Algoritmo de Dijkstra

Integrantes del equipo:

1. Porfirio Damián Escamilla Huerta.
2. Omar Porfirio García.
3. Carlos David Zamora Gutiérrez.

Licenciatura en Matemática Algorítmica.

Escuela Superior de Física y Matemáticas.

Grupo: 3AM1.

Teoría de Gráficas.

Profesor: David Fernández Bretón.

El siguiente trabajo presenta una implementación en Python del Algoritmo de Dijkstra para encontrar el camino más corto en una gráfica.

Primero importaremos el módulo `sys` de Python para poder utilizar `sys.maxsize` que es el valor más grande que una variable puede almacenar en Python (depende de la arquitectura), este valor lo usaremos como nuestro ∞.

```
import sys
```

```
sys.maxsize
```

```
9223372036854775807
```

El programa utiliza una lista de 3-tuplas, donde cada tupla es de la forma:

(extremo, extremo , peso)

y nos sirve para representar las aristas de la gráfica:

```
aristas = [
    ("Buenavista","Guerrero",6),
    ("Guerrero","Garibaldi",7),
    ("Garibaldi","Lagunilla",6),
    ("Guerrero","Hidalgo",8),
    ("Garibaldi","Bellas Artes",11),
    ("San Cosme","Revolucion",6),
    ("Revolucion","Hidalgo",5),
    ("Hidalgo","Bellas Artes",9),
    ("Bellas Artes","Allende",10),
    ("Allende","Zocalo",5),
    ("Hidalgo","Juarez",4),
    ("Juarez","Balderas",14),
    ("Bellas Artes","San Juan de Letran",5),
    ("San Juan de Letran","Salto del agua",6),
    ("Cuauhtemoc","Balderas",13),
```

```
    ("Balderas","Salto del agua",9),
]
```

En la siguiente parte del código lo que hacemos es crear una lista de los vértices a partir de la lista de aristas. Después inicializamos la matriz `M`, que será nuestra **matriz de adyacencia** como una matriz de ceros.

```
vertices = []
# Agregamos todos los vértices:
for a in aristas:
  vertices.append(a[0])
  vertices.append(a[1])

# Eliminamos los vértices repetidos:
vertices = list(dict.fromkeys(vertices))

# Inicializamos la matriz de adyacencia en ceros
M =[ [0 for v in vertices] for v in vertices]
```

```
vertices
```

```
['Buenavista',
 'Guerrero',
 'Garibaldi',
 'Lagunilla',
 'Hidalgo',
 'Bellas Artes',
 'San Cosme',
 'Revolucion',
 'Allende',
 'Zocalo',
 'Juarez',
 'Balderas',
 'San Juan de Letran',
 'Salto del agua',
 'Cuauhtemoc']
```

```
M
```

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Con nuestra matriz `M` inicializada en ceros y las listas de aristas y vértices, ahora podemos crear nuestra matriz de adyacencia:

```
for a in aristas:
    i1 = vertices.index(a[0])
    i2 = vertices.index(a[1])
    M[i1][i2] = a[2]
    M[i2][i1] = a[2]
```

```
M
```

```
[[0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [6, 0, 7, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 7, 0, 6, 0, 11, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 8, 0, 0, 0, 9, 0, 5, 0, 0, 4, 0, 0, 0, 0],
 [0, 0, 11, 0, 9, 0, 0, 0, 10, 0, 0, 0, 5, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 5, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 10, 0, 0, 0, 5, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 14, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 0, 0, 9, 13],
 [0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 6, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 6, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0]]
```

Ahora creamos la función

`dijktra(idx_origen, idx_destino)`

que recibe como parámetros:

- `idx_origen`: índice del vértice de origen respecto a la lista `aristas`.
- `idx_destino`: índice del vértice de destino respecto a la lista `aristas`.

Dentro de la función , lo primero que hacemos es crear las listas `marcados` y `no_marcados` que representan los vértices ya marcados y los que no respectivamente.

También creamos la lista `info_vertices`, que contiene un diccionario por cada vértice, ese diccionario nos guarda la siguiente información:

- El nombre del vértice.

- El camino desde el vértice de origen (se inicializa con el propio vértice de origen).
- El tiempo (se inicializa con `sys.maxsize`, nuestro ∞).

Después agregamos el vértice de origen a marcados y lo quitamos de los no marcados.

Luego creamos un bucle que se ejecuta mientras el vértice de destino no se encuentre en la lista de marcados.

Dentro del bucle tomamos el vértice actual como el último de marcados. Luego vemos en su matriz de adyacencia la información de los vértices vecinos. Calculamos el tiempo desde el vértice de origen a cada vecino pasando por el vértice actual y si es menor que el tiempo que ya correspondía a ese vértice, actualizamos la información.

Después revisamos la información de los vértices no marcados y elegimos el de menor tiempo para agregarlo a marcados y quitarlo de los no marcados.

Una vez terminado el bucle, imprimimos el camino más corto.

```
def dijkstra(idx_origen, idx_destino):

  marcados = []
  no_marcados = [v for v in vertices]
  info_vertices=[ {"v": v ,"camino":[vertices[idx_origen]], "tiempo":sys.maxsize} for v in v

  info_vertices[idx_origen]["tiempo"] = 0
  marcados.append(vertices[idx_origen])
  del no_marcados[vertices.index(marcados[-1])]

  while vertices[idx_destino] not in marcados:
    v_act = marcados[-1]
    idx_act = vertices.index(v_act)

    for j in M[idx_act]:
      if j != 0:
        t = info_vertices[idx_act]["tiempo"] + j
        pos = M[idx_act].index(j)
        if info_vertices[pos]["tiempo"] > t:
          info_vertices[pos]["tiempo"] = t
          info_vertices[pos]["camino"] = info_vertices[idx_act]["camino"].copy()
          info_vertices[pos]["camino"].append(vertices[pos])

    minimo = sys.maxsize

    for nm in no_marcados:
      idx_nuevo = vertices.index(nm)
      t = info_vertices[idx_nuevo]["tiempo"]
```

```
      if t < minimo:
        minimo = t
        v_act = nm
    marcados.append(v_act)
    no_marcados.remove(v_act)

  print(f'El camino más corto es: {info_vertices[idx_destino]["camino"]}')
```

Pedimos al usuario que elija el vértice de origen y el vértice de destino.

```
i = 1
for v in vertices:
  print(f"{i}. {v}")
  i+=1


while True:
  entrada = input("Elige el origen:")
  if entrada.isdigit():
    o = int(entrada) - 1
    if 0 <= o < len(vertices):
      break

i = 1
for v in vertices:
  print(f"{i}. {v}")
  i+=1

while True:
  entrada = input("Elige el destino:")
  if entrada.isdigit():
    d = int(entrada) - 1
    if 0 <= d < len(vertices):
      break



dijkstra(o,d)
```

```
1. Buenavista
2. Guerrero
3. Garibaldi
4. Lagunilla
5. Hidalgo
6. Bellas Artes
7. San Cosme
8. Revolucion
```

```
9. Allende
10. Zocalo
11. Juarez
12. Balderas
13. San Juan de Letran
14. Salto del agua
15. Cuauhtemoc
1. Buenavista
2. Guerrero
3. Garibaldi
4. Lagunilla
5. Hidalgo
6. Bellas Artes
7. San Cosme
8. Revolucion
9. Allende
10. Zocalo
11. Juarez
12. Balderas
13. San Juan de Letran
14. Salto del agua
15. Cuauhtemoc
El camino más corto es: ['Lagunilla', 'Garibaldi', 'Guerrero', 'Hidalgo', 'Revolucion']
```