Universidad Autónoma de Nuevo León

FCFM

Matemáticas Computacionales

Algoritmo Dijkstra

Ana Cecilia García Esquivel Matrícula: 1749760

Resumen:

En este reporte hablaré sobre uno de los algoritmos más utilizados en problemas de optimización, el denominado algoritmo Dijkstra, así como pondré un ejemplo de dicho algoritmo y además de unos ejemplos de resultados obtenidos mediante este algoritmo.

Dijkstra

El algoritmo Dijkstra, es un algoritmo que nos ayuda a encontrar la ruta a seguir más corta entre una cierta cantidad de nodos, es por eso que es muy utilizado en problemas de optimización, sobre todo de rutas, claro está. Dijkstra es muy útil ya que no está restringido para cierta cantidad de nodos, es decir, puedes elegir la cantidad de nodos que uno quiera y con ayuda de los pesos (magnitudes) de cada camino, Dijkstra facilitará la tarea de hallar la ruta más corta para recorrer todos y cada uno de los nodos.

¿Cómo funciona?

Este algoritmo parte de un nodo inicial, el cual ingresamos con la instrucción “print(g.shortest(nombre de nodo inicial))”, a partir de ahí Dijkstra evalúa el peso de las aristas (caminos) que conducen hacia los nodos “hijos” del nodo en gestión y determina cual peso es menor y se decide por esa arista, al llegar al siguiente nodo vuelve a hacer la evaluación y así continúa hasta haber recorrido todos los nodos. Cabe mencionar que el valor de los pesos deben ser positivos ya que estamos considerando dichos pesos como magnitudes, si existen valores

negativos para el peso de las aristas se deben incluir otra clase de algoritmos. Como este algoritmo se utiliza o se aplica con grafos, debemos tener previamente definida la función de Grafos() para poder utilizarla y que todo funcione correctamente.

El siguiente código lo hizo el profesor de nuestra clase de Matemáticas Computacionales que corresponde únicamente al algoritmo Dijkstra (sólo la función principal), todo lo que se necesita para que funcione está en el código subido a GitHub.

```
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
```

Enseguida se muestran en tabla algunos de los resultados obtenidos al utilizar este algoritmo:

Grafo de 5 nodos y 10 aristas

Nodo	Recorrido	Distancia mín. recorrida
A	d, a	4
B	d, b	3
C	d, a, c	5
D	d	0
E	d, e	8

Grafo de 10 nodos y 20 aristas

Nodos	Recorrido	Distancia mínima
A	a	0
B	A, b	1
C	A, b, c	3
D	A, d	2
E	a, b, e	5
F	A, b, c, f	7
G	A, b, c, f, j, g	12
H	A, b, c, h	8
I	A, b, c, f, i	9
J	A, b, c, f, j	10

Grafo de 15 nodos y 30 aristas

Nodos	Recorrido	Distancia mín. recorrida
A	a	0
B	A, b	1
C	A, b, c	3
D	a, b, c, d	6
E	a, b, c, d, e,	10
F	A, b, f	4
G	A, b, f, g	10
H	A, b, c, h	7
I	A, b, c, h, i	9
J	A, b, c, h, j	14
K	A, b, f, k	13
L	A, b, c, h, i, l	14
M	A, b, c, h, i, l, m	20
N	A, b, c, h, i, n	15
O	A, b, c, h, i, l, o	

Conclusiones

Considero que este algoritmo es muy útil y fácil de usar una vez ya creado, y que probablemente como matemáticos usaremos muy seguido en proyectos futuros, como por ejemplo para la clase de Matemáticas Discretas.