Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

Matemáticas Computacionales

Ismael Medina Robledo

1744617

Reporte de Algoritmo de Dijkstra

- **Introducción**

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene.

- **Algoritmo**

Este algoritmo recorre los nodos, partiendo del nodo inicial, donde recorre todos los posibles caminos que conectan a un nodo con otro, después de recorrer y calcular la distancia más corta de todas las rutas, el algoritmo termina.

Pseudocódigo

```
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, w): # Dijkstra's algorithm
                        q = [(0, v, ())] # arreglo "q" de las "Tuplas" de lo que se va a almacenar dondo 0 es la distancia, v el nodo y ()
                        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
                                        if u == w: #si es el nodo final#
                                                return list(flatten(p))[::-1] + [u] #se regresa el camino
                                p = (u, p) #Tupla del nodo y el camino
                                for n in self[u].neighbors: #Para cada hijo del nodo actual
                                        if n not in visited: #si no lo hemos visitado
                                                el = self.vecinos[u][n] #se toma la distancia del nodo acutal hacia el nodo hijo
                                                heappush(q, (l + el, n, p)) #Se agrega al arreglo "q" la distancia actual mas la ditanacia
 return None
```

- **Conclusiones**

Este algoritmo funciona de manera efectiva cuando los pesos de las aristas (nodos) son mayores que 1. Aunque el recorrido que realiza el algoritmo es muy variado, y varia por los pesos de las aristas, las conexiones que tienen los nodos dentro del grafo. Un algoritmo muy interesante que no es muy intuitivo de realizar.