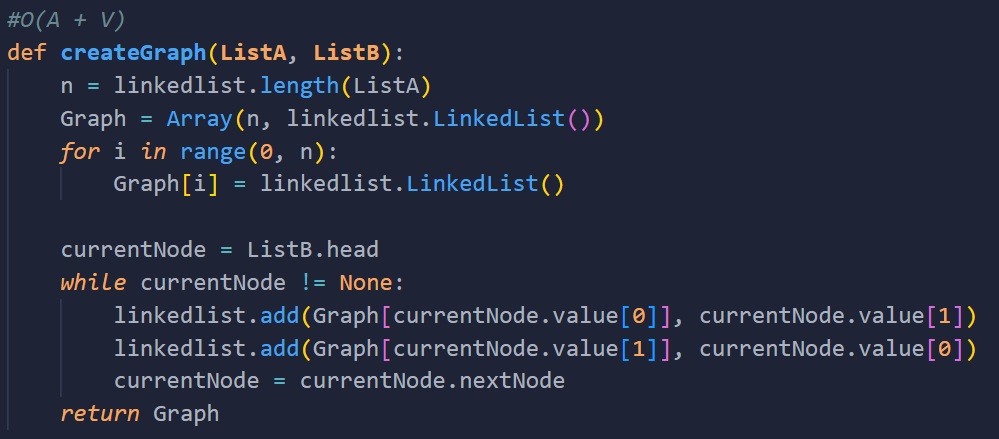
**Trabajo Práctico**

**Grafos**

**Tomás Rando - 14004 - LCC**

**1)**

**createGraph(List1, List2)**



**2)**

**existPath(Grafo, v1, v2)**



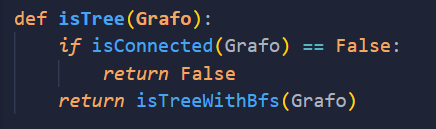
**3)**

**isConnected(Grafo)**



**4)**

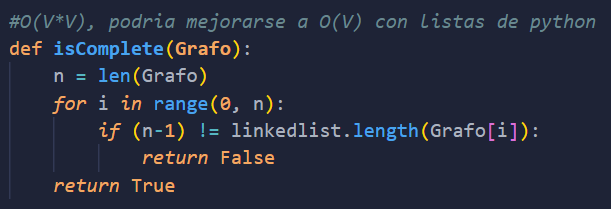
**isTree(Grafo)**





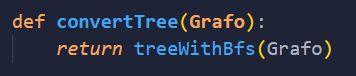
**5)**

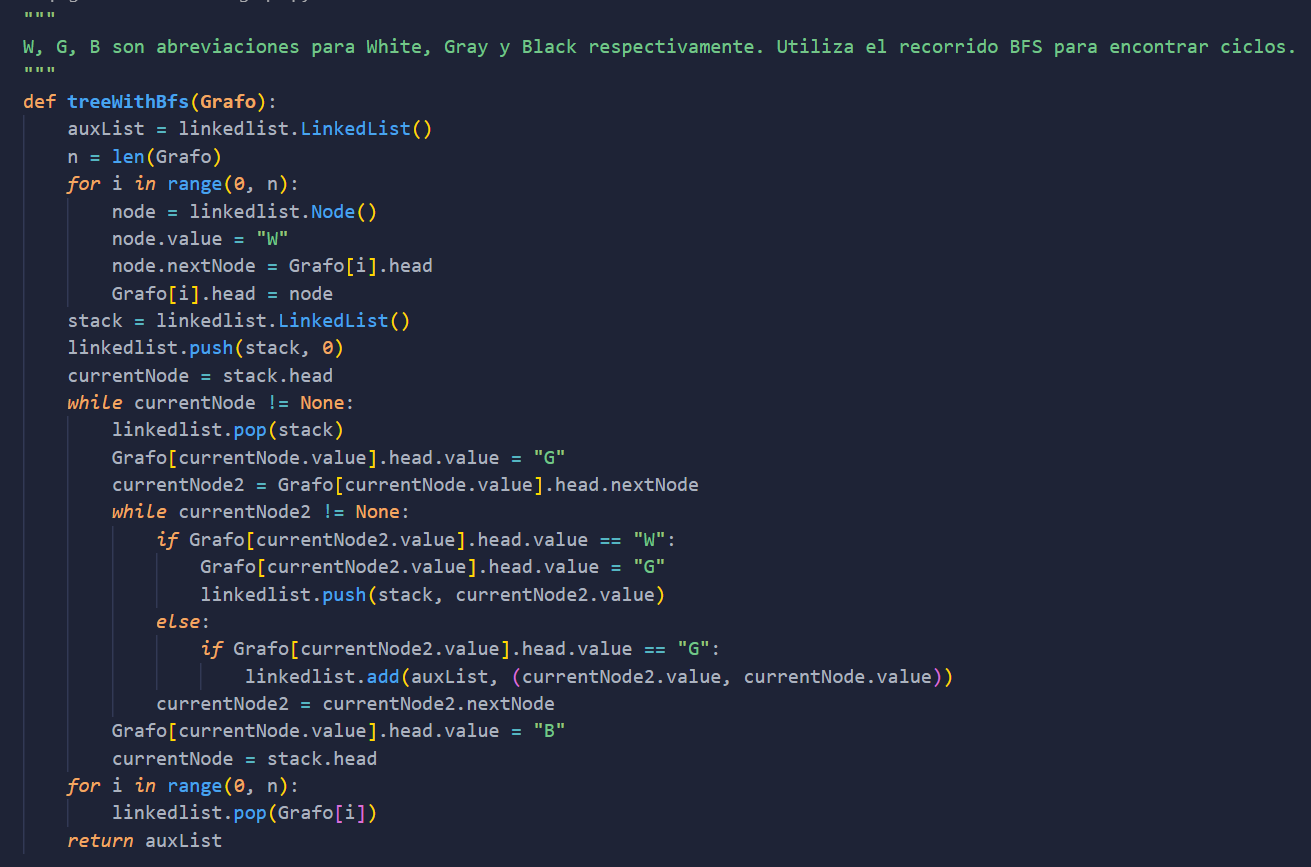
**isComplete(Grafo)**



**6)**

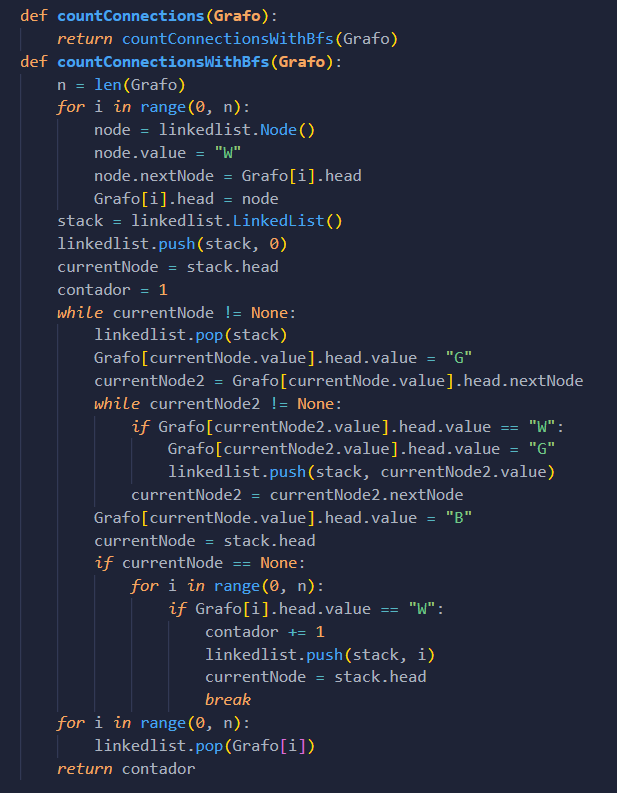
**convertTree(Grafo)**





**7)**

**countConnections(Grafo)**



**convertToBFSTree(Grafo, v)**

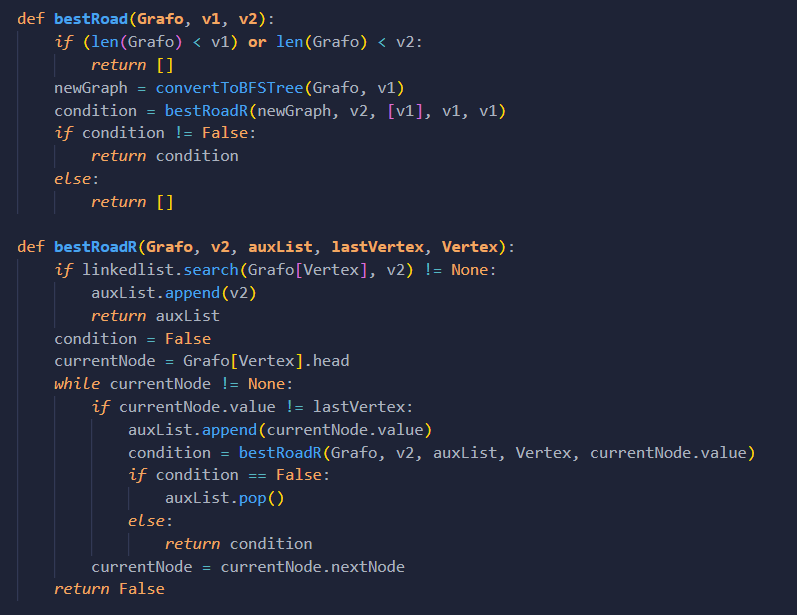


**convertToDFSTree(Grafo, v)**



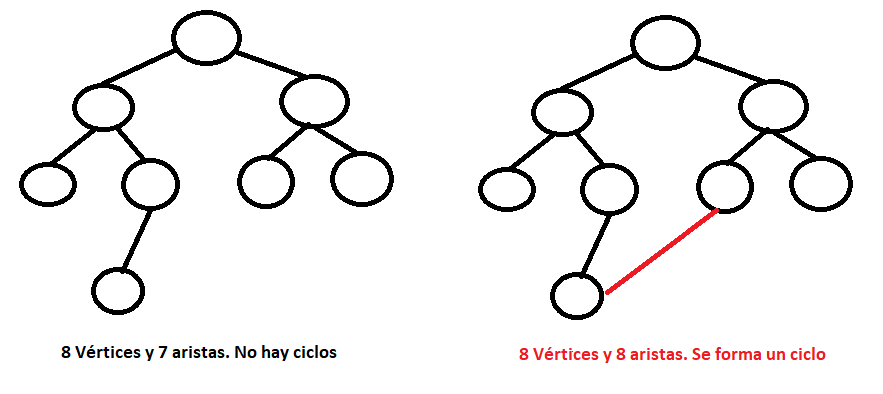
**10)**

**bestRoad(Grafo, v1, v2)**



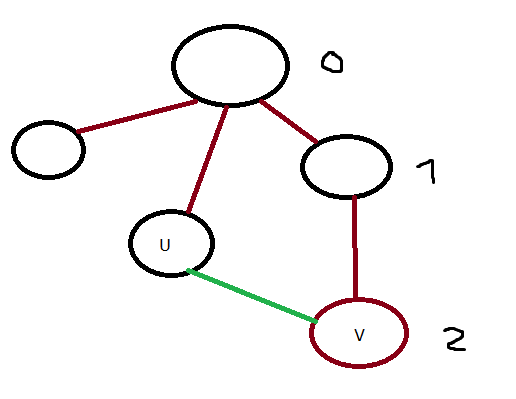
**12)**

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.



**13)**

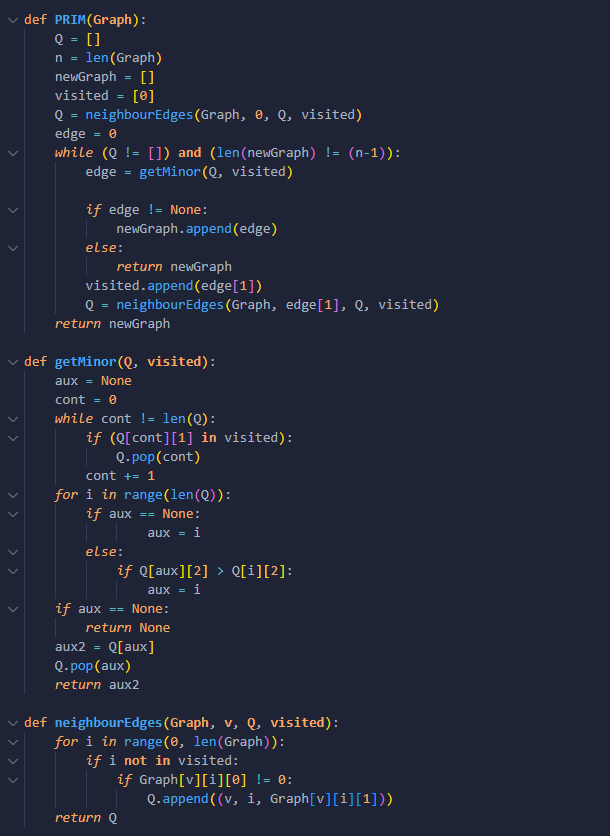
Si una arista está conectada tanto a v como a u, eso significa que v está un nivel anterior o superior al de u. El algoritmo BFS recorre el grafo por niveles, añadiendo las aristas al árbol teniendo en cuenta eso. Si una arista (v,u) no ha sido agregada, significa que el vértice v fue encontrado añadiendo 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.

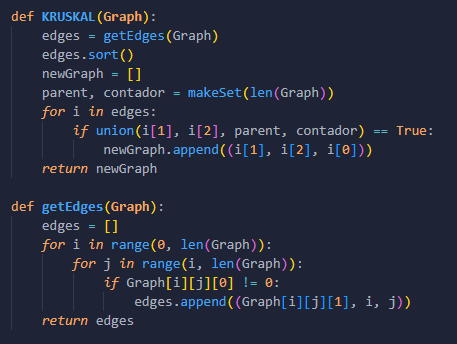
**14)**

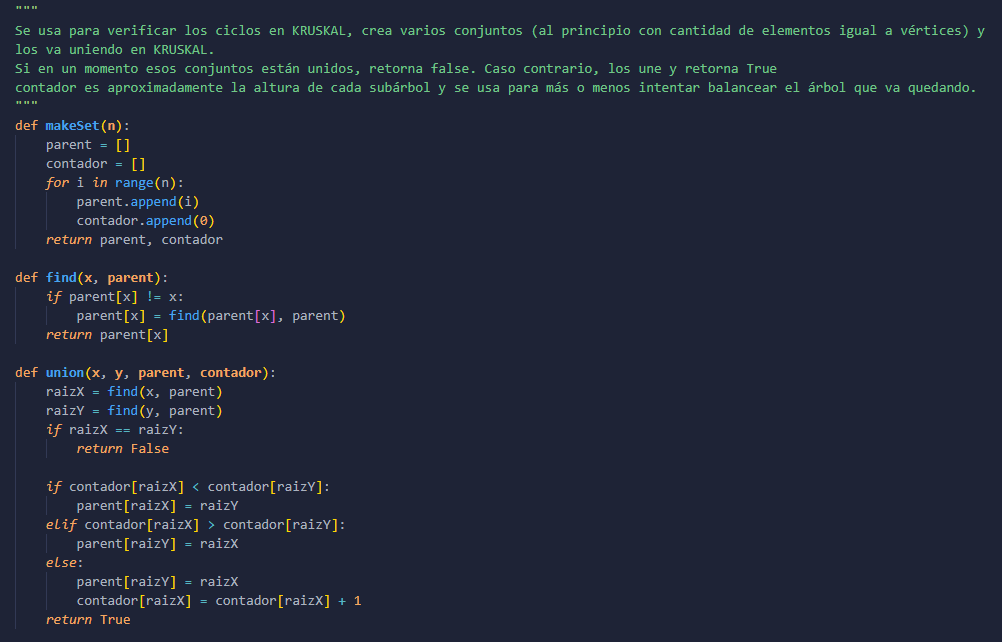
**PRIM(Grafo)**



**15)**

**KRUSKAL(Grafo)**

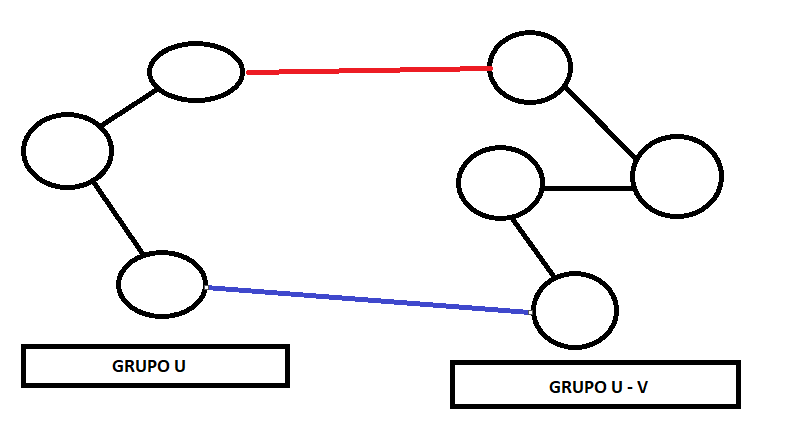




Si tenemos dos árboles abarcadores mínimos U y V-U y existe al menos una arista (u, v) tal que u pertenece a U y v pertenece a V-U, entonces, sí o sí pertenecerá al árbol abarcador mínimo (siempre que sea de coste mínimo). Esto se puede afirmar gracias a que la arista es de costo mínimo, puesto que un AACM como su nombre lo índica, posee todas las aristas de menor valor, y, al existir por lo menos la arista (u, v), entonces el árbol U y el árbol V-U deberán conectarse necesariamente, de lo contrario quedaría un bosque.

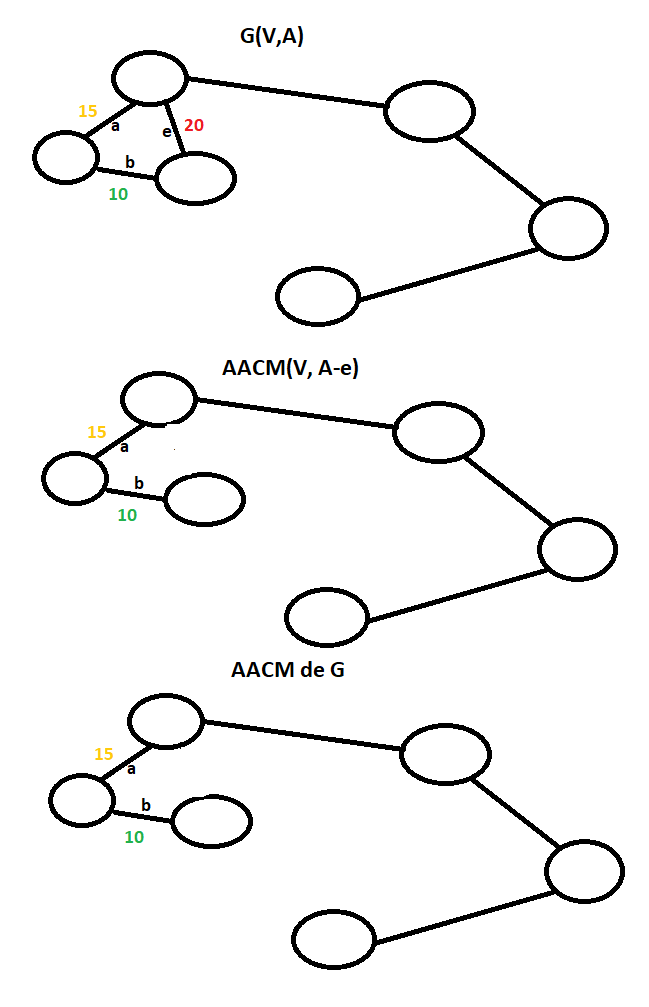
Si hubiesen dos aristas que conectasen U y V-U, llámese (u1, v1), igualmente (u, v) sería la que prevalecería, puesto que, nuevamente, es la arista con menor coste entre las dos. Si sucediese que ambas tuvieran el mismo valor, pues habrían dos árboles abarcadores posibles, y seguiría siendo verdadero, ya que se puede seleccionar el que tiene a (u,v).

En el ejemplo se muestra como se seleccionaría la roja en el caso de que tuviera menor valor que la azul, y si ambas tuviesen el mismo, pues se podría elegir nuevamente el de la roja para formar el AACM



**17)**

Sabemos que en un AACM no pueden existir ciclos y siempre se seleccionan las aristas de menor costo. Por ello, si se quita la arista que provoca el ciclo y además es la de mayor coste, entonces se generará un grafo sin ese ciclo, en el que si se hace el AACM no habrá problema al agregar aristas, ya que no se producirá un ciclo. Ese AACM será igual que el AACM del grafo original, ya que, al ser un árbol de costo mínimo, siempre se seleccionarán las aristas de menor coste, provocando que la arista eliminada (e), no se encuentre igualmente.



**18)**

Esto es demostrado por la propiedad de Kruskal, si G(V, E) es un grafo conexo, no dirigido y ponderado.

Si A es un subconjunto de E que está incluido en algún AACM

Si C = (Vc, Ec) es una componente conexa en el bosque Ga = (V, A).

Si (u, v) es una arista de costo mínimo que conecta C con otro componente de Ga, entonces existe un AACM para G que contiene (u, v) entre sus aristas.

Por ello, el enunciado es verdadero.

1. En vez de seleccionar y guardar las aristas de menor coste, guardar las de mayor coste
2. Seleccionar las aristas en un orden arbitrario, sin tener en cuenta el coste.
3. Se podría intentar cambiar los valores de las aristas pertenecientes a E con el menor coste posible. De esta manera, existe al menos un AACM que cumpla con lo solicitado

**20)**

Si se tiene en cuenta que toda arista tiene coste mínimo, entonces es posible usar un recorrido para encontrar las aristas a agregar. Ya que es posible agregar cualquier arista siempre y cuando no se forme un ciclo

Matriz =[len(Graph), len(Graph)]

Bfs(Graph, 0):

for i in Graph

v = i

for i in ady[v]

If i == “W”

M[v, i] = 1

M[i, v] = 1

**21**)

**shortestPath(Grafo, s, v) (o dijkstra)**

