REPORTE DEL ALGORTIMO DISJKTRA

Elaborado por: Luis Daniel Honorato Hernández

Licenciatura en Matemáticas

1729448

INTRODUCCION

En este presente reporte se abordará el tema del algoritmo de Dijkstra en el cual se expondrá las ideas de lo que es el algoritmo y que se realiza a través de este además de una explicación un poco ya más clara con el código que se proporcionará adelante aunado con una tabla en donde se concentraran los resultados un poco ya mejor planteados y finalizando con una breve conclusión de lo aprendido en este reporte.

ALGORITMO DIJSKTRA

El algoritmo Dijkstra se trata acerca de un algoritmo que su principal función es encontrar un camino o recorrido más corto de los demás presentes y esto se hace a través de nodos y vértices que formaran un grafo en el cual para su comienzo se posicionara en el nodo origen a explorar por el usuario y después va comparando los pesos de cada arista que parte del nodo inicial y si este peso a comparar es menor a el otro o anterior se posicionara ahora en ese nodo y después tendrá que ver los hijos es decir los que están conectados con ese nodo y verificando los pesos menores hasta llegar a tu objetivo final y aunque no lo crean este algoritmo sirve de mucha ayuda en el momento en que tú quieres determinar la ruta corta de entre millones de ciudades para llegar al destino deseado o cuando se entregan paquetes a domicilio o rutas de viaje para el trabajo en fin son muchas de las aplicaciones de este grafo aunque pueden existir inconvenientes en el tiempo, clima o tráfico y entre otras cosas que no puedan definir a ese camino como el más corto así que se deben de verificar varias opciones al momento de realizar el algoritmo.

```
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() #es el conjunto de  nodos
        self.E = dict() #es el conjunto de aristas
        self.vecinos = dict() #conjunto de los hijos de cada nodo

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

    def conecta(self, v, u, peso=1): #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)



    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 shortests(self,v):#algoritmo de dijkstra
        q=[(0,v,())]#arreglo q de las tuplas de lo que se va a almacenar donde 0 es la distancia,
v el nodo y() el camino hacia el
        dist=dict()#diccionario de distancias
        visited=set()#conjunto de visitados
```

```
        while len(q)>0:#mientras exista un nodo pendiente
            (l,u,p)=heappop(q)#se toma la tupla con la distancia menor
            if u not in visited:#si no lo hemos visitado
                visited.add(u)#se agrega a visitados
                dist[u]=(l,u,list(flatten(p))[::-1]+[u])#agrega el diccionario
            p=(u,p)#tupla del nodo y el camino
            for n in self.vecinos[u]:#para cada hijo del nodo actual
                if n not in visited:#si no lo hemos visitado
                    el=self.E[(u,n)]#se toma la distancia del nodo actual mas la distancia hacia el nodo hijo
                    heappush(q,(l+el,n,p))#se agrega el arreglo q la distancia actual mas la distancia hacia el nodo hijo n hacia donde se va y el camino
        return dist #regresa el diccionario de distancias


    def kruskal(self):
        e=deepcoy(self.E)
        arbol=Grafo()
        peso=0
        comp=dict()
        t=sorted(e.keys(),key=lambda k:e[k],reverse=True)
        while len(t)>0:
            #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
#GRAFO 1
g=Grafo
g.conecta('a','b', 11)
g.conecta('a','f', 1)
g.conecta('a','d', 2)
g.conecta('b','f', 3)
g.conecta('b','e', 7)
g.conecta('f','e', 15)
g.conecta('d','e', 1)
#GRAFO 2
g=Grafo
g.conecta('a','b', 11)
g.conecta('a','e', 1)
g.conecta('a','g', 2)
g.conecta('b','c', 3)
g.conecta('c','d', 7)
g.conecta('a','e', 15)
g.conecta('e','f', 1)
g.conecta('f','d', 11)
g.conecta('a','g', 1)
g.conecta('g','h', 2)
g.conecta('h','i', 3)
g.conecta('f','i', 7)
g.conecta('i','k',0)
#GRAFO 3
```

```
g=Grafo
g.conecta('a','b', 11)
g.conecta('a','f', 1)
g.conecta('a','i', 2)
g.conecta('a','l', 3)
g.conecta('b','c', 7)
g.conecta('c','d', 15)
g.conecta('d','e', 1)
g.conecta('f','g', 11)
g.conecta('g','h', 1)
g.conecta('i','j', 2)
g.conecta('j','k', 3)
g.conecta('l','m', 7)
g.conecta('m','n', 3)
g.conecta('n','o', 7)
#GRAFO 4
g=Grafo
g.conecta('a','b', 11)
g.conecta('a','f', 1)
g.conecta('a','j', 2)
g.conecta('a','n', 3)
g.conecta('a','r', 7)
g.conecta('b','c', 15)
g.conecta('c','d', 1)
g.conecta('d','e', 11)
g.conecta('f','g', 1)
g.conecta('g','h', 2)
g.conecta('h','i', 3)
g.conecta('j','k', 7)
g.conecta('k','l', 3)
```

```
g.conecta('l','m', 7)
g.conecta('n','o', 4)
g.conecta('o','p', 2)
g.conecta('p','q', 3)
g.conecta('r','s', 7)
g.conecta('s','t', 3)
#GRAFO 5
g=Grafo
g.conecta('a','b', 11)
g.conecta('a','f', 1)
g.conecta('a','j', 2)
g.conecta('a','n', 3)
g.conecta('a','r', 7)
g.conecta('b','c', 15)
g.conecta('c','d', 1)
g.conecta('d','e', 11)
g.conecta('f','g', 1)
g.conecta('g','h', 2)
g.conecta('h','i', 3)
g.conecta('j','k', 7)
g.conecta('k','l', 3)
g.conecta('l','m', 7)
g.conecta('n','o', 4)
g.conecta('o','p', 2)
g.conecta('p','q', 3)
g.conecta('r','s', 7)
g.conecta('s','t', 3)
g.conecta('t','u', 4)
g.conecta('u','v', 2)
g.conecta('v','y', 3)
```

```
g.conecta('e','z', 7)
g.conecta('i','w', 3)


print(g.kruskal())
print(g.shortests('a'))
```

CONCLUSION

Con este trabajo aprendí que si son muy útiles los grafos de este tipo de algoritmo en la vida diaria ya que son muy indispensables y necesarios cuando se trata de hacer recorridos en un menor tiempo o distancia para poder realizar nuestras cosas como las tenemos planeadas para no perder ni un segundo de nuestras vidas y esto se ve muy eficiente al momento de que se corre el programa