1)

```python
#Ejercicio 1
def createGraph(vertices:list, aristas:list):
    """Implementa la operación crear grafo"""
    graph_list = []

    for i in range(len(vertices)):
        graph_list.append([])

    for a in aristas:
        index = a[0]
        node = a[1]
        graph_list[index].append(node)
        index = a[1]
        node = a[0]
        graph_list[index].append(node)

    return graph_list
```

2)

```python
#Ejercicio 2
def existPath(Grafo, v1, v2):
    def dfs(v,v2,visit):
        if v == v2:
            return True
        visit.append(v)
        for vecino in Grafo[v]:
            if vecino not in visit:
                if dfs(vecino,v2,visit):
                    return True
        return False
    #Inicio
    visit = []
    return dfs(v1,v2,visit)
```

3)

```python
#Ejercicio 3
def isConnected(graph):
    conexo = []
    for i in range(len(graph)):
        conexo.append(existPath(graph,i,len(graph)-1))

    for c in conexo:
        if c != True:
            return False
    return True
```

4)

```python
#Ejercicio 4
def isTree(graph):
    marked = [False] * len(graph)
    padres = [-1] * len(graph)
    aristas_bucle = []
    def dfs(v,marked,padres):
        if marked[v] == True:
            return False
        marked[v] = True
        for g in graph[v]:
            if not marked[g]:
                padres[g] = v
                dfs(g,marked,padres)
            elif g != padres[v]:
                aristas_bucle.append([g,v])
    dfs(0,marked,padres)
    return all(marked) and not aristas_bucle
```

5)

```python
#Ejercicio 5
def isComplete(graph):
    #El grado de todos los vertice va hacer igual al total de los vertices - 1
    grado = len(graph) - 1
    for g in graph:
        if grado != len(graph):
            return False
    return True
```

6)

```python
#Ejercicio 6
def convertTree(graph):
    marked = [False] * len(graph)
    padres = [-1] * len(graph)
    aristas_eliminadas = []
    def dfs(v,marked,aristas):
        marked[v] = True
        for g in graph[v]:
            if not marked[g]:
                padres[g] = v
                dfs(g,marked,aristas)
            elif padres[v] != g:
                aristas_eliminadas.append([g,v])

    dfs(0,marked,aristas_eliminadas)

    return aristas_eliminadas
```

7)

```python
#Ejercicio 7
def countConnection(Grafo):
    n = len(Grafo)
    visitados = [False] * n
    cant_cc = 0

    def dfs(v):
        visitados[v] = True
        for vecino in Grafo[v]:
            if not visitados[vecino]:
                dfs(vecino)

    for i in range(n):
        if not visitados[i]:
            dfs(i)
            cant_cc += 1

    return cant_cc
```

8)

```python
def bearthfirstsearch(graph,vertices):
    queue = myqueue.Queue()
    marked = [False] * len(vertices)
    queue.enqueue(vertices[0])
    bfs_list = []

    new_graph = [[] for _ in range(len(vertices))]
    m = vertices[0]

    while not queue.is_empty():
        v = queue.dequeue().value
        if not marked[v]:
            if m != v:
                new_graph[m].append(v)
            marked[v] = True
            bfs_list.append(v)
            for g in graph[v]:
                if not marked[g]:
                    queue.enqueue(g)
                    m = v

    return new_graph
```

9)

```python
def depthfirstsearch(graph,vertices):
    marked = [False] * len(vertices)
    pila = stack.Stack()
    pila.push(vertices[0])
    new_list = []
    n = 0
    while not pila.is_empty():
        v = pila.pop().value
        if not marked[v]:
            marked[v] = True
            new_list.append(v)
            for g in graph[v]:
                if not marked[g]:
                    pila.push(g)
    return new_list

def convertToDFSTree(dfs_list):
    new_graph = [[] for _ in range(len(vertices))]
    m = dfs_list[0]
    for v in range(1,len(dfs_list)):
        new_graph[m].append(dfs_list[v])
        m = dfs_list[v]
    return new_graph
```

10)

12)
Si el grafo g es un árbol es conexo y acíclico, tiene por propiedad (n-1) aristas y si se le agrega una nueva pasaria a tener (n-1+1) = (n) lo cual es una contradicción ya que debe tener n-1, en otras palabras la arista nueva que se agrega crea un ciclo en el grafo
13)
Supongamos por contradicción que la arista (u, v) no pertenece al árbol BFS y los niveles de u y v difieren en más de 1. Sea L_u el nivel de u en el árbol BFS y L_v el nivel de v en el árbol BFS. Sin pérdida de generalidad, supongamos que L_u > L_v + 1.

Como u está en el nivel L_u y v está en el nivel L_v, entonces hay un camino de longitud L_u - L_v entre u y v que no usa la arista (u, v) y es completamente contenido en el árbol BFS. Sea w el último vértice en este camino antes de v. Entonces el nivel de w en el árbol BFS debe ser L_v + 1, ya que de lo contrario, el camino de u a v usando la arista (u, v) y luego siguiendo el árbol BFS desde v a w tendría longitud L_u - L_v + 1, lo cual contradice nuestra suposición de que L_u > L_v + 1.

Ahora consideremos el árbol BFS que se forma a partir de w como raíz. Como w es un vértice del nivel L_v + 1 en el árbol BFS original, entonces en el nuevo árbol BFS, w es la raíz y v está en el nivel 1. Además, como (u, v) no está en el árbol BFS original, entonces (u, w) tampoco está en el árbol BFS original, y por lo tanto, (u, w) debe ser una arista hacia un nivel inferior a L_u.

Pero esto significa que hay un camino de u a w de longitud L_u - (L_v + 1) + 1 = L_u - L_v que no usa la arista (u, w) y es completamente contenido en el árbol BFS con raíz en w. Por lo tanto, el árbol BFS con raíz en w contiene un camino más corto de u a v que el árbol BFS original, lo cual es una contradicción, ya que el árbol BFS original es el árbol BFS mínimo. Por lo tanto, nuestra suposición inicial de que la arista (u, v) no pertenece al árbol BFS y los niveles de u y v difieren en más de 1 es falsa, y se sigue que si la arista (u, v) no pertenece al árbol BFS, entonces los niveles de u y v difieren a lo sumo en 1.