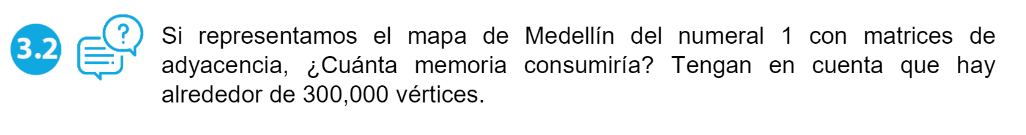
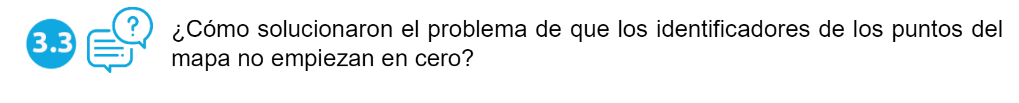
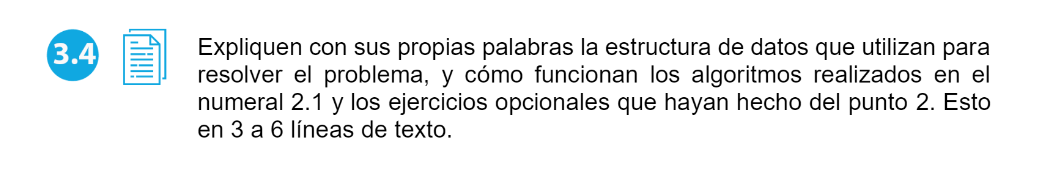
Para resolver este problema usamos tablas de hash, dentro de esta hay una pareja como key la cual es **Id** del vértice y **nombre** del vértice, y el valor es una LinkedList de parejas la cual es el **ID** del otro vértice y a que **distancia** esta de ese vértice. Usamos bufferReader y un StringTokenizer para poder separar los datos dentro del txt, además usamos Longs en vez de int porque algunas id eran demasiado grandes para poder ser almacenadas en un int.

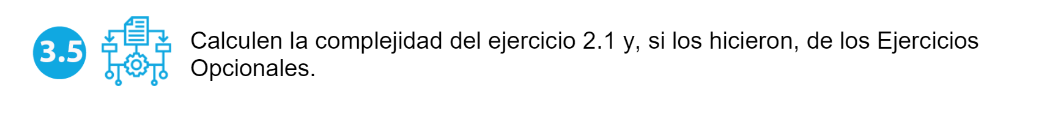
Una matriz de adyacencia es una la cual se comporta por el numero de elementos al cuadrado por lo que si son 300,000 serian: 90’000.000.000

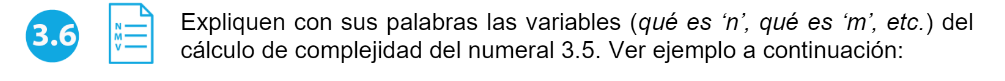


No nos ocurrió este problema, no se si fue debido a que utilizamos long en vez de int.



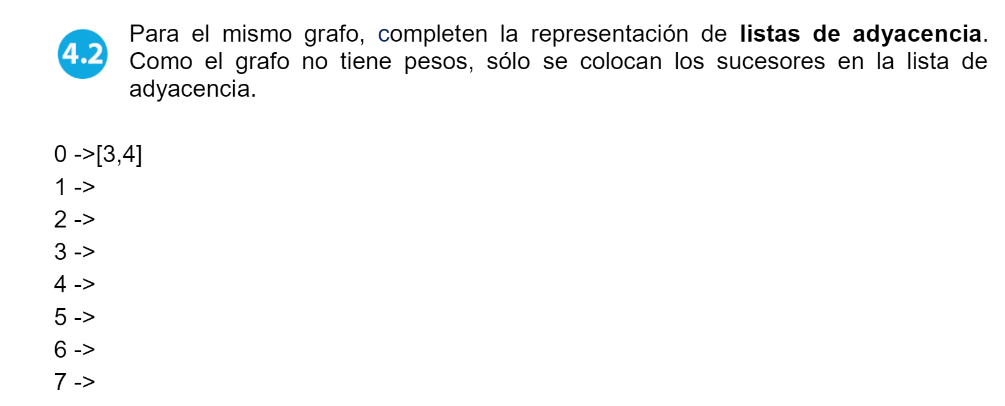
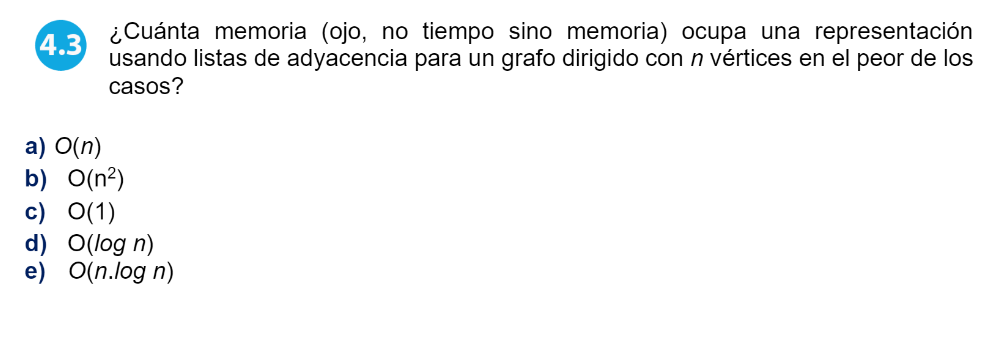
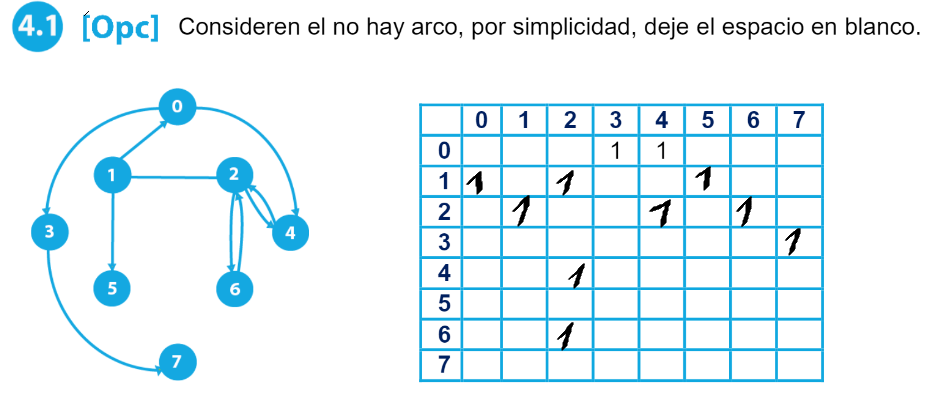
Para la implementación de los grafos usamos matrices de adjacencia. Inicializamos todos pesos en 0, asi sabemos que si dos vertices no estan conectados tendran peso 0.

El metodo para determinar si es Bicolorable o no, se basa en la teoria de grafos, con los grafos bipartitos La clase a travez de una lista saca los sucesores de cada nodo y revisa por cada sucesor que los sucesores de el mismo no esten en los sucesores del nodo inicial.

El algoritmo que estamos utilizando revisa los sucesores de cada nodo para ver si coinciden con los sucesores de otro de los nodos, teniendo esto en cuenta, la complejidad del mismo dependeria de la cantidad de nodos y de la cantidad de sucesores de cada nodo, entonces esta estaria dada por n\*m.

n: es la cantidad de nodos del grafo.

m: es la cantidad de sucesores de cada uno de los nodos.



🡨

0 ->[3,4]

1 ->[0,2,5]

2 ->[1,4,6]

3 ->[7]

4 ->[2]

5 ->[]

6 ->[2]

7 ->[]