Estructura Pila.

¿Qué es?

Esta estructura funciona prácticamente como en la vida real es un conjunto de objetos que están uno sobre otro de manera vertical, al agregar un nuevo elemento a la pila este se va a colocar en la parte superior, ya que en una pila el último elemento en ingresar será el tope de la misma. Esto es debido a que las pilas son una estructura de tipo LIFO.

¿Qué entendimos?

Pues es una estructura en la cual si nosotros por ejemplo nosotros metemos los valores 1, 2, 3, 100 y 84563 ya tenemos nuestra pila y si nosotros queremos obtener 2 valores al tratar de obtenerlos este nos va a dar el valor de 100 y 84563 por que fueron los 2 últimos valores en ingresarse a la pila y asi es como trabaja esta clase los valores se van apilando uno arriba de otro.

¿Cómo lo programamos?

Bueno en este algoritmo primero con __init__() establecemos la lista como vacía, después con obtener() la definimos y agregamos un self.pila.pop() para nosotros poder ingresar los elementos de nuestra pila, luego meter() y agregamos un self.pila.append(e) para poder ingresar los valores de nuestra pila y se acomoden como debe ser el primer elemento abajo y último elemento ingresado en la parte superior y por ultimo con longitud() este nos ayuda para saber la longitud que va a tener nuestra pila de cosas.

```
class Pila:
    def __init__(self):
        self.pila= []
    def obtener(self):
        return self.pila.pop()
    def meter(self,e):
        self.pila.append(e)
        return len(self.pila)
    @property
    def longitud(self):
        return len(self.pila)
```

Estructura Fila.

¿Qué es?

Esta es una estructura en la cual al ingresar tu un valor o un elemento este será tanto el primero en entrar como el primero en salir. Esto es debido a que la fila es una estructura de tipo FIFO(First in, First out).

¿Qué entendimos?

En esta estructura nosotros podemos trabajar por ejemplo ingresando los elementos 1, 2, 3, 183, 1837506 y debido a como se trabaja con esta clase al tratar de obtener por ejemplo 3 elementos de este, se van a tomar 1 , 2 y 3 que fueron los primero en ingresarse.

¿Cómo lo programamos?

En este algoritmo primero con __init__() establecemos la lista como vacía, después con obtener() la definimos y agregamos un self.pila.pop() para poder ingresar los elementos de nuestra fila, luego meter() agregamos un self.fila.insert(0,e) para que los elemento se acomoden de tal manera que el primero en ingresarse sea el primero en obtenerse y por ultimo con longitud() pues recibimos la cantidad de elementos que contiene nuestra fila.

```
class Fila:
    def __init__(self):
        self.fila= []
    def obtener(self):
        return self.fila.pop()
    def meter(self,e):
        self.fila.insert(0,e)
        return len(self.fila)
    @property
    def longitud(self):
        return len(self.fila)
```

Grafo.

¿Qué es?

Un grafo es un conjunto de objetos que pueden ser llamados vértices o nodos y estos son unidos por enlaces llamados aristas y estos permiten representar relaciones binarias entre elementos de un conjunto

¿Qué entendimos?

Pues que un grafo es un algoritmo que te ayuda por ejemplo como cuando se quiere estudiar una red de computadoras con un grafo se puede representar y estudiar esta red en este caso los vértices representarían terminales y los aristas conexiones.

¿Cómo lo programamos?

Bueno primero con __init__(self) aquí se empieza a definir un conjunto y un mapeo de pesos de aristas y otro para los vecinos, luego en agrega(self,v) es donde se define la vecindad de "v" en este caso que inicialmente no tiene nada, después en conecta(self,v,u,peso=1) es donde se relacionan los elementos del grafo en ambos sentidos y ya.

```
class Grafo:

    def __init__(self):
        self.V = set()
        self.E = dict()
        self.vecinos = dict()

    def agrega(self,v):
        self.V.add(v)
        if not v in self.vecinos:
            self.vecinos[v] = set()
    def conecta(self, v, u, peso = 1):
        self.agrega(v)
        self.agrega(u)
        self.E[(v, u)] = self.E[(u, v)] = peso
        self.vecinos[v].add(u)
        self.vecinos[u].add(v)

    def complemento(self):
        comp= Grafo()
        for v in self.V:
            for w in self.V:
                if v != w and (v, w) not in self.E:
                    comp.conecta(v, w, 1)
            return comp
```

BFS.

¿Qué es?

Bueno dado un grafo el BFS va recorriendo los nodos padres primero y revisa si tiene hijos después si tiene hijos revisa uno y regresa al padre y revisa al segundo hijo después revisa si los hijos tienen hijos y los revisa de la misma manera y si ya no tienen hijos pues ahí se queda.

¿Qué entendimos?

Bueno dado un grafo el BFS lo que hace es ir revisando cada nodo usando la clase fila, la manera en la que los revisa es primero los nodos padres y después a sus hijos luego regresa al nodo padre y revisa si no tiene mas hijos si no tiene hijos pues ahí termina.

¿Cómo lo programamos?

El BFS se programó usando los algoritmos antes vistos se usó la clase Fila para la manera en la que se iban a estar acomodando y obteniendo los elementos y el algoritmo de grafo para poder construir nuestro grafo y por ultimo definimos en BFS como se iban a usar los 2 algoritmos anteriores para revisar los nodos en forma de Filas.

```
class Fila:
    def __init__(self):
        self.fila= []
    def obtener(self):
        return self.fila.pop()
    def meter(self,e):
        self.fila.insert(0,e)
        return len(self.fila)
    @property
    def longitud(self):
        return len(self.fila)

class Grafo:

    def __init__(self):
        self.V = set()
        self.E = dict()
        self.vecinos = dict()

    def agrega(self,v):
        self.V.add(v)
        if not v in self.vecinos:
            self.vecinos[v] = set()
    def conecta(self, v, u, peso = 1):
        self.agrega(v)
        self.agrega(u)
        self.E[(v, u)] = self.E[(u, v)] = peso
        self.vecinos[v].add(u)
        self.vecinos[u].add(v)

    def complemento(self):
        comp= Grafo()
        for v in self.V:
            for w in self.V:
                if v != w and (v, w) not in self.E:
                    comp.conecta(v, w, 1)
```

```
def BFS(grafo,ni):
        visitados=[ni]
        f=Fila()
        f.meter(ni)
        while (f.longitud>0):
            na=f.obtener()
            visitados.append(na)
            ln= grafo.vecinos[na]
            for nodo in ln:
                if nodo not in visitados:
                    f.meter(nodo)

        return visitados
```

DFS.

¿Qué es?

Bueno dado un grafo el DFS lo que hace es que va recorriendo cada nodo , este los va revisando desde el nodo padre hasta su primer hijo y si ese hijo tiene hijo pues también lo revisa luego revisando todo como si estuviera apilado.

¿Qué entendimos?

Bueno dado un grafo el DFS lo que hace es ir revisando cada nodo usando la clase pila, la manera en la que los revisa es primero los nodos padres y después a sus hijos luego a los hijos de sus hijos luego regresa al nodo padre y revisa al segundo hijo y si tiene hijos los revisa, si el nodo padre ya no tiene más hijos ahí termina.

¿Cómo lo programamos?

El DFS se programó usando los algoritmos antes vistos se usó la clase Pila para la manera en la que se iban a estar acomodando y obteniendo los elementos y el algoritmo de grafo para poder construir nuestro grafo y por ultimo definimos en DFS como se iban a usar los 2 algoritmos anteriores para revisar los nodos de arriba que es el nodo padre hasta los hijos de ese nodo y los hijos de ese nodo revisando estos en forma de una pila.

```
class Pila:
    def __init__ (self):
        self.pila = []
    def obtener (self):
        return self.pila.pop ()
    def meter (self, e):
        self.pila.append (e)
        return len (self.pila)
    @property
    def longitud (self):
        return len (self.pila)

class Grafo:

    def __init__(self):
        self.V = set()
        self.E = dict()
        self.vecinos = dict()

    def agrega(self,v):
        self.V.add(v)
        if not v in self.vecinos:
            self.vecinos[v] = set()
    def conecta(self, v, u, peso = 1):
        self.agrega(v)
        self.agrega(u)
        self.E[(v, u)] = self.E[(u, v)] = peso
        self.vecinos[v].add(u)
        self.vecinos[u].add(v)

    def complemento(self):
        comp= Grafo()
        for v in self.V:
            for w in self.V:
                if v != w and (v, w) not in self.E:
                    comp.conecta(v, w, 1)
```

```
def DFS(grafo,ni):
        visitados=[ni]
        p=Pila()
        p.meter(ni)
        while (p.longitud>0):
            na=p.obtener()
            visitados.append(na)
            ln= grafo.vecinos[na]
            for nodo in ln:
                if nodo not in visitados:
                    p.meter(nodo)

        return visitados
```

Ruiz Fraser Francisco Guillermo 1837506