Reporte de algoritmo de kruskal

Yadira Yaneth Marroquín de la Peña

Matricula: 1701504

yaz_marroquin@outlook.com

https://github.com/yadira02/1701504MatComp

4 de noviembre del 2017

En el siguiente reporte tiene como objetivo mostrar y analizar los algoritmos de kruskal para ver su funcionamiento en Python. Como objetivo del programa de kruskal encontrar todas las aristas del grafo que conecten a todos los vertices y la suma de los pesos para determinar el peso mas minimo.

1. Problema del Agente Viajero

1.1 Definición

El programa tiene como objetivo encontrar un recorrido completo que conecte todos los nodos de una red, visitándolos tan solo una vez y volviendo al punto de partida, y que encuentre menor distancia total de la ruta. Este tipo de problemas tiene gran aplicación en el ámbito de la logística y distribución, así como en la programación de curvas de producción.

El problema del agente viajero tiene una variación importante, y esta depende de que las distancias entre un nodo y otro sean simétricas o no, es decir, que la distancia entre A y B sea igual a la distancia entre B y A, puesto que en la práctica es muy poco probable que así sea. La cantidad de rutas posibles en una red está determinada por la ecuación:

$$(n-1)!$$

Es decir que en una red de 5 nodos la cantidad de rutas probables es igual a (5-1)! = 24, y a medida que el número de nodos aumente la cantidad de rutas posibles crece factorialmente. En el caso de que el problema sea simétrico la cantidad de rutas posibles se reduce a la mitad, es decir:

$$((n-1)!) / 2$$

Lo cual significa un ahorro significativo en el tiempo de procesamiento de rutas de gran tamaño.

1.2 Porque el problema del agente viajero es difícil

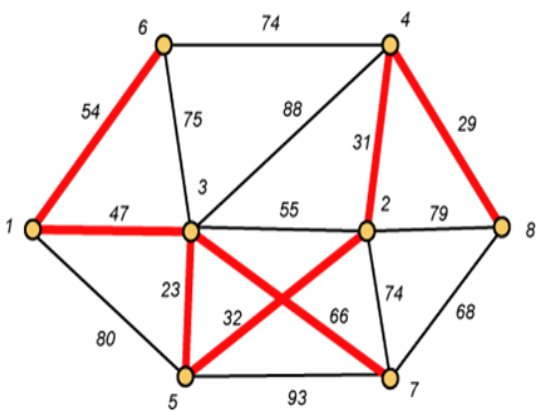
Es un programa considerado un problema difícil ya que la computadora no hay un algoritmo que puede resolver en un tiempo polinomio, es decir no podemos garantizar la mejor solución. El algoritmo nos muestra el mejor resultado para dicho problema el cual tienen que ser extensamente estudiado debido al grado de complejidad y arrojar las mejores soluciones que son arrojadas cuando está en consideración la gran cantidad de rutas.

1.3 Árbol de Expansión mínima

El algoritmo se basa en una propiedad clave de los árboles que permite estar seguros de sí un arco debe pertenecer al árbol o no, y usar esta propiedad para seleccionar cada arco. Nótese en el algoritmo, que siempre que se añade un arco (u,v), éste será siempre la conexión más corta (menor coste)alcanzable desde el nodo u al resto del grafo G. Así que por definición éste deberá ser parte del árbol. Dado un grafo conexo, no dirigido y con pesos en las aristas, un árbol de expansión mínima es un árbol compuesto por todos los vértices y cuya suma de sus aristas es la de menor peso.

CARACTERÍSTICAS

- Algoritmo basado en las aristas
- Agregar las aristas , uno a la vez, en orden de peso creciente
- El algoritmo mantiene A – un bosque de árboles. Una arista es aceptada si se conecta vértices de distintos árboles
- Necesitamos una estructura de datos que mantiene una partición, es decir una colección de conjuntos disjuntos



1.4 Algoritmo de kruskal

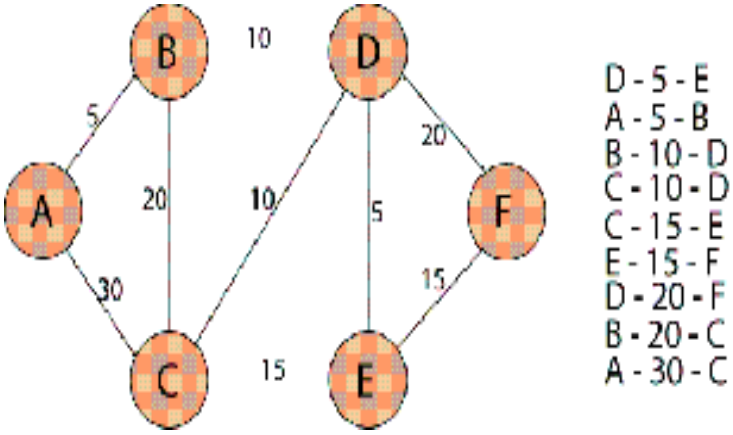
El objetivo del algoritmo de Kruskal es construir un árbol formado por arcos sucesivamente seleccionados de mínimo peso a partir de un grafo con pesos en los arcos. Por ejemplo, un grafo completo de cuatro nodos (todos relacionados con todos) tendría 16 árboles.

Como hemos ordenado las aristas por peso comenzaremos con la arista de menor peso, si los vértices que contienen dicha arista no están en la misma componente conexa entonces los unimos para formar una sola componente mediante para revisar si están o no en la misma componente al hacer esto estamos evitando que se creen ciclos y que la arista que une dos vértices siempre sea la mínima posible.

La formulación del MST también ha sido aplicada para hallar soluciones en diversas áreas (diseño de redes de transporte, diseño de redes de telecomunicaciones - TV por cable, sistemas distribuidos, interpretación de datos climatológicos, visión artificial - análisis de imágenes - extracción de rasgos de parentesco, análisis de clusters y búsqueda de superestructuras de quasar, plegamiento de proteínas, reconocimiento de células cancerosas, y otros).

1.5 Código de Pseudocódigo

```
1  método Kruskal(Grafo):
2      inicializamos MST como vacío
3      inicializamos estructura unión-find
4      ordenamos las aristas del grafo por peso de menor a mayor.
5      para cada arista e que une los vértices u y v
6          si u y v no están en la misma componente
7              agregamos la arista e al MST
8              realizamos la unión de las componentes de u y v
```



1.6 Código del Grafo

```
def kruskal(self):
    e = deepcopy(self.E)
    arbol = Grafo()
    peso = 0
    comp = dict()
    t = sorted(e.keys(), key = lambda k: e[k], reverse=True)
    nuevo = set()
    while len(t) > 0 and len(nuevo) < len(self.V):
        #print(len(t))
        arista = t.pop()
        w = e[arista]
        del e[arista]
        (u,v) = arista
        c = comp.get(v, {v})
        if u not in c:
            #print('u ',u, 'v ',v ,'c ', c)
            arbol.conecta(u,v,w)
            peso += w
            nuevo = c.union(comp.get(u,{u}))
            for i in nuevo:
                comp[i]= nuevo
    print('MST con peso', peso, ':', nuevo, '\n', arbol.E)
    return arbol
```

1.7 Método del vecino más cercano

El método Análisis de vecinos más próximos también puede utilizarse para calcular valores para un destino continuo. En esta situación, la media o el valor objetivo medio de los vecinos más próximos se utiliza para obtener el valor predicho del nuevo caso. Un algoritmo heurístico diseñado para solucionar el problema del agente viajero, no asegura una solución óptima, sin embargo suele proporcionar buenas soluciones, y tiene un tiempo de cálculo muy eficiente.

El método de desarrollo es muy similar al utilizado para resolver problemas de árbol de expansión mínima. Cuando se selecciona una arista nueva podríamos estar tomando una rama que sale de cualquier vértice del árbol. Aquí estamos construyendo un ciclo, no un árbol, así que siempre tomamos una rama que sale del extremo del camino que hemos construido hasta ese momento. Al final, añadimos una arista que une el vértice con el vértice inicio, para completar el recorrido.

1.8 Código del Vecino más cercano

```
def vecinoMasCercano(self):
    lv = list(self.V)   ##lista de vertices
    random.shuffle(lv)
    ni = lv.pop()
    inicial = ni
    lv2=list()
    lv2.append(ni)
    peso=0
    while len(lv2)<len(self.V):
        le = list()
        ln=list()
        ln = self.vecinos[ni]
        for nv in ln:
            if nv not in lv2:
                le.append((nv,self.E[(ni,nv)]))
        sorted(le, key = lambda le: le[1] )
        t=le[0]
        lv2.append(t[0])
        peso=peso+t[1]
        ni=t[0]
    peso=peso+self.E[lv2[-1], inicial]
    lv2.append(inicial)
    return (lv2,peso)
```

1.9 Solución Exacta

Para encontrar una solución exacta para el problema del agente viajero podemos hacer lo siguiente, teniendo n vértices calcular todos los caminos posibles que se pueden formar, encontrar n! , entonces debemos encontrar todas las permutaciones posibles y lo mostraremos con el algoritmo.

```
def permutation(lst):
    if len(lst) == 0:
        return []
    if len(lst) == 1:
        return [lst]
    l = [] # empty list that will store current permutation
    for i in range(len(lst)):
       m = lst[i]
       remLst = lst[:i] + lst[i+1:]
       for p in permutation(remLst):
           l.append([m] + p)
    return l
```

2. Experimento

En el siguiente programa se realizara un experimento en el cual tomaremos de 10 lugares turísticos donde se consideren como vértices y los usaremos para el encontrar el lugar de menos tiempo en recorrer.

Los 10 lugares son:

- Allende
- Montemorelos
- Congregación calles
- Cola de caballo
- General Terán
- Estanzuela
- Rayones
- Laguna de Sánchez
- Linares
- Uni

Se encontraran diferentes ciclos en el grafo, lo importante es tratar pasar por aristas de tal manera que la suma de los pesos de las aristas visitadas sea la menor posible. Para realizar esta tarea utilizaremos nuestro algoritmo de Kruskal además del siguiente código:

```
m = Grafo()
m.conecta('allende', 'montemorelos', 24)
m.conecta('allende', 'congregacion calles', 14)
m.conecta('allende', 'cola de caballo', 20)
m.conecta('allende', 'general teran', 42)
m.conecta('allende', 'estanzuela', 44)
m.conecta('allende', 'rayones', 66)
m.conecta('allende', 'laguna de sanchez', 61)
m.conecta('allende', 'linares', 72)
m.conecta('allende', 'uni', 70)

m.conecta('montemorelos','congregacion calles', 13)
m.conecta('montemorelos','cola de caballo', 45)
m.conecta('montemorelos','general teran', 18)
m.conecta('montemorelos','estanzuela', 67)
m.conecta('montemorelos','rayones', 61)
m.conecta('montemorelos','laguna de sanchez', 84)
m.conecta('montemorelos','linares',51)
m.conecta('montemorelos','uni', 93)

m.conecta('congregacion calles','cola de caballo',32 )
m.conecta('congregacion calles','general teran', 31)
m.conecta('congregacion calles','estanzuela', 55)
m.conecta('congregacion calles','rayones', 56)
m.conecta('congregacion calles','laguna de sanchez',71)
m.conecta('congregacion calles','linares', 61)
m.conecta('congregacion calles','uni', 81)
```

```
m.conecta('cola de caballo','general teran', 62)
m.conecta('cola de caballo','estanzuela', 26)
m.conecta('cola de caballo','rayones',86 )
m.conecta('cola de caballo','laguna de sanchez', 39)
m.conecta('cola de caballo','linares', 92)
m.conecta('cola de caballo','uni', 52)

m.conecta('general teran','estanzuela', 85)
m.conecta('general teran','rayones',79)
m.conecta('general teran','laguna de sanchez', 101)
m.conecta('general teran','linares', 67)
m.conecta('general teran','uni', 111)

m.conecta('estanzuela','rayones', 109)
m.conecta('estanzuela','laguna de sanchez', 65)
m.conecta('estanzuela','linares', 116)
m.conecta('estanzuela','uni', 24)

m.conecta('rayones','laguna de sanchez', 126)
m.conecta('rayones','linares', 110)
m.conecta('rayones','uni', 133 )

m.conecta('laguna de sanchez','linares', 131)
m.conecta('laguna de sanchez','uni', 89)

m.conecta('linares','uni', 140)
```

2.2 Resultados

La ruta más óptima para visitar todos los municipios utilizando MST:

Municipio inicial	Municipio final	Peso
Congregación calles	Montemorelos	13
Congregación calles	Allende	14
General Terán	Montemorelos	18
Cola de caballo	Allende	20
Uni	Estanzuela	24
Estanzuela	Cola de caballo	26
Laguna de Sánchez	Cola de caballo	39
Linares	Montemorelos	51
Rayones	Congregación calles	56

Con un peso: 261

Después un largo recorrido observaremos cuales las otras 3 rutas más cortas.

1. Con un peso de: 506

Municipio inicial	Municipio final	Peso
Allende	Cola de caballo	20
Cola de caballo	Laguna de Sánchez	39
Laguna de Sánchez	Estanzuela	65
Estanzuela	Uni	24
Uni	Congregación calles	81
Congregación calles	Rayones	56
Rayones	Montemorelos	61
Montemorelos	Linares	51
Linares	General Terán	67
General Terán	Allende	42

2. con peso de: 508

Municipio inicial	Municipio final	Peso
General Terán	Montemorelos	18
Montemorelos	Congregación calles	13
Congregación calles	Rayones	56
Rayones	Allende	66
Allende	Cola de caballo	20
Cola de caballo	Laguna de Sánchez	39
Laguna de Sánchez	Estanzuela	65
Estanzuela	Uni	24
Uni	Linares	140
Linares	General Terán	67

3. Con peso de: 509

Municipio inicial	Municipio final	Peso
Cola de caballo	Laguna de Sánchez	39
Laguna de Sánchez	Estanzuela	65
Estanzuela	Uni	24
Uni	Allende	70
Allende	Congregación calles	14
Congregación calles	Rayones	56
Rayones	Montemorelos	61
Montemorelos	Linares	51
Linares	General Terán	67
General Terán	Cola de caballo	62

Tiempo de ejecución

Tiempo que tarda el programa en realizar las permutaciones 331.72906414686634

2.3 Conclusión.

En mi opinión el programa de Kruskal nos ayuda a encontrar la distancia más cortas en cada camino que deseas tomar. Los algoritmos en el futuro nos ayudara a encontrar algoritmos exactos para alguna manera obtener la solución de manera más rápida, este trabajo observe como un programa nos ayuda a ver y obtener la distancia más corta entre municipios que estás buscando y así no batallar. Pero como la tecnología avanza cada día más y con la ayuda de los maestros que nos da a conocer las nuevas ideas.

Lo importante de este algoritmo es que nos enseña las diferentes formas en que se puede intercambiar cada país las distancias diferentes que se obtiene y los valores del peso cada grafo. En cada programa en Python nos deje un aprendizaje importante en la vida.