UNIVERSIDAD SIMON BOLIVAR
LABORATORIO DE ALGORITMOS Y ESTRUCTURAS III
SECCION 2
GRUPO 20
INTEGRANTES: Alberto Cols, 09-10177
Matteo Ferrando, 09-10285

INFORME PROYECTO 1

Para la implementación de los grafos utilizamos una lista de adyacencias implementada con un Arreglo Dinámico que contiene pares (A, B), siendo A el nodo de ID correspondiente a la posición en el arreglo y B un apuntador a una Lista<Lados> que contiene todos los sucesores del nodo A; el Arreglo Dinámico y Lista son una implementación de la interfaz Secuencia<T>.

Pudimos haber hecho un Arreglo de apuntadores a Listas para simplificar el problema, siendo el indice donde se encontraba este apuntador el ID del Nodo, pero la decisión de hacer un Arreglo de Pares fue para poder almacenar cada Nodo y no simplemente tener un indicio de cuanto valdría el ID del nodo según la posición del arreglo, sin embargo haciendo esto nos esta trayendo "problemas" con la memoria, ya que no podemos agregar más de 1.800.000 nodos aproximadamente. Entonces, a pesar de este problema, lo implementamos de esta manera ya que según el enunciado del Proyecto, debemos agregar Nodos a nuestro grafo, no simplemente tener indicios de que Nodo podría estar almacenado en cada posición, y, aún más importante, de la otra forma, no podríamos agregar un Nodo cuyo ID no dependa directamente de la posición del arreglo en la que será almacenado.

Además, utilizamos arreglos dinámicos ya que, a pesar de que el costo de clonar un arreglo para poder agrandarlo es elevado, al sacar un promedio de todas las operaciones que se realizarán nos dimos cuenta de que el Costo Amortizado es constante, por lo cual, el programa trabajará de una forma eficiente. Mediante esta estructura (Lista de Adyacencias), los métodos para obtener la lista de sucesores de un nodo y para agregar un lado o un nodo cualquiera, son de orden constante; y para poder eliminar un lado o determinar si un nodo es sucesor de otro, tenemos métodos cuya complejidad es O(numero de lados).

Por último, al implementar el archivo GrafoNoDirigido.java decidimos hacerlo subclase de GrafoDirigido.java para facilitar la implementación. Esta idea surgió al preguntarnos "¿Es un grafo no dirigido un grafo dirigido?" y, aunque no se cumple exactamente esta proposición, dedujimos que sí, ya que puede observarse que cada lado “no dirigido” son dos lados dirigidos, uno que va del origen al destino y el otro en sentido contrario, por lo que ciertos métodos de nuestra super clase podrían ser re-utilizados en dicha sub-clase. Aparte de estas dos clases, tenemos la clase Nodo.java, Lado.java y Par.java que son las clases de los objetos creados para poder implementar el grafo, y al final, tendremos la interfaz para tipos genéricos Secuencia.java que posee dos implementaciones: Arreglo.java y Lista.java