Universidad Autónoma de Nuevo León

FCFM

Reporte de estructuras de python

Ana Cecilia García Esquivel Matrícula: 1749760

Resumen

En este reporte se mencionarán las estructuras: fila, pila, grafo, BFS, DFS; qué son y cómo funcionan, además de un código de cada una de dichas estructuras.

Fila

La estructura de datos *Fila* es una lista, donde podemos ir guardando elementos. Dichos elementos se van acomodando en forma de una fila, tal como su nombre lo dice, de modo que al ingresar los elementos en la lista cuando los queramos visualizar u obtener, la estructura lo arrojará en el mismo orden de como entraron, es decir, el primer elemento que ingreses es el primero que saldrá cuando lo queramos ver.

El siguiente algoritmo es un ejemplo de fila, hecho en clase.

```
class fila:
        def _init_(self):
                self.fila=[]
        def obtener (self):
                return self.fila.pop()
        def meter (self,e):
                self.fila.append(e)
                return len(self.pila)
        @property
        def longitud(self):
                return len(self.pila)
l = fila()
l.meter(1)
l.meter(2)
l.meter(3)
```

```
print(l.longitud)
print(l.obtener())
```

Pila

Esta estructura de datos también es una lista, pero a diferencia de las filas, en este tipo de lista los elemento que se van ingresando se van empilando, así como una “pila” de cajas, de modo que el último elemento puesto en la pila es el primero en salir cuando pedimos obtener los elementos.

El siguiente código es un ejemplo de esta estructura.

```
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)
p= Pila()
p.meter(1)
p.meter(2)
p.meter(3)
print(p.longitud)
print(p.obtener())
```

Grafo

Un grafo es una representación de datos conectados entre sí, donde cada dato puede ser un nodo, y hay aristas que los conectan. Esto es muy utilizado para ejercicios de optimización donde se requiere saber de rutas más cortas y efectivas, en las cuales se visiten a todos los nodos a partir de alguno central.

Aquí hay un ejemplo de un código de un grafo:

```
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)
```

BFS

El algoritmo de búsqueda por amplitud sirve para recorrer todos los nodos en un grafo, teniendo un nodo raíz o inicial y luego buscando a sus alrededores a los nodos más cercanos a él. Para esto usa la clase fila().

A continuación se mostrará un ejemplo de la función BFS

```
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
```

DFS

Este algoritmo es el conocido como Búsqueda por profundidad. Este también busca recorrer todos y cada uno de los nodos de un grafo, pero a diferencia del BFS este algoritmos usa la clase pila() para hacerlo, y además si llega a algún nodo de donde ya no hay más recorrido posible, se regresa a alguno que sí con el fin de poder recorrer todos los nodos.

En seguida se muestra un ejemplo del código de la función DFS

```
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
```