Introducción

Caminos de peso mínimo

Otro problema de teoría de gráficas que se vuelve mucho más versatil en las aplicaciones en su versión ponderada es el de encontrar caminos cortos entre dos vértices. En una gráfica sin pesos lo que nos interesa para conectar dos vértices óptimamente es simplemente que se minimice su número de aristas. En las gráficas con aristas ponderadas lo que nos interesará es encontrar un camino cuyo peso total sea mínimo.

Problema. Dada una gráfica con aristas ponderadas \(G\) y dos vértices \(u\) y \(v\), encontrar la distancia ponderada de \(u\) a \(v\).

Problema. Dada una gráfica con aristas ponderadas \(G\) y dos vértices \(u\) y \(v\), encontrar un camino de menor peso de \(u\) a \(v\).

Problema. Dada una gráfica con aristas ponderadas \(G\), encontrar todas las distancias ponderadas entre cada par de vértices.

Algoritmo de Dijkstra

```
import networkx as nx
import matplotlib.pyplot as plt
import sympy as sp

def dijkstra(G,u,v):
    masinf=float('inf')
    vertices=list(G.nodes)
    distancias={w:masinf for w in vertices}
    fijos={w:False for w in vertices}
    padres={w:None for w in vertices}
    distancias[u]=0
    fijos[u]=True
    nuevo_fijo=u

    while not(all(fijos.values())):
        # Acualizar distancias.
        for w in G.neighbors(nuevo_fijo):
            if fijos[w]==False:
                nueva_dist=distancias[nuevo_fijo]+G[nuevo_fijo][w]['weight']
                if distancias[w]>nueva_dist:
                    distancias[w]=nueva_dist
                    padres[w]=nuevo_fijo

        # Encontrar el nuevo a fijar.
        mas_chica=masinf
        for w in vertices:
            if fijos[w]==False and distancias[w]<mas_chica:
                optimo=w
                mas_chica=distancias[w]
        nuevo_fijo=optimo
        fijos[nuevo_fijo]=True

        # Cuando fije el vértice final v, dar el camino.
        if nuevo_fijo==v:
            camino=[v]
            while camino[0]!=u:
                camino=[padres[camino[0]]]+camino
            return distancias[v], camino
```

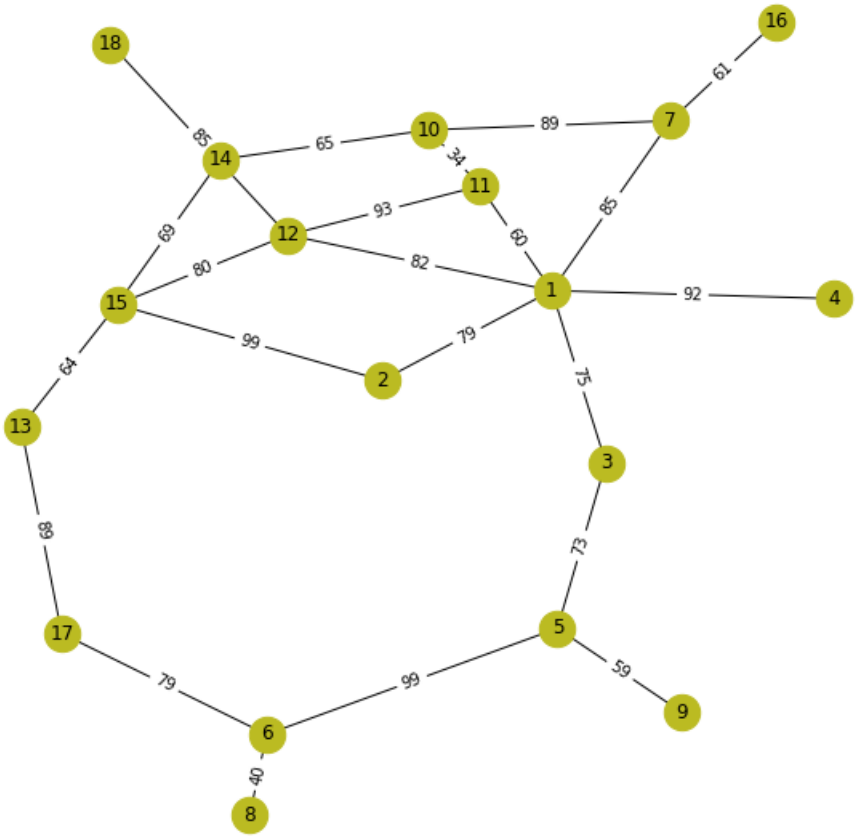
Hagamos un ejemplo de aplicación de Dijkstra.

```
G=nx.Graph()
G.add_weighted_edges_from([
    (1, 7, 85), (7, 16, 61), (12, 18, 68), (10, 11, 34),
    (14, 10, 65), (6, 5, 99), (15, 13, 64), (1, 11, 60),
    (18, 12, 85), (11, 12, 93), (1, 2, 79), (1, 4, 92),
    (17, 13, 89), (8, 6, 40), (15, 2, 99), (1, 3, 75),
    (15, 14, 69), (6, 17, 79), (9, 5, 59),
    (1, 12, 82), (7, 10, 89), (5, 3, 73), (15, 12, 80)
    ])
```

```
KKL=nx.kamada_kawai_layout(G)
labels = nx.get_edge_attributes(G,'weight')

fig, ax= plt.subplots(1)
fig.set_size_inches(10,10)
nx.draw_kamada_kawai(G,ax=ax,with_labels=True, node_color='#bbbb22',node_size=500)
nx.draw_networkx_edge_labels(G,KKL,ax=ax,edge_labels=labels)

D=dijkstra(G,7,17)
plt.show()
print("El camino más corto del vértice 7 al 17 es {} y pesa {}".format(D[1],D[0]))
```




```
El camino más corto del vértice 7 al 17 es [7, 10, 14, 15, 13, 17] y pesa 376
```

Algoritmo de Floyd-Warshall

Daremos una implementación de Floyd-Warshall en donde iremos llevando varias matrices de distancias correspondientes a las distintas etapas de inclusión de vértices como vértices intermedios.

```
def floydwarshall(G):
    masinf=float('inf')
    vertices=list(G.nodes)
    n=len(vertices)
    matrices=[]
    M=[[None for i in range(n)] for j in range(n)]

    for i in range(n):
        for j in range(n):
            if i==j:
                M[i][j]=0
            elif (vertices[i],vertices[j]) in G.edges or (vertices[j],vertices[i]) in G.edges:
                M[i][j]=G[vertices[i]][vertices[j]]['weight']
            else:
                M[i][j]=masinf
    matrices.append(M)

    for k in range(n):
        M=[[None for i in range(n)] for j in range(n)]
        for i in range(n):
            for j in range(n):
                M[i][j]=min(matrices[-1][i][j],matrices[-1][i][k]+matrices[-1][k][j])
        matrices.append(M)

    return(matrices[-1])
```

```
G=nx.Graph()
G.add_weighted_edges_from([
    (1, 7, 85), (7, 16, 61), (12, 18, 68), (10, 11, 34),
    (14, 10, 65), (6, 5, 99), (15, 13, 64), (1, 11, 60),
    (18, 12, 85), (11, 12, 93), (1, 2, 79), (1, 4, 92),
    (17, 13, 89), (8, 6, 40), (15, 2, 99), (1, 3, 75),
    (15, 14, 69), (6, 17, 79), (9, 5, 59),
    (1, 12, 82), (7, 10, 89), (5, 3, 73), (15, 12, 80)
    ])

print(list(G.nodes))
print(G.edges)
display(sp.Matrix(floydwarshall(G)))
```

```
[1, 7, 16, 12, 18, 10, 11, 14, 6, 5, 15, 13, 2, 4, 17, 8, 3, 9]
[(1, 7), (1, 11), (1, 2), (1, 4), (1, 3), (1, 12), (7, 16), (7, 10), (12, 18), (12, 11), (12, 15), (10, 11), (10, 14), (14, 15
```

$$\begin{split}\displaystyle \left[\begin{array}{ccccccccccccccc}0 & 85 & 146 & 82 & 167 & 94 & 60 & 159 & 247 & 148 & 162 & 226 & 79 & 92 & 315 & 287 & 75 & 207\\85 & 0 & 61 & 167 & 252 & 89 & 123 & 154 & 332 & 233 & 223 & 287 & 164 & 177 & 376 & 372 & 160 & 292\\146 & 61 & 0 & 228 & 313 & 150 & 184 & 215 & 393 & 294 & 284 & 348 & 225 & 238 & 437 & 433 & 221 & 353\\82 & 167 & 228 & 0 & 85 & 127 & 93 & 149 & 312 & 230 & 80 & 144 & 161 & 174 & 233 & 352 & 157 & 289\\167 & 252 & 313 & 85 & 0 & 212 & 178 & 234 & 397 & 315 & 165 & 229 & 246 & 259 & 318 & 437 & 242 & 374\\94 & 89 & 150 & 127 & 212 & 0 & 34 & 65 & 341 & 242 & 134 & 198 & 173 & 186 & 287 & 381 & 169 & 301\\60 & 123 & 184 & 93 & 178 & 34 & 0 & 99 & 307 & 208 & 168 & 232 & 139 & 152 & 321 & 347 & 135 & 267\\159 & 154 & 215 & 149 & 234 & 65 & 99 & 0 & 301 & 307 & 69 & 133 & 168 & 251 & 222 & 341 & 234 & 366\\247 & 332 & 393 & 312 & 397 & 341 & 307 & 301 & 0 & 99 & 232 & 168 & 326 & 339 & 79 & 40 & 172 & 158\\148 & 233 & 294 & 230 & 315 & 242 & 208 & 307 & 99 & 0 & 310 & 267 & 227 & 240 & 178 & 139 & 73 & 59\\162 & 223 & 284 & 80 & 165 & 134 & 168 & 69 & 232 & 310 & 0 & 64 & 99 & 254 & 153 & 272 & 237 & 369\\226 & 287 & 348 & 144 & 229 & 198 & 232 & 133 & 168 & 267 & 64 & 0 & 163 & 318 & 89 & 208 & 301 & 326\\79 & 164 & 225 & 161 & 246 & 173 & 139 & 168 & 326 & 227 & 99 & 163 & 0 & 171 & 252 & 366 & 154 & 286\\92 & 177 & 238 & 174 & 259 & 186 & 152 & 251 & 339 & 240 & 254 & 318 & 171 & 0 & 407 & 379 & 167 & 299\\315 & 376 & 437 & 233 & 318 & 287 & 321 & 222 & 79 & 178 & 153 & 89 & 252 & 407 & 0 & 119 & 251 & 237\\287 & 372 & 433 & 352 & 437 & 381 & 347 & 341 & 40 & 139 & 272 & 208 & 366 & 379 & 119 & 0 & 212 & 198\\75 & 160 & 221 & 157 & 242 & 169 & 135 & 234 & 172 & 73 & 237 & 301 & 154 & 167 & 251 & 212 & 0 & 132\\207 & 292 & 353 & 289 & 374 & 301 & 267 & 366 & 158 & 59 & 369 & 326 & 286 & 299 & 237 & 198 & 132 & 0\end{array}\right]\end{split}$$

Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. Ejecuta manualmente los algorimos de Dijkstra y de Floyd-Warshall en la siguiente gráfica. Para Dijkstra, deberás encontrar el camino más corto entre los vértices marcados con 1 y 7. (poner gráfica)
2. Explica cómo puedes usar el problema de Floyd-Warshall como subrutina para:
    - Encontrar los dos vértices más alejados de una gráfica.
    - Encontrar la distancia promedio en una gráfica.
    - Determinar cuántas parejas de vértices están a distancia ponderada menor a 100.
    - Encontrar el vértice que queda «más cerca de todos», es decir, aquel para el cual la distancia a su vértice más lejano, sea lo más chica posible.
3. Adapta el algoritmo de Floyd-Warshall para que además de almacenar las distancias mínimas, también almacene un camino de peso mínimo entre dos vértices.
4. Tienes una base de datos de canciones tal que para cada canción guarda muchas características: ritmo, tiempo, estilo, autor, letra, etc. ¿Cómo podrías utilizar el algoritmo de Dijkstra para crear una lista de reproducción que te lleve suavemente de una canción dada a otra? ¿Qué gráfica tendrías que construir? ¿Qué pesos le pondrías a las aristas y por qué?
5. Es posible que exista más de un camino de peso mínimo entre dos vértices. Diseña un algoritmo que encuentre el mejor de ellos

en donde el criterio de desempate sea que, además de tener peso mínimo, también tenga la menor cantidad de aristas.

Anterior
Árboles de peso mínimo: algoritmos de Prim y Kruskal

Siguiente
Redes, flujos y flujos máximos

Por Leonardo Ignacio Martínez Sandoval