Práctica 4 de Algorítmica
(Algoritmos voraces)

Ordenamiento de los vértices de un grafo

Curso 2017-2018

Ordenamiento de los vértices de un grafo

En esta práctica de **una sola sesión** vamos a resolver, de manera aproximada, un problema que abordaremos nuevamente, esta vez de manera exacta, en la prática 5.

Dado un grafo dirigido $G = (V, E)$ con $N = |V|$ vértices, se dice que un orden definido sobre sus vértices $v_1, v_2, \ldots, v_N$ es un **orden topológico** si no existe ninguna arista (v_i, v_j) con $j < i$. Dicho de otra manera: si dibujamos el grafo situando los vértices ordenadamente de izquierda a derecha no aparece ninguna arista hacia atrás.

Como sabrás, un grafo con ciclos no admite orden topológico. Nos interesa generalizar el concepto de orden topológico para grafos dirigidos **que puedan tener ciclos**. Ya puestos, asumiremos también que se trata de **grafos ponderados**, es decir, que las aristas tienen pesos.

Ya que en general no podemos prohibir la aparición de aristas hacia atrás, simplemente las vamos a penalizar.

Descripción del problema

Queremos encontrar un ordenamiento de los vértices del grafo que maximice la suma de las aristas que van hacia la derecha menos la suma de las aristas que van hacia la izquierda.

Ordenamiento de los vértices de un grafo

Podemos representar el grafo mediante una **matriz de adyacencia** que representaremos en Python 3 como una matriz de enteros de la biblioteca `numpy`.

```
import numpy as np
```

El siguiente código crea una instancia aleatoria para un grafo con `N` vértices:

```
def create_graph(N,maxvalue=1000):
  G = np.random.randint(maxvalue, size=(N, N))
  for i in range(N):
    G[i][i] = 0
  return G
```

Donde la función `np.random.randint` nos devuelve, en este caso, una matriz de valores enteros alteatorios, aunque luego ponemos la diagonal la cero porque suponemos que los vértices no tienen bucles.

Por otra parte, el siguiente código genera una permutación aleatoria de los vértices (numerados entre `0` y `N-1`):

```
def generate_random_ordering(G):
  # G is a square numpy matrix of integers
  N = G.shape[0] # G.shape would return something like (10,10)
  return np.random.permutation(N)
```

Ordenamiento de los vértices de un grafo

Ejemplo:

```
>>> G = create_graph(5)
>>> G
array([[  0, 626, 112, 809, 481],
       [  7,   0, 342,  64, 624],
       [286, 183,   0, 183, 609],
       [763, 311, 107,   0, 425],
       [415, 363, 259, 844,   0]])
>>> generate_random_ordering(G)
array([4, 3, 1, 2, 0])
```

El siguiente código nos permite calcular la función objetivo que nos interesa maximizar:

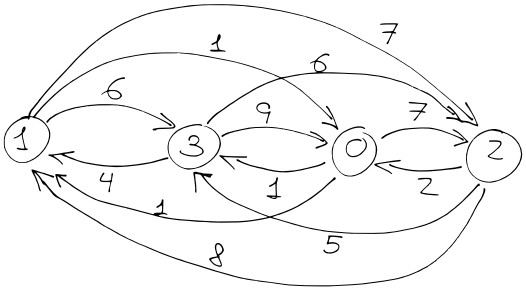
```
def evaluate(G,ordering):
  # assume that G.shape is of type (N,N) and ordering.shape is of type
  # (N) and is a permutation of values 0,...,N-1
  N = G.shape[0]
  return sum(G[ordering[i]][ordering[j]]-G[ordering[j]][ordering[i]]
             for i in range(N) for j in range(i+1,N))
```

Ordenamiento de los vértices de un grafo

Probemos ahora un ejemplo algo más pequeño y veamos cómo funciona `evaluate`:

```
>>> G = create_graph(4,10)
>>> G
array([[0, 1, 7, 1],
       [1, 0, 7, 6],
       [2, 8, 0, 5],
       [9, 4, 6, 0]])
>>> ordering = generate_random_ordering(G)
>>> ordering
array([1, 3, 0, 2])
```

Que corresponde al siguiente grafo:



Si lo evaluamos nos dará `(6,1,7,9,6,7) - (4,1,8,1,5,2) = 36 - 21 = 15`

Ordenamiento de los vértices de un grafo

Ejercicio 1 Se pide escribir una función `generate_greedy_ordering` que utilice un algoritmo voraz para obtener un ordenamiento de los vértices del grafo que intente maximizar el resultado de la función `evaluate`. Se sugiere la siguiente estrategia voraz:

Seleccionar los vértices de uno en uno de manera voraz. En cada instante, elegir el vértice que maximice el valor que aporta en esa posición a la función `evalute`. Es decir, que maximice un score formado por los arcos que van entre los vértices ya colocados a él (suman hacia la derecha, restan hacia la izquierda) y los que van de él al resto de vértices no seleccionados (de él hacia ellos sumando, de ellos hacia él restando).

Este algoritmo NO es óptimo

```
G= [[0 1 0 1]
    [0 0 2 2]
    [1 0 0 0]
    [0 0 1 0]]
```

La estrategia voraz da `[1 3 2 0]` de valor `(2,2,0,1,0,1) - (0,0,1,0,1,0) = 6 - 2 = 4`, mientras que la solución óptima podría ser `[0, 1, 3, 2]` con valor `(1,1,0,2,2,1) - (0,0,1,0,0,0) = 7 - 1 = 6`. La traza del algoritmo voraz:

```
fijados=[]        elijo el máximo de [(1, 0), (3, 1), (-2, 2), (-2, 3)]
fijados=[1]       elijo el máximo de [(-1, 0), (2, 2), (2, 3)]
fijados=[1, 3]    elijo el máximo de [(-3, 0), (4, 2)]
fijados=[1, 3, 2] elijo el máximo de [(-1, 0)]
```

Ordenamiento de los vértices de un grafo

Ejercicio 2 Haz una función que manipule un grafo `G` tomando pares de aristas opuestas de manera que a cada uno de esos pares se les reste el menor de los dos pesos. Por ejemplo, si `G[1][2]` vale `100` y `G[2][1]` vale `30`, tras procesar el grafo tendremos que `G[1][2]` valdrá `70` y `G[2][1]` valdrá `0`.

¿Tiene algún efecto esta operación sobre el resultado de `evaluate`? Compruébalo experimentalmente.

Ejercicio 3 (opcional) Si no has implementado el ejercicio 1 de manera que no sea necesario recalcular de cero el score de todos los vértices, se pide en este ejercicio que lo hagas de manera incremental. Para ello, basta con tener asociado a cada vértice la suma de las aristas que van desde los vértices ya elegidos por el algoritmo voraz y los que van a vértices por seleccionar. Cada vez que selecciones un vértice has de actualizar los scores de los vértices todavía no seleccionados.