Universidad Autónoma de Nuevo León Facultad de Ciencias Físico Matemáticas

Artemio Guajardo Aparicio Grupo: 001 Matrícula: 1590417

Reporte de las estructuras: Pila, Fila, Grafo, BFS y DFS

Resumen: Este reporte habla sobre las estructuras de datos ya mencionadas en el título, sobre que son y sus implementaciones a Python.

Pila

Las pilas son un tipo de lista conocidas como listas LIFO (Last In, First Out: El último en entrar es el primero en salir). Aquí los elementos se acomodan de manera que solo el elemento que está más "encima" puede ser leído y solo podemos añadir elementos encima de ésta. Aquí solo se puede colocar un elemento al final o quitar un elemento del final. La pila se aplica en el área de informática debido a que es simple y da respuesta a numerosos procesos.

La implementación de una Pila a Python fue dada en su mayoría por el maestro y es la siguiente:

```
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)
```

Fila

Al igual que pila, es un tipo de lista, pero esta es conocida como lista FIFO (First In First Out: El primero en entrar es el primero en salir). La diferencia es que aquí se añaden elementos nuevos al final o se quitan elementos viejos del principio.

La implementación de una Pila a Python fue dada en su mayoría por el maestro y es la siguiente:

```
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

Los grafos, a muy grandes rasgos se puede decir que son conjuntos de objetos (nosotros les llamamos nodos), unidos por enlaces que nos ayudan a representar relaciones entre dichos objetos. Nosotros los representamos como conjuntos de puntos unidos por líneas.

La implementación de un grafo a Python fue realizada después de analizar una fuente en internet y quedo como:

```
class Grafos:
    def __init__(self):
        self.V = set() #Un conjunto
        self.E = dict() #Un mapeo de pesos de aristas
        self.vecinos = dict() #Un mapeo

    def agrega(self,v):
        self.V.add(v)
        if not v in self.vecinos: #vecindad de v
            self.vecinos[v] = set() #Inicialmente no tiene nada

    def conecta(self,v,u,peso=1):
        self.agrega(v)
        self.agrega(u)
        self.E[(v,u)]=self.E[(u,v)]=peso #En ambos sentidos
        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
```

Instrucciones para generar un grafo

Para generar un grafo, cuyos nodos son ‘a’, ‘b’, ‘c’, y ‘d’ debemos hacer lo siguiente:

- Primero inicializamos una variable asignándole la función *Grafo()* para trabajar con más facilidad.

- Después haciendo uso de la función “conecta” vamos relacionando los nodos que queremos que tengan aristas entre ellos
- Al final imprimimos los datos relacionados que deseemos imprimir

Un ejemplo para el código del grafo antes mencionado sería:

```
G = Grafo()
G.conecta('a', 'b')
G.conecta('a', 'c')
G.conecta('b', 'c')
G.conecta('c', 'd')
print(G.vecinos['a'])
print(G.V)
print(G.E)
```

Se pueden conectar los nodos del grafo de la forma que se desee.

Todo lo anterior se debía ver primero para poder comprender mejor los conceptos de búsqueda por amplitud (BFS) y búsqueda por profundidad (DFS).

Búsqueda por amplitud (BFS)

La BFS es un algoritmo basado en la estructura First In First Out (FIFO) ya antes mencionada, es decir, hace uso de una Fila para recorrer los nodos de un grafo en orden creciente, es decir, al localizar un nodo éste es asignado como la raíz, de esta manera visita todos sus nodos alrededor y uno de estos nodos de alrededor se asignan como la nueva raíz según la posición que tenga en la fila. Básicamente la BFS sirve para explorar un grafo.

Su implementación en Python es la siguiente:

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

Agregando previamente las funciones *Fila()* y *Grafo().*

Búsqueda por profundidad (DFS)

Este algoritmo es muy parecido a la BFS, con la diferencia de que éste está basado en la estructura Last In First Out, también antes mencionada, es decir, este algoritmo hace uso de Pila para recorrer los nodos de un grafo localizando los recorridos posibles y en el caso de no poder continuar, vuelve al punto donde existen nuevos recorridos posibles con el fin de visitar todos los nodos.

Su implementación en Python es la siguiente:

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

Agregando previamente las funciones *Pila()* y *Grafo().*