Reporte de practica: Algoritmo Kruskal

Nombre: Diana Carolina Lozoya Pérez **Matricula:**1725012

Materia: Matemáticas Computacionales **Grupo:**001 **Fecha:** 05/11/17

Resumen: Este reporte tiene como objetivo mostrar y describir un algoritmo llamado Kruskal, posteriormente en base a este algoritmo crear una solución aproximada al problema del agente viajero, que también será explicado eventualmente para poder comprender la aplicación del algoritmo y el uso que le estamos dando.

I. ¿Qué es el Problema del Agente Viajero?

El Problema del Agente Viajero – Travel Salesman Problem (TSP por sus siglas en inglés) tiene como objetivo encontrar un recorrido completo que conecte todos los lugares (nodos) de una red(grafo), visitándolos tan solo una vez y volviendo al punto de partida (nodo inicial), y que además minimice distancia total de la ruta.

Además, este tipo de problemas tiene gran aplicación en el ámbito de logística y distribución, así como en la programación de curvas de producción.

a) Lo difícil del problema:

Este problema tiene una variación importante, ya que esta depende de que las distancias entre un nodo y otro sean simétricas o no, es decir, que la distancia de A a B sea la misma que de B a 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.
Para tratar de resolver este problema aremos uso de dos algoritmos:

a) Algoritmo de aproximación
b) Heurística

II. ¿Qué es un Algoritmo de aproximación?

Un algoritmo de aproximación es un algoritmo que entrega una solución con una garantía teórica de cercanía al óptimo.
Se dice que un algoritmo de aproximación se da cuando nos entrega soluciones factibles, pero no necesariamente optimas, ya que factible es referente a algo que puede hacerse y optimo quiere decir que no existe algo mejor.

III. ¿Qué es un árbol de expansión mínima?

Antes de entrar de lleno, a la definición del algoritmo de Kruskal, es importante que antes aclaremos que es un árbol de expansión mínima, ya que este concepto es fundamental para poder comprender el funcionamiento de nuestro algoritmo.
Así, un árbol de expansión es un árbol compuesto por todos los vértices y algunas (posiblemente todas) de las aristas de G. Al ser creado un árbol no existirán ciclos, además debe existir una ruta entre cada par de vértices. Donde G es un grafo conexo no dirigido.
Entonces un árbol de expansión mínima es un árbol compuesto por todos los vértices y cuya suma de sus aristas es la de menor peso.
El problema de hallar el Árbol de Expansión Mínima (MST) puede ser resuelto con varios algoritmos, los más conocidos con Prim y Kruskal ambos usan técnicas voraces (greedy).

IV. ¿Qué es el Algoritmo de Kruskal?

Algoritmo descubierto por Joseph B. Kruskal en 1956 en los laboratorios Bell. El objetivo del algoritmo de Kruskal es construir un árbol (subgrafo sin ciclos) formado por aristas sucesivamente seleccionadas de mínimo peso a partir de un grafo con pesos en las aristas.

¿Cómo funciona el algoritmo de Kruskal?

Primeramente, ordenaremos las aristas del grafo por su peso de menor a mayor. Mediante la técnica greedy Kruskal intentará unir cada arista siempre y cuando no se forme un ciclo, ello se realizará mediante Union-Find. 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 Unión (x , y), para revisar si están o no en la misma componente conexa usamos la función SameComponent(x , y) al hacer esto estamos evitando que se creen ciclos y que la arista que une dos vértices siempre sea la mínima posible.

Este algoritmo nos va a ayudar para poder dar una solución al Problema del Agente Viajero.

Pseudocódigo de Kruskal

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

Código en Python:

```python
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
```

V. Vecino más cercano

Es un algoritmo de aproximación, que en este caso nos ayudara a resolver el problema del agente viajero, como este es un algoritmo de aproximación no nos dará un resultado exacto, pero, sin embargo, por su facilidad de implementación y el tiempo que requiere para darnos una solución aproximada lo vuelve una buena opción para obtener resultados rápidamente.

A continuación, se muestra el código que será utilizado, en Python:

```
def vecinoMasCercano(self):
    ni = random.choice(list(self.V))
    result=[ni]
    while len(result) < len(self.V):
        ln = set(self.vecinos[ni])
        le = dict()
        res =(ln-set(result))
        for nv in res:
            le[nv]=self.E[(ni,nv)]
        menor = min(le, key=le.get)
        result.append(menor)
        ni=menor
    return result
```

VI. Método para la solución Exacta del PAV

Dentro de este método encontramos el algoritmo de heurística, pues el siguiente código a mostrar, será el que nos ayudará a conseguir la solución exacta.

```
print("SOLUCION EXACTA\n")
tim=time.clock()
b=g.PAV()
camino=b
dfs=camino
best=0
for i in range(len(dfs)-1):
    best+=G.aristas[(dfs[i],dfs[i+1])]
best+=G.aristas[(dfs[-1],dfs[0])]
print("El camino mejor fue: ")
for k in camino:
    print(k,'->')
print(camino[0])
print("\n Con costo de: ",best)
print("Con tiempo de: ",time.clock()-tim)
```

VII. Ejemplo

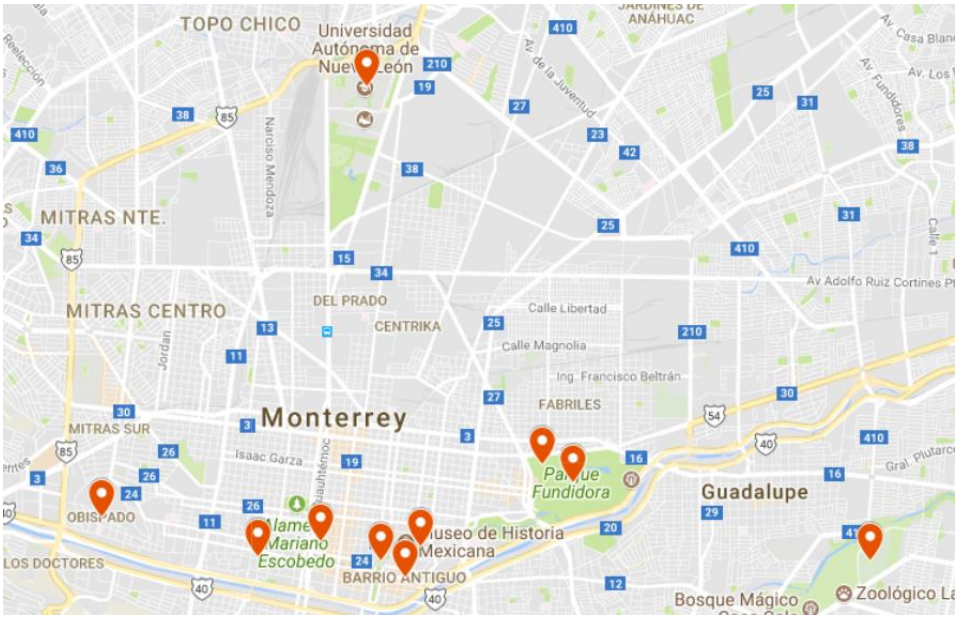
Ahora, probaremos el algoritmo de Kruskal, con el objetivo de solucionar un problema de encontrar la ruta óptima. El problema expresa que necesitamos encontrar la ruta más optima entre 10 lugares dados de una ciudad, esto con la finalidad de visitar todos y consumir la ruta más eficaz de todas.

Primeramente, definimos nuestros lugares a visitar(nodos) y las distancias entre los mismos (aristas)

Lugares:

1. Paseo Santa lucia
2. Macroplaza
3. Barrio Antiguo
4. Estadio BBVA Bancomer
5. Basílica Municipal ‘’La Purísima’’
6. Ciudad Universitaria (UANL)
7. Arena Monterrey
8. Obelisco a la fundación de monterrey
9. Parque Fundidora
10. Cerro del Obispado

En un mapa, su visualización seria así:



Ahora debemos de obtener las distancias del lugar 1 a todos los lugares, así como del lugar 2 a todos los restantes y así sucesivamente. Se mostrarán en forma de tabla.

Lugar	Distancia al Lugar 1
Macroplaza	550 m
Barrio Antiguo	481 m
Estadio BBVA	6.4 km
La Purísima	2.24 km
Ciudad Universitaria	6.26 km
Arena Monterrey	1.96 km
Obelisco	1.41 km
Fundidora	2.22 km
Obispado	4.33 km

Lugar	Distancia al lugar 2
Santa lucia	550 m
Barrio Antiguo	387 m
Estadio BBVA	6.56 km
La Purísima	1.64 km
Ciudad Universitaria	6.36 km
Arena Monterrey	2.49 km
Obelisco	787 m
Fundidora	2.79 km
Obispado	3.76 km

Lugar	Distancia al Lugar 3
Santa lucia	481 m
Macroplaza	387 m
Estadio BBVA	6.27 km
La Purísima	2.02 km
Ciudad Universitaria	6.62 km
Arena Monterrey	2.42 km
Obelisco	1.22 km
Fundidora	2.57 km
Obispado	4.14 km

Lugar	Distancia al Lugar 4
Santa lucia	6.4 km
Macroplaza	6.56 km
Barrio Antiguo	6.27 km
La Purísima	8.30 km
Ciudad Universitaria	9.38 km
Arena Monterrey	4.61 km
Obelisco	7.46 km
Fundidora	4.15 km
Obispado	10.4 km

Lugar	Distancia al lugar 5
Santa lucia	2.24 km
Macroplaza	1.64 km
Barrio Antiguo	2.02 km
Estadio BBVA	8.30 km
Ciudad Universitaria	6.45 km
Arena Monterrey	4.03 km
Obelisco	878 m
Fundidora	4.38 km
Obispado	2.12 km

Lugar	Distancia al lugar 7
Santa lucia	1.96 km
Macroplaza	2.49 km
Barrio Antiguo	2.42 km
Estadio BBVA	4.61 km
La Purísima	4.03 km
Ciudad Universitaria	5.61 km
Obelisco	3.17 km
Fundidora	473 m
Obispado	5.95 km

Lugar	Distancia al lugar 6
Santa lucia	6.26 km
Macroplaza	6.36 km
Barrio Antiguo	6.62 km
Estadio BBVA	9.38 km
La Purísima	6.45 km
Arena Monterrey	5.61 km
Obelisco	6.11 km
Fundidora	5.98 km
Obispado	6.90 km

Lugar	Distancia al lugar 8
Santa lucia	1.41 km
Macroplaza	787 m
Barrio Antiguo	1.22 km
Estadio BBVA	7.46 km
La Purísima	878 m
Ciudad Universitaria	6.11 km
Arena Monterrey	3.17 km
Fundidora	3.53 km
Obispado	2.90 km

Lugar	Distancia al lugar 9
Santa lucia	2.22 km
Macroplaza	2.79 km
Barrio Antiguo	2.57 km
Estadio BBVA	4.15 km
La Purísima	4.38 km
Ciudad Universitaria	5.98 km
Arena Monterrey	473 m
Obelisco	3.53 km
Obispado	6.35 km

Lugar	Distancia al lugar 10
Santa lucia	4.33 km
Macroplaza	3.76 km
Barrio Antiguo	4.14 km
Estadio BBVA	10.4 km
La Purísima	2.12 km
Ciudad Universitaria	6.90 km
Arena Monterrey	5.95 km
Obelisco	2.90 km
Fundidora	6.35 km

Luego de obtener las distancias correspondientes, pasamos a interpretarlas en nuestro grafo. Esto queda de la siguiente manera:

```
g=grafo()
g.conecta("Santa Lucia","Macroplaza",0.55)
g.conecta("Santa Lucia","Barrio Antiguo",0.48)
g.conecta("Santa Lucia","Estadio BBVA",6.4)
g.conecta("Santa Lucia","La Purisima",2.24)
g.conecta("Santa Lucia","Ciudad Universitaria",6.26)
g.conecta("Santa Lucia","Arena Monterrey",1.96)
g.conecta("Santa Lucia","Obelisco",6.4)
g.conecta("Santa Lucia","Fundidora",2.24)
g.conecta("Santa Lucia","Obispado",6.26)

g.conecta("Macroplaza","Santa Lucia",0.55)
g.conecta("Macroplaza","Barrio Antiguo",0.38)
g.conecta("Macroplaza","Estadio BBVA",6.56)
g.conecta("Macroplaza","La Purisima",1.64)
g.conecta("Macroplaza","Ciudad Universitaria",6.36)
g.conecta("Macroplaza","Arena Monterrey",2.49)
g.conecta("Macroplaza","Obelisco",0.78)
g.conecta("Macroplaza","Fundidora",2.79)
g.conecta("Macroplaza","Obispado",3.76)

g.conecta("Barrio Antiguo","Santa Lucia",0.48)
g.conecta("Barrio Antiguo","Macroplaza",0.38)
g.conecta("Barrio Antiguo","Estadio BBVA",6.27)
g.conecta("Barrio Antiguo","La Purisima",2.02)
g.conecta("Barrio Antiguo","Ciudad Universitaria",6.62)
g.conecta("Barrio Antiguo","Arena Monterrey",2.42)
g.conecta("Barrio Antiguo","Obelisco",1.22)
g.conecta("Barrio Antiguo","Fundidora",2.57)
g.conecta("Barrio Antiguo","Obispado",4.14)
```

```
g.conecta("Estadio BBVA", "Santa Lucia",6.4)
g.conecta("Estadio BBVA", "Macroplaza",6.56)
g.conecta("Estadio BBVA", "Barrio Antiguo",6.27)
g.conecta("Estadio BBVA", "La purisima",8.3)
g.conecta("Estadio BBVA", "Ciudad Universitaria",9.38)
g.conecta("Estadio BBVA", "Arena Monterrey",4.61)
g.conecta("Estadio BBVA", "Obelsico",7.46)
g.conecta("Estadio BBVA", "Fundidora",4.15)
g.conecta("Estadio BBVA", "Obispado",10.4)

g.conecta("La Purisima","Santa Lucia",2.24)
g.conecta("La Purisima","Macroplaza",1.64)
g.conecta("La Purisima","Barrio Antiguo",2.02)
g.conecta("La Purisima","Estadio BBVA",8.30)
g.conecta("La Purisima","Ciudad Universitaria",6.45)
g.conecta("La Purisima","Arena Monterrey",4.03)
g.conecta("La Purisima","Obelisco",0.87)
g.conecta("La Purisima","Fundidora",4.38)
g.conecta("La Purisima","Obispado",2.12)

g.conecta("Ciudad universitaria","Santa Lucia",6.26)
g.conecta("Ciudad universitaria","Macroplaza",6.36)
g.conecta("Ciudad universitaria","Barrio Antiguo",6.62)
g.conecta("Ciudad universitaria","Estadio BBVA",9.38)
g.conecta("Ciudad universitaria","La Purisima",6.45)
g.conecta("Ciudad universitaria","Arena Monterrey",5.61)
g.conecta("Ciudad universitaria","Obelisco",6.11)
g.conecta("Ciudad universitaria","Fundidora",5.98)
g.conecta("Ciudad universitaria","Obispado",6.9)
```

```
g.conecta("Arena Monterrey","Santa Lucia",1.96)
g.conecta("Arena Monterrey","Macroplaza",2.49)
g.conecta("Arena Monterrey","Barrio Antiguo",2.42)
g.conecta("Arena Monterrey","Estadio BBVA",4.61)
g.conecta("Arena Monterrey","La Purisima",4.03)
g.conecta("Arena Monterrey","Ciudad Universitaria",5.61)
g.conecta("Arena Monterrey","Obelisco",3.17)
g.conecta("Arena Monterrey","Fundidora",0.47)
g.conecta("Arena Monterrey","Obispado",5.95)

g.conecta("Obelisco","Santa Lucia",1.41)
g.conecta("Obelisco","Macroplaza",0.78)
g.conecta("Obelisco","Barrio Antiguo",122)
g.conecta("Obelisco","Estadio BBVA",7.46)
g.conecta("Obelisco","La Purisima",0.87)
g.conecta("Obelisco","Ciudad Universitaria",6.11)
g.conecta("Obelisco","Arena Monterrey",3.17)
g.conecta("Obelisco","Fundidora",3.53)
g.conecta("Obelisco","Obispado",2.9)

g.conecta("Fundidora","Santa Lucia",2.22)
g.conecta("Fundidora","Macroplaza",2.79)
g.conecta("Fundidora","Barrio Antiguo",2.57)
g.conecta("Fundidora","Estadio BBVA",4.15)
g.conecta("Fundidora","La Purisima",4.38)
g.conecta("Fundidora","Ciudad Universitaria",5.98)
g.conecta("Fundidora","Arena Monterrey",0.43)
g.conecta("Fundidora","Obelisco",3.53)
g.conecta("Fundidora","Obispado",6.35)

g.conecta("Obispado","Santa Lucia",4.33)
g.conecta("Obispado","Macroplaza",3.76)
g.conecta("Obispado","Barrio Antiguo",4.14)
g.conecta("Obispado","Estadio BBVA",10.4)
g.conecta("Obispado","La Purisima",2.12)
g.conecta("Obispado","Ciudad Universitaria",6.9)
g.conecta("Obispado","Arena Monterrey",5.95)
g.conecta("Obispado","Obelisco",2.9)
g.conecta("Obispado","Fundidora",6.35)
```

De esta manera podemos, ya hemos introducido todos los datos de nuestra ruta a nuestro grafo, el cual posteriormente utilizaremos para poder emplear el Algoritmo de Kruskal.

VIII. Análisis de resultados

Después de emplear dicho algoritmo, lo que nos entrega como resultado son múltiples rutas con múltiples costos, pero recordemos que este algoritmo es de aproximación, por lo que vamos a elegir la ruta con el costo más bajo, la cual es:

```
Estadio BBVA Fundidora 4.15
Fundidora Arena Monterrey 0.43
Arena Monterrey Santa Lucia 1.96
Santa Lucia Barrio Antiguo 0.48
Barrio Antiguo Macroplaza 0.38
Macroplaza Obelisco 0.78
Obelisco La Purisima 0.87
La Purisima Obispado 2.12
Obispado Ciudad Universitaria 6.9
Ciudad Universitaria Estadio BBVA 9.38
costo: 27.450000000000003
```

Ahora probando el algoritmo para la solución exacta, tenemos como resultado:

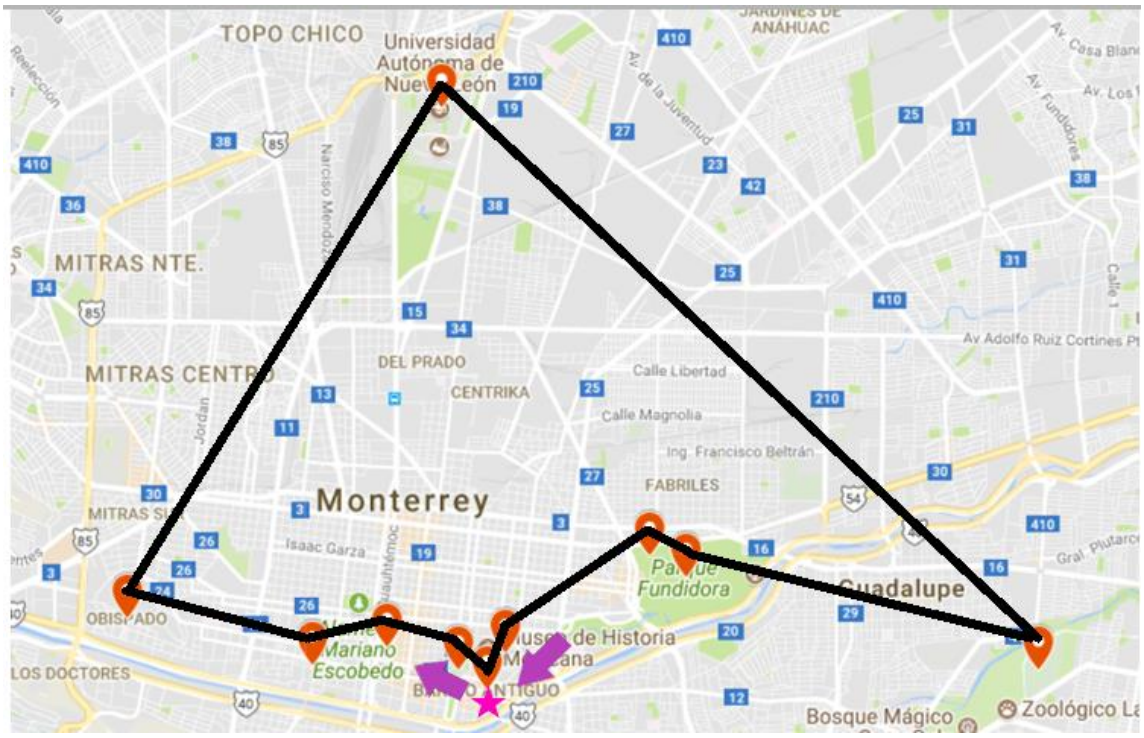
```
SOLUCION EXACTA

El camino mejor fue:
Barrio Antiguo ->
Macroplaza ->
Obelisco ->
La Purisima ->
Obispado ->
Ciudad Universitaria ->
Estadio BBVA ->
Fundidora ->
Arena Monterrey ->
Santa Lucia ->
Barrio Antiguo

 Con costo de:  27.45
Con tiempo de:  102.88702436938863
```

IX. Solución:

Entonces tomaremos el resultado del método exacto, y lo ilustraremos en el mapa, quedando de la siguiente manera:



Iniciando en donde esta ★ siguiendo el sentido de las flechas. De esta manera obtendremos el recorrido mas corto y con un costo menor. Nótese que en ningún momento se tomó a algún nodo inicial, es decir, el algoritmo hizo pruebas, análisis de rutas, hasta que determino cual es el mejor lugar para iniciar, obteniendo así, la ruta mas optima y, en consecuencia, el resultado exacto.

X. Conclusión

Como se pudo observar, las dos soluciones que se obtuvieron en los dos algoritmos diferentes son iguales, aunque el método de aproximación es muy rápido, en esta ocasión coincidió con el costo del método exacto, sin embargo, si ponemos atención, la ruta es diferente. Esto se debe a que el de aproximación pues muestra todas las posibles rutas, mientras que el exacto solo una. De este ultimo método, podemos ver que el tiempo que tardo en calcular la ruta fue de 1 minuto con 42 segundos aproximadamente, lo que representa que funciona muy bien para haber analizado todas las rutas posibles entre 10 lugares distintos.

El hecho de querer dar solución parece sencillo, pero cuando nos damos cuenta de que en este problema pueden existir una infinidad de soluciones en cuanto aumentan nuestros lugares de visita o simplemente nuestros nodos. Pues resulta difícil demostrar que la ruta o camino mas corto que encontramos en realidad es el más corto. Por lo que se crea el ultimo algoritmo que se supone nos ayuda a determinar el exacto, es decir, el mejor de todos. Sin embargo, se tarda relativamente más que el de aproximación, ya que va comprobando las rutas conforme va avanzando, mientras que el otro simplemente genera todas las rutas y las compara entre sí. Claramente el método exacto es más preciso, es decir nos da la respuesta sin rodeos, pero si es tardado. Mientras que el otro nos da los datos al instante, pero no podemos asegurar que sean los mejores.

A mi consideración, me quedo con el método exacto, ya que preferiría esperar un par de minutos más a que mi computador me dé la solución exacta a arriesgarme en un método de aproximación. Aunque este sea rápido y la mayoría de las veces acierte con la respuesta correcta. Prefiero esperar y tener algo concreto que estar probando, pues, aunque la mayoría de las veces de buenos resultados, siempre puede haber una posibilidad de que exista un resultado mejor.

Nota: Para mas soluciones y ver como funciona el ejemplo que se planteo, ejecutar el código que se encuentra en el repositorio con el nombre de: “Codigo Kruskal”.