

Matemáticas computacionales

Reporte de dijsktra

Alumno: Isaac Emanuel Segovia Olay

M. 1748957

Maestro: Lic. José Anastacio Hernández Saldaña

20-octubre-2017

Dijsktra


En este algoritmo nos dimos cuenta de que sirve para saber cual es el camino más corto al momento de darle un nodo inicial con el resto de los nodos en un grafo el dónde en cada arista tiene un peso. Este algoritmo explora todos los caminos más cortos que parten desde el nodo inicial y llevan al resto de los nodos, cuando por fin se obtiene el camino más corto el algoritmo se detiene.

También se puede decir que es usado para encontrar la ruta más corta, ya que cada arista tiene un peso, el algoritmo encontrará el camino con menos peso. Hay que mencionar que este algoritmo sólo funciona con pesos positivos.

¿Cómo se programa?

Para empezar, hay que marcar todos los vértices como no visitados. Empezando del nodo inicial, evaluamos las aristas para poder encontrar el que tenga un camino más corto, ósea, con menor peso, como se mencionó antes. Después se compara la arista con los demás nodos para ver si se puede llegar más rápido, una vez que ya tengamos el nodo más cercano, se repite otra vez el proceso, así estamos obteniendo la arista con el menor peso que se pueda. Se finaliza cuando llega hasta nuestro nodo distinto.

Este es el algoritmo que se utilizó para realizar la actividad.

 dijkstra.py - C:/Users/isaac/Desktop/dijkstra.py (3.6.2)

File Edit Format Run Options Window Help

```
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): # Algoritmo Dijkstra
        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
```