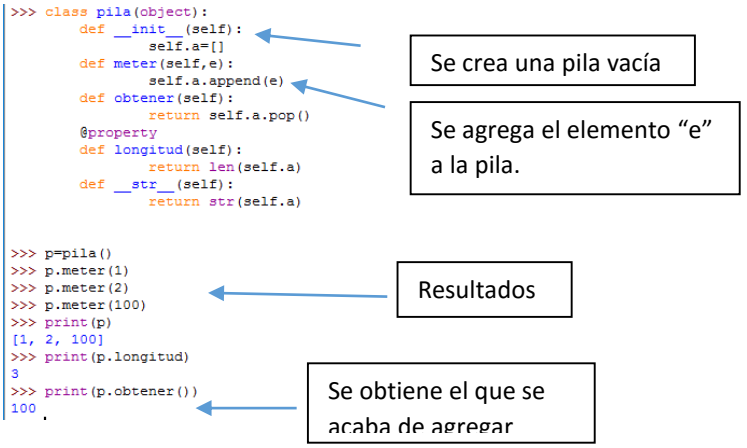
Karen Eunice Oviedo Garza

Introducción a la teoría de la complejidad y análisis de algoritmos.

Se dará una descripción y definición de funciones con mis propias palabras en lo visto en el capítulo 2.

Class pila

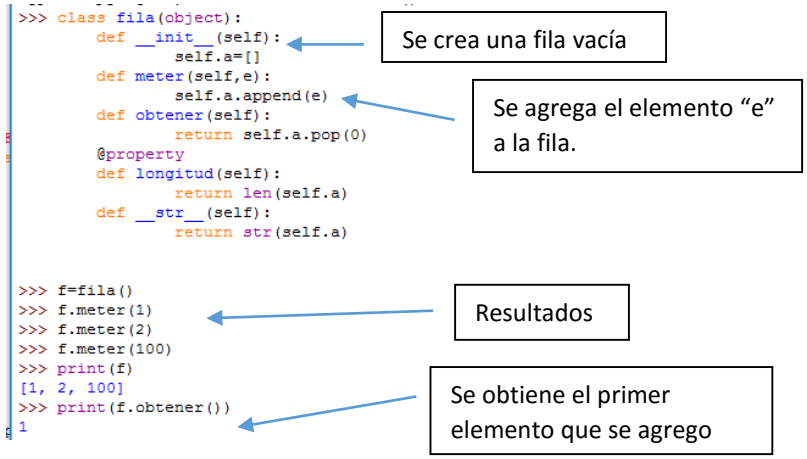
Un ejemplo para comprender sería una pila de platos, al lavarlos y secarlos se van apilando hacia arriba, por lo tanto el plato que primero se podría tomar seria el que está más arriba.




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


>>> p=pila()
>>> p.meter(1)
>>> p.meter(2)
>>> p.meter(100)
>>> print(p)
[1, 2, 100]
>>> print(p.longitud)
3
>>> print(p.obtener())
100
```

Class fila

Un ejemplo para comprender seria en una fila para pagar algun producto,a la persona que atienden primero es a la que esta primero en la fila dicha persona es la que lleva mas tiempo esperando por lo tanto se obtiene el elemento que esta mas lejano.




```
>>> class fila(object):
        def __init__(self):
                self.a=[]
        def meter(self,e):
                self.a.append(e)
        def obtener(self):
                return self.a.pop(0)
        @property
        def longitud(self):
                return len(self.a)
        def __str__(self):
                return str(self.a)


>>> f=fila()
>>> f.meter(1)
>>> f.meter(2)
>>> f.meter(100)
>>> print(f)
[1, 2, 100]
>>> print(f.obtener())
1
```

Graficas

Una grafica esta compuesta de vertices que tambien pueden ser llamados nodos,la union entre un vertice y otro se forma la arista,si existe el vertice “a” que esta conectado al vertice “b”,la arista es llamada “ab”.

Karen Eunice Oviedo Garza

```
>>> class grafica:
        def __init__(self):
                self.vertices=set()
                self.aristas= dict()
                self.vecinos=dict()
        def agrega(self,v):
                self.vertices.add(v)
                if not v in self.vecinos:
                        self.vecinos[v]=set()
        def conecta(self, u,v, peso=1):
                self.agrega(v)
                self.agrega(u)
                self.aristas[(v,u)]=self.aristas[(u,v)]=peso
                self.vecinos[v].add(u)
                self.vecinos[u].add(v)
        def __str__(self):
                return "Aristas= " +str(self.aristas)+"\nVertices"+str(self.vertices)


>>> g=grafica()
>>> g.conecta("a","b")
>>> g.conecta("b","c")
>>> g.conecta("a","d")
>>> print(g)
Aristas= {('b', 'a'): 1, ('a', 'b'): 1, ('c', 'b'): 1, ('b', 'c'): 1, ('d', 'a'): 1, ('a', 'd'): 1}
Vertices{'d', 'c', 'b', 'a'}
```

Se inicializa la gráfica.

Va a contener vértices conectados para que se formen las aristas, al estar conectados se hacen vecinos unos de otros

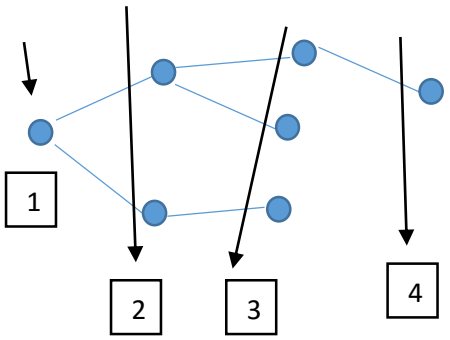
Si un vértice no está unido a otro se va a unir.

Convierte enteros en cadena de caracteres.

Resultados

Breath First Search

El BFS nos ordena los vertices ,primero los que se encuentran conectados al principal,despues los que estan conectados a los secundarios y asi sucesivamente. Utiliza la clase fila.



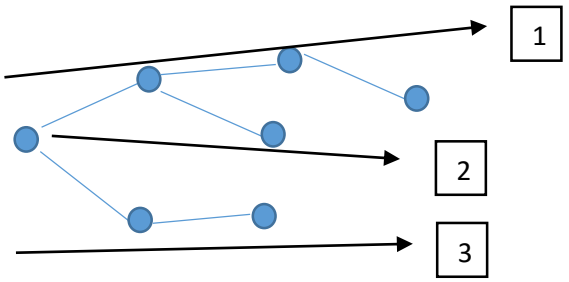

```
                self.vecinos=dict()
        def agrega(self,v):
                self.vertices.add(v)
                if not v in self.vecinos:
                        self.vecinos[v]=set()
        def conecta(self, u,v, peso=1):
                self.agrega(v)
                self.agrega(u)
                self.aristas[(v,u)]=self.aristas[(u,v)]=peso
                self.vecinos[v].add(u)
                self.vecinos[u].add(v)
        def __str__(self):
                return "Aristas= " +str(self.aristas)+"\nVertices"+str(self.vert
ices)


>>> def BFS(grafo, vertice):
        vistos = []
        por_ver = fila()
        por_ver.meter(vertice)
        while por_ver.longitud >0:
                vertice = por_ver.obtener()
                if vertice not in vistos:
                        vistos.append(vertice)
                        for vecino in grafo.vecinos[vertice]:
                                por_ver.meter(vecino)
        return vistos

>>> g=grafica()
>>> g.conecta("a","b")
>>> g.conecta("a","c")
>>> g.conecta("b","d")
>>> g.conecta("c","e")
>>> print(g)
Aristas= {('b', 'a'): 1, ('a', 'b'): 1, ('c', 'a'): 1, ('a', 'c'): 1, ('d', 'b')
: 1, ('b', 'd'): 1, ('e', 'c'): 1, ('c', 'e'): 1}
Vertices{'d', 'e', 'c', 'a', 'b'}
>>> BFS(g,"a")
['a', 'c', 'b', 'e', 'd']
```

Resultados

Karen Eunice Oviedo Garza

Deep First Search

El DFS nos ordena los vertices ,primero una de las lineas que se conectan al vertice principal y despues la siguiente linea y asi sucesivamente.Utiliza la clase pila.




```
                self.vecinos=dict()
        def agrega(self,v):
                self.vertices.add(v)
                if not v in self.vecinos:
                        self.vecinos[v]=set()
        def conecta(self, u,v, peso=1):
                self.agrega(v)
                self.agrega(u)
                self.aristas[(v,u)]=self.aristas[(u,v)]=peso
                self.vecinos[v].add(u)
                self.vecinos[u].add(v)
        def __str__(self):
                return "Aristas= " +str(self.aristas)+"\nVertices"+str(self.vert
ices)


>>> def DFS(grafo, vertice):
        vistos = []
        por_ver = pila()
        por_ver.meter(vertice)
        while por_ver.longitud >0:
                vertice = por_ver.obtener()
                if vertice not in vistos:
                        vistos.append(vertice)
                        for vecino in grafo.vecinos[vertice]:
                                por_ver.meter(vecino)
        return vistos

>>> g=grafica()
>>> g.conecta("a","b")
>>> g.conecta("a","c")
>>> g.conecta("b","d")
>>> g.conecta("c","e")
>>> print(g)
Aristas= {('b', 'a'): 1, ('a', 'b'): 1, ('c', 'a'): 1, ('a', 'c'): 1, ('d', 'b')
: 1, ('b', 'd'): 1, ('e', 'c'): 1, ('c', 'e'): 1}
Vertices{'e', 'a', 'd', 'b', 'c'}
>>> DFS(g,"a")
['a', 'c', 'e', 'b', 'd']
```

Resultados