Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software.

Nombre de la Práctica Grafos con listas de adyacencia.

Numero de Práctica 31.

Nombre de la carrera Ingeniería de Software.

Nombre de la materia Lab. Estructuras de Datos.

Nombre del alumno Jesús Manuel Juárez Pasillas.

Nombre del docente Aldonso Becerra Sánchez.

Fecha: 22/11/2021.

Práctica 31: Grafos con listas de adyacencias.

Introducción:

Los grafos con listas de adyacencia nos permiten representar grafos utilizando memoria dinámica, lo que conlleva que no se utilicen arreglos donde se almacenan los vértices y tampoco se utilicen las matrices de adyacencia, en cambio de utilizar estas dos cosas se utiliza una lista encadenada la cual almacena listas encadenadas de cada vértice con sus adyacencias. El primer elemento de estas listas siempre es el nodo real, los demás vértices agregados son las adyacencias de ese vértice y representan las aristas que tiene el nodo con los demás vértices.

Desarrollo:

En el desarrollo de esta practica se pide que se obtengan los componentes conexos de un grafo comenzando de un vértice pasado como parámetro. Para esto se hizo uso del recorrido en profundidad el cual implementa su funcionalidad y se agrega algunas cosas más para seguir agregando componentes.

La idea es que dado un vértice del cual comenzar se obtenga todos los vértices que se pueden recorrer comenzando de un origen. Luego de haber terminado este recorrido y si aún hay vértices sin marcar, haga el primer vértice sin marcar el vértice origen y con esto sacar otra vez su recorrido, y así hasta que todos los nodos estén recorridos.

Para cada componente conexo obtenido (el cual será una lista encadenada), se irán agregando a una lista la cual contendrá todos los componentes conexos obtenidos del grafo comenzando por el vértice indicado. Para poder obtener esto se hicieron dos métodos dentro de la clase GrafoListaAdyacencia:

- Obtiene todos los componentes conexos del grafo comenzando sobre un vértice dado. Cada componente conexo es una ListaEncadenada la cual se agrega a la lista que se regresa.
  `public ListaEncadenada componenetesConexos(Object origen).`

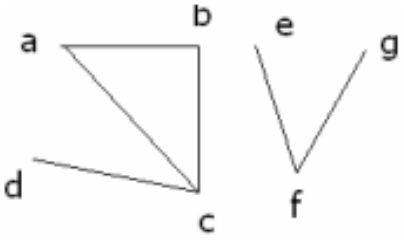
- Verifica si es que todos los vértices están marcados, si no es así agrega el primero que encuentra que no está marcado a la pila y lo marca, si todos están marcados no agrega nada.
  `private void verificarVerticesNoMarcados(ListaEncadenada marcados,ListaPila pila).`

Nota: Toda la documentación del proyecto esta agregada en la carpeta “doc” dentro de la carpeta del proyecto (“edylab_2021_31/doc”).

Capturas del programa funcionando:

La prueba esta almacenada en el paquete pruebas, y la clase se llama **PruebaComConexos.**

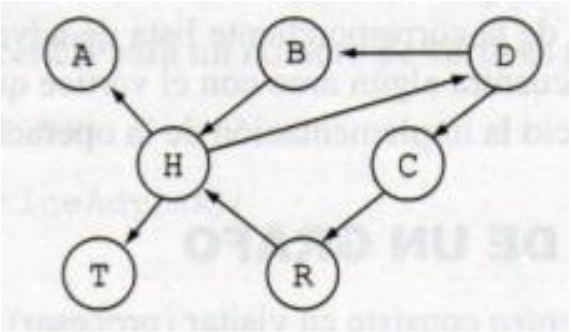
Para la prueba, se hicieron los dos grafos que vienen en las indicaciones de la practica.

1. 

.
    a. En el primer grafo se le asigno como origen en vertice **a:**

```
a -> b -> c -> null
b -> a -> c -> null
c -> a -> b -> d -> null
d -> c -> null
e -> f -> null
f -> e -> g -> null
g -> f -> null
Componentes conexos:
a -> c -> d -> b -> null
e -> f -> g -> null
```

Como se puede observar se obtuvieron 2 componentes conexos, los cuales cubren todos los nodos del grafo.

2. 

.

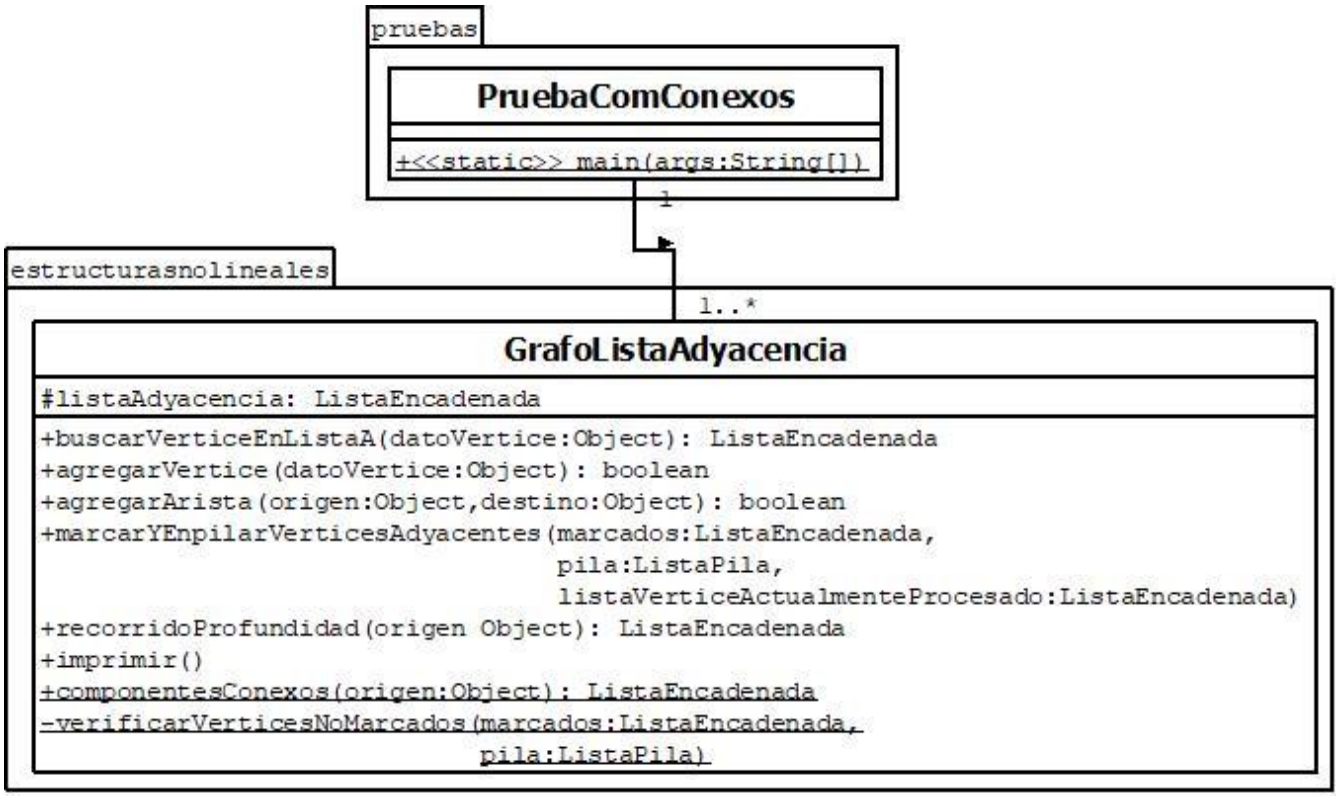
a. En este grafo se tomo como origen el vertice **A.**

```
A -> null
B -> H -> null
C -> R -> null
D -> B -> C -> null
H -> A -> T -> D -> null
T -> null
R -> H -> null
Componentes conexos:
A -> null
B -> H -> D -> C -> R -> T -> null
Process finished with exit code 0
```

En este grafo también obtiene dos componentes conexos los cuales cubren todos los nodos del grafo.

Código agregado:

Solo los métodos llamados **componentesConexos** y **verificarVerticesNoMarcados** son nuevos. Y solo la clase **PruebaComConexos** es nueva.



Pre-evaluación:

Pre-Evaluación para prácticas de Laboratorio de Estructuras de Datos	PRE-EVALUACIÓN DEL ALUMNO
CUMPLE CON LA FUNCIONALIDAD SOLICITADA.	Sí
DISPONE DE CÓDIGO AUTO-DOCUMENTADO.	Sí
DISPONE DE CÓDIGO DOCUMENTADO A NIVEL DE CLASE Y MÉTODO.	Sí
DISPONE DE INDENTACIÓN CORRECTA.	Sí
CUMPLE LA POO.	Sí
DISPONE DE UNA FORMA FÁCIL DE UTILIZAR EL PROGRAMA PARA EL USUARIO.	Sí
DISPONE DE UN REPORTE CON FORMATO IDC.	Sí
LA INFORMACIÓN DEL REPORTE ESTÁ LIBRE DE ERRORES DE ORTOGRAFÍA.	Sí
SE ENTREGÓ EN TIEMPO Y FORMA LA PRÁCTICA.	Sí
INCLUYE LA DOCUMENTACIÓN GENERADA CON JAVADOC.	Sí
INCLUYE EL CÓDIGO AGREGADO EN FORMATO UML.	Sí
INCLUYE LAS CAPTURAS DE PANTALLA DEL PROGRAMA FUNCIONANDO.	Sí
LA PRÁCTICA ESTÁ TOTALMENTE REALIZADA (ESPECIFIQUE EL PORCENTAJE COMPLETADO).	100%
Observaciones:	

Conclusión:

Los grafos manipulados con listas de adyacencia nos permiten almacenar los nodos y sus adyacentes en una lista de listas, en las cuales las listas contenidas en la lista principal, el primer elemento de estas es el vértice y los demás son los vértices adyacentes, ósea sus aristas. Con esto nos permite tener una gran variedad de nodos sin importar el número de aristas que se tengan, ya que si se tienen muchos nodos y pocas aristas las matrices de adyacencia serian un problema debido a que almacenaría demasiados ceros, en cambio con las listas solo almacena las adyacencias que le corresponden.