Trabajo Práctico
Grafos
Tomás Rando - 14004 - LCC

1)
createGraph(List1, List2)

```
#O(A + V)
def createGraph(ListA, ListB):
    n = linkedlist.length(ListA)
    Graph = Array(n, linkedlist.LinkedList())
    for i in range(0, n):
        Graph[i] = linkedlist.LinkedList()

    currentNode = ListB.head
    while currentNode != None:
        linkedlist.add(Graph[currentNode.value[0]], currentNode.value[1])
        linkedlist.add(Graph[currentNode.value[1]], currentNode.value[0])
        currentNode = currentNode.nextNode
    return Graph
```

2)
existPath(Grafo, v1, v2)

```
def existPath(Grafo, v1, v2):
    n = len(Grafo)
    for i in range(0, n):
        node = linkedlist.Node()
        node.value = "W"
        node.nextNode = Grafo[i].head
        Grafo[i].head = node
    stack = linkedlist.LinkedList()
    linkedlist.push(stack, v1)
    currentNode = stack.head
    condition = False
    while currentNode != None:
        linkedlist.pop(stack)
        Grafo[currentNode.value].head.value = "G"
        currentNode2 = Grafo[currentNode.value].head.nextNode
        while currentNode2 != None:
            if Grafo[currentNode2.value].head.value == "W":
                Grafo[currentNode2.value].head.value = "G"
                linkedlist.push(stack, currentNode2.value)
            if currentNode2.value == v2:
                condition = True
                stack.head = None
                break
            currentNode2 = currentNode2.nextNode
        Grafo[currentNode.value].head.value = "B"
        currentNode = stack.head
    for i in range(0, n):
        linkedlist.pop(Grafo[i])
    return condition
```

3)
isConnected(Grafo)

```
def isConnected(Grafo):
    return isConnectedWithBfs(Grafo)
def isConnectedWithBfs(Grafo):
    n = len(Grafo)
    for i in range(0, n):
        node = linkedlist.Node()
        node.value = "W"
        node.nextNode = Grafo[i].head
        Grafo[i].head = node
    stack = linkedlist.LinkedList()
    linkedlist.push(stack, 0)
    currentNode = stack.head
    while currentNode != None:
        linkedlist.pop(stack)
        Grafo[currentNode.value].head.value = "G"
        currentNode2 = Grafo[currentNode.value].head.nextNode
        while currentNode2 != None:
            if Grafo[currentNode2.value].head.value == "W":
                Grafo[currentNode2.value].head.value = "G"
                linkedlist.push(stack, currentNode2.value)
            currentNode2 = currentNode2.nextNode
        Grafo[currentNode.value].head.value = "B"
        currentNode = stack.head
    condition = True
    for i in range(0, n):
        if Grafo[i].head.value == "W":
            condition = False
        linkedlist.pop(Grafo[i])
    return condition
```

4)
isTree(Grafo)

```
def isTree(Grafo):
    if isConnected(Grafo) == False:
        return False
    return isTreeWithBfs(Grafo)
```

```
def isTreeWithBfs(Grafo):
    n = len(Grafo)
    for i in range(0, n):
        node = linkedlist.Node()
        node.value = "W"
        node.nextNode = Grafo[i].head
        Grafo[i].head = node
    stack = linkedlist.LinkedList()
    linkedlist.push(stack, 0)
    currentNode = stack.head
    condition = True
    while currentNode != None:
        linkedlist.pop(stack)
        Grafo[currentNode.value].head.value = "G"
        currentNode2 = Grafo[currentNode.value].head.nextNode
        while currentNode2 != None:
            if Grafo[currentNode2.value].head.value == "W":
                Grafo[currentNode2.value].head.value = "G"
                linkedlist.push(stack, currentNode2.value)
            else:
                if Grafo[currentNode2.value].head.value == "G":
                    condition = False
            currentNode2 = currentNode2.nextNode

        if condition == True:
            Grafo[currentNode.value].head.value = "B"
            currentNode = stack.head
        else:
            break
    for i in range(0, n):
        linkedlist.pop(Grafo[i])
    return condition
```

5)
isComplete(Grafo)

```
#O(V*V), podria mejorarse a O(V) con listas de python
def isComplete(Grafo):
    n = len(Grafo)
    for i in range(0, n):
        if (n-1) != linkedlist.length(Grafo[i]):
            return False
    return True
```

6)
convertTree(Grafo)

```
def convertTree(Grafo):
    return treeWithBfs(Grafo)
```

```
"""
W, G, B son abreviaciones para White, Gray y Black respectivamente. Utiliza el recorrido BFS para encontrar ciclos.
"""
def treeWithBfs(Grafo):
    auxList = linkedlist.LinkedList()
    n = len(Grafo)
    for i in range(0, n):
        node = linkedlist.Node()
        node.value = "W"
        node.nextNode = Grafo[i].head
        Grafo[i].head = node
    stack = linkedlist.LinkedList()
    linkedlist.push(stack, 0)
    currentNode = stack.head
    while currentNode != None:
        linkedlist.pop(stack)
        Grafo[currentNode.value].head.value = "G"
        currentNode2 = Grafo[currentNode.value].head.nextNode
        while currentNode2 != None:
            if Grafo[currentNode2.value].head.value == "W":
                Grafo[currentNode2.value].head.value = "G"
                linkedlist.push(stack, currentNode2.value)
            else:
                if Grafo[currentNode2.value].head.value == "G":
                    linkedlist.add(auxList, (currentNode2.value, currentNode.value))
            currentNode2 = currentNode2.nextNode
        Grafo[currentNode.value].head.value = "B"
        currentNode = stack.head
    for i in range(0, n):
        linkedlist.pop(Grafo[i])
    return auxList
```

7)
countConnections(Grafo)

```
def countConnections(Grafo):
    return countConnectionsWithBfs(Grafo)
def countConnectionsWithBfs(Grafo):
    n = len(Grafo)
    for i in range(0, n):
        node = linkedlist.Node()
        node.value = "W"
        node.nextNode = Grafo[i].head
        Grafo[i].head = node
    stack = linkedlist.LinkedList()
    linkedlist.push(stack, 0)
    currentNode = stack.head
    contador = 1
    while currentNode != None:
        linkedlist.pop(stack)
        Grafo[currentNode.value].head.value = "G"
        currentNode2 = Grafo[currentNode.value].head.nextNode
        while currentNode2 != None:
            if Grafo[currentNode2.value].head.value == "W":
                Grafo[currentNode2.value].head.value = "G"
                linkedlist.push(stack, currentNode2.value)
            currentNode2 = currentNode2.nextNode
        Grafo[currentNode.value].head.value = "B"
        currentNode = stack.head
        if currentNode == None:
            for i in range(0, n):
                if Grafo[i].head.value == "W":
                    contador += 1
                    linkedlist.push(stack, i)
                    currentNode = stack.head
                    break
    for i in range(0, n):
        linkedlist.pop(Grafo[i])
    return contador
```

8)

convertToBFSTree(Grafo, v)

```
def convertToBFSTree(Grafo, v):
    n = len(Grafo)
    newGraph = Array(n, linkedlist.LinkedList())
    for i in range(0, n):
        newGraph[i] = linkedlist.LinkedList()
        node = linkedlist.Node()
        node.value = "W"
        node.nextNode = Grafo[i].head
        Grafo[i].head = node
    stack = linkedlist.LinkedList()
    linkedlist.push(stack, v)
    currentNode = stack.head
    while currentNode != None:
        linkedlist.pop(stack)
        Grafo[currentNode.value].head.value = "G"
        currentNode2 = Grafo[currentNode.value].head.nextNode
        while currentNode2 != None:
            if Grafo[currentNode2.value].head.value == "W":
                Grafo[currentNode2.value].head.value = "G"
                linkedlist.push(stack, currentNode2.value)
                linkedlist.add(newGraph[currentNode.value], currentNode2.value)
                linkedlist.add(newGraph[currentNode2.value], currentNode.value)
            currentNode2 = currentNode2.nextNode
        Grafo[currentNode.value].head.value = "B"
        currentNode = stack.head
    for i in range(0, n):
        linkedlist.pop(Grafo[i])
    return newGraph
```

9)

convertToDFSTree(Grafo, v)

```
def convertToDFSTree(Grafo, v):
    n = len(Grafo)
    newGraph = Array(n, linkedlist.LinkedList())
    for i in range(0, n):
        newGraph[i] = linkedlist.LinkedList()
        node = linkedlist.Node()
        node.value = "W"
        node.nextNode = Grafo[i].head
        Grafo[i].head = node
    Grafo[v].head.value = "G"
    newGraph = convertToDFSTreeR(Grafo, newGraph, v, v)
    for i in range(0, n):
        if Grafo[i].head.value == "W":
            Grafo[i].head.value = "G"
            newGraph = convertToDFSTreeR(Grafo, newGraph, i, i)
    for i in range(0, n):
        linkedlist.pop(Grafo[i])
    return newGraph

def convertToDFSTreeR(Grafo, newGraph, lastVertex, v):
    currentNode = Grafo[v].head.nextNode
    while currentNode != None:
        if currentNode.value != lastVertex:
            if Grafo[currentNode.value].head.value == "W":
                Grafo[currentNode.value].head.value = "G"
                linkedlist.add(newGraph[v], currentNode.value)
                linkedlist.add(newGraph[currentNode.value], v)
                convertToDFSTreeR(Grafo, newGraph, v, currentNode.value)
        currentNode = currentNode.nextNode
    Grafo[v].head.value = "B"
    return newGraph
```

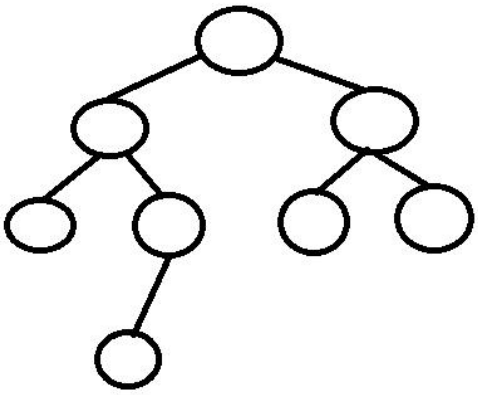
10)

bestRoad(Grafo, v1, v2)

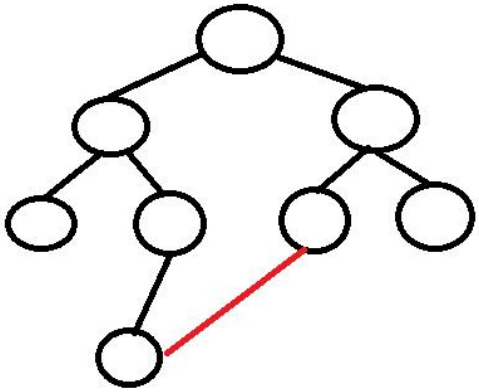
```
def bestRoad(Grafo, v1, v2):
    if (len(Grafo) < v1) or len(Grafo) < v2:
        return []
    newGraph = convertToBFSTree(Grafo, v1)
    condition = bestRoadR(newGraph, v2, [v1], v1, v1)
    if condition != False:
        return condition
    else:
        return []

def bestRoadR(Grafo, v2, auxList, lastVertex, Vertex):
    if linkedlist.search(Grafo[Vertex], v2) != None:
        auxList.append(v2)
        return auxList
    condition = False
    currentNode = Grafo[Vertex].head
    while currentNode != None:
        if currentNode.value != lastVertex:
            auxList.append(currentNode.value)
            condition = bestRoadR(Grafo, v2, auxList, Vertex, currentNode.value)
            if condition == False:
                auxList.pop()
            else:
                return condition
        currentNode = currentNode.nextNode
    return False
```

12)

Por propiedad, un grafo de n vértices tiene n-1 aristas en total. Si tenemos un árbol normal de m vértices, entonces este tendrá m - 1 aristas. Si le agregamos una arista más, obtenemos m aristas, por lo que no se cumple la propiedad y es falso.



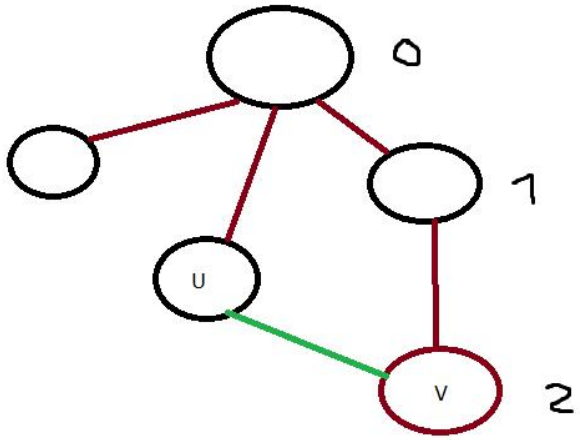
8 Vértices y 7 aristas. No hay ciclos



8 Vértices y 8 aristas. Se forma un ciclo

13)

Si una arista está conectada tanto a v como a u, eso significa que v está un nivel anterior o superior al de u. El algoritmo BFS recorre el grafo por niveles, añadiendo las aristas al árbol teniendo en cuenta eso. Si una arista (v,u) no ha sido agregada, significa que el vértice v fue encontrado añadiendo los vértices adyacentes a otro vértice del nivel de u y que se encuentra en el nivel inferior a este. Puesto que si no lo estuviese, hubiese sido agregado al recorrer u, y la arista (v, u) si existiría en el árbol BFS.



Por ejemplo, en este árbol, la arista (u,v) (La verde), no fue agregada porque el BFS recorrió añadiendo los vértices por niveles, y en el nivel 1 recorrió desde los vértices de la derecha hacia la izquierda y se encontró una arista que une a v. Por lo explicado anteriormente, el nivel del vértice u y v, difiere en 1 a lo sumo, puesto que de lo contrario, hubiese sido agregada al agregar las aristas de u, ya que va recorriendo por niveles.