REPORTE DE ESTRUCTURA DE DATOS

Elaborado por: Luis Daniel Honorato Hernández

Licenciatura en Matemáticas

1729448

INTRODUCCION

Este reporte abordara contenido acerca de la unidad numero dos llamada estructura de datos en la cual se encuentran distintos tipos de clases las cuales son fila y pila y estas dos se usan dentro de un grafo para realizar búsquedas por profundidad o anchura de todo lo antes mencionado se explicara acerca de que es desde mi punto de vista, el funcionamiento de cada uno y un código donde se explicara a detalle lo que se está queriendo realizar al finalizar se hará una breve conclusión de lo aprendido durante este reporte.

Una fila o también llamada cola consiste en acomodar un conjunto de agrupaciones ya sea en los sistemas informáticos transportes y operaciones en los cuales están acomodados de forma horizontal y funcionan en encolar y desencolar los objetos introducidos es decir en pocas palabras viéndolo en un ejemplo “si se tiene una autopista de carros se desencola el primer carro y te devolverá el primer valor”.

Una pila se vería como una estructura de datos similar a cuando tienes una lista de datos en los cuales los datos que se introducen salen por el mismo lugar algunos ejemplos aplicados a este concepto serian una pila de datos acomodados uno sobre otro, una pila de libros, las estanterías del supermercado y el ponerse muchas camisas.

Un grafo funciona igual como un árbol genealógico ya que cualquier nodo (vértice) señala a cualquier otro nodo(vértice) que pertenece del anterior además estos almacenan datos que tienen algún tipo de relación ya sea un parentesco o puestos de trabajo o descendencia de algún poder.

Para el recorrido de un grafo se realizan dos tipos de técnicas las cuales son búsqueda de profundidad el cual consiste en alejarse del vértice inicial para recorrer los demás vértices a través de ciclos mientras que la búsqueda por anchura busca visitar primero a los vértices vecinos del vértice inicial después los vecinos de estos y así consecutivamente.

```
class Pila:
    def __init__(self):
        self.pila=[]#lista a utilizar para acomodar los objetos
    def obtener(self):#aqui no se encuentra la e que es “dato” ya que lo que se desea es desapilar y no encontrar el elemento agregado
        return self.pila.pop()
    def meter(self,e):#aqui se usa la e que es “dato” para agregar un elemento a la lista
        self.pila.append(e)
        return len(self.pila)
    @property
    def longitud(self):#aqui se obtiene la longitud del arreglo al finalizar la pila
        return len(self.pila)


p=Pila()
p.meter(1)
p.meter(2)
p.meter(3)
p.meter(5)
print(p.longitud)
print(p.obtener())
print(p.obtener())
print(p.obtener())
print(p.obtener())
print(p.longitud)
```

```
class Cola:
    def __init__(self):#lista a utilizar para acomodar objetos
        self.cola=[]
    def obtener(self):#aqui se agrega el ultimo elemento al final de la lista
        return self.cola.pop(0)
    def meter(self,e):#aqui se agrega un elemento a la lista
        self.cola.append(e)
        return len(self.cola)
    @property
    def longitud(self):#aqui se regresa la longitud final de la cola
        return len(self.cola)

c=Cola()
c.meter(1)
c.meter("wafle")
c.meter(2)
c.meter("muñeca")

print(c.longitud)
print(c.obtener())
print(c.obtener())
print(c.obtener())
print(c.obtener())
print(c.longitud)
```

```
class Grafo:


    def __init__(self):
        self.V = set()#conjunto de vertices
        self.E = dict()#conjunto de aristas
        self.vecinos = dict()


    def agrega(self, v):#se agrega un vértice
        self.V.add(v)
        if not v in self.vecinos: #si el vértice no esta agregado aquí se hace la adicion
de ello
            self.vecinos[v] = set()


    def conecta(self, v, u, peso=1): #aqui se van conectando los lazos del grafo
        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)


    @property
    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
```

```
from grafo import Grafo
G = Grafo()
G.conecta('a', 'b', 5)
G.conecta('a', 'c', 7)
G.conecta('b', 'c', 2)
G.conecta('c', 'd', 4)
print(G.vecinos['a'])
print(G.V)
print(G.E)
G2 = G.complemento()

class Cola:
    def __init__(self):
        self.cola=[]
    def obtener(self):
        return self.cola.pop(0)
    def meter(self,e):
        self.cola.append(e)
        return len(self.cola)
    @property
    def longitud(self):
        return len(self.cola)

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,u,v,peso=1):
        self.agrega(u)
        self.agrega(v)
        self.E[(u,v)]=self,E[(v,u)]=peso
        self.vecinos[u].add(v)
        self.vecinos[v].add(u)

    @property
    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(u,w,1)
    return comp

def BFS(graph,ni):
    visitados=[ni]
    f=Cola()
    f.meter(ni)
    while f.longitud>0:
```

```
        na=f.obtener()
        vecinos=graph.vecinos[na]
        for nodo in vecinos:
            if nodo not in visitados:
                visitados.append(nodo)
                f.meter(nodo)
    return visitados

graph=Grafo()
graph.conecta(1,3)
graph.conecta(3,4)
graoh.conecta(4,8)
graph.conecta(8,9)
graph.conecta(1,6)
graph.conecta(6,7)
graph.conecta(7,5)
graph.conecta(5,2)
print(BFS(graph,1))
```

```
class Pila:
    def __init__(self):
        self.pila=[]
    def obtener(self):
        return self.pila.pop(0)
    def meter(self,e):
        self.pila.append(e)
        return len(self.pila)
    @property
    def longitud(self):
        return len(self.pila)

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,u,v,peso=1):
        self.agrega(u)
        self.agrega(v)
        self.E[(u,v)]=self,E[(v,u)]=peso
```

```
        self.vecinos[u].add(v)
        self.vecinos[v].add(u)

    @property
    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(u,w,1)
    return comp

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

graph=Grafo()
graph.conecta(1,3)
```

```
graph.conecta(3,4)
graoh.conecta(4,8)
graph.conecta(8,9)
graph.conecta(1,6)
graph.conecta(6,7)
graph.conecta(7,5)
graph.conecta(5,2)
print(DFS(graph,1))
```

CONCLUSION

De este presente reporte aprendí acerca de la diferencia entre fila y pila ya que estos dos conceptos si los había escuchado pero no muy a fondo como los vi en clase además pude conocer el funcionamiento de los grafos y más aplicados al momento de programarlos aunque tuve algunas dificultades al momento de comprenderlos y me di cuenta que es muy útil todo esto al momento de hacer un orden en acumulación de datos para llevar el orden de una manera práctica y flexible y por ello debe quedar muy en claro cada concepto y particularidad vista