Arborescencia de Mínimo Peso. Algoritmo de Edmonds.

Optimización en Redes.

Laura Henríquez Cazorla
25/02/2024

El algoritmo de Edmonds está compuesto de tres pasos principalmente: reducción, contracción y expansión. El programa está compuesto por cinco funciones en las que estos pasos son el foco del mismo. Estas son:

def leer_documento(grafos_doc)

- Input:
  grafos_doc: archivo tipo texto, desde el que leeremos las aristas y sus pesos del grafo.

- Objetivo:
  Leer los datos y obtener el grado del mismo. Llamar a la función arborescencia, que nos dará mediante el método de Edmonds la arborescencia de mínimo peso.

- Output:
  No devuelve nada, los datos obtenidos los imprimirá arborescencia.

def reduccion(G,raiz)

- Input:
  G: Digrafo (del tipo nx.DiGraph()).
  Raíz: raíz de la arborescencia.

- Objetivo:
  Reducir la arborescencia, es decir, tomar la arista de mínimo peso que incide en cada nodo, menos en la raíz.

- Output:
  R: Digrafo reducido (del tipo nx.DiGraph()).

def contraccion(G, G_reducido,e):

- Input:
  G: Digrafo (del tipo nx.DiGraph()).
  G_reducido: digrafo reducido (del tipo nx.DiGraph()), obtenido de la función reduccion.

- Objetivo:
  Contraer el digrafo reducido. Para ello, tomamos uno de los ciclos que tiene el dígrafo reducido, y este pasa a ser un nodo. A continuación, actualizaremos las aristas y sus pesos (en el caso de salir del ciclo se mantiene el peso, y si entra, se restará esta a la de mínimo peso que incidía en este nodo). Por último, va almacenando en una lista (e), aquellas aristas en las que al menos uno de los dos nodos es un ciclo, con sus nuevos pesos y el nodo del ciclo que inicialmente pertenecía a la arista.

- Output:
  R: Digrafo contraído (del tipo nx.DiGraph()).

e: lista con sublistas, donde se almacenan aquellas aristas en las que al menos uno de los dos nodos es un ciclo: [nodo1, nodo2, nodo del que entra/sale, peso actualizado]

def expandir (G,G_reducido, ciclos, e)

- Input:
  G: Digrafo inicial (del tipo nx.DiGraph()).
  G_reducido: Digrafo reducido (del tipo nx.DiGraph()).
  ciclos: lista con los ciclos que inicialmente había en el primer grafo reducido.
  e: lista que va almacenando los datos de las aristas actualizadas (es la obtenida con la función de contracción).

- Objetivo:
  Expandir el dígrafo reducido, y así obtener la arborescencia de mínimo peso. Para ello, iremos usaremos el vector e, cuando en el nodo final de una arista aparezca algún ciclo. En el caso de no haber ningún ciclo en esa arista, simplemente se añade al grafo resultado.

- Output:
  R: Digrafo expandido (del tipo nx.DiGraph()).

def arborescencia (G,raiz)

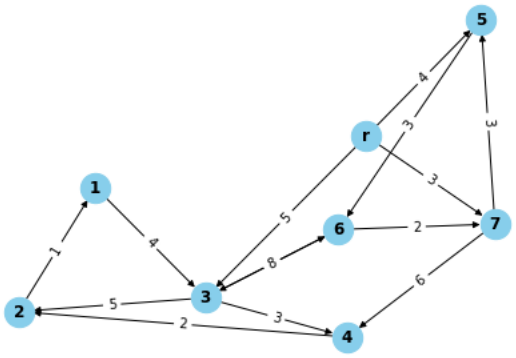
- Input:
  G: Digrafo (del tipo nx.DiGraph()).
  Raiz: raíz de la arborescencia.

- Objetivo:
  Obtener la arborescencia de mínimo peso mediante el algoritmo de Edmonds. Es la encargada de unir las tres anteriores. Finalmente, dibuja el digrafo.

- Output:
  resultado_final: arborescencia buscada (aristas de un dígrafo del tipo tipo nx.DiGraph()).
  peso: entero con el peso de la arborescencia final.

Para el caso del ejercicio 9, el archivo grafos.txt adjuntado, es el correspondiente al mismo. Veamos qué es lo que va ocurriendo.

Digrafo inicial: [('r', '3', 5), ('r', '5', 4), ('r', '7', 3), ('3', '2', 5), ('3', '4', 3), ('3', '6', 6), ('5', '6', 3), ('7', '5', 3), ('7', '4', 6), ('1', '3', 4), ('2', '1', 1), ('4', '2', 2), ('6', '3', 8), ('6', '7', 2)]

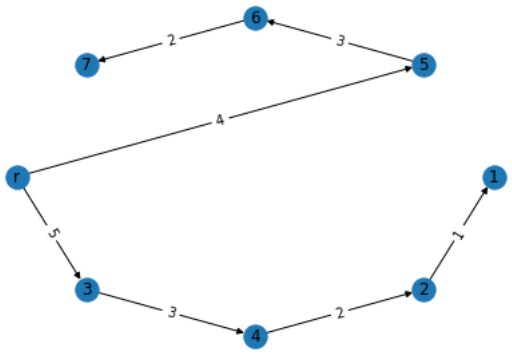


Tras reducirlo nos queda: [('3', '4'), ('5', '6'), ('7', '5'), ('1', '3'), ('2', '1'), ('4', '2'), ('6', '7')]

Como existen dos ciclos aplicamos la contracción (bucles (5,6,7) y (1,2,3,4)).
Tras aplicar la contracción: [('r', '3'), ('r', ('7', '5', '6')), ('3', '2'), ('3', '4'), ('3', ('7', '5', '6')), (('7', '5', '6'), '4'), (('7', '5', '6'), '3'), ('2', '1'), ('4', '2'), ('1', '3')]
Lo reducimos: [('r', ('7', '5', '6')), ('3', '4'), ('2', '1'), ('4', '2'), ('1', '3')]

Sigue existiendo un ciclo, así que volvemos a aplicar la contracción:
Contracción: [('r', ('2', '1', '3', '4')), ('r', ('7', '5', '6')), (('2', '1', '3', '4'), ('7', '5', '6')), (('7', '5', '6'), ('2', '1', '3', '4'))]
Reducción: [('r', ('2', '1', '3', '4')), ('r', ('7', '5', '6'))]

Ya no quedan más ciclos, por lo que únicamente falta expandirlo:
La arborescencia buscada es: [('r', '3'), ('r', '5'), ('3', '4'), ('4', '2'), ('2', '1'), ('5', '6'), ('6', '7')]
El peso es: 20



Que es la arborescencia de mínimo peso obtenida por el algoritmo de Edmonds

Otra manera de obtener la arborescencia de mínimo peso mediante el algoritmo de Edmonds es usando clases ya definidas en NetworkX:

```
def edmonds_algorithm(G, raiz):
    #llamamos a la clase Edmonds
    arborescencia = nx.algorithms.tree.branchings.Edmonds(G, raiz)
    #dentro de la clase de Edmonds, llamamos a aquella funcion que nos calcula el óptimo
    #(tomando las aristas)
    optimo=arborescencia.find_optimum(attr="weight",
        default=1,
        kind="min",
        style="arborescence",
        preserve_attrs=False,
        partition=None,
        seed=None).edges
    #calculamos el peso de la arborescencia
    peso_total = 0
    for u, v, peso in optimo(data='weight', default=1):
        peso_total += peso
    return optimo,peso_total
```