
# INSTITUTO POLITÉCNICO NACIONAL

## ESCUELA SUPERIOR DE CÓMPUTO

Análisis y Diseño de Algoritmos

## PROYECTO FINAL

Implementación de algoritmo Prim
para resolución de problemas

3CM2

Ochoa Cisneros Juan Carlos

Ortiz Becerril Alan

Recio Pérez Luisa Georgina

Profesora: Elizabeth Moreno Galván

15/01/2023

## Introducción

Para este algoritmo de grafos vamos a considerar una situación que es bastante común, como encontrar el camino más corto al momento de hacer un cableado eléctrico para reducción de material y costos. Por este motivo recurrimos al algoritmo Prim el cual radica en la construcción de un **árbol de expansión** de ponderación mínima. Formalmente definimos el árbol de expansión mínimo $T$ para un grafo $G=(V,E)$ como sigue $T$ es un subconjunto acíclico de $E$ que conecta todos los vértices de $V$. Se minimiza la suma de las ponderaciones de las aristas de $T$.

## Justificación

Como hemos visto a lo largo de la materia de Análisis y Diseño de Algoritmos desde la antigüedad se han utilizado los algoritmos para la facilitación de tareas que requieren un proceso repetitivo. En este caso se analizaron múltiples algoritmos, los casos para su utilización y cuando evitarlos.

Para esto se utilizará el algoritmo Prim el cual se suele utilizar para diseñar una red de carreteras que une pequeños pueblos periféricos, una red de tuberías para agua potable o gas natural, una red eléctrica, canales de riego, una red de fibra óptica, algoritmos de búsqueda de caminos utilizados en IA (Inteligencia Artificial), desarrollo de juegos, ciencias cognitivas. Sin irnos tan lejos podemos ver la aplicación de este algoritmo en una aplicación que utilizamos diariamente como lo es Google Maps ya que busca el camino mas corto hacia tu destino.

Nuestro proyecto busca implementar este algoritmo para la resolución de una red eléctrica para la reducción de costos y material además de sentar un importante precedente de cooperación técnica y universitaria, que podría resultar de provecho para futuras experiencias inclusive sirviendo como base de un trabajo terminal en caso de retomar la idea a futuro.

### Problema por resolver

Programar el camino más corto sobre un cableado eléctrico. Dado un nodo ”n” que introduzca el usuario o que el programa lo determine automáticamente.

1. El programa debe de preguntar el nodo a elegir al usuario.
2. Dependiendo de los valores de origen, peso, destinos entregados al grafo resultante se imprime la lista de visitados.
3. Del archivo de salida calculado se genera el grafo resultante.

# Implementación

El lenguaje en el cual se decidió programar fue en Python ya que tiene funciones y librerías para calcular permutaciones sin ningún ciclo, así como también permite graficar el resultado de manera sencilla. Al inicio se elige el nodo de origen que después se agregará a la lista de visitados, así como sus adyacentes a una lista ordenada.

Igualmente, para la lista utilizamos el método append() el cual agrega el elemento completo al final de la lista. Si el elemento es una secuencia como una lista, una tupla o un diccionario, la secuencia completa se añadirá como un elemento de la lista ordenada, mientras que esta no este vacía se eliminan los vértices. Si el destino no se encuentra en la lista de visitados se agrega a esa misma lista además de poner los adyacentes en la ordenada que no se hayan visitado

```
import os
import networkx as nx
import matplotlib.pyplot as plt

#   CREAR UN GRAFO DIRIGIDO, PARA ELLO EL USUARIO INGRESA EL NUMERO DE
#   VERTICES Y EL NUMERO DE ARISTAS. SE CREA UN DICCIONARIO VACIO
#   EN EL CUAL SE GUARDAN LOS VERTICES

def crearGrafoDirigido(n_vertices, n_aristas):
    grafo = {}
    for i in range(n_vertices):
        grafo[i] = []
    for i in range(n_aristas):
        origen = int(input("Ingrese la luminaria de origen: "))
        destino = int(input("Ingrese la luminaria de destino: "))
        peso = float(input("Ingrese la distancia entre estas luminarias:
"))
        grafo[origen].append((destino, peso))
    return grafo

#   A CONTINUACIÓN SE PIDE AL USUARIO QUE INGRESE EL VERTICE ORIGEN, EL
VERTICE DESTINO
#   Y EL PESO DE LA ARISTA, ESTA INFORMACIÓN SERA GUARDADA EN EL
DICCIONARIO
#   POR ULTIMO SE IMPRIME EL GRAFO CREADO

n_vertices = int(input("Ingrese el número de luminarias: "))
n_aristas = int(input("Ingrese el número de posibles conexiones entre
ellas: "))
grafo = crearGrafoDirigido(n_vertices, n_aristas)
print(grafo)

listaVisitados = []
grafoResultante = {}
listaOrdenada = []

#    COMIENZO DEL ALGORITMO DE PRIM
#1.- ELEGIR NODO ORIGEN AL AZAR O PEDIRLO AL USUARIO
origen = int(input("\nIngresa el número de la lampara de origen: "))
#2.- AGREGARLO A LA LISTA DE VISITADOS
listaVisitados.append(origen)

#3.- AGREGAR SUS ADYACENTES A LA LISTA ORDENADA
for destino, peso in grafo[origen]:
  listaOrdenada.append((origen, destino, peso))
'''ORDENAMIENTO INSERT PARA LA LISTA'''
pos=0
act=0
listAux=[]
```

Se aplica el ordenamiento de la lista aplicando .sort  después se añade el vértice al grafo resultante que contiene 3 valores, origen, destino y peso. Los cuales se agregarán posteriormente al grafo. En donde se puede observar que en la posición 0 es el valor del nodo origen.

```
for i in range(len(listaOrdenada)):
    listAux=listaOrdenada[i]
    act=listaOrdenada[i][2]
    pos=i
    while pos> 0 and listaOrdenada[pos-1][2] > act:
        listaOrdenada[pos] = listaOrdenada[pos-1]
        pos=pos-1
    listaOrdenada[pos]=listAux

#4.- MIENTRAS LA LISTA ORDENADA NO ESTE VACIA, HACER:
while listaOrdenada:
  #5.-TOMAR VERTICE DE LA LISTA ORDENADA Y ELIMINARLO
  vertice = listaOrdenada.pop(0)
  d = vertice[1]

  #6.-SI EL DESTINO NO ESTA EN LA LISTA DE VISITADOS
  if d not in listaVisitados:
    #7.- AGREGAR A LA LISTA DE VISITADOS EN NODO DESTINO
    listaVisitados.append(d)
    #8.- AGREGAR A LA LISTA ORDENADA LOS ADYACENTES DEL NODO DESTINO
    #"d" QUE NO HAN SIDO VISITADOS
    for key, lista in grafo[d]:
      if key not in listaVisitados:
        listaOrdenada.append((d, key, lista))
    #####ORDENAMIENTO APLICADO A LA LISTA :
    listaOrdenada = [(c,a,b) for a,b,c in listaOrdenada]
    listaOrdenada.sort()
    listaOrdenada = [(a,b,c) for c,a,b in listaOrdenada]
    #9.-AGREGAR VERTICE AL GRAFO RESULTANTE
    # PARA COMPRENDER MEJOR, EN LAS SIGUIENTES LINEAS SE TOMA EL
"VERTICE", QUE EN ESTE CASO
    # ES UNA TUPLA QUE CONTIENE TRES VALORES; EL VERTICE EN SU POSICIÓN 0
ES EL VALOR DEL NODO ORIGEN
    # EL VÉRTICE EN SU POSICIÓN 1 ES EL NODO DESTINO, Y EL VÉRTICE EN SU
POSICIÓN 2 ES EL PESO DE LA ARISTA ENTRE AMBOS NODOS,
    # Y A CONTINUACIÓN SE AGREGAN ESOS VALORES AL GRAFO
    origen  = vertice[0]
    destino = vertice[1]
    peso    = vertice[2]

    if origen in grafoResultante:
      if destino in grafoResultante:
        lista = grafoResultante[origen]
        grafoResultante[origen] = lista + [(destino, peso)]
        lista = grafoResultante[destino]
        lista.append((origen, peso))
        grafoResultante[destino] = lista
      else:
        grafoResultante[destino] = [(origen, peso)]
        lista = grafoResultante[origen]
        lista.append((destino, peso))
        grafoResultante[origen] = lista
    elif destino in grafoResultante:
      grafoResultante[origen] = [(destino, peso)]
      lista = grafoResultante [destino]
      lista.append((origen, peso))
      grafoResultante[destino] = lista
    else:
      grafoResultante[destino] = [(origen, peso)]
      grafoResultante[origen] = [(destino, peso)]
```

Y así sucesivamente se van añadiendo los valores al grafo resultante. Finalmente tenemos el grafo resultante imprimiendo la lista y el grafo. Para esto se utilizó la librería NetworkX ya que es una de las más sencillas de utilizar. Es posible asignar atributos tanto a los nodos como a las aristas por ejemplo podemos especificar la ponderación mediante el atributo weight.

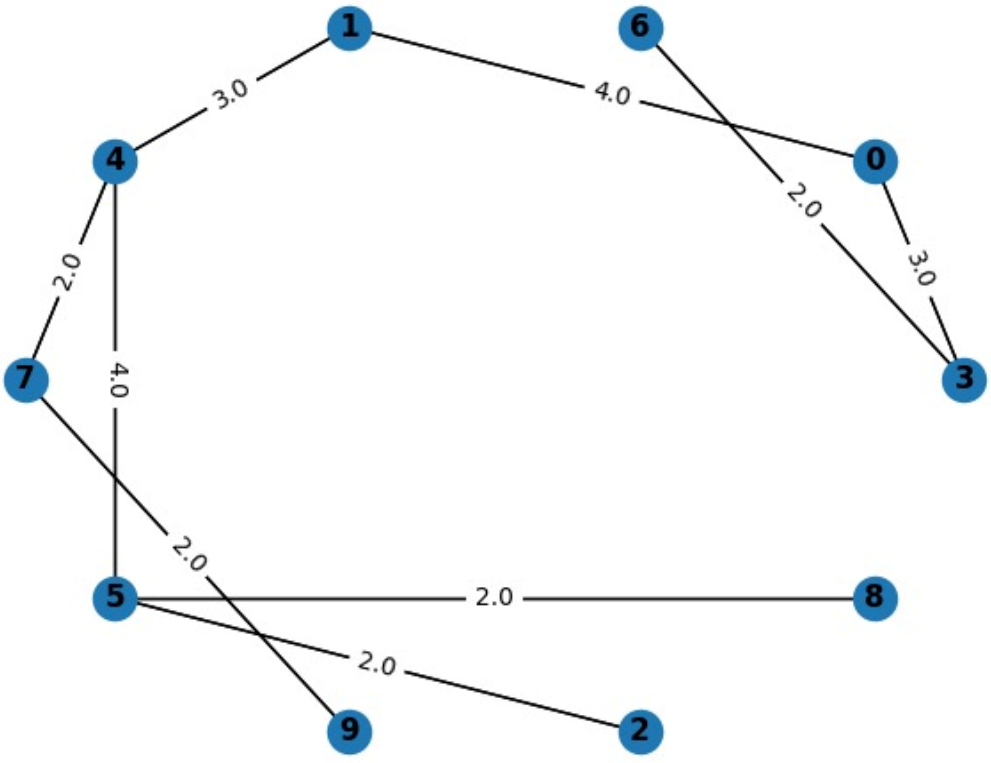
- Θ Es importante aclarar que al momento de agregar la librería Networkx para graficar a pesar de agregar funciones y el comando debido a las limitaciones de esta no se consiguió una visualización más sencilla del grafo
- Θ También agregar la librería matplotlib para el correcto funcionamiento.

```
print("\n\nTrayectoria resultante:\n")
for key, lista in grafoResultante.items():
  print(key)
  print(lista)

#   ESTA FUNCIÓN ES UNA FUNCIÓN DE AYUDA PARA DIBUJAR UN GRAFO.
#   TOMA UN GRAFO COMO PARÁMETRO Y USA LA LIBRERÍA MATPLOTLIB PARA
DIBUJARLO.
#   EL GRAFO SE DIBUJA CON UN LAYOUT CIRCULAR, LOS NODOS SE DIBUJAN CON
ETIQUETAS,
#   LOS ARCOS SE DIBUJAN SIN FLECHAS Y LOS PESOS SE DIBUJAN COMO
ETIQUETAS EN LOS ARCOS


def dibujarGrafo(grafo):
  G=nx.DiGraph()
  for origen, lista in grafo.items():
    for destino, peso in lista:
      G.add_edge(origen, destino, weight=peso)
  labels = nx.get_edge_attributes(G,'weight')
  pos=nx.circular_layout(G)
  nx.draw(G, pos, with_labels=True, arrows=False, font_weight='bold')
  nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)
  plt.show()

dibujarGrafo(grafoResultante)
```



## Instructivo

Al iniciar el programa se le preguntara al usuario el numero de luminarias totales luego las posibles conexiones entre ellas, recordando que los grafos tienen exactamente n*2 aristas (consideremos que de X a Y no es lo mismo de Y a X aunque tengan el mismo peso). EL grafo se empieza a llenar con los datos los cuales los pide en el orden de: la lampara de origen, la de destino y finalmente el peso, así hasta tomar todas las luminarias totales finalizando con la trayectoria resultante.

## Conclusión

Con este programa nos forzamos a buscar soluciones más sencillas también tuvimos algunas dificultades con las librerías y en general el entorno de Python ya que no estábamos tan familiarizados a comparación de C++ o Java. Pero elegimos este leguaje ya que nos proporciona varias herramientas al momento de generar grafos. Aunque una de sus desventajas sería que fue un poco complicado al momento de lidiar con los errores porque a pesar de tenerlos aun así nos corría.

## Bibliografía


- Software Foundation, P. (2022, 13 enero). 3.10.2 Documentation. Documentación de Python. Recuperado 15 de enero de 2023, de https://docs.python.org/es/3/
- Manejo de grafos con NetworkX en Python. (2020, 12 febrero). https://www.ellaberintodefalken.com/2020/02/grafos-con-networkx.html
- Algoritmo de Prim. (2016, 7 junio). Estructura de Datos II. https://estructurasite.wordpress.com/algoritmo-de-prim/
- Manejo de grafos con NetworkX en Python. (2020b, febrero 12).