```
#Undirected graph where each node is connected to their adjacent node on both way
graph = {"francfurt":["mainhain", "wulzburg", "kassel"],
         "mainhain": ["karlsuhe", "francfurt"],
         "wulzburg": ["francfurt", "numburg", "earthfurt"],
         "kassel":   ["francfurt","munchen"],
         "karlsuhe": ["ausburg","mainhain"],
         "numburg":  ["statgaurd", "wulzburg"],
         "ausburg":["karlsuhe"],
         "statgaurd":["numburg"],
         "earthfurt":["wulzburg"],
         "munchen":["kassel"]
        }


visited_list=[] #creating a list which have been visited means that we have trave


node1=input("enter node: ") #taking input from the user means user will decide fr

     enter node: francfurt
```

DFS(Depth First Search): In DFS we explore as far as possible along a branch before backtracking. We start from a parent and visit to its unvisited neighbours until all the neighboers are visited. When all neighbors are visited then we start backtracking.

```
def DFS(node1):
  """
  This function is meant to show how the DFS(Depth First Search) algorithm.We are
  all the nodes.Give all the nodes from parent to (top most) lowest node in a bra

  Parameter:node1 which is the parent node in level 0.
  precondition:key of a dictionary(which is our graph) where particular nodes(key
  """
  if node1 not in visited_list: #as initially visited_list is empty so the node w
    print(node1) #printing each node
    visited_list.append(node1) #appending the node to visited list
    for i in graph[node1]: #the node (which is a key of dictionary(graph)) callin
      DFS(i) #calling the function recursively


DFS(node1)

     francfurt
     mainhain
     karlsuhe
     ausburg
     wulzburg
     numburg
     statgaurd
     earthfurt
     kassel
     munchen
```

BFS

```
visited_list=[] #created empty list to add all the nodes level wise

queue_list=[] #created empty list this will keep updating in each level and after adding a
```

BFS(Breadth First Search): In BFS we traverse the graph or tree level by level. we start from level 0 and visit the adjacent of each node. It has completeness property(means it will definately give the answer).it is optimal

```
def BFS(node1):
  """
  This function is meant to show that how the BFS(breath first search algorithm w
  In this serch we search the goal level by level but here we are traversing all
  level by level.

  Parameter:node1 is the parent node
  precondition:node1 is the parent node means it is first one to start with which

  """
  if node1 not in visited_list: #checking if there is node1 in the visited_list
    visited_list.append(node1) #appending node1 to the visiting list
  for i in graph[node1]: #here loop variable i will be a node(or element of the l
    if i not in visited_list and i not in queue_list: #checking if the i is in th
      queue_list.append(i) #appending the i to the queue_list
  if queue_list: #this is checking whether the queue_list is empty or not if it i
    node1=queue_list.pop(0) #poping out the first element of the queue_list and a
    BFS(node1) #calling the function recursively with node1 and it will go on for


BFS("kassel") #calling the function
print(queue_list) #printing the queue_list which will be empty at the end
print(visited_list) #printing the visited list
```

```
[]
['kassel', 'francfurt', 'munchen', 'mainhain', 'wulzburg', 'karlsuhe', 'numburg', 'ea
```

✓ 0s completed at 9:54 AM