-first search from each vertex. As soon as we try to go from the current vertex back to the source vertex, we have found the shortest cycle containing the source vertex. At this point we can stop the BFS, and start a new BFS from the next vertex. From all such cycles (at most one from each BFS) choose the shortest.
6. Find all the edges that lie on any shortest path between a given pair of vertices (a,b). To do this, run two breadth first searches: one from a and one from b. Let da[] be the array containing shortest distances obtained from the first BFS (from a) and db[] be the array containing shortest distances obtained from the second BFS from b. Now for every edge (u,v) it is easy to check whether that edge lies on any shortest path between aa and bb: the criterion is the condition da[u]+1+db[v]=da[b].
7. Find all the vertices on any shortest path between a given pair of vertices (a,b). To accomplish that, run two breadth first searches: one from a and one from b. Let da[] be the array containing shortest distances obtained from the first BFS (from a) and db[] be the array containing shortest distances obtained from the second BFS (from b). Now for each vertex it is easy to check whether it lies on any shortest path between a and b: the criterion is the condition da[v]+db[v]=da[b].
8. Find the shortest path of even length from a source vertex s to a target vertex t in an unweighted graph: For this, we must construct an auxiliary graph, whose vertices are the state (v,c), where v - the current node, c=0 or c=1 - the current parity. Any edge (a,b) of the original graph in this new column will turn into two edges ((u,0),(v,1)) and ((u,1),(v,0)). After that we run a BFS to find the shortest path from the starting vertex (s,0) to the end vertex (t,0).