Fila, Pila, Grafo, BFS, DFS

Matemáticas computacionales

Ismael Medina Robledo

1744617

9 de octubre de 2017

Resumen

Una estructura de datos es una forma de organización computacional. Además se explicara acerca de Grafos, la búsqueda por profundidad DFS, búsqueda por anchura BFS, fila y pila.

- **Fila**

Es un código que almacena datos, los cuales pueden ser impresos para su visualización o ver la longitud de la fila.

```
>>> 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.fila)
        @property
        def longitud(self):
                return len(self.fila)


>>> l=fila()
>>> l.meter(2)
1
>>>
```

- **Pila**

Es un código idéntico a fila, con la diferencia que el último dato almacenado es el primero que se leerá.

```
>>> 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)
```

- **Grafo**

Es un conjunto de datos unidos por enlaces llamados 'aristas' que representan relaciones entre los elementos. Como si se tratara de un diagrama de árbol en donde las ramas son los datos.

```
        >>> class Grafo:
        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
```

- **BFS**

Es una búsqueda por anchura que es utilizada para los grafos. En donde el algoritmo recorre el grafo de izquierda a derecha y después de arriba a abajo, ordenando los datos en una fila o pila según se necesite.

- **DFS**

Es una búsqueda por profundidad con este algoritmo se puede recorrer todas las aristas de un grafo de manera ordenada.

Conclusión

Las estructuras de datos son parte importante para formar algoritmos eficientes y eficaces. Es por eso que es importante saber crear una buena estructura de datos.