| | |
|---|---|
| **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | Laboratorio de docencia |

# Laboratorios de computación salas A y B

*Profesor:* Jesús Cruz Navarro

*Asignatura:* Estructura de datos y algoritmos II

*Grupo:* 02

*No de Práctica(s):* 06

*Integrante(s):* Hernandez Sarabia Jesus Ivan

*No. de Equipo de cómputo empleado:* No empleado

*No. de Lista o Brigada:*

*Semestre:* 2022-1

*Fecha de entrega:* 27 de octubre de 2021

*Observaciones:*

CALIFICACIÓN: _____

## Objetivo:

El estudiante conocerá las formas de representar un grafo e identificará las características necesarias para entender el algoritmo de búsqueda de expansión

## Actividades:

- Diseñar e implementar las clases Vértice y Grafo, con los métodos AgregarVertice y AgregarArista, como los vistos en clase (usando como parámetros los nombres de los vértices, en lugar de pasar un objeto de tipo Vértice, como la práctica de la coordinación).

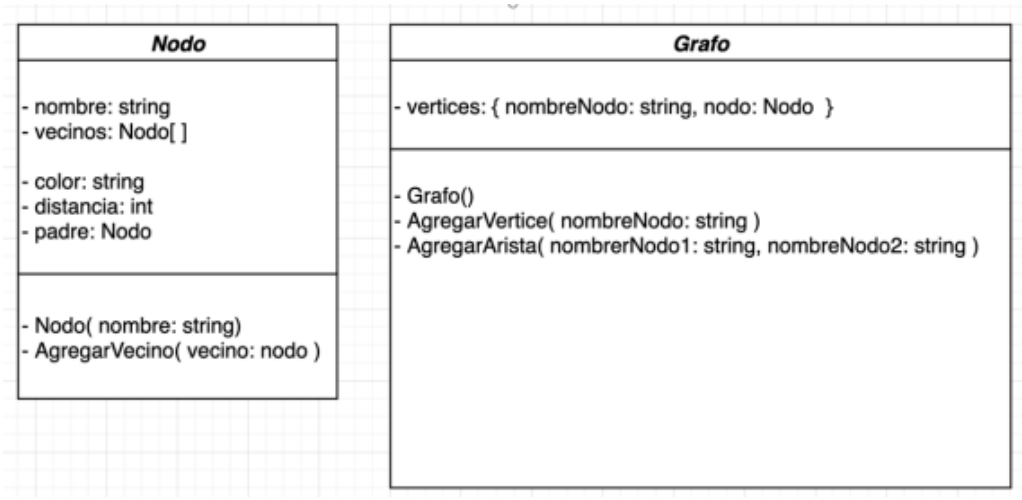
Se debe validar que:
- En la clase Vértice, al agregar vecino, no exista ya un vecino con ese nombre de vértice.
- En la clase Grafo, al agregar vértice, no exista ya un vértice con ese nombre.
- En la clase Grafo, al agregar arista, que existan los vértices entre la arista.

En caso de error, se debe imprimir un mensaje de error en consola y no hacerlo.

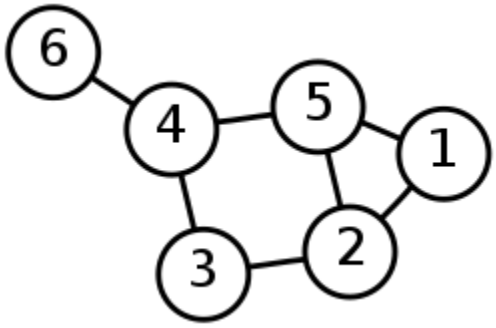
Los vértices de la clase nodo se deben guardar en un diccionario y acceder a ellos utilizando su llave, no iterando sobre estos.

Además, ambas clases deben sobreescribir los métodos __str__ y __repr__ para poder imprimir el grafo desde la función print(). Para el caso de la clase grafo al imprimir se debe imprimir algo similar a una lista de adyacencia.

Diagrama de Clases:

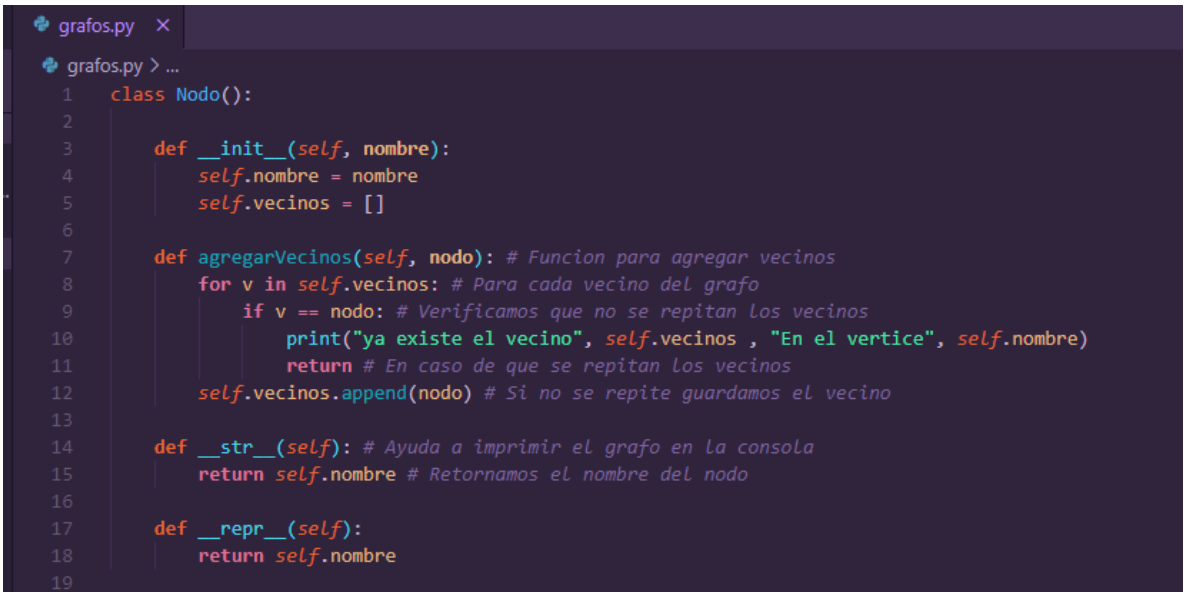



Ejemplo de grafo empleado para este ejercicio.



Código Fuente:

Clase Nodo ():
Con sus respectivas funciones creadas.



```python
class Nodo():

    def __init__(self, nombre):
        self.nombre = nombre
        self.vecinos = []

    def agregarVecinos(self, nodo): # Funcion para agregar vecinos
        for v in self.vecinos: # Para cada vecino del grafo
            if v == nodo: # Verificamos que no se repitan los vecinos
                print("ya existe el vecino", self.vecinos , "En el vertice", self.nombre)
                return # En caso de que se repitan los vecinos
        self.vecinos.append(nodo) # Si no se repite guardamos el vecino

    def __str__(self): # Ayuda a imprimir el grafo en la consola
        return self.nombre # Retornamos el nombre del nodo

    def __repr__(self):
        return self.nombre
```

Clase grafo ():
Con sus respectivas funciones creadas.

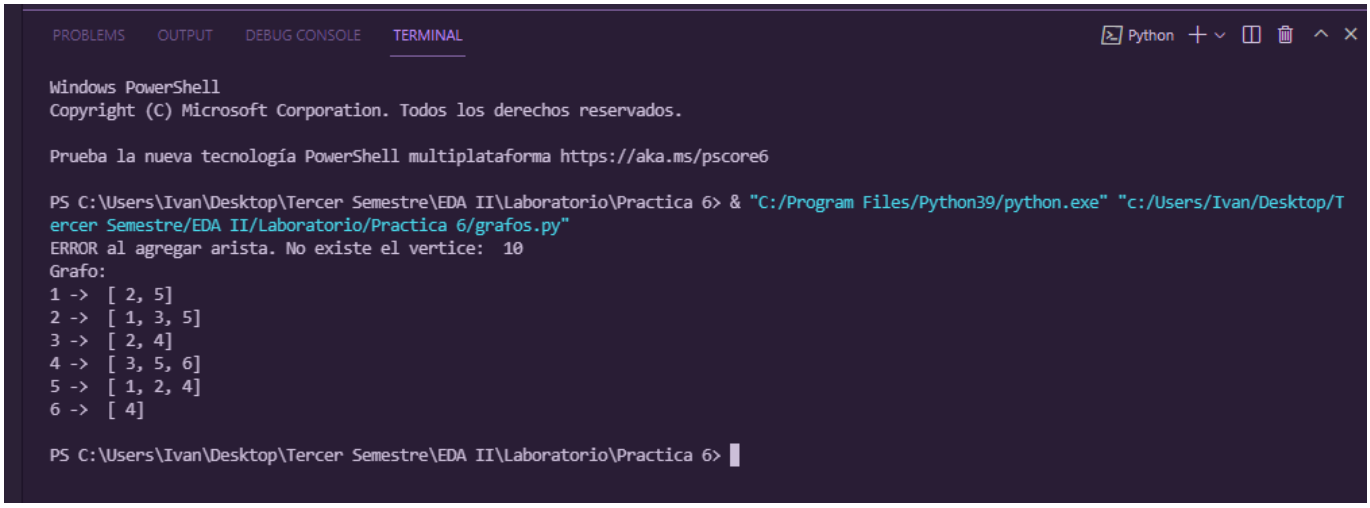
```python
class grafo(): # Clase grafo

    def __init__(self):
        self.vertice= {}

    def agregarVertice(self, nombreNodo):
        for v in self.vertice:
            if v == nombreNodo: # Para cuando repetimos los nodos
                print('Ya existe el nodo con el nombre: ', nombreNodo)
        nodoNuevo = Nodo(nombreNodo) # Se agraga el nodo si este no esta repetido
        self.vertice[nombreNodo] = nodoNuevo

    def agregarArista(self, nombreNodo1, nombreNodo2):
        if nombreNodo1 in self.vertice:
            nodo1 = self.vertice[nombreNodo1]
        else:
            print('ERROR al agregar arista. No existe el vertice: ', nombreNodo1)
            return
        if nombreNodo2 in self.vertice:
            nodo2 = self.vertice[nombreNodo2]
        else:
            print('ERROR al agregar arista. No existe el vertice: ', nombreNodo2)
            return

        nodo1 = self.vertice[nombreNodo1]
        nodo2 = self.vertice[nombreNodo2]

        nodo1.agregarVecinos(nodo2)
        nodo2.agregarVecinos(nodo1)
```

```python
        # Funcion para poder imprimir los grafos en consola
    def __str__(self):
        print("Grafo: ")
        s = ""
        for v in self.vertice:   # Para cada valor de cada vertice
            s += self.vertice[v].nombre + " -> " # Guardamos los nombres de los vertices
            s += " [ "

            for w in self.vertice[v].vecinos:
                s += w.nombre + ", " # Guardamos los vecinos
            arr = s[len(s)-2]
            if arr == ",": # Quitamos los espacios
                s = s[:-2]
            s += "]"
            s += "\n"
        return s

    def __repr__(self):
        s = " " # Servira para guardar los nombres de los vertices
        for v in self.vertice:
            s += self.vertices[v].nombre + " - "
        s = s[:-2]
        return s
```

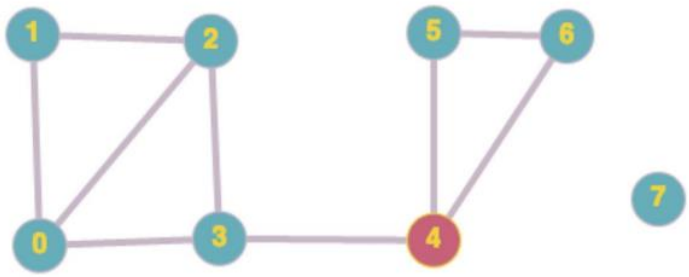
Main del programa para agregar los vértices y las aristas del grafo.

```python
g = grafo() # Creamos el grafo

# Agregamos los verices
g.agregarVertice('1')
g.agregarVertice('2')
g.agregarVertice('3')
g.agregarVertice('4')
g.agregarVertice('5')
g.agregarVertice('6')

g.agregarArista('10', '0') # Ejemplo de vertice que no se encuentra

g.agregarArista('1', '2') # Agregamos las aristas
g.agregarArista('1', '5')

#g.agregarArista('2', '1')
g.agregarArista('2', '3')
g.agregarArista('2', '5')

#g.agregarArista('3', '2')
g.agregarArista('3', '4')

#g.agregarArista('4', '3')
g.agregarArista('4', '5')
g.agregarArista('4', '6')

#g.agregarArista('5', '1')
#g.agregarArista('5', '2')
#g.agregarArista('5', '4')

#g.agregarArista('6', '4')

print(g)
```

## Ejecución del programa:



b) Desarrollar un programa que utilice las clases generadas y genere un grafo como el visto en clase (Ejemplo Básico) e imprima su lista de adyacencia. Además, pruebe los casos de error y muestre los mensajes en pantalla. Genere el siguiente grafo:

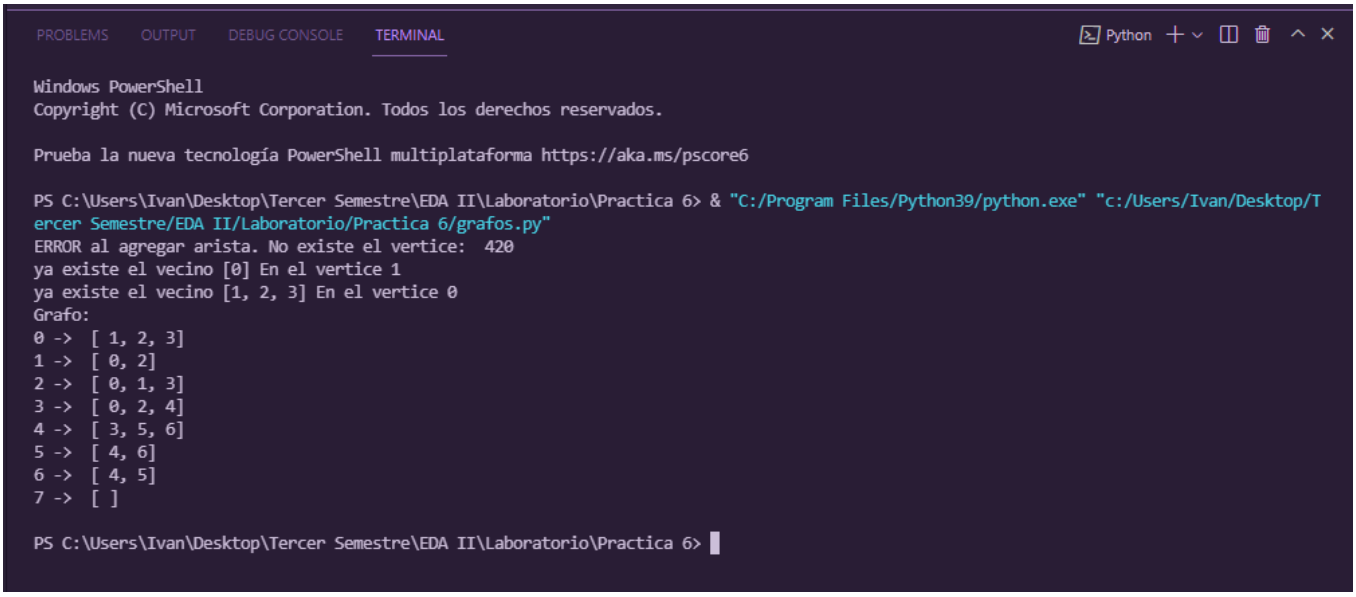



Para este programa tendremos el mismo código implementado en inciso pasado simplemente agregaremos sus respectivos vértices y aristas para poder mostrar lo solicitado:

grafos.py

```
73
74  g = grafo() # Creamos el grafo
75
76  g.agregarVertice('0') # Agregamos los verices
77  g.agregarVertice('1')
78  g.agregarVertice('2')
79  g.agregarVertice('3')
80  g.agregarVertice('4')
81  g.agregarVertice('5')
82  g.agregarVertice('6')
83  g.agregarVertice('7')
84
85  g.agregarArista('420', '0') # Ejemplo de vertice que no se encuentra
86  g.agregarArista('0', '1') # Agregamos las aristas
87  g.agregarArista('0', '2')
88  g.agregarArista('0', '3')
89
90  g.agregarArista('1', '0')
91  #g.agregarArista('1', '2')
92
93  #g.agregarArista('2', '0')
94  g.agregarArista('2', '1')
95  g.agregarArista('2', '3')
96
97  #g.agregarArista('3', '0')
98  #g.agregarArista('3', '2')
99  g.agregarArista('3', '4')
100
101 #g.agregarArista('4', '0')
102 g.agregarArista('4', '5')
103 g.agregarArista('4', '6')
104
105 #g.agregarArista('5', '4')
106 g.agregarArista('5', '6')
107
108 #g.agregarArista('6', '4')
109 #g.agregarArista('6', '5')
110
111 print(g)
```

## Ejecución del programa:



## Conclusión:

Hernández Sarabia Jesús Ivan: Una vez realizada la practica de laboratorio puedo concluir que uno de los usos de los grafos es, encontrar elemento que se relacionan o conectan entre sí, como puede ser los ejemplos realizados en la práctica, donde mostrado un diagrama en donde se observan elementos conectados unos a otros, podemos obtener el grafo a partir de dicho diagrama gracias al programa implementado en la practica. También puedo concluir que muchos momentos de nuestra vida cotidiana hacemos usos de estos grafos, como pueden las redes sociales, las líneas del metro, entre muchos otros ejemplos en los que hacemos usos de dichos grafos.