Durante el siguiente reporte estaré hablando del algoritmo Dijkstra,  diré que es, como funciona además de que pondré el código con el que estuvimos trabajando.

El algoritmo Dijkstra es un algoritmo el que usa el camino más corto y fácil dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Este funciona de manera en la que parte de un vértice que será ingresado a partir de estas evaluaremos sus adyacentes, buscamos el que está más cerca del vértice principal o central como lo quieran llamar lo tomamos como punto intermedio y vemos si podemos llegar más rápido a través de este vértice a los demás, así estamos hasta que funcione bien por consiguiente elegimos al vértice más cercano y vamos repitiendo el proceso esto y hasta cuando se hará esto pues hasta que el vértice menos utilizado sea su destino, esto es un poco de cómo funciona el algoritmo Dijkstra.

Ya explicando un poco de que es Dijkstra y cómo funciona ahora lo que en verdad nos interesa es como programarlo en Python, primero que nada tenemos que definir una función que tenga como dato un nodo (un nodo es Se trata de una estructura de datos que puede utilizarse para la implementación de nuevas **estructuras),** después creamos un arreglo donde estarán las tuplas de lo que se está juntando y después de hacer esto creamos un diccionario donde marcamos las distancias y un conjunto para ir guardando todas los ya visitado y usamos un ciclo while, este va a hacer repetición de las instrucciones en el mientras halla en el grafo modos no visitados

Codigo de Dijksta

```
from heapq import heappop, heappush


from copy import deepcopy




def flatten(L):


    while len(L) > 0:
```

```
        yield L[0]

        L = L[1]



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




def shortest(self, v):

    q = [(0, v, ())]

    dist = dict()

    visited = set()

    while len(q) > 0:
        (l, u, p) = heappop(q)
        if u not in visited:

            visited.add(u)

            dist[u] = (l,u,list(flatten(p))[::-1] + [u])
        p = (u, p)
        for n in self.vecinos[u]:
            if n not in visited:
```

```
            el = self.E[(u,n)]

            heappush(q, (l + el, n, p))

    return dist



def kruskal(self):

    e = deepcopy(self.E)

    arbol = Grafo()

    peso = 0

    comp = dict()

    t = sorted(e.keys(), key = lambda k: e[k], reverse=True)            nuevo = set()

    while len(t) > 0 and len(nuevo) < len(self.V):

        #print(len(t))

        arista = t.pop()
```

```
    w = e[arista]

    del e[arista]

    (u,v) = arista

    c = comp.get(v, {v})

    if u not in c:

        #print('u ',u, 'v ',v ,'c ', c)

        arbol.conecta(u,v,w)

        peso += w

        nuevo = c.union(comp.get(u,{u}))

        for i in nuevo:

            comp[i]= nuevo

print('MST con peso', peso, ':', nuevo, '\n', arbol.E)

return arbol
```

```
g= Grafo()

g.conecta('a','b', 1)

g.conecta('a','c', 1)

g.conecta('a','d', 1)

g.conecta('a','e', 1)

g.conecta('c','e', 1)

g.conecta('c','f', 10)

g.conecta('b','f', 1)




print(g.kruskal())

print(g.shortest('c'))
```

## Tabla de como quedaron los resultados del algoritmo

Para empezar de 5 modos y 10 aristas

| p | modos | min(d) |
|---|---|---|
| a-a | a | 0 |
| a-c | c | 1 |
| a-e | e | 1 |

| | | |
|---|---|---|
| a-b | b | 2 |
| a-d | d | 2 |

Despues de 10 modos y 20 aristas

| p | modos | min(d) |
|---|---|---|
| c-c | c | 0 |
| c-d | d | 1 |
| c-j | j | 1 |
| c-d-a | a | 2 |
| c-b | b | 2 |
| c-e | e | 2 |
| c-e-f | f | 3 |
| c-d-a-i | i | 3 |
| c-d-a-i-h | h | 6 |
| c-d-a-i-h-g | g | 7 |

15 modos y 30 aristas

| p | modos | min(d) |
|---|---|---|
| k-k | k | 0 |
| k-j | j | 1 |
| k-l | l | 5 |
| k-j-o | o | 5 |
| k-l-f | f | 6 |
| k-l-m | m | 6 |
| k-j-o-n | n | 6 |
| k-l-f-a | a | 7 |
| k-l-f-b | b | 7 |
| k-j-o-n-d | d | 7 |
| k-u-o-n-d-e | e | 8 |
| k-l-f-b-h | h | 8 |
| k-l-f-b-i | i | 8 |
| k-l-f-a-c | c | 9 |
| k-l-f-g | g | 9 |

20 modos y 40 aristas

| p | modos | min(d) |
|---|---|---|
| s-s | s | 0 |
| s-e | e | 1 |
| s-t | t | 1 |
| s-e-d | d | 2 |
| s-e-r | r | 2 |
| s-e-f | f | 3 |
| s-e-d-n | n | 3 |
| s-e-r-q | q | 3 |
| s-e-f-a | a | 4 |
| s-e-f-b | b | 4 |
| s-e-f-l | l | 4 |
| s-e-d-e-m | m | 4 |
| s-e-f-o | o | 4 |
| s-e-f-b-h | h | 5 |
| s-e-f-b-i | i | 5 |
| s-e-f-b-p | p | 5 |
| s-e-f-a-c | c | 6 |
| s-e-f-g | g | 6 |
| s-e-f-o-j | j | 8 |
| s-e-f-o-j-k | k | 9 |

## Conclusiones

En esta etapa estuvimos viendo algo que ya habíamos visto que son grafos, y como dije en el reporte antepasado mientras explicaba el profe sobre estos no pude asistir a clase, entonces viendo videos o cosas en internet y logre entender un poco más y a continuación se hablara un poco más a detalle sobre estos y que hemos complementado estos temas y aunque me falta mucho en este tema tengo que ponerme a repasar más y después que vimos sobre el algoritmo de Dijksta.