Breadth - first search :-

- One of the most basic & essential searching algorithms on graphs.

— As a result of how the algorithm works, the path found by BFS to any node is the shortest path to that node, i.e. the path that contains the smallest number of edges.

— The algorithm works in $O(n+m)$ time, where n : number of vertices & m : number of edges.

Description :-

The algorithm :-

— Takes as input an unweighted

graph and the id of the source vertex 's'.

— Input graph can be either directed / undirected.

Analogous to :-

* Fire spreading on the graph

At 0th step only the source s is on fire.

At each step, the fire burning at each vertex spreads to all of its neighbors.

In one iteration of the algorithm, the "ring of fire" is expanded in width by one unit. (hence the name)

Detailed description :-

1) Create a queue 'q' which will contain the vertices to be processed and a Boolean array 'used[]' which indicates for each vertex, if it has been hit (or visited) or not.

2) Initially, push the source s to the queue and set used[s] = true, and for all other vertices v set used[v] = false.

3) Loop untill the queue is empty and in each iteration, pop a vertex from the front of the queue.

4) Iterate through all the edges going out of this vertex and if some of these edges go to vertices that are not already hit,

set them on fire and place them in the queue.

5) As a result, when queue is empty, the "ring of fire" contains all vertices reachable from the source 's', with each vertex reached in the shortest possible way.

6) We can also calculate the lengths of the shortest paths (which requires maintaing an array of path lengths d[]) as well as some information to restore all of these shortest paths (it is necessary to maintain an array of "parents" P[] which stores for each vertex the vertex from which we reached it).

Applications of BFS :- (V. imp.)

① Find the shortest path from a source to other vertices in an unweighted graph.

* ② Find all components in an undirected graph in $O(n+e)$ time.

③ finding a solution to a problem or a game with the least number of moves, if each state of the game can be represented by a vertex of the graph, and the transitions from one state to the other are the edges of the graph.

* (4) Finding the shortest path in a graph with weights 0 or 1.

* See '0-1 BFS' problem

* (5) Finding the shortest cycle in a directed unweighted graph.

* (6) Find all the edges that lie on any shortest path b/w a given pair of vertices (a, b).

* (7) Find all the vertices on any shortest path b/w a given pair of vertices (a, b)

* (8) Find the shortest path of even length from a source vertex 's' to a target vertex t in an unweighted graph.

Solutions :-

2 ⇒ Run BFS starting from each vertex, except for vertices which have already been visited from previous runs.

Thus, we perform normal BFS from each of the vertices, but don't reset the array 'used[]' each and every time we get a new connected component, and the total running time will be $O(n+m)$.

4 ⇒ Modify the normal BFS :-

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.

5 ⇒ Start a BFS 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 (atmost one from each BFS) choose the shortest.

6 ⇒ Run BFS twice :-

one from a and one from b.

Let $d_a[]$ be the array containing shortest distances obtained from the first BFS (from a).

Similarly $d_b[]$.

Now for every edge (u, v) it is easy to check whether that edge lies on any shortest path b/w a and b : the criterion is the condition :-

$$d_a[u] + 1 + d_b[v] = d_a[b]$$

7 ⇒ Again run 2 BFS

$d_a[]$ $d_b[]$ as defined above

Now for each vertex we can check if it lies on any shortest path b/w a and b :-

$$d_a[v] + d_b[v] = d_a[b]$$

8 ⇒ Construct an auxiliary graph, whose vertices are the state (v, c) where v – current node, $c=0$ or $c=1$ – the current parity.

Any edge (u, v) 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 $(t,0)$.