Algorithm 8.1 (Breadth-First Search)

1. Let $S = \{v\}$ [v is the root]; $T = \emptyset$ [initially there are no edges in the tree].
2. $C = N(v)$ [C is the current set of vertices being processed].
3. $l(v) = 0$ [label root as vertex 0]; $p(v) = v$; $b^* = v$ [keeps track of current vertex we are branching from]; $i = 1$ [initializes variable i used to help label the vertices]; remove b^* from all adjacency lists.
4. For each $w \in N(b^*)$, place w in S and place edge $b{*}w \in T$; assign successive labels $l(w) = i$ and $p(w) = p(b^*), w$ to vertices of C. Add one to i after each vertex is labeled; remove w from all adjacency lists [this ensures that a vertex w gets labels $l(w)$ and $p(w)$ just once].
5. Define a new b^* to be the vertex x in C such that $l(x)$ is minimum; remove b^* from C, and return to step 4. If, however, C is empty, stop. If every vertex of G has been labeled, a spanning tree has been found. If not, then G is disconnected, but a spanning tree of the component containing the root has been found.

```
#Breadth First Search
def BFS(self,v):
    if v in self.Graph:
        #step 1
        graph = copy.deepcopy(self.Graph)       #create copy of self.Graph
        S = [v]                            #list S
        label = []                          #list to hold labels, index refers to label
        T = []                               #empty list of edges
        #step 2
        C = graph[v]            #list of v's neighbors
        C.sort()
        #step 3
        label.append(v)  #label v as 0
        bstar = v        #bstar = vertex we're branching to

        for item in graph:
            if bstar in graph[item]:          #remove bstar from adjacency lists
                graph[item].remove(bstar)

        #step 4
        while(True):
            neighbors = graph[bstar]        #list of adjacent vertices

            for vert in label:
                if vert in neighbors:       #neighbors = adjacent vertices that haven't been visited
                    neighbors.remove(vert)

            for item in neighbors:          #iterate through neighbors
                S.append(item)          #add vertex to S
                S.sort()
                T.append((bstar,item))  #add edge to T
                label.append(item)      #label item

            graph2 = copy.deepcopy(graph)       #create copy of graph bc can't edit something you're iterating

            for thing in C:
                for item in graph:
                    if thing in graph2[item]:       #remove bstar from adjacency lists
                        graph2[item].remove(thing)


            graph = graph2          #update graph

            #step 5
            if (len(C) == 0):           #halting condition
                return T

            else:
                bstar = C[0]            # define a new bstar to be min vertex in C
                C.remove(bstar)
```