Reporte de Dijkstra

Andrea Estefanía Galván Irigoyen

M.1735583 gpo.001

Matemáticas Computacionales

En este reporte se mostrará de lo que se trata el código de dijkstra, para qué nos puede servir y como se programa en Python para facilitarnos ciertas cosas.

Cuando tenemos un grafo con un punto de origen (un vértice) y tenemos otros vértices a los que queremos legar pero cada uno con longitudes distintas en este caso pesos de las aristas y queremos tener la opción del camino para recorrer todos los vértices de tal manera que dicho camino sea el más corto; con el algoritmo dijkstra nos da esa opción de optimización con estos casos sólo hay que agregar un grafo con los vértices que desees, los pesos de cada arista y el vértice de origen.

Para programar este algoritmo ocuparemos al iniciar colcar “from heapq import heappop, heappush”, siguiente de esto un grafo, después definiremos un shortest con un arreglo que tendrá las tuplas almacenadas, después ingresar los visitados y poner que cuando exista un nodo pendiente se tomará la tupla con la distancia menor y asi se agregará a visitados,, con un for para cada nodo si no ha sido visitado se toma el peso hacia el nodo de origen y se agregara a los visitados.

20 de octubre del 2017

CÓDIGO

```
from heapq import heappop, heappush
>>> def flatten(L):
    while len(L) > 0:
        yield L[0]
        L = L[1]
          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
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
```

Conclusión

Para este reporte tarde un poco en entender lo que se hacía con cada función en el código de dijkstra, pero leyendo los comentarios que venían en el código pude entenderlos mejor aparte de investigarlo un poco, en donde si me tarde un poco fue en probar el programa para los 25 nodos ya que fueron muchas aristas y fueron muchos datos.

20 de octubre del 2017