```
from collections import deque

# A class to represent a graph object
class Graph:
  def __init__(self, edges, n):
    # Initialize an adjacency list for the graph
    self.adjList = [[] for _ in range(n)]

    # Populate the adjacency list with edges
    for (src, dest) in edges:
      self.adjList[src].append(dest)
      self.adjList[dest].append(src)

def recursiveBFS(graph, q, discovered):
  # Base case: if the queue is empty, return
  if not q:
    return

  # Dequeue a vertex from the queue
  v = q.popleft()
  print(v, end=' ')

  # Explore all the unvisited neighbors of the dequeued vertex
  for u in graph.adjList[v]:
    if not discovered[u]:
      # Mark the neighbor as discovered and enqueue it
      discovered[u] = True
      q.append(u)

  # Recursively call BFS with the updated queue and discovered set
  recursiveBFS(graph, q, discovered)


if __name__ == '__main__':
  # Define the edges and number of vertices
  edges = [
    (1, 2), (1, 3), (1, 4), (2, 5), (2, 6), (5, 9),
    (5, 10), (4, 7), (4, 8), (7, 11), (7, 12)
  ]
  n = 15

  # Create a Graph object with the given edges and number of vertices
  graph = Graph(edges, n)

  # Create a list to keep track of discovered vertices
  discovered = [False] * n

  # Create a queue using deque to perform BFS
  q = deque()

  # Iterate through all the vertices
  for i in range(n):
    if not discovered[i]:
      # Mark the vertex as discovered
      discovered[i] = True
      # Enqueue the vertex
      q.append(i)
      # Call the recursiveBFS function to explore the vertex and its neighbors
      recursiveBFS(graph, q, discovered)
```

```
#BFS (Breadth-First Search) is a graph traversal algorithm that explores all the vertices of a graph in breadth-first order. It starts at

# Here is a step-by-step explanation of the BFS algorithm:

# Initialize an empty queue and an empty set to keep track of visited vertices.
# Enqueue the source vertex into the queue and mark it as visited.
# While the queue is not empty, do the following:
# a. Dequeue a vertex from the queue.
# b. Process the vertex (print it, perform some operation, etc.).
# c. Enqueue all the unvisited neighbors of the vertex and mark them as visited.
# Repeat steps 3 until the queue becomes empty
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```