FACULTAD DE INGENIERIA, ARQUITECTURA Y DISEÑO

Profesor Leonardo Esqueda

PROYECTO FINAL
ESTRUCTURA DE DATOS 2
CAMINO MAS CORTO

- **Yanelis Ayala** **4-754-1723**
- **Zulendis De Gracia** **9-745-165**
- **Rafael Galastica** **3-732-2031**
- **Julio R. Gatica** **PE-10-585**
- **Alfredo Guerra** **13-9603-783**
- **Víctor Hidalgo** **8-772-2136**
- **Diego Martínez** **8-878-1209**
- **Ariel Ortega** **8-808-2210**
- **Edsel Rangel** **8-908-1302**
- **Carlos Rivera** **8-759-2457**
- **Edgar Rojas** **AO-8924-08**

Grupo 1

Panamá, República de Panamá

2019

Índice General

INTRODUCCIÓN

El siguiente proyecto tiene como objetivo de analizar todas las posibles rutas de un grafo cuyo objetivo sea la ruta más corta. El algoritmo que utilizamos para obtener la ruta más corta es el algoritmo de Bellman-Ford (algoritmo de Bell-End-Ford) genera el camino más corto en un grafo no dirigido en el que el peso de alguna de las aristas puede ser negativo. El algoritmo de Dijkstra resuelve este mismo problema en un tiempo menor, pero requiere que los pesos de las aristas no sean negativos, salvo que el grafo sea dirigido y sin ciclos. Por lo que el Algoritmo Bellman-Ford normalmente se utiliza cuando hay aristas con peso negativo. Este algoritmo fue desarrollado por Richard Bellman, Samuel End y Lester Ford.

MATRIZ 1

La matriz de incidencia

CORUÑA - ALBACETE													
	CORUÑA	VIGO	VALLADOLID	BILBAO	ZARAGOZA	BARCELONA	MADRID	VALENCIA	JAEN	SEVILLA	GRANADA	MURCIA	ALBACETE
CORUÑA	0	1	1	0	0	0	0	0	0	0	0	0	0
VIGO	1	0	1	0	0	0	0	0	0	0	0	0	0
VALLADOLID	1	1	0	1	0	0	1	0	0	0	0	0	0
BILBAO	0	0	1	0	1	0	1	0	0	0	0	0	0
ZARAGOZA	0	0	0	1	0	1	1	0	0	0	0	0	0
BARCELONA	0	0	0	0	1	0	0	1	0	0	0	0	0
MADRID	0	0	1	1	1	0	0	0	1	0	0	0	1
VALENCIA	0	0	0	0	0	1	0	0	0	0	0	1	1
JAEN	0	0	0	0	0	0	1	0	0	1	1	0	0
SEVILLA	0	0	0	0	0	0	0	0	1	0	1	0	0
GRANADA	0	0	0	0	0	0	0	0	1	1	0	1	0
MURCIA	0	0	0	0	0	0	0	1	0	0	1	0	1
ALBACETE	0	0	0	0	0	0	1	1	0	0	0	1	0

La matriz de incidencia es una matriz binaria (sus elementos sólo pueden ser unos o ceros) que se utiliza como una forma de representar relaciones binarias.

MATRIZ 2

La matriz de Adyacencia

CORUÑA - ALBACETE CON VALORES													
	CORUÑA	VIGO	VALLADOLID	BILBAO	ZARAGOZA	BARCELONA	MADRID	VALENCIA	JAEN	SEVILLA	GRANADA	MURCIA	ALBACETE
CORUÑA	0	171	455	0	0	0	0	0	0	0	0	0	0
VIGO	171	0	356	0	0	0	0	0	0	0	0	0	0
VALLADOLID	455	356	0	280	0	0	395	0	0	0	0	0	0
BILBAO	0	0	280	0	324	0	395	0	0	0	0	0	0
ZARAGOZA	0	0	0	324	0	296	325	0	0	0	0	0	0
BARCELONA	0	0	0	0	296	0	0	349	0	0	0	0	0
MADRID	0	0	193	395	325	0	0	0	335	0	0	0	251
VALENCIA	0	0	0	0	0	349	0	0	0	0	0	241	191
JAEN	0	0	0	0	0	0	335	0	0	242	99	0	0
SEVILLA	0	0	0	0	0	0	0	0	242	0	256	0	0
GRANADA	0	0	0	0	0	0	0	0	99	256	0	278	0
MURCIA	0	0	0	0	0	0	0	241	0	0	278	0	150
ALBACETE	0	0	0	0	0	0	251	191	0	0	0	150	0

La **matriz de adyacencia** es una matriz cuadrada que se utiliza como una forma de representar relaciones binarias. Como se trata de un grafo no dirigido, la matriz obtenida es simétrica:

CODIGO PYTHON:

LA RUTA MAS CORTA

```python
import math
# importamos la librería math
# Cada elemento de este diccionario contiene una posición del camino, y cómo
# valor tiene una lista con el calculo del camino más corto, y el origen del
# mismo
valores = {
    "coruña": [math.inf, ""],
    "vigo": [math.inf, ""],
    "valladolid": [math.inf, ""],
    "oviedo": [math.inf, ""],
    "bilbao": [math.inf, ""],
    "madrid": [math.inf, ""],
    "zaragoza": [math.inf, ""],
    "gerona": [math.inf, ""],
    "barcelona": [math.inf, ""],
    "badajoz": [math.inf, ""],
    "albacete": [math.inf, ""],
    "valencia": [math.inf, ""],
    "murcia": [math.inf, ""],
    "jaen": [math.inf, ""],
    "sevilla": [math.inf, ""],
    "granada": [math.inf, ""],
    "cadiz": [math.inf, ""]
}

# aquí establecemos cada uno de los caminos en una sola dirección y el valor
# que tiene cada camino
caminos = [
    ["coruña", "vigo", 171],
    ["coruña", "valladolid", 455],
    ["vigo", "valladolid", 356],
    ["valladolid", "bilbao", 280],
    ["valladolid", "madrid", 193],
    ["oviedo", "bilbao", 304],
    ["bilbao", "madrid", 395],
    ["bilbao", "zaragoza", 324],
    ["madrid", "badajoz", 403],
    ["madrid", "jaen", 335],
    ["madrid", "albacete", 251],
    ["madrid", "zaragoza", 325],
    ["zaragoza", "barcelona", 296],
    ["gerona", "barcelona", 100],
    ["barcelona", "valencia", 349],
    ["valencia", "albacete", 191],
    ["valencia", "murcia", 241],
    ["albacete", "murcia", 150],
    ["jaen", "sevilla", 242],
    ["jaen", "granada", 99],
    ["sevilla", "cadiz", 125],
    ["sevilla", "granada", 256],
]


def setValores(origen,destino,valor):
    """
```

```
    Función que actualiza el valor del diccionario valores, actualizando
    el valor al más bajo e indicando de que punto viene el camino más corto
    Tiene que recibir:
        origen -> punto inicial
        destino -> punto final
        valor -> valor de ese tramo + el valor que tiene el origen
    Devuelve True o False, dependiendo si ha disminuido el valor entre dos puntos
    """
    if valor < valores[destino][0]:
        # guardamos el nuevo valor mas bajo
        valores[destino][0] = valor

        # guardamos de donde viene el valor mas bajo
        valores[destino][1] = origen
        return True
    return False


# definimos el inicio y el destino
inicio = "coruña"
final = "madrid"

valores[inicio][0] = 0

# realizamos un bucle hasta que no haya ningun otro cambio de valores
while True:
    cancel = True

    # recorremos cada uno de los caminos
    for i in caminos:

        # enviamos los datos del camino
        if setValores(i[0], i[1], valores[i[0]][0] + i[2]):
            cancel = False

        # enviamos los datos del camino de forma invertida
        if setValores(i[1], i[0], valores[i[1]][0] + i[2]):
            cancel = False

    # finalizamos el bucle cuando ya no hay ningun cambio en los valores
    if cancel:
        break

# iniciamos la busqueda del camino mas corto
camino = [final]

while True:
    if camino[-1] == inicio:
        break
    camino.append(valores[camino[-1]][1])

print("El camino mas corto desde el punto '{}' y el punto '{}' es:
{}".format(inicio, final, camino[::-1]))
```