**ESTRUCTURAS DE DATOS**

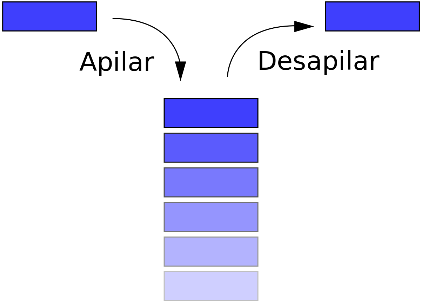
Brenda Espinosa Coronado – 1729180 – Ciencias Computacionales

Una estructura de datos es una colección de datos (normalmente de tipo simple) que se caracterizan por su organización y las operaciones que se definen en ellos. Por tanto, una estructura de datos vendrá caracterizada tanto por unas ciertas relaciones entre los datos que la construyen como las operaciones posibles en ellas.

Las estructuras de datos nos permiten resolver un problema de manera más sencilla gracias a que las reglas que las rigen nunca cambian, así que puedes asumir que ciertas cosas son siempre ciertas.

**Tipos de estructuras**

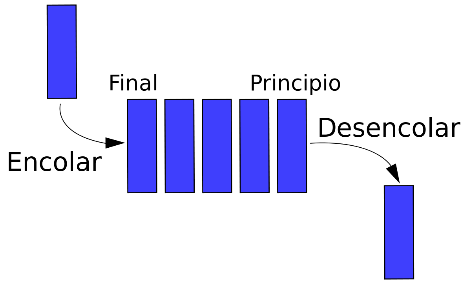
***Pila***  
Una pila es una lista especial donde todas las operaciones manipulan el primer elemento de la lista; se añade al frente y remueve del frente y se implementa como una lista enlazada manteniendo como puntero al primer elemento.



*Código:*

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)

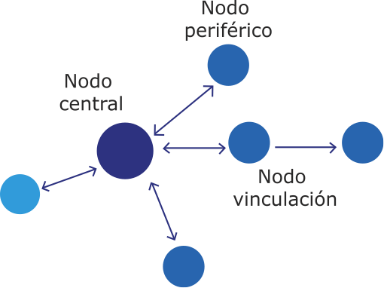
***Fila***  
Una fila es una estructura donde los elementos nuevos llegan al final, pero el procesamiento se hace desde el primer elemento. También las filas están fácilmente implementadas como listas enlazadas, manteniendo un puntero al comienzo de la fila y otro al final.



*Código:*

class Fila:  
 def \_\_init\_\_(self):  
 self.fila = []  
 def obtener(self):  
 return self.fila.pop(0)  
 def meter(self,e):  
 self.fila.append(e)  
 return len(self.fila)  
 @property  
 def longitud(self):  
 return len(self.fila)

***Grafo***  
Un grafo GG es un par de conjuntos G=(V,E)G=(V,E).  
V es un conjunto de n vértices u,v,w ∈ V mientras E es un conjunto de m aristas; |V|=n es el orden del grafo y |E|=m se llama el tamaño del grafo. Las aristas son típicamente pares de vértices, {u,v} ∈ E, en cual caso E⊆V× V  se escribe (u,v) como notación alternativa en vez de representar la arista como un subconjunto tal cual.



*Código:*

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

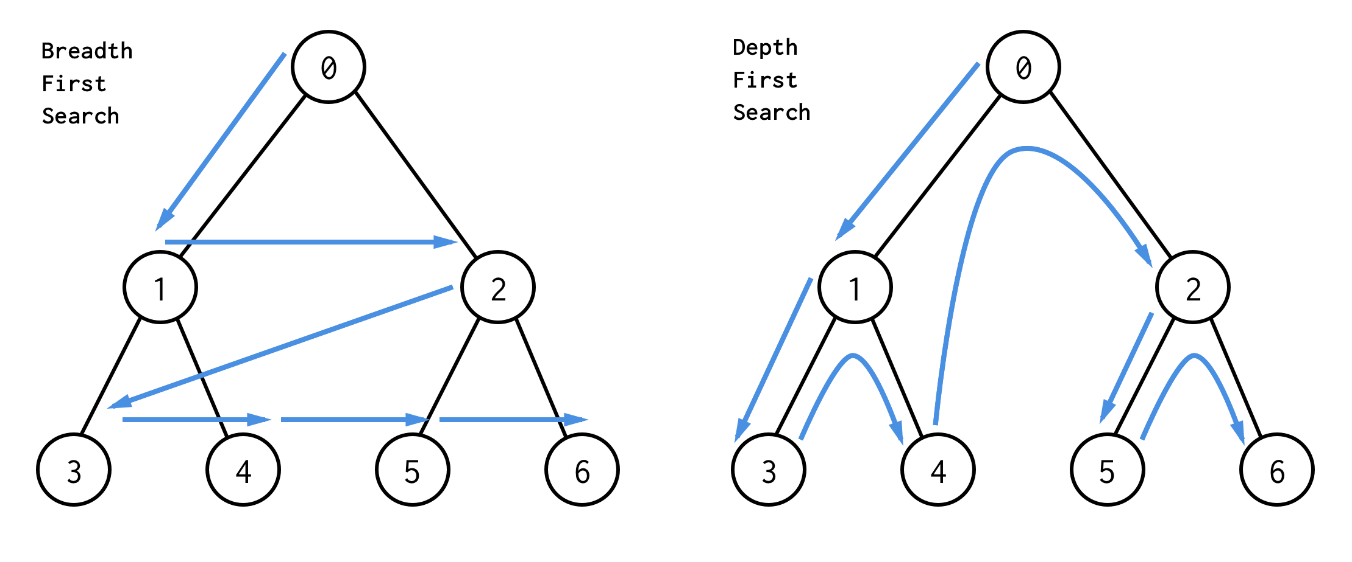
Mientras que pila y fila nos sirven para hacer búsquedas, el grafo nos ayuda a representar estructuras más complejas.

Un recorrido es un método sistemático para visitar cada vértice (por lo menos una vez). Una búsqueda es un recorrido cuyo propósito es encontrar un vértice del G (de nuestro grafo) que tenga una cierta propiedad. Algoritmos de recorrido y/o búsqueda comúnmente utilizan colas o pilas como estructuras auxiliares que guían el proceso.

Para esto, utilizamos las dos búsquedas siguientes:

***Búsqueda por profundidad DFS***

Un recorrido DFS puede progresar en varias maneras. El orden de visitas a los vértices depende de cómo se elige a cuál vecino se va. Las opciones son "visitar" antes o después de llamar la subrutina para los vecinos; si la visita se realiza antes de la llamada recursiva se llama el pre-orden. En el caso de visitar después, se llama post-orden.

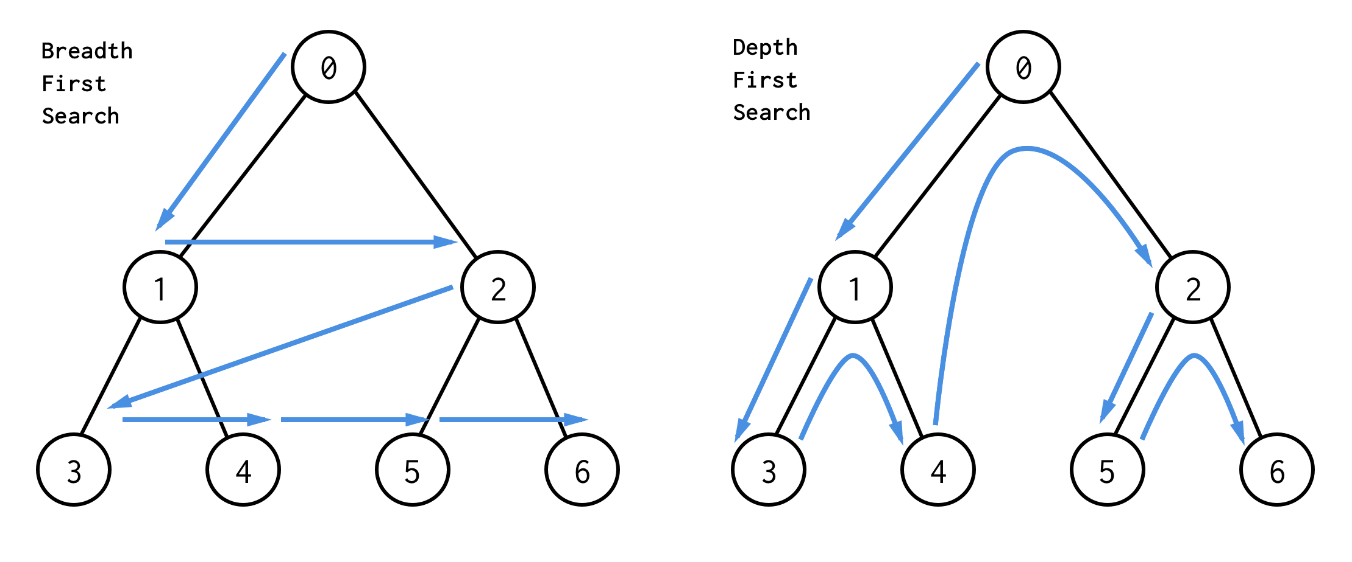


*Código:*

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

***Búsqueda por amplitud BFS***

El uso del BFS incluyen obviamente la búsqueda y el cálculo de distancias desde un vértice específico (hay que recordar cuál vértice puso a cuál en la cola, darle distancia cero al inicial y luego a los demás uno más el valor de quién lo agregó) y la detección de ciclos (por ver si se vuelve a un vértice ya visitado).



*Código:*

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

El beneficio de usar estructuras es que nos facilitan visualizar la manera más rápida de llegar al final de nuestro grafo, viendo los posibles caminos que podemos tomar para llegar a esto.