# HOJA DE EJERCICIOS 10 – ALGORITMOS AVANZADOS UNSAAC

## Ejercicio 1.

## K-Center Problem

Dados p puntos, necesitamos escoger k (k <= p) centros, tal que la distancia máxima de un punto a un centro es minimizada. Se podría reescribir como: Se necesita construir k almacenes, dado un mapa de p ciudades conectadas. La mejor forma de construir un almacén, es tener presente que debe ser cercano a las ciudades. En otras palabras, la distancia máxima de una ciudad a un almacén debe ser mínima.

Analiza el siguiente seudocódigo (Enfoque voraz):

***1. Elija un centro arbitrario, p1.***

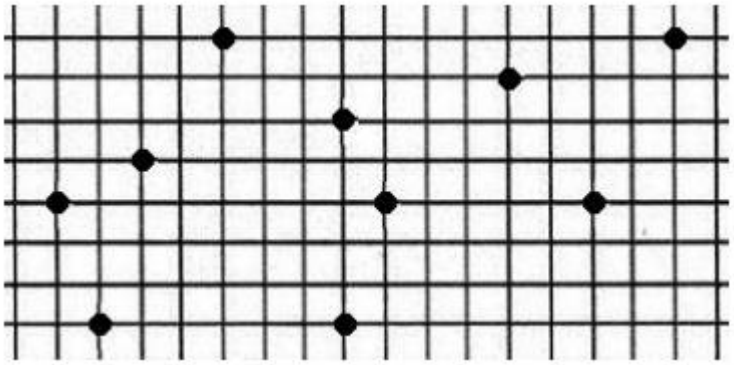
***2. Para cada punto restante P1, P2,... PN-1, calcule la distancia mínima desde los centros ya elegidos.***

***3. Elija el nuevo centro con la mayor distancia desde los centros ya elegidos, es decir, max***

***((dist (p1, P1), dist (p1, P2),... dist (p1, pN-1)).***

***4. Continúe con este procedimiento hasta encontrar todos los centros k.***

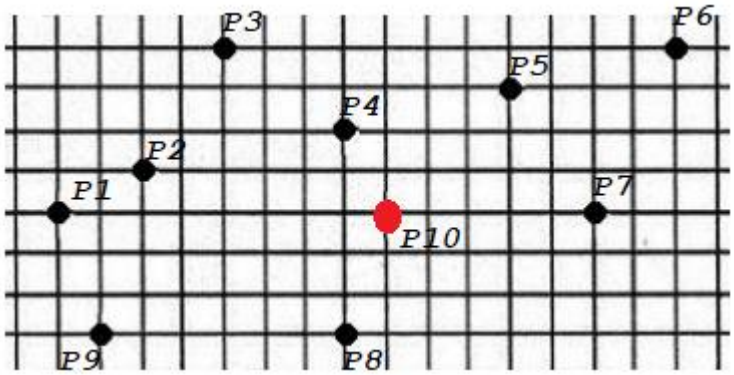
**A. Explique el cálculo de k=4 centros en la siguiente imagen (10 ciudades), paso a paso considerando que cada cuadrado es de 10 kilómetros.**



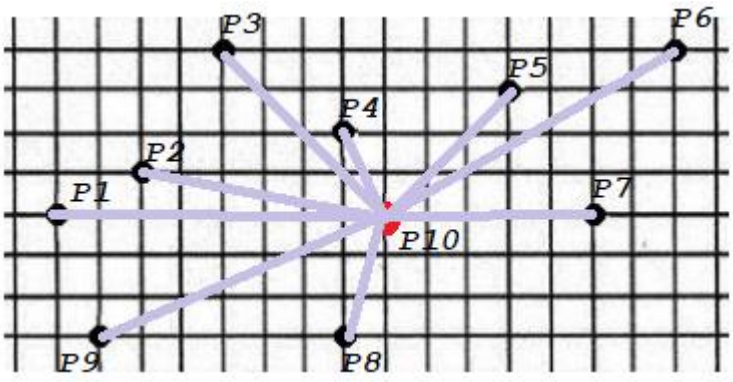
Solución:

PRIMERA ITERACION:

Escogemos un punto arbitrario



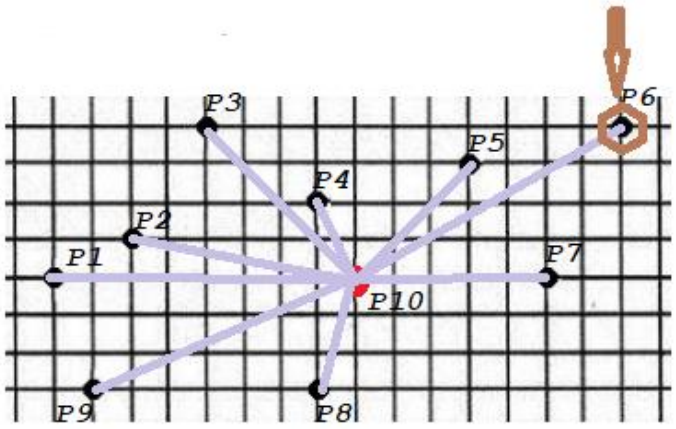
Hallamos las distancias que hay del centro escogido con los otros nodos (con el teorema de Pitágoras)




```
P10->P1 = 80.00 Km
P10->P2 = 60.83 Km
P10->P3 = 56.57 Km
P10->P4 = 22.36 Km
P10->P5 = 42.43 Km
P10->P6 = 80.62 Km
P10->P7 = 50.00 Km
P10->P8 = 31.62 Km
P10->P9 = 76.16 Km
```

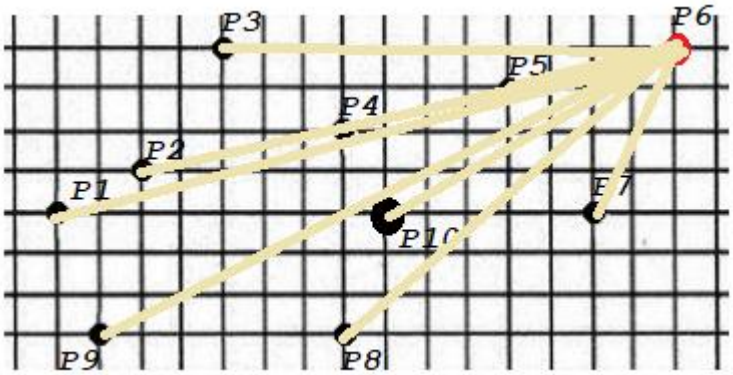
Como es el primer nodo, no se procede a comparar este centro con otros.

Elegimos el nuevo centro (el que tenga mayor distancia)




## SEGUNDA ITERACION:

Hallamos las distancias del nuevo centro con los puntos

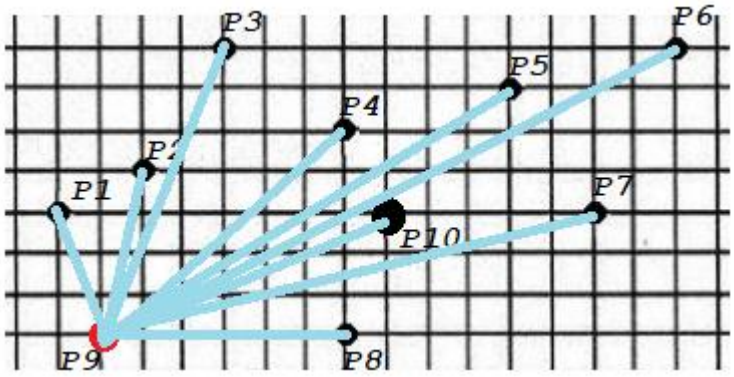



```
P6->P1 = 155.24 Km
P6->P2 = 133.42 Km
P6->P3 = 110.0 Km
P6->P4 = 82.46 Km
P6->P5 = 41.23 Km
P6->P7 = 44.72 Km
P6->P8 = 106.30 Km
P6->P9 = 156.52 Km
P6->P10 = 80.62 Km
```

## TERCERA ITERACION:

Ahora el nuevo centro seria el nodo P9

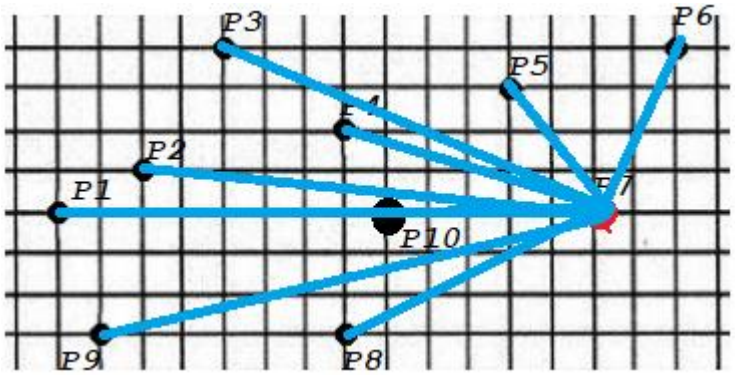
Hallamos las distancias:




```
P9->P1 = 31.62    km
P9->P2 = 41.23    km
P9->P3 = 76.16    km
P9->P4 = 78.10    km
P9->P5 = 116.62   km
P9->P6 = 156.52   km
P9->P10 = 76.16   km
P9->P7 = 123.69   km
P9->P8 = 60.00    km
```

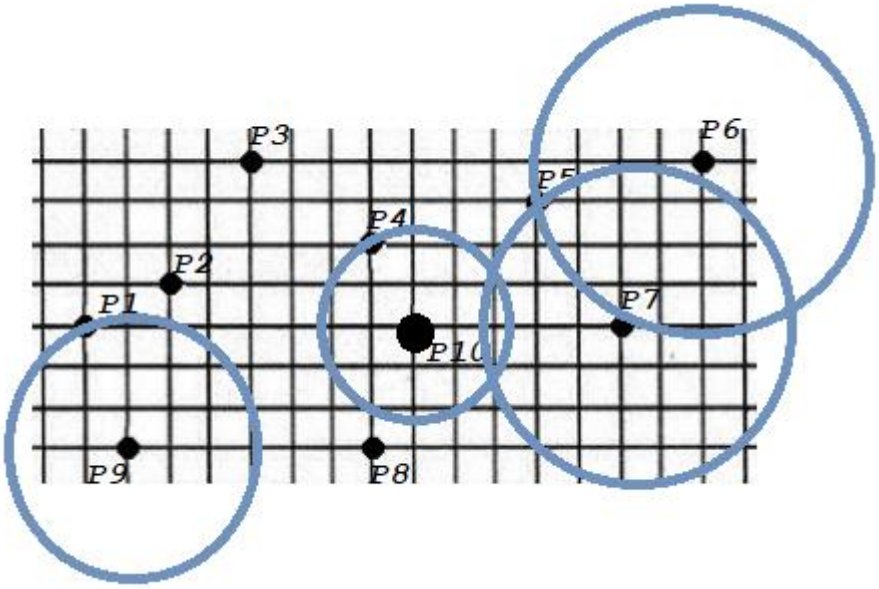
CUARTA ITERACION:

Hallamos las distancias del nuevo centro:




```
P7->P6 = 44.72
P7->P5 = 36.05
P7->P3 = 98.49
P7->P4 = 63.25
P7->P2 = 110.45
P7->P10 = 50
P7->P1 = 130.0
P7->P9 = 123.69
P7->P8 = 67.08
```

Quedándonos los 4 centros de la siguiente forma:



**B. ¿Qué tan eficiente considera el algoritmo anterior?**

Se puede observar que el algoritmo voraz no nos da la solución óptima ya que hay algunas ciudades que están muy alejadas de los centros (almacenes)

**C. Revisa el paper: "Approximation Algorithms for the Vertex K-Center Problem: Survey and Experimental Evaluation", lista los algoritmos presentados, lee las conclusiones e indica cual es el mejor en la práctica para largas instancias.**

Los algoritmos que se encuentran en el paper, son:

- Algoritmo SH
- Algoritmo GON
- Algoritmo HS
- Algoritmo CDS
- Algoritmo CDSH

El mejor algoritmo para largas instancias es el algoritmo de CDSH por la eficiencia y el tiempo de ejecución.

## Ejercicio 2

## Vertex Cover

Prueba el siguiente algoritmo de aproximación que resuelve el problema del Vertex Cover en Python:

```
from collections import defaultdict
class Graph:
        def __init__(self, vertices):
                self.V = vertices
                self.graph = defaultdict(list)
        def addEdge(self, u, v):
                self.graph[u].append(v)
        def printVertexCover(self):
                visited = [False] * (self.V)
                for u in range(self.V):
                        if not visited[u]:
                                for v in self.graph[u]:
                                        if not visited[v]:
                                                visited[v] = True
                                                visited[u] = True
                                                break
                for j in range(self.V):
                        if visited[j]:
                                print(j, end = ' ')
                print()
g = Graph(7)
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(3, 4)
g.addEdge(4, 5)
g.addEdge(5, 6)
g.printVertexCover()
```

**Entiende y comenta con detalle el código anterior.**

```
#Se crea la clase Grafo
class Graph:
    #Constructor de la clase
    def __init__(self,vertices):
        #Inicializa sus atributos
        self.V=vertices
        self.graph=defaultdict(list)

    #Agregar nodo
    def addEdge(self,u,v):
        self.graph[u].append(v)

    #Mostrar los vertices
    def printVertexCover(self):
        #inicializar una lista con valor falso
        visited=[False]*(self.V)
        #Verifixa si los nodos ya fueron visitados anteriormente
        for u in range(self.V):
            if not visited[u]:
                for v in self.graph[u]:
                    if not visited[v]:
                        visited[v]=True
                        visited[u]=False
                        break
        for j in range(self.V):
            if visited[j]:
                print(j,end=" ")
        print()

#Inicializamos la clase con 7 vertices
g=Graph(7)
#Agregar nodos
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(3, 4)
g.addEdge(4, 5)
g.addEdge(5, 6)
#Mostrar el vertex cover
g.printVertexCover()
```

## Ejercicio 3

## Travel Salesman Problem

Aplica el algoritmo de aproximación que resuelve el problema del viajero:

• (Algoritmo de doble arbol)

1. Computar un MST.

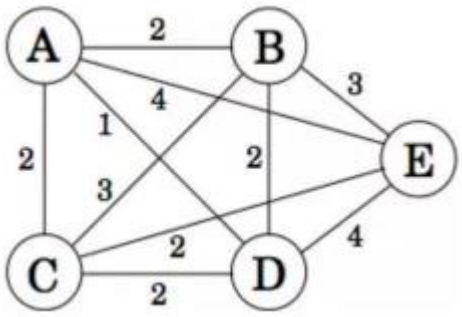
2. Remplazar cada arista del MST por dos copias de si mismo.

3. Encontrar un Tour Euleriano.

4. Reduce el Tour para obtener un ciclo Hamiltoniano.

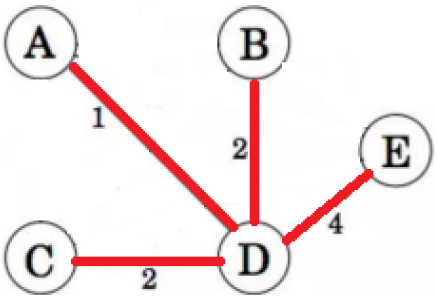
5. Retorna el ciclo Hamiltoniano.

A la siguiente instancia:

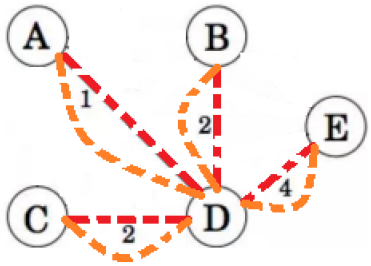



Muestra a detalle cada paso, y la obtención de una solución factible.

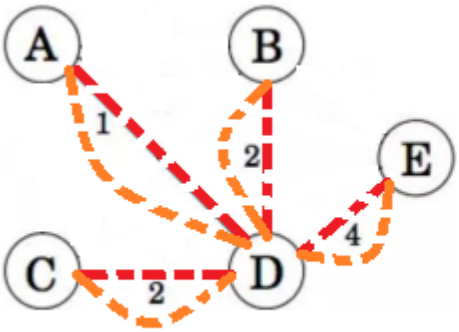
1.- Encontramos el árbol MST

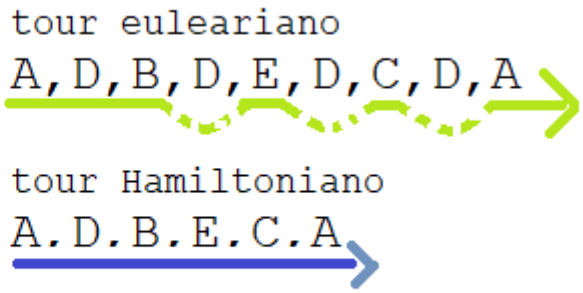



2.- Remplazar cada arista del MST por dos copias de si mismo

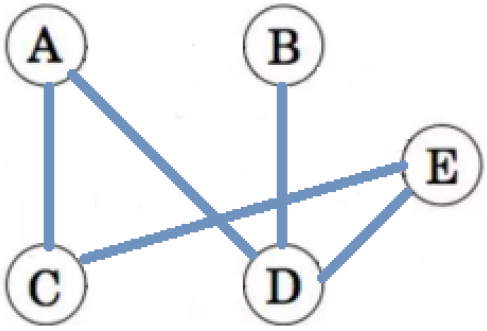


3.- Encontrar un Tour Euleriano.

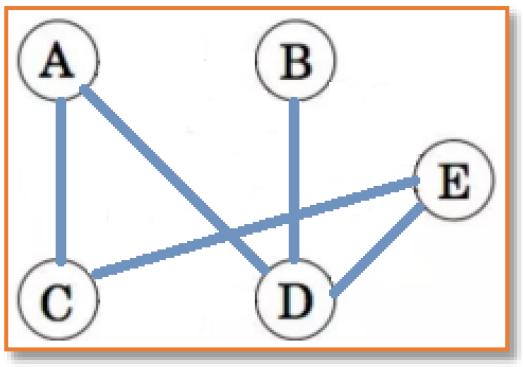



4.- Reduce el Tour para obtener un ciclo Hamiltoniano.




5.- Retornamos el resultado



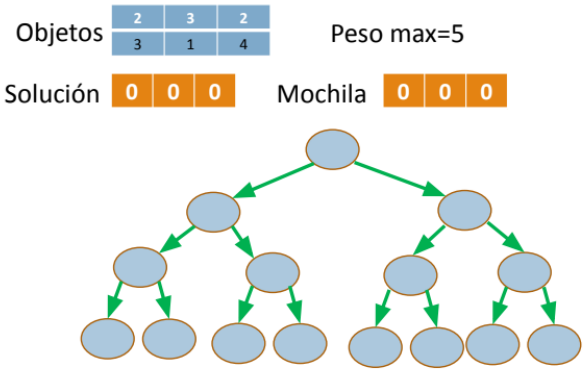

## Ejercicio 4.

**Investiga algún otro problema de aproximación distinto a los vistos en clase. Explícalo y muestra un ejemplo.**

### El problema de la mochila

El problema de la mochila, comúnmente abreviado por KP (del inglés Knapsack problem).Modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de objetos, cada uno con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo. De tal forma que con los algoritmos de aproximación encuentra la solución óptima y que cumpla los requisitos de este problema. Ejemplo:

Para el problema siguiente:




La solución óptima, es: