Universidad de Costa Rica

Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0217 – Estructuras de Datos Abstractas y Algoritmos para Ingeniería
II ciclo 2020

Algoritmo de Borůvka

Leonardo Gabriel Alfaro Patiño - B60224
Profesor: Juan Carlos Coto Ulate
Grupo 01

14 de diciembre del 2020

Índice

Índice de figuras

Índice de tablas

1. Introducción

Otakar Borůvka (nacido un 10 de mayo de 1899 en Uherský Ostroh - muerte 22 de julio 1995 en Brno) fue un checo matemático hoy más conocido por su trabajo en la teoría de grafos, mucho antes de que se tratara de una disciplina matemática establecida.

Figura 1: Otakar Borůvka

Creado por dicho ingeniero electricista ruso Otakar Boruvka quien formuló dicho algoritmo bastante eficiente cuando enfrentó el trabajo de la electrificación del sureste de Moravia en 1928. [1]

Cuando se tiene un grafo conexo pesado, es de mucha utilidad determinar cuál es el árbol generador de mínimo peso, puesto que con él queda determinada la forma de intercomunicar todos los nodos del grafo con el mínimo costo o la mínima distancia, etc. Una aplicación típica ocurre en el diseño de redes de comunicación donde los nodos representan ciudades y las aristas son las posibles líneas de comunicación entre las ciudades. El valor asociado a cada arco representa el costo de seleccionar esa línea para la red. [1]

Este algoritmo, como los de Kruskal o Prim, trata de encontrar caminos mínimos o máximos en un grafo con pesos, buscando su árbol minimal o maximal, respectivamente, es decir, el camino mas corto o el mas largo. [2]

Su aplicabilidad tiene una amplia gama de casos: construcción de carreteras con paso obligado por ciertos puntos y con un coste mínimo o con una distancia mínima; instalación de una red de distribución de productos a través de distintos puntos y con coste mínimo; organización de rutas turísticas con distancias mínimas, etcétera. [2]

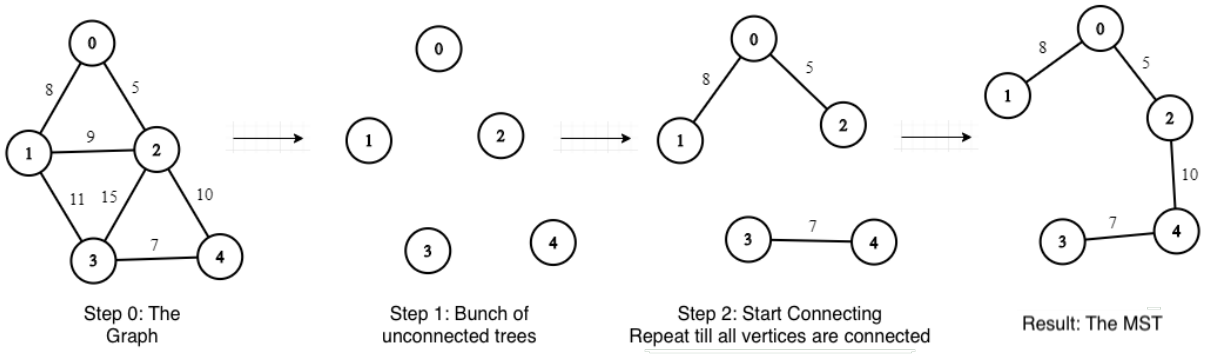

Figura 2: Algoritmo de Boruvka

2. Discusión

2.1. Vistazo general o abstracto del algoritmo

Para poder continuar con el desarrollo del trabajo es necesario declarar y conocer a perpetualidad el significado de algunos términos, además de la descripción básica de un grafo. Del PDF del curso se tiene la siguiente definición de grafo, *Un grafo es un conjunto de nodos asociados entre si mediante arcos*, los cuales pueden tener o no tener **peso** y pueden o no tener **dirección**. Un grafo con pesos se denomina **pesado** y un grafo con dirección se denomina **dirigido**. DE manera generalizado los grafos permiten arcos reflexivos es decir, arcos cuyo nodo fuente y blanco sean el mismo. Sin embargo, en la practica, los arcos reflexivos tienden a no ser de mucha utilidad y por lo tanto no se declaran explícitamente.

Un grafo es definido matemáticamente de la siguiente manera: Un grafo **G** es un par ordenado de un juego **V** de vértices y un juego de **A** aristas.

$$G = (V, A) \tag{1}$$

En donde el orden importa ya que es un par ordenado.

$$(a, b) \neq (b, a) \tag{2}$$

Lo anterior solo es cierto si $a = b$

En este algoritmo de Boruvka-Sollin se hace uso de una matriz de incidencia especial **M** del grafo cuyos elementos m_{ij} son los peso respectivos p_{ij} de cada arco v_i a v_j. En primer lugar describiéremos el algoritmo para encontrar el árbol minimal. [2]

Plantear la matriz M de incidencia del grafo dado.

1. Seleccionar el elemento mínimo de **M**. Si existen varios iguales, seleccionar cualquiera de ellos p_{ij}. Tomar nota de su fila i y de su columna j. El primer lado del árbol sera ij.

2. Eliminar de **M** las columnas asociadas con i y con j.

3. Marcar las filas i, j.

4. Seleccionar el elemento mínimo p_{ij} de todas las filas marcadas son tener en cuenta los elementos de las columnas ya suprimidas. Añadir al árbol el nuevo lado ij.

5. Si aun hay columnas **M** para eliminar, volver al paso 3. En caso contrario, terminar.

Este es el algoritmo básico de Boruvka, mas sin embargo esto es una implementación mas a un nivel matemático ya que Boruvka era un matemático y no fue hasta muchos años después que un científico en computación (Sollin), dio uso a este en dicha área científica.

El algoritmo aplico a un lenguaje de alto nivel tiene básicamente 3 pasos:

1. Primer paso es crear al grafo

2. Seleccionar un grupo de arboles sin conectar (numero de arboles = numero de vértices)

3. Mientras hayan arboles sin conectar, para cada árbol sin conectar se debe: encontrar su arista con menor peso y añadir este arista para conectar otro árbol

2.2. Complejidad del algoritmo

Con el Algoritmo de Boruvka se puede obtener una Complejidad de $O(logV)$ iteraciones en el bucle externo antes de terminar, y por lo tanto su Complejidad temporal es $O(ElogV)$, donde E es el número de arcos, y V es el número de vértices de G. En Grafos planos puede implementarse para ejecutarse en tiempo lineal, eliminando los arcos de menor peso entre cada par de nodos después de cada etapa del algoritmo. [1]

2.3. Implementación del algoritmo en Python

Se procede a realizar la implementación del código en Python, consta esta implementación de una serie de pasos que deben realizarse cuantas veces sea necesario, hasta encontrar el MST (Minimum spanning tree). Cabe recalcar que datos o información sobre el código fueron obtenidos de paginas como Stackoverflow, así que se agradece a dichos autores por la ayuda ya que implementar el algoritmo de raíz sin conocimiento previo es complicado y se sale un poco del enfoque del proyecto. Se introduce el siguiente grafo a continuación en el código, primeramente se explica la solución abstracta y seguidamente se indica la solución en código.

2.3.1. Solución con pseudo-código o gráfica

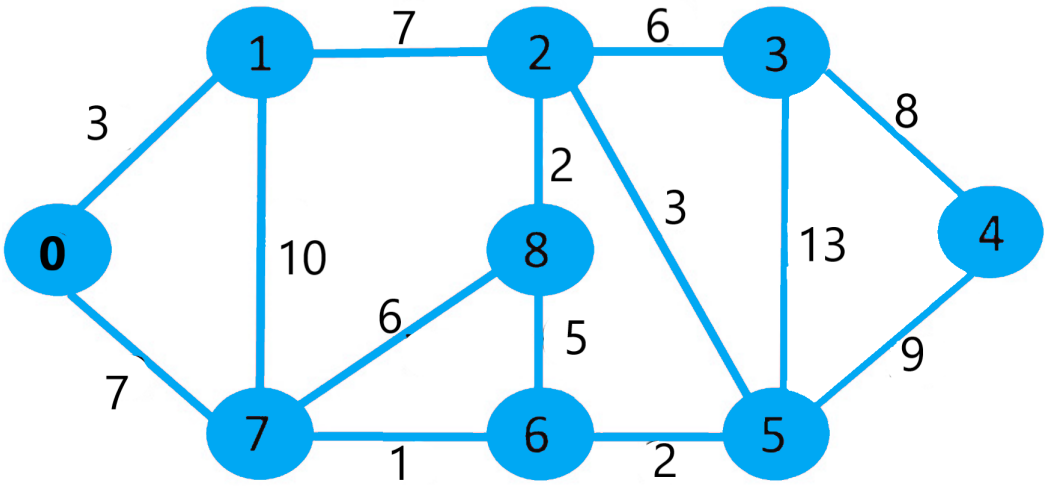

Figura 3: Paso 1 creación del grafo

En el primer paso se crea el grafo que se en el cual se quiere implementar dicho algoritmo, se introducen los nodos y los arcos, cada uno con su respectivo peso.

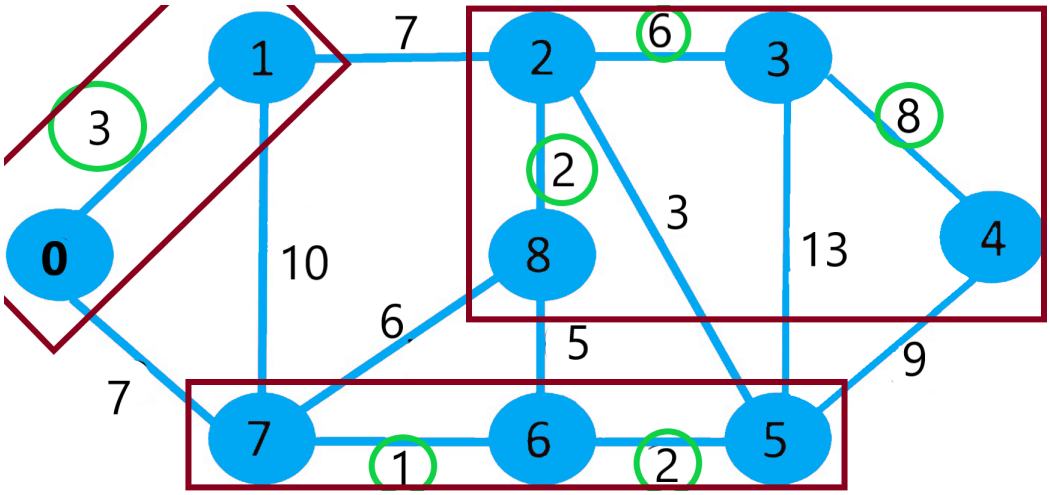

Figura 4: Paso 2, para cada nodo se encuentra el arco menos pesado que lo conecte con otro nodo

Por ejemplo en este caso para el nodo 0, seria el arco 0-1 o para el nodo 7 seria el arco 7-6, este proceso se repite para todos los nodos encontrando el arco de menor peso que lo conecte con otro nodo.

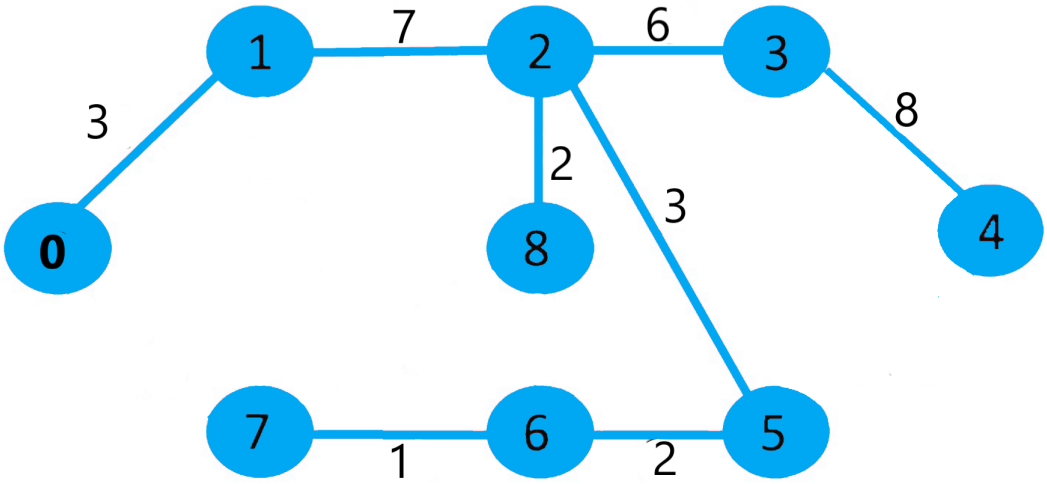

Figura 5: Grafo finalizado con el árbol de menor extensión encontrado

2.3.2. Solución con Python

Se presentara a continuación fragmentos del código para explicar su funcionamiento, de igual manera se adjunta todo el código en el repositorio donde se encuentra el trabajo.

```
#|*********************************************************************;
#* Project            : Proyecto Final, Estructuras de Datos Abstractas y
    Algoritmos para Ingenieria
#*
#* Program name       : Algoritmo de Boruvka para encontrar el MST en un
    grafo pesado
#*
#* Author             : Leonardo Alfaro
#*
#* Date created       : 13/12/2020
#*
#* Purpose            : To implement Boruvka's algorithm in Python
#*
#*********************************************************************;

# Needed for constructing the graph
from collections import defaultdict

# We define the class Graph for creating such graph
# Creates the Graph
class Graph:
    def union(self, parent, rank, x, y): # Function to unify different nodes
    one we find a MST
        xroot = self.find(parent, x) # next x root is defined by parent and
   next x
        yroot = self.find(parent, y) # same as x root
        #
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
```

```
26          elif rank[xroot] > rank[yroot]:
27              parent[yroot] = xroot
28          else:
29              parent[yroot] = xroot
30              rank[xroot] += 1
31      # Constructor method for the vertices
32      def __init__(self, vertices):
33          self.V = vertices
34          self.graph = []
35      # Method for adding edges, u and v are sides w is weight
36      def addEdge(self, u, v, w):
37          self.graph.append([u, v, w])
38      # Find method for finding is equal to index i, if so return i
39      def find(self, parent, i):
40          if parent[i] == i:
41              return i
42          return self.find(parent, parent[i])
43
44  # Building the MST
45  # Graph after applying Boruvka's algorithm
46      def boruvkaMST(self):
47          parent = [] # Parent vertices
48          rank = [] # Rank is the weight or "peso" of the edge
49          cheapest = [] # Cheapest basically means the least "heavy"
50          numTrees = self.V
51          MSTweight = 0 # MST wieght is zero and we add to it as we progress
52          for node in range(self.V):
53              parent.append(node)
54              rank.append(0)
55              cheapest = [-1] * self.V
56          # We combine components until all components are not combined into a
        single MST
57          while numTrees > 1:
58              for i in range(len(self.graph)):
59                  u,v,w = self.graph[i]
60                  set1 = self.find(parent, u)
61                  set2 = self.find(parent, v)
62
63                  if set1 != set2:
64                      if cheapest[set1] == -1 or cheapest[set1][2] > w:
65                          cheapest[set1] = [u, v, w]
66                      if cheapest[set2] == -1 or cheapest[set2][2] > w:
67                          cheapest[set2] = [u, v, w]
68          # The above picked are the cheapest edges, add them to the MST
69              for node in range(self.V):
70                  if cheapest[node] != -1:
71                      u,v,w = cheapest[node]
72                      set1 = self.find(parent, u)
```

```
                    set2 = self.find(parent, v)

                    if set1 != set2:
                        MSTweight += w
                        self.union(parent, rank, set1, set2)
                        print("Edge %d-%d has weight %d is included in MST"
    % (u,v,w))
                        numTrees = numTrees - 1

            cheapest = [-1] * self.V
        print("Weight of MST is %d" % MSTweight)
# Creating the graph
g = Graph(4)

g.addEdge(0, 1, 11)
g.addEdge(0, 2, 5)
g.addEdge(0, 3, 6)
g.addEdge(1, 3, 10)

g.boruvkaMST()
```

Se obtiene la siguiente salida en la terminal al correr el código:

```
leoalfaro@MBP-de-Leonardo:~/Documents/algoritmos$ python algoritmo_de_boruvka.py
Edge 0-2 has weight 5 is included in MST
Edge 1-3 has weight 10 is included in MST
Edge 0-3 has weight 6 is included in MST
Weight of MST is 21
leoalfaro@MBP-de-Leonardo:~/Documents/algoritmos$
```

Figura 6: Salida obtenida en la terminal al ejecutar el código

3. Conclusiones

1. Un grafo es un conjunto de nodos asociados entre si mediante arcos, los cuales pueden tener o no tener peso y pueden o no tener dirección. U
2. Otakar Boruvka fue un matemático checo que creo el algoritmo que se lleva su nombre.
3. Dicho algoritmo permite encontrar el árbol de menor peso en un grafo pesado.
4. Es usado en un amplio espectro de casos, como: construcción de carreteras, instalación de redes eléctricas, organización de rutas turísticas con distancias mínimas, entre otros.
5. La complejidad del algoritmo es $O(logV)$ iteraciones en el bucle externo antes de terminar, y por lo tanto su Complejidad temporal es $O(ElogV)$, donde E es el número de arcos, y V es el número de vértices de G.

6. Los pasos para realizar el algoritmo son sencillos en naturaleza pero debe ser repetido n iteraciones para lograr el resultado del MST, esto lo hace fácil de implementar en código en lenguajes como Python.

7. A la hora de implementar el código en Python primeramente se debe definir la clase Graph y dentro de esta los métodos encargados del algoritmo, así también como el método constructor, seguidamente se construye el MST con ayuda de dichos métodos.

Referencias

[1] R. M. M. P. Alfredo Caicedo Barrero, Graciela Wagner de García, *Introducción a la teoría de grafos*. Ediciones Elizcom, 2010.

[2] M. García, *Matemática Discreta*. Paraninfo, 2015.