Universidad de Costa Rica

Estructuras de datos abstractas y algoritmos para ingeniería

IE0217

Proyecto de investigación

Algoritmo de Prim

Estudiante:
Jimena González Jiménez (B83443)

11 de diciembre de 2020

Índice

1. Glosario

1. $G(V, E)$: manera de referirse a un grafo no dirigido. En este caso ”V” es para *vertex*, que en inglés significa vértice y se refiere a los nodos. Por otro lado, ”E” es para *edge* que en inglés quiere decir arista o arco.

2. Cabe recalcar que los términos arista y arco, y nodo y vértice son intercambiables cuando se habla de grafos.

3. Un corte de energía eléctrica es cuando el suministro de energía se aisla parcial o totalmente de la red de distribución o de los centros de acopio [1].

4. Algoritmo de Dijkstra: Al igual que con el algoritmo de Prim, se comienza con un grafo pesado no dirigido. Con base en este grafo, se construye un árbol recubridor mínimo entre los nodos [2].

2. Intoducción

Antes de hablar del algoritmo de Prim, es necesario definir algunos conceptos generales. Esto, con la finalidad de entender mejor el algoritmo y sus aplicaciones. Primero, un algoritmo se puede definir como un proceso computacional compuesto por varios pasos que recibe un valor(*input*) y lo transforma en otro (*output*). También se pueden ver como herramientas para resolver problemas computacionales ya definidos [3]. Cuando el problema especifica qué relación debe existir entre el input y el *output*, se debe buscar un algoritmo capaz de satisfacer esto. Por lo tanto, distintos algoritmos se adaptan a distintos programas y lo que pida el problema es lo que define el algoritmo necesario para resolverlo [3].

Ya que se mencionaron aspectos generales de los algoritmos, se hablará de los algoritmos avaros, el algoritmo de Prim es de este tipo, y de árboles recubridores mínimos. Primeramente, un algoritmo avaro es uno que busca gratificación instantánea. Esto quiere decir que buscan la solución más eficiente para cada etapa del proceso. Con la esperanza de que esto lleve a una eficiencia global [4]. Por otro lado, los árboles recubridores mínimos para un grafo no dirigido $G(V, E)$ buscan unir todos los nodos por medio de los arcos de menor peso y por consiguiente reduzca el peso de cada arista [5].

Por último, el algoritmo de Prim es un algoritmo avaro que busca generar un árbol recubridor mínimo mediante la expansión de sub-árboles. Este surgió en el año 1957 [6], justo cuando las ciencias de la computación empezaron a crecer, como una de las múltiples soluciones al problema del árbol recubridor mínimo. Eso sí, Robert Clay Prim no fue el único que presentó una saloución a este problema. Algunos otros científicos que propusieron algoritmos para resolver este problema fueron: Joseph Kruskal y Edsger W. Dijkstra. De hecho, en ocasiones al algoritmo de Prim se le conoce como el algoritmo de Prim-Dijkstra, debido a que los algoritmos funcionan de la misma manera [7]. El nodo inicial se selecciona arbitrariamente de los que componen el árbol inicial. Luego, el árbol se va expandiendo al seleccionar el nodo más cercano que no esté en el árbol. El algoritmo termina una vez que todos los nodos del grafo original estén contenidos en el árbol. Por la manera en la que se construye el árbol, el

número de iteraciones qur toma construirlo está dado por $n - 1$. Donde n corresponde a la cantidad de nodos en el grafo original [5].

2.1. Ejemplo ilustrado

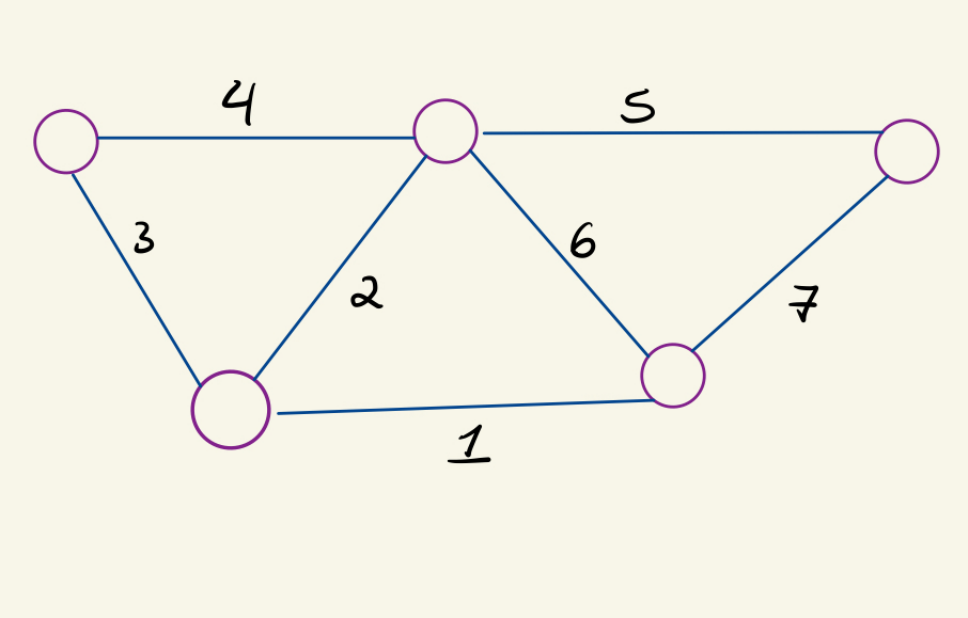

Figura 1: Se empieza con un grafo pesado [8].

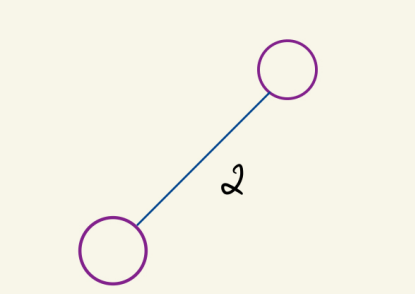

Figura 2: Se selecciona un nodo cualquiera y se busca el arco menos pesado y se añade [8].

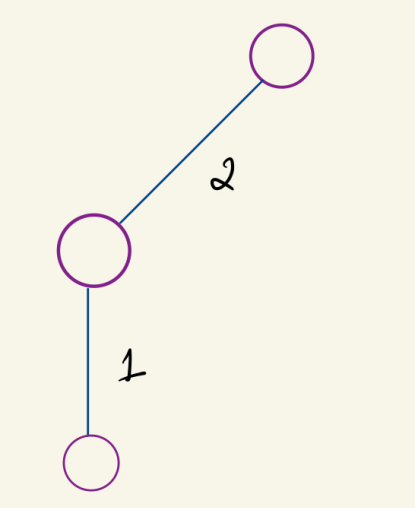

Figura 3: Se repite el proceso con el segundo nodo que se añadió [8].

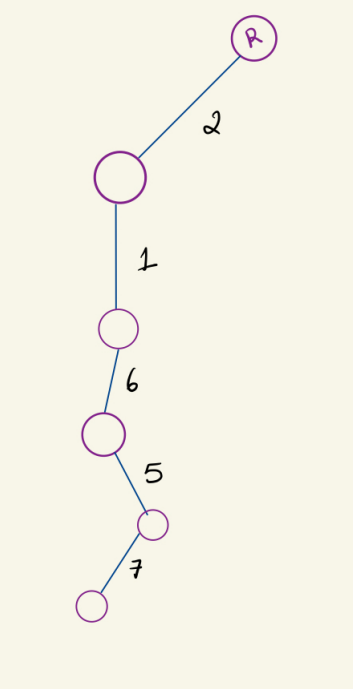

Figura 4: Finalmente, se repite el proceso hasta tener el árbol completo [8].

2.2. Ejemplo en Python

Además del ejemplo ilustrado, se realizó el ejemplo en Python que se muestra a continuación. El código de este ejemplo se basó en uno encontrado en un blog de programación [9].

```
import sys

class Graph():

  def __init__(self, nodos):
    self.N = nodos
    self.grafo = [[0 for column in range(nodos)]
          for row in range(nodos)]

  def printMST(self, parent):
    print ("Arco \tPeso")
    for i in range(1, self.N):
```

```
      print (parent[i], "-", i, "\t", self.grafo[i][ parent[i] ])

  def minKey(self, key, mstSet):

    min = sys.maxsize

    for n in range(self.N):
      if key[n] < min and mstSet[n] == False:
        min = key[n]
        min_index = n

    return min_index

  def primMST(self):

    key = [sys.maxsize] * self.N
    parent = [None] * self.N
    key[0] = 0
    mstSet = [False] * self.N

    parent[0] = -1

    for cout in range(self.N):

      u = self.minKey(key, mstSet)

      mstSet[u] = True

      for n in range(self.N):

        if self.grafo[u][n] > 0 and mstSet[n] == False and key[n] > self.grafo[u][n]:
          key[n] = self.grafo[u][n]
          parent[n] = u

    self.printMST(parent)

g = Graph(5)
g.grafo = [ [3, 6, 5, 9, 1],
      [8, 0, 7, 5, 2],
      [9, 9, 9, 9, 9],
      [2, 9, 1, 0, 2],
      [7, 4, 2, 4, 9]]

```

```
56 g.primMST();
```

Este funciona con matrices cuadradas 5x5. Ya que los grafos utilizados por el algoritmo de Prim se pueden representar como una matriz. Con un generados de matrices en línea, se generaron tres matrices distintas y estas se sustituyeron en la última parte del código. Luego, se volvió a correr y efectivamente cada resultado es distinto, pero se puede observar como el algoritmo de Prim busca el arco menos pesado en cada nodo.

Figura 5: Resultado de la primera vez que se corrió el código.

Figura 6: Resultado de la segunda vez que se corrió el código, luego de cambiar la matriz.

Figura 7: Resultado de correr el mismo código una tercera vez con una matriz distinta.

3. Discusión

En esta sección, se discutirán algunas aplicaciones del algoritmo de Prim. El primero, es su aplicación a la hora de configurar sistemas de energía. Después, se hablará sobre su implementación en la identificación de áreas aisladas en desastres naturales. Por último, se discutirá su aplicación en la distribución del recurso hídrico.

Según TD Sudhakar y el doctor KN Srinivas, es posible rastrear el flujo de energía eléctrica empleando el algoritmo de Prim. Esto vendría a sustituir la normativa aplicada usualmente en estos casos. El propósito de sustituir esta normativa es porque su uso puede causar cortes en cascada. Para evitar estos cortes, se propone hacer este rastreo aplicando el algoritmo de Prim. Para aplicar dicho algoritmo, se asume que cada rama de distribución es equivalente a cada arco, tomando en cuenta su peso. Cabe recalcar que los resultados fueron obtenidos con el sistema de 16 buses del IEEE [1].

El artículo consultado para esta sección plantea que es posible usar un algoritmo basado en el algoritmo de Prim para mapear el factor de riesgo de desastres naturales en comunidades y rutas. Estas se asocian a los nodos, arcos y pesos de los arcos de un árbol recubridor mínimo. Cada área forma un árbol por separado. Si llegara a suceder algo y si algún área no se logra comunicar con el árbol principal a tiempo, se marca como área aislada. En resumen, el árbol formado por el subconjunto de arcos encontrados por el algoritmo no solo incluye los vértices conectados en el grafo, pero la suma de los pesos de todos los arcos [10].

La última aplicación del algoritmo de Prim que se discutirá es para modelar el sistema de tuberías para distribución de agua. Debido al incremento en la población, se ha incrementado la demanda de agua potable, por lo que es necesario mejorar el servicio de distribución de agua. Esto se hace ampliando el sistema de tuberías, lo cual se tiene que hacer de manera óptima ya que esto disminuye el costo de operación. El sistema de distribución se modela como un grafo pesado no dirigido, donde los extremos de las tuberías son los nodos y cada tubo sería un arco, cuyos pesos están definidos por la longitud de los tubos. Asumiendo que todos

los tubos tienen el mismo diámetro y como se mencionó anteriormente, que el peso de cada arco es dado por la longitud del tubo en metros. Para aplicar el algoritmo, se escoge un nodo como raíz y con cada iteración se va cubriendo el área en la que se está trabajando. Cuando terminan las iteraciones, se sabrá la cantidad de tubos necesarias para abastecer el área en cuestión. La cual será optimizada debido a que se está trabajando con un algoritmo avaro [6].

4. Conclusiones

Finalmente, hay que reconocer que el algoritmo de Prim es una herramienta muy útil y no solo en el campo de la ingeniería eléctrica. Porque, como se explicó en la discusión, se puede emplear desde el diseño de redes de distribución de energía eléctrica hasta para mitigación de desastres naturales. Aun así, cabe recalcar que por ser un algoritmo avaro, el algoritmo de Prim no es adecuado para resolver algunos otros problemas.

Referencias

[1] T. Sudhakar and K. Srinivas, “Power system reconfiguration based on prim’s algorithm,” *1st International Conference on Electrical Energy Systems*, vol. 1, pp. 12–20, 2011.

[2] D. EW, “A note on two problems in conexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, Massachusets: Massachusets Institute of Technology, 2009.

[4] M. Soltys, *An Introduction to the Analysis of Algorithms*. Toh Tuk Link, Singapore: World Scientific Publishing Co. Pte. Ltd, 2012.

[5] D.Kapanadevi, “Effective searching path in graph using prim’s algorithm,” *International Journal of Computer Organization Trends*, vol. 3, no. 8, pp. 310–313, 2013.

[6] M. Sinaga, E. Manurung, and S. Manullang, “Prim’s algorithm to model pipe network at the water supply company,” *Journal of Pysics: Conference Series*, vol. 1188, pp. 1–7, 2019.

[7] J. Nešetřil, “A few remarks on the history of mst-problem,” *Archivum Mathematicum*, vol. 33, no. 1, pp. 15–22, 1997.

[8] Programiz, “Prim’s algorithm.”

[9] G. for Geeks, “Prim’s minimum spanning tree.”

[10] W.-C. Wan, M.-C. Hsieh, and C.-H. Huang, “Applying prim’s algorithm to identify isolated areas for natural disaster prevention and protection,” *Scientific Research Publishing*, vol. 10, no. 7, pp. 417–431, 2018.