NOMBRE: ALVARO BLANCO
LEGAJO: 10622
TRABAJO PRACTICO Grafos parte I y II

Ejercicio 1

```
def createGraph(vertices,edges):
  Graph = {}
  for vertex in vertices:
    Graph[vertex] = []
  for vertex1, vertex2 in edges:
    Graph[vertex1].append(vertex2)
    Graph[vertex2].append(vertex1)
  return Graph
```

Ejercicio 2

```
def existPath(Grafo, v1, v2):
  visited = set()
  stack = [v1]
  while stack:
    vertex = stack.pop()
    if vertex == v2:
      return True
    if vertex not in visited:
      visited.add(vertex)
      stack.extend(Grafo[vertex])
  return False
```

Ejercicio 3

```
def isConnected(Grafo):
  vertices = list(Grafo.keys())
  for i in range(len(vertices)):
    v1 = vertices[i]
    for j in range(i + 1, len(vertices)):
      v2 = vertices[j]
      flag = existPath(Grafo, v1, v2)
      if flag == False:
        return False

  return True
```

Ejercicio 4

```
def isTree(Grafo):
  n = len(Grafo)
  edges = countEdges(Grafo)
  if isConnected(Grafo) and hasCycleDFS(Grafo) == False and edges == n-1:
    return True
  else:
    return False
```

```python
def countEdges(Grafo):
  contEdges = 0
  for vertex in Grafo:
    contEdges += len(Grafo[vertex])
  return contEdges // 2
```

```python
def hasCycleDFS(Grafo):
  visited = set()
  stack = []
  for vertex in Grafo:
    if vertex not in visited:
      stack.append((vertex, None))
      while stack:
        curr, prev = stack.pop()
        visited.add(curr)
        for neighbor in Grafo[curr]:
          if neighbor not in visited:
            stack.append((neighbor, curr))
          elif neighbor != prev:
            return True
  return False
```

Ejercicio 5

```python
def isComplete(Grafo):
  n = len(Grafo)
  for vertex in Grafo:
    if len(Grafo[vertex]) != n - 1:
      return False
  return True
```

Ejercicio 7

```python
def countConnections(Grafo):
  visited = set()
  count = 0
  for vertex in Grafo:
    if vertex not in visited:
      DFS_2(Grafo, vertex, visited)
      count += 1
  return count
```

Ejercicio 8

```python
def convertToBFSTree(Grafo, v):
  visited = set()
  q = Queue()
  visited.add(v)
  q.put(v)
  bfs = {v: []}
  while not q.empty():
    vertex = q.get()
    for neighbor in Grafo[vertex]:
      if neighbor not in visited:
        visited.add(neighbor)
        q.put(neighbor)
        bfs[vertex].append(neighbor)
        bfs[neighbor] = []
  return bfs
```

Ejercicio 9

```
def convertToDFSTree(Grafo, v):
  DFS_Tree = {v: []}
  visited = set()
  DFS(Grafo, v, v, DFS_Tree, visited)
  for vertex in Grafo:
    if vertex not in visited:
      DFS_Tree[vertex] = []
      DFS(Grafo, vertex, vertex, DFS_Tree, visited)
  return DFS_Tree

def DFS(Grafo, v, root, DFS_Tree, visited):
  visited.add(v)
  for adj_vertex in Grafo[v]:
    if adj_vertex not in visited:
      DFS_Tree[root].append(adj_vertex)
      DFS_Tree[adj_vertex] = []
      DFS(Grafo, adj_vertex, root, DFS_Tree, visited)
```

Ejercicio 10

```
def bestRoad(Grafo, v1, v2):
  visited = set()
  q = Queue()
  visited.add(v1)
  q.put((v1, []))

  while not q.empty():
    vertex, ruta = q.get()
    if vertex == v2:
      return ruta + [vertex]
    for neighbor in Grafo[vertex]:
      if neighbor not in visited:
        visited.add(neighbor)
        q.put((neighbor, ruta + [vertex]))
  return []
```

Ejercicio 12

Si estamos en presencia de un arbol eso significa que hay solo un camino de la raiz a cada nodo del arbol, si yo agrego una arista entre cualquier par de nodos del arbol esto nos lleva a tener 2 caminos distintos para llegar al mismo nodo, lo que es en efecto un ciclo.

Ejercicio 13

Si la arista (u,v) no pertenece al arbol BFS. Esto significa que la unica forma en que se puede llegar a v desde la raiz es a traves de otro vertice, digamos w, que ya ha sido descubierto y explorando la arista (w,v). Por lo tanto, v no puede estar en un nivel superior al de w en el arbol BFS, ya que de lo contrario se habria descubierto antes que w y no se habria necesitado una arista adicional para alcanzarlo.