DECANATURA DE DIVISIÓN DE EDUCACIÓN ABIERTA Y A DISTANCIA
FACULTAD DE CIENCIAS Y TECNOLOGÍAS
TEORIA DE GRAFOS
Código SAC (16887)
TEÓRICO PRÁCTICA
EVALUACIÓN PRACTICA 2020-2

OBJETIVOS DE LA EVALUACIÓN

- Identificar las características de sistemas que pueden ser representados y estudiados desde la teoría de grafos
- Comprender los principios sobre los cuales la teoría de grafos representa, evalúa y analiza sistemas discretos
- Aplicar algoritmos de optimización de caminos propios de la teoría de grafos.

VALORACIÓN DEL ESPACIO ACADÉMICO

Este espacio académico se valorará de acuerdo con lo estipulado en el Capítulo VII del Reglamento Estudiantil Particular de la VUAD. Así:

Evaluación Teórica:
-La Evaluación en línea (o presencial) tiene un valor del 50% de la calificación total del espacio académico.
-La Evaluación Distancia tiene un valor del 40% de la calificación total del espacio académico.
-Foro académico tiene un valor del 5% de la calificación total del espacio académico.
-Chat académico tiene un valor del 5% de la calificación total del espacio académico
Evaluación Teórico práctico:
Evaluación presencial tiene un valor del 50%;
Evaluación a distancia tiene un valor del 25%;
Evaluación práctica valor del 25%.

Evaluación Práctico:
Evaluación práctica tendrá un valor del 100%.

Nota: Tenga en cuenta que la adecuada citación, bajo las Normas APA 6ª edición, hace parte de los criterios de evaluación. Por ende, todo punto o trabajo que contenga fragmentos o en su totalidad no corresponda a su autoría o no tenga la citación adecuada tendrá como consecuencia la anulación del punto o evaluación.

ACTIVIDADES A DESARROLLAR

Estimados estudiantes:
Antes de comenzar el desarrollo de la Evaluación Distancia se le recomienda ir al Aula Virtual y descargar la "Rúbrica de Evaluación", ya que allí se encuentran claros los criterios a partir de los cuales se realizará la valoración y ponderación de las respuestas dadas a cada una de las preguntas. **¡Éxitos!**

Actividades a Desarrollar

Tomada como Referencia:
https://runestone.academy/runestone/static/pythoned/Graphs/toctree.html

1. **Primer Momento de Evaluación. El tipo abstracto de datos grafo**

El tipo abstracto de datos (TAD) grafo está definido como sigue:

- Grafo() crea un grafo nuevo y vacío.
- agregarVertice(vert) agrega una instancia de Vertice al grafo.
- agregarArista(deVertice, aVertice) agrega al grafo una nueva arista dirigida que conecta dos vértices.
- agregarArista(deVertice, aVertice, ponderacion) agrega al grafo una nueva arista ponderada y dirigida que conecta dos vértices.
- obtenerVertice(claveVert) encuentra el vértice en el grafo con nombre claveVert.
- obtenerVertices() devuelve la lista de todos los vértices en el grafo.
- in devuelve True para una instrucción de la forma vertice in grafo, si el vértice dado está en el grafo, False de lo contrario.

Actividad a Desarrollar. Implementar el TAD grafo en Lenguaje de Programación Python o Java.

2. **Segundo Momento de Evaluación. Una matriz de adyacencia**

Una de las maneras más fáciles de implementar un grafo es usar una matriz bidimensional. En esta implementación de matriz, cada una de las filas y columnas representa un vértice en el grafo. El valor que se almacena en la celda en la intersección de la fila vv y la columna ww indica si hay una arista desde el vértice vv al vértice ww. Cuando dos vértices están conectados por una arista, decimos que son **adyacentes**. La Figura 3 ilustra la matriz de adyacencia para el grafo de la Figura 2. Un valor en una celda representa la ponderación de la arista que une el vértice vv con el vértice ww.

	V0	V1	V2	V3	V4	V5
V0		5				2
V1			4			
V2				9		
V3					7	3
V4	1					
V5			1		8	

Figura 1: Una representación de un grafo mediante una matriz de adyacencia

La ventaja de la matriz de adyacencia es que es simple, y que para grafos pequeños es fácil ver qué nodos están conectados a otros nodos. Sin embargo, note que la mayoría de las celdas de la matriz están vacías. Dado que la mayoría de las celdas están vacías decimos que esta matriz es “rala”. Una matriz no es una forma muy eficiente de almacenar datos ralos. De hecho, en Lenguaje de Programación Python o Java usted debe incluso esforzarse por crear una estructura de matriz como la de la Figura 1.

La matriz de adyacencia es una buena implementación para un grafo cuando el número de aristas es grande. Pero ¿qué entendemos por grande? ¿Cuántas aristas se necesitarían para llenar la matriz? Puesto que hay una fila y una columna para cada vértice en el grafo, el número de aristas requeridas para llenar la matriz es |V|2|V|2. Una matriz está llena cuando cada vértice está conectado a todos los otros vértices. Hay pocos problemas reales que se aproximan a este tipo de conectividad. Los problemas que veremos en este capítulo se refieren a grafos que están conectados de forma rala.
Actividad a Desarrollar. Implementar la matriz de adyacencia en Lenguaje de Programación Python o Java

Tercer Momento de Evaluación. Una Lista de adyacencia.

Una forma más eficiente, respecto al uso del espacio, de implementar un grafo conectado de forma rala es usar una lista de adyacencia. En una implementación de lista de adyacencia mantenemos una lista maestra de todos los vértices en el objeto Grafo y además cada objeto Vértice en el grafo mantiene una lista de los otros vértices a los que está conectado. En nuestra implementación de la clase Vertice usaremos un diccionario en lugar de una lista donde las claves del diccionario son los vértices, y los valores son las ponderaciones. La Figura 4 ilustra la representación mediante una lista de adyacencia para el grafo de la Figura 2.

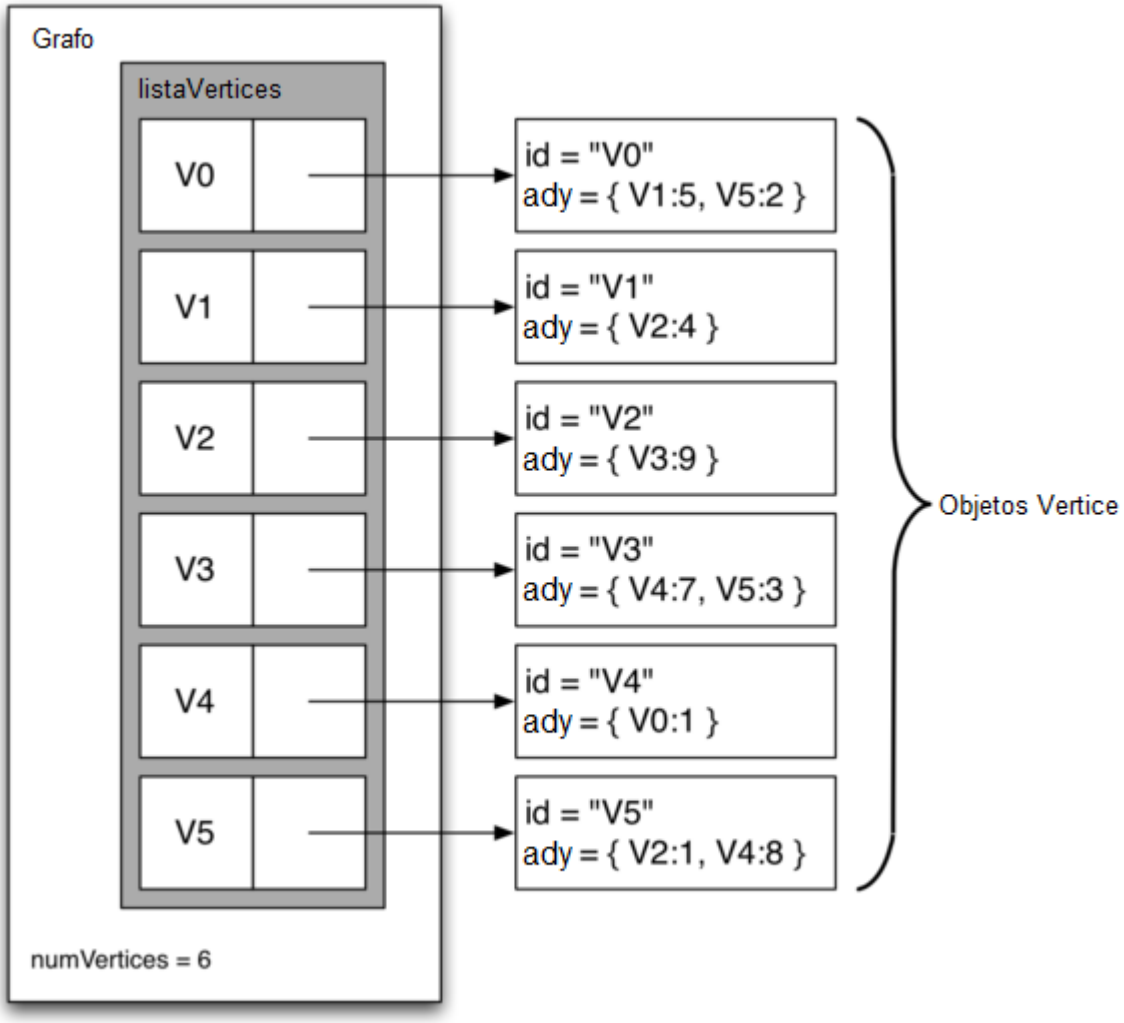

Figura 2: Representación mediante una lista de adyacencia de un grafo

La ventaja de la implementación mediante una lista de adyacencia es que nos permite representar de forma compacta un grafo ralo. La lista de adyacencia también nos permite encontrar fácilmente todos los enlaces que están directamente conectados a un vértice particular.

Actividad a Desarrollar. Implementar la matriz de adyacencia en Lenguaje de Programación Python o Java