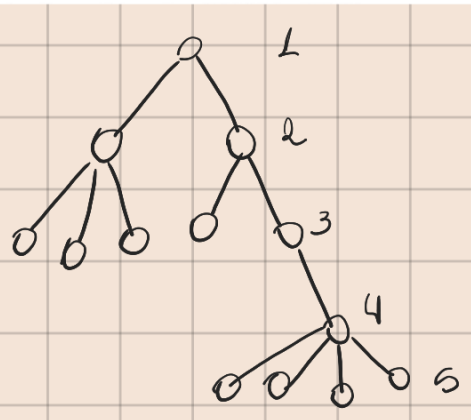
Recorridos en profundidad

1. ⋆Sea T un árbol generador de un grafo (conexo) G con raíz r, y sean V y W los vértices que están a distancia par e impar de r, respectivamente.

   a) Observar que si existe una arista $vw \in E(G) \setminus E(T)$ tal que $v, w \in V$ o $v, w \in W$, entonces el único ciclo de $T \cup \{vw\}$ tiene longitud impar.



Si $v, w \in V \longrightarrow d(v,r) + d(w,r) \equiv 0(2) \rightarrow$ si $\exists\, vw \in E(G) \setminus E(T)$

$\rightarrow$ si considero $P_v + P_w + vw \rightarrow$ tengo un ciclo. Como $P_v + P_w \equiv 0(2) \rightarrow$

$P_v + P_w + vw \equiv 1(2)$ (camino de v y w a r)

$\rightarrow T \cup \{vw\}$ tiene un único ciclo de longitud impar

(es único xq T es árbol)

Para $v, w \in W$ vale lo mismo pues $d(w,r) + d(v,r) \equiv 0(2)$

   b) Observar también que si toda arista de $E(G) \setminus E(T)$ une un vértice de V con otro de W, entonces (V, W) es una bipartición de G y, por lo tanto, G es bipartito.

Si $\forall\, vw \,/\, vw \in E(G) \setminus E(T)$ $v \in V$ y $w \in W \longrightarrow T \cup \{vw\}$ tiene un! ciclo par $\rightarrow \forall\, c$: ciclo de G, la longitud de c es par $\rightarrow G$ es bipartito

c) A partir de las observaciones anteriores, diseñar un algoritmo lineal para determinar si un grafo conexo G es bipartito. En caso afirmativo, el algoritmo debe retornar una bipartición de G. En caso negativo, el algoritmo debe retornar un ciclo impar de G. **Explicitar cómo es la implementación del algoritmo**; no es necesario incluir el código.

Algoritmo

1. Paso de lista de aristas a lista de adyacencias
2. Hago DFS sobre mi grafo y me guardo el árbol T → no va a tener cross edges
3. Recorro el árbol T y guardo el conjunto de vértices a distancias par de r en V y los que están a distancia impar de r en W. A medida que guardo los marco. verde par, rojo impar.
4. Me guardo en u el 1er vértice de W
5. Dado $v \in V(G)$ y mientras u no esté marcado de verde $O(n+m)$
   busco si en $N(v)$ hay algún vértice en verde. Si lo hay guardo en u ese vértice y me guardo v. Sino, no hago nada
6. Guardo en z el 1er vértice de V
7. Dado $w \in W$ y mientras z no sea rojo $O(n+m)$
   busco si en $N(w)$ hay algún rojo, si lo hay lo guardo en z y me guardo w. Sino, no hago nada
8. Si u = rojo y z = verde → devuelvo W y V
9. Si u = verde → busco el ancestro común x más cercano a u y a b en T y devuelvo $P_{uax} + P_{xab} + u$

10. Si u ≠ verde y z = rojo → busco el ancestro común más cercano a w y a z en T y devuelvo $P_{zax} + P_{xaw} + z$.

d) Generalizar el algoritmo del inciso anterior a grafos no necesariamente conexos observando que un grafo G es bipartito si y solo si sus componentes conexas son bipartitas.

Para generalizar el algoritmo puedo aplicarlo en c/componente y devolver lo mismo que antes solo que le sumo la marca que le corresponde a la componente

Algoritmo

1. Paso de lista de ajustar a adyacencias
2. Hago DFS sobre el grafo y guardo el bosque T.
3. Aplico el algoritmo anterior sobre c/componente y devuelvo la raíz de la componente y el resultado. Si devolvió un ciclo en algún momento ahí tenemos la justificación de por qué NO es bipartito