# Experimentos con el algoritmo de Floyd-Warshall en `python`

Yessica Reyna Fernández

Flujo en Redes

22 de abril de 2018

## 1. Introducción

En base a la estructura ya usada por los grafos en la práctica anterior[1], se

## 2. Grafo circular base

Primeramente con respecto a la posición de los nodos, se posicionan en forma de circunferencia, entonces se realiza la transformación a coordenadas en radianes para la circunferencia a formar; a continuación se debe definir en ese caso la posición del angulo en donde ira cada punto para esta transformación por lo cual se deberá tomar una medida equitativa para la cantidad de nodos que se tiene, tomando en cuenta la ecuación paramétrica de la circunferencia con centro en $(a, b)$ donde $x = a + rCos(t)$ y $y = b + rSen(t)$ con $t \in [0, 2\pi]$, definiendo lo siguiente como parte básica del grafo:

```
1  angulo=2*pi/n #n=cantidad de nodos
2  r=0.3 #radio fijo de la circunferencia
3  c=(0.5,0.5) #posición del centro de la circunferencia
4  self.pos[v] = (c[0]+(r * cos(angulo * v)), c[1]+(r* sin(angulo * v))) #dentro de
       la función nodoscrear() con v=indice del nodo a crear
```

Ahora bien hablando de la forma de trazar las aristas entre nodos se mide en forma de un parámetro $k$, el cual varia dependiendo de la cantidad de nodos; por lo que se busca que cada vez que se corra el código las repeticiones que habrá de cada tamaño este dada por el valor $\lfloor n/2 \rfloor$ ademas de que el valor de $k$ realiza las conexiones entre nodos; asimismo se define `E[(u,v)]` como las distancias euclidianas entre cada par de nodos y `vecinos[v].add(i+(j+1))` lo que agrega al nodo $i + (j + 1)$ entre los vecinos de $v$, todo esto descrito en lo siguiente:

```
1  G.conecta(k) #función de conexiones en aristas
2  for j in range(k): #recorrer tamaño de k
3      for i in range(len(self.V)): #recorriendo cada nodo
4          if i< (len(self.V)-(j+1)): #siempre que el indice no pase el valor k
5              self.E[(i, i+(j+1))] = self.E[(i+(j+1), i)]=self.euclidiana(i,(i+(j+1)))
6              self.vecinos[i].add(i+(j+1))
7              self.vecinos[i+(j+1)].add(i)
```

En relación con esto se define la función para integrar aristas aleatorias al grafo que depende de una probabilidad que aumenta con el valor de $k$, agregando esa nueva arista siempre que no se encuentre considerada en el, ademas de que su probabilidad para entrar al grafo sea mas pequeña que un numero que se genera aleatoriamente:

```
prob=2**-(k)
G.conectaaleatorio(prob)
    if  m is not w and (m,w) not in self.E:
        if  random()< prob:
            self.E[(m,w)]=self.E[(w,m)]=self.euclidiana(m,w)
            self.vecinos[m].add(w)
            self.vecinos[w].add(m)
```

Así pues esto definiría lo mas básico por elaborar par aun grafo base circular, tratando en las siguientes secciones otros aspectos que se pueden agregar al grafo.

# 3. Promedios de densidad

Como se ha tratado en practicas anteriores, el algoritmo de Floyd-Warshall crea un vector con todos los pares de distancias entre un par de puntos dados $u$ y $v$.
Por lo cual se puede

# 4. Densidad de cluster

# 5. Resultados