Daniela Estefanía Villanueva Andrade

1684742

20 Octubre 2017

Reporte de algoritmo de Dijkstra.

El objetivo principal de éste algoritmo es encontrar el camino más corto desde un nodo o vértice que consideramos nuestro origen a otro de los nodos, lo anterior desarrollado a partir de un grafo que como sabemos está compuesto por los subconjuntos de vértices y aristas que a su vez tienen cierto peso y distancia existente entre ellos.

Ahora bien, la idea de éste algoritmo es ir explorando los caminos más cortos que parten de nuestro nodo origen a todos los demás, al obtener el camino más corto de nuestro punto de partida a los vértices que componen al grafo entonces, el algoritmo se detiene.

Para programar nuestro algoritmo de Dijkstra hicimos uso del grafo realizado con anterioridad; visitando cada uno de los nodos.

Código de Algoritmo de Dijkstra.

```
>>> 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()
```

```
        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 shortests(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
```

Recorrido	Distancia
c-a	1
c-f	1
c-c	0
c-b	2
b-f	2

Recorrido	Distancia
g-g	0
i-g	3
g-h	5
g-j	5
j-l	9
h-i	8
i-l	12
i-g	12
g-i	3
j-k	6

Recorrido	Distancia
l-g	12
l-4	4
g-i	3
k-j	6
m-j	6
k-l	10
m-l	2
l-n	3
g-m	11
g-l	6
o-p	7

0-c	10
j-m	6
c-p	7