Universidad Autónoma de Nuevo León Facultad de Ciencias Físico Matemáticas

Artemio Guajardo Aparicio Grupo: 001 Matrícula: 1590417 Fecha: 20/10/2017

Algoritmo Dijkstra

Introducción

Para obtener rutas cortas en varias cosas existen muchos algoritmos, en este caso veremos el algoritmo de Dijkstra para obtener la ruta más corta en un grafo. Veremos para que sirve, como fue que fue implementado y como trabaja utilizándolo en grafos con diferentes cantidades de nodos y aristas.

Descripción e implementación del algoritmo

Como ya se mencionó en la introducción, este algoritmo sirve para determinar la ruta más corta desde un nodo inicial hacia los demás nodos, lo cual requiere un grafo que tenga pesos en las aristas ya que si todos los pesos en las aristas de nuestro grafo son de 1 es más conveniente usar BFS (mencionado en reportes anteriores).

Este algoritmo consiste en ir explorando los caminos que parten del nodo inicial y que llevan a todos los demás nodos, cuando se obtiene el camino más corto desde el nodo inicial al resto de nodos el algoritmo habrá finalizado.

Para su implementación necesitaremos obviamente la función grafo y ya en nuestra función Dijkstra primero declaramos un arreglo donde tendremos las Tuplas de lo que se va a almacenar donde especificaremos la distancia, el nodo y el camino hacia él, después inicializamos variables para tener nuestro diccionario de distancias y nuestro conjunto de nodos visitados. Después tomamos la tupla con la distancia menor y agregamos a visitados los nodos que no habíamos visitado, de igual manera se agregan al diccionario, luego para cada hijo del nodo en el que estamos, si no lo hemos visitado se toma la distancia del nodo actual hacia el nodo hijo y se agrega al arreglo la distancia actual más la distancia hacia el nodo hijo, el nodo hijo hacia donde se va y el camino, es decir los nodos por los que pasa. Al final obtendremos la mínima distancia, los nodos y los nodos recorridos.

La implementación directamente en Python queda de la siguiente manera:

```
def shortest(self, v):  # Dijkstra's algorithm
    q = [(0, v, ())]  # arreglo "q" de las "Tuplas" de lo que se va a almacenar
    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 al 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 acutal hacia el nodo hijo
          heappush(q, (l + el, n, p))  #Se agrega al arreglo "q" la distancia actual más la
distancia hacia el nodo hijo, el nodo hijo n hacia donde se va, y el camino
    return dist  #regresa el diccionario de distancias
```

Esta es solo la implementación para el algoritmo Dijkstra, pero se necesita también en nuestro programa un algoritmo para grafos.

Tablas con resultados de pruebas realizadas para grafos de diferentes tamaños

Para un grafo con 5 nodos y 10 aristas:

Nodo	Recorrido	Distancia mínima
a	c-e-d-a	6
c	c-c	0
b	c-b	6
e	c-e	4
d	c-e-d	5

Para un grafo con 10 nodos y 20 aristas

Nodo	Recorrido	Distancia mínima
a	f-c-a	4
c	f-c	1
b	f-c-a-b	10
e	f-j-e	11
d	f-d	10
g	f-g	15
f	f-f	0
i	f-c-a-i	10
h	f-c-a-h	6
j	f-j	3

Para un grafo con 15 nodos y 30 aristas

Nodo	Recorrido	Distancia mínima
a	j-e-k-a	14
c	j-e-k-a-b-c	22
b	j-e-k-a-b	16
e	j-e	6
d	j-e-k-a-b-d	24
g	j-e-h-g	36
f	j-e-k-a-b-f	28
i	j-e-n-i	33
h	j-e-h	27
k	j-e-k	9
j	j-j	0
m	j-e-m	24
l	j-e-k-l	16
o	j-e-o	20
n	j-e-n	22

Para un grafo con 20 nodos y 40 aristas

Nodo	Recorrido	Distancia mínima
a	r-o-f-l-a	32
c	r-o-f-l-a-t	38
b	r-o-f-l-b	37
e	r-o-f-l-a-t-e	40
d	r-m-j-p-d	44
g	r-o-f-l-a-t-g	42
f	r-o-f	25
i	r-o-f-l-a-t-i	44
h	r-o-f-l-a-t-g-h	43
k	r-k	12
j	r-m-j	25
m	r-m	14
l	r-o-f-l	27
o	r-o	16
n	r-n	27
q	r-q	18
p	r-m-j-p	38
s	r-s	20
r	r-r	0
t	r-o-f-l-a-t	34

Para un grafo con 25 nodos y 50 aristas

Nodo	Recorrido	Distancia mínima
a	v-x-a	12
c	v-w-r-y-c	14
b	v-x-b	15
e	v-w-u-e	10
d	v-w-d	6
g	v-g	11
f	v-w-u-f	13
i	v-w-u-i	21
h	v-w-h	8
k	v-w-s-k	15
j	v-w-u-j	23
m	v-m	17
l	v-w-u-l	27
o	v-w-o	11
n	v-w-r-y-n	8
q	v-x-a-q	15
p	v-w-u-p	19
s	v-w-s	3
r	v-w-r	2
u	v-w-u	7
t	v-w-t	5
w	v-w	1
v	v-v	0
y	v-w-r-y	6
x	v-x	3

Conclusiones

Como ya se había mencionado antes este algoritmo funciona cuando los pesos de las aristas no son 1 (en tal caso se usaría BFS), y con las tablas podemos observar que el recorrido realizado por el algoritmo varía según la conexión de los nodos, mientras que la distancia más corta varía según la conexión de los nodos y los pesos de las aristas. Cabe mencionar que en clase se mencionó que este algoritmo no funciona para grafos con aristas cuyos pesos sean negativos. El algoritmo es eficiente mientras los pesos de las aristas sean positivos.