

# **SEGUNDA RONDA DE ADMISIÓN**

## EXAMEN DE DESARROLLO

RODRÍGUEZ SÁNCHEZ EDUARDO

SOLUCIÓN DEL PROBLEMA  
POR EL METODO DE ALGORITMO DEL VECINO MÁS PROXIMO

19 DE JUNIO DE 2020

# PLANTEAMIENTO DEL PROBLEMA

Dado un conjunto de  $n$  ciudades, el problema del agente viajero tiene como objetivo visitar todas las ciudades una sola vez, regresando a la ciudad inicial y recorriendo la menor distancia posible. Obtener la ruta más óptima.

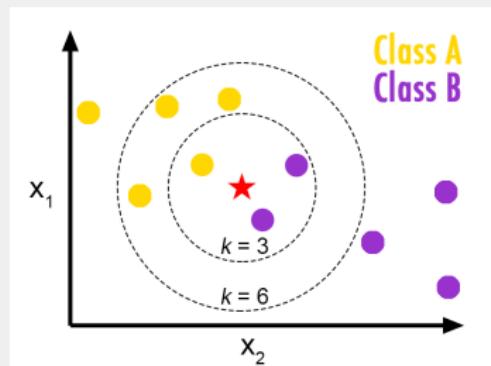
# RESTRICCIONES

1. Aplicar la técnica del vecino más cercano
2. Utilizar la distancia euclídea para conocer la distancia entre ciudades
3. El conjunto de datos es pequeño con  $n = 20$
4. La codificación se realiza en lenguaje Java

# MARCO TEÓRICO, SOLUCIÓN SIMPLE

El vector son las coordenadas de la ciudad, el algoritmo que genera la ruta para visitar todas las ciudades una vez, regresando a la ciudad de origen es :

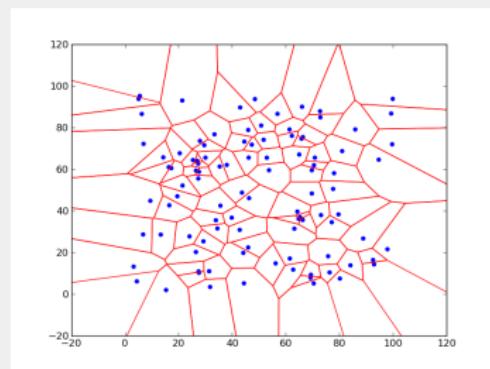
1. Selecciona ciudad inicial
2. Ubicar la ciudad mas cercana
3. Marcar la nueva ciudad como actual
4. Marcar la ciudad previa como visitada
5. Si todas las ciudades han sido visitadas termina, de lo contrario ir a 2



# MARCO TEÓRICO, SOLUCIÓN OPTIMA

Empleando Geometria Computacional se pueden ocupar otros métodos para optimizar la solución.

Una solución optima  $\Theta(n \log n)$  consiste en construir diagramas de Voronoi del conjunto de vectores  $S$ . El diagrama de Voronoi consiste en una colección de  $n$  polígonos convexos

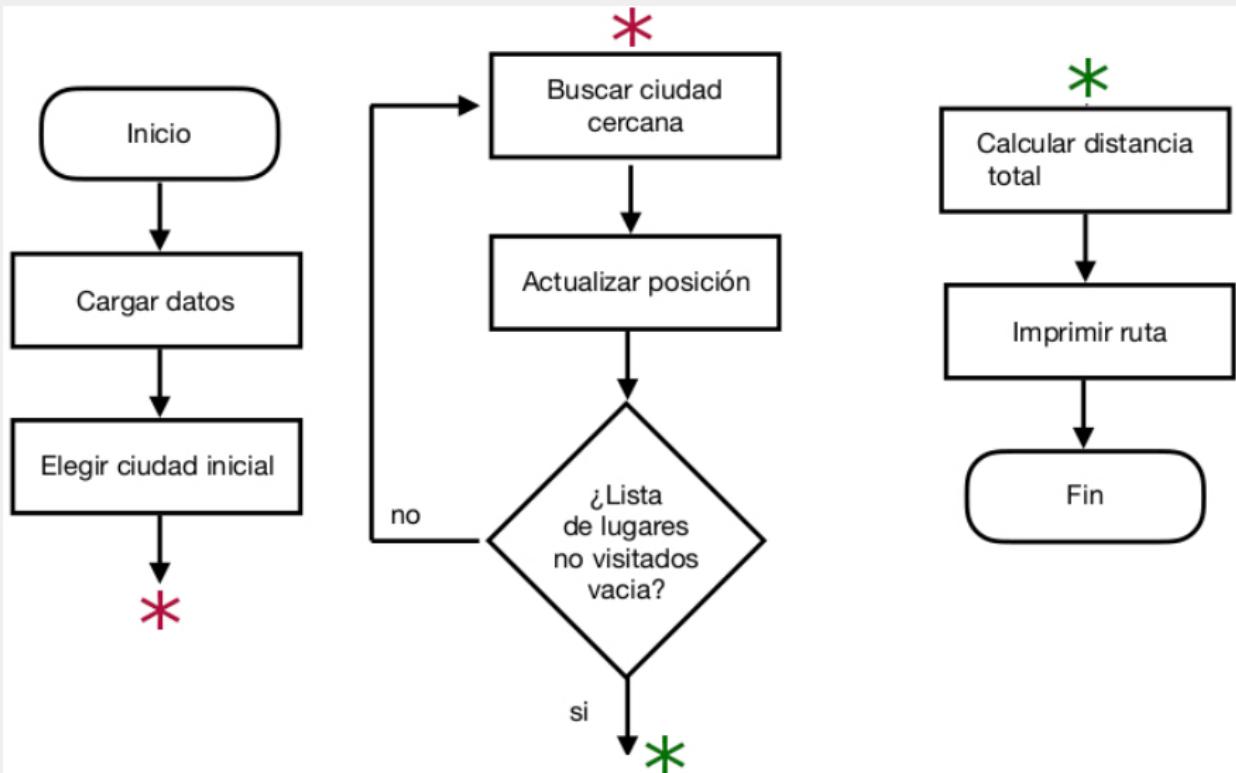


# METODOLOGÍA UTILIZADA

La solución inicial solo busca al elemento mas cercano del conjunto de vectores. Una propuesta alternativa es dividir el conjunto en bloques y calcular las distancias en cada bloque, aplicando el concepto de divide y vencerás.

En este caso todos los datos se encuentran dentro de un area de 100 unidades cuadradas, ubicando cuatro cuadrantes se divide al conjunto S en cuatro subconjuntos.

# DIAGRAMA DE FLUJO, PROGRAMA PRINCIPAL



# PSEUDOCÓDIGO, BUSCAR CIUDAD CERCANA

```
Inicializar variables e Iterador que recorre la lista
while(iterador tenga elementos){
    Carga nuevo valor de la lista;
    Calcula distancia;
    if(es el primer lugar visitado){
        guardar distancia minima de vertice a ciudad visitada;
        Actualizar bandera de primera vez;
        Actualiza el valor de la posicion al pivote;
    }
    if(distancia lugar visitado < distancia minima){
        Actualiza valor de nueva distancia minima;
        Actualiza el valor de la posicion al pivote;
    }
    incrementa en uno pivote;
}
Actualiza el vertice a la posicion actual
Elimina ciudad de sitios no visitados.
Regresa el vertice
}
```

# RESULTADOS

La tabla de datos agrupa la ciudad inicial con su respectiva distancia calculada.

La distancia mínima calculada es de 387 y pertenece a la ruta con ciudad inicial I, la distancia máxima calculada es de 470 y pertenece a la ruta con ciudad inicial E.

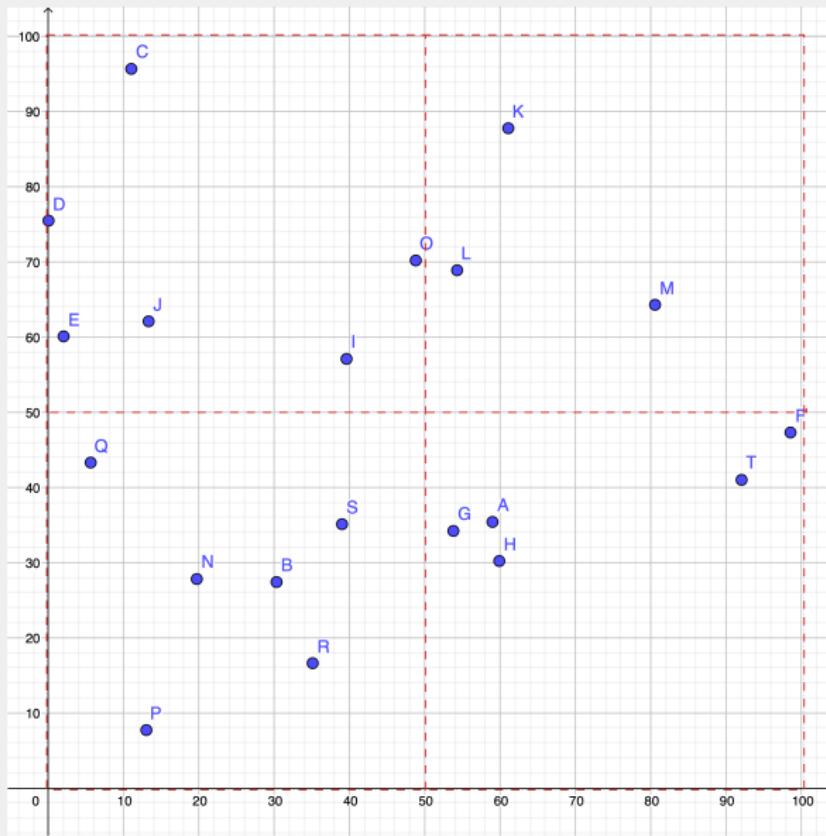
Ciudad inicial	Distancia recorrida
A	404
B	460
C	451
D	451
E	470
F	439
G	414
H	403
I	387
J	455
K	419
L	411
M	404
N	452
O	416
P	465
Q	456
R	452
S	435
T	430

# TABLA

Ciudad	Distancia	Ciudad	Distancia
A	404	M	404
B	460	N	452
C	451	O	416
D	451	P	465
E	470	Q	456
F	439	R	452
G	414	S	435
H	403	T	430
I	387		
J	455		
K	419		
L	411		

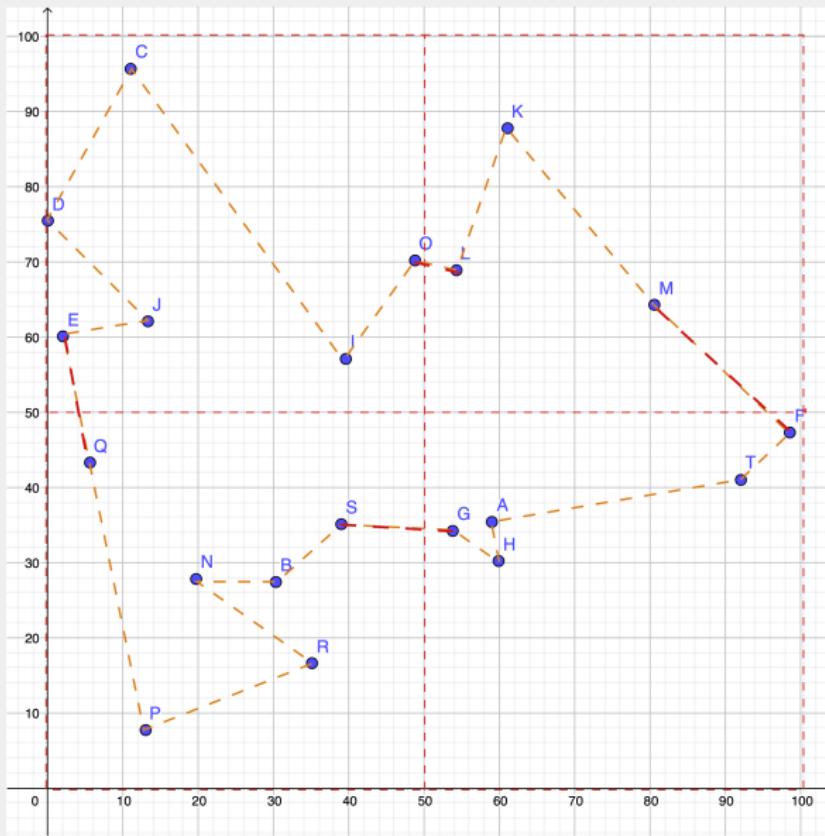
**Cuadro:** Distancia de cada ruta

# MAPA DE PUNTOS



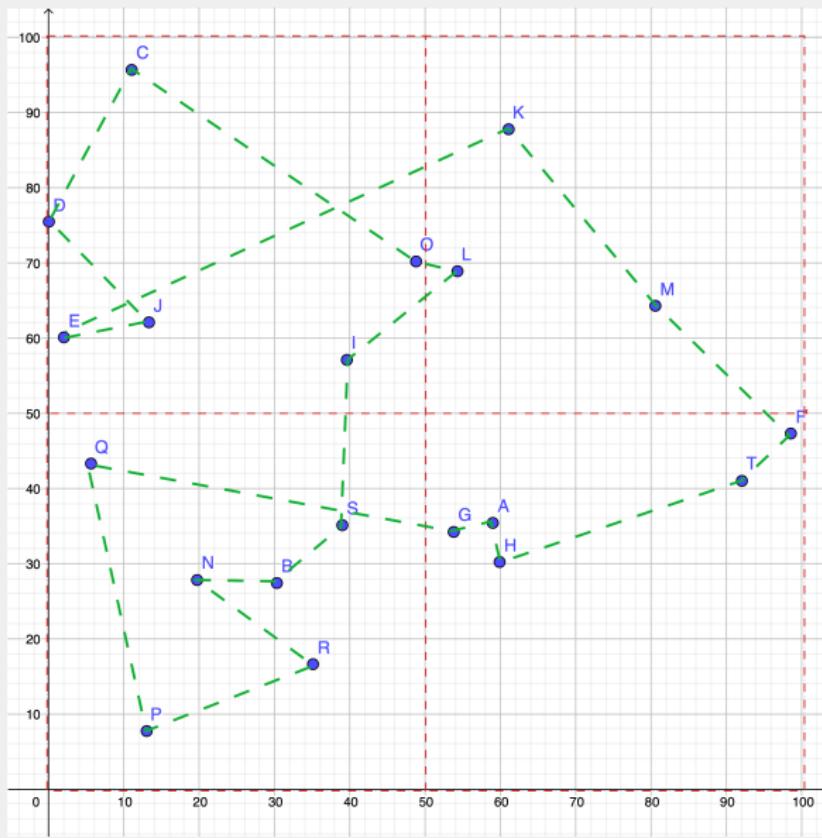
A cada ciudad se le asigna un nombre comenzando con la primer letra del abecedario.

# RUTA MÁS CORTA, CIUDAD INICIAL I



Los resultados de la ruta sirven para diseñar un algoritmo que calcule rutas similares en cualquier ciudad. Se observa que la ruta además visita un solo cuadrante a la vez

# RUTA MÁS LARGA, CIUDAD INICIAL E



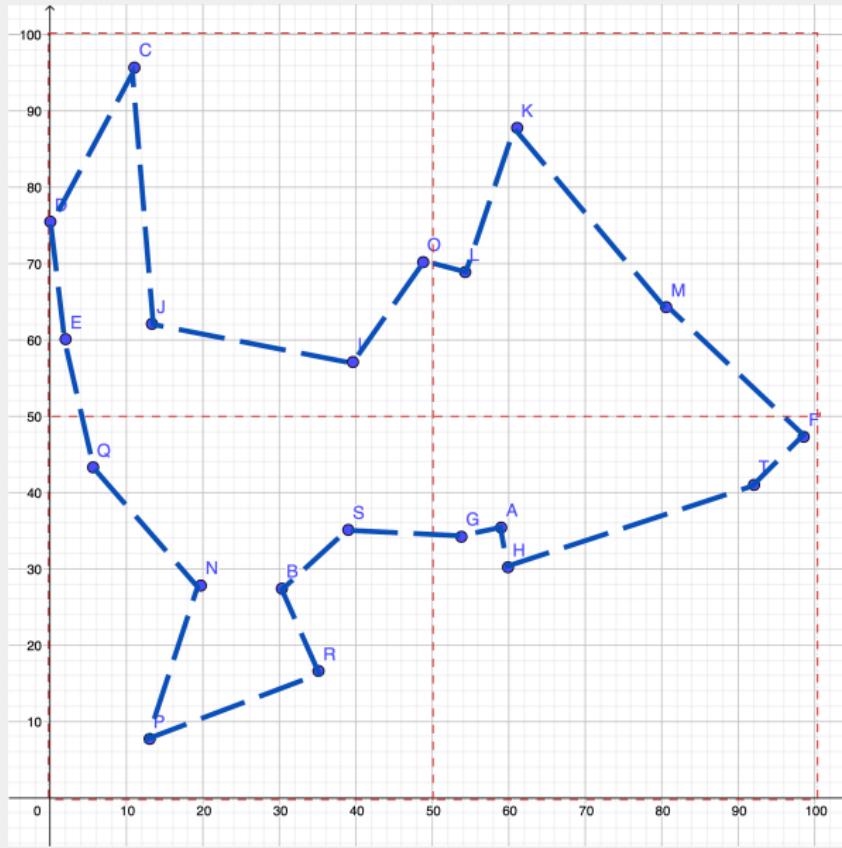
Durante el  
recorrido de la ruta  
los cuadrantes se  
visitán más de una  
vez

## RUTA MÁS OPTIMA

Para encontrar la ruta más optima se aplican las siguientes restricciones:

- Al igual que las ciudades, los cuadrantes se visitan una sola vez
- El mapa se divide en cuatro cuadrantes iguales, organizando los datos en grupos
- Se buscan aquellos puntos más cercanos a los subejes que forman los cuadrantes
- Una vez encontrados se obtienen las ciudades mas cercanas para identificar los puntos que van a conectarse con otros cuadrantes, esto es, de todos los puntos cercanos a los subejes se buscan los pares de cuadrantes opuestos para identificar las conexiones entre cuadrantes

# RUTA MÁS OPTIMA



# PSEUDOCÓDIGO, RUTA MÁS OPTIMA

```
Cargar Datos
Elegir ciudad inicial
Organizar datos por cuadrantes en ArrayList:
for(cada uno de los cuadrantes){
    extraer puntos y guardarlos en una lista
}
Definir puntos limite cercanos a subejes:
for(cada ciudad del cuadrante){
    medir distancia Respecto a subeje X, guardar mas cercano
    medir distancia Respecto a subeje Y, guardar mas cercano
}
Validar cada ciudad cercana a subejes:
    Buscar su punto mas cercano del cuadrante opuesto
    comparar con el pto definido como limite del
        cuadrante opuesto
if(ciudad mas cercana es distinta del punto limite){
    Actualiza la ciudad mas cercana como nuevo punto limite
}
Crear trayectoria en cada cuadrante:
    Ubicar los dos puntos limite
    Obtener las permutaciones del resto de puntos
    Conectar puntos teniendo siempre a los puntos limite en
        los extremos
Formatear Resultados e imprimir ruta
```

# EJECUCIÓN DE ALGORITMO SIMPLE

```
Output - JavaApplication (run) × Usages
ant -f "/Users/delta9/NetBeansProjects/PCyTI v2/JavaApplication" -Dnb.internal.action.name=run run
init:
Deleting: /Users/delta9/NetBeansProjects/PCyTI v2/JavaApplication/build/built-jar.properties
deps-jar:
Updating property file: /Users/delta9/NetBeansProjects/PCyTI v2/JavaApplication/build/built-jar.properties
compile:
run:
A
Distancia recorrida:404.88715
La Ruta es:
A, H, G, S, B, N, R, P, Q, E, J, D, C, O, L, I, K, M, F, T.
BUILD SUCCESSFUL (total time: 14 seconds)
```

# EJECUCION DE ALGORITMO MODIFICADO

```
Output - JavaApplication (run-single) x Usages
ant -f "/Users/delta9/NetBeansProjects/PCyTI v2/JavaApplication" -Dnb.internal.acti
init:
Deleting: /Users/delta9/NetBeansProjects/PCyTI v2/JavaApplication/build/built-jar.p
deps-jar:
Updating property file: /Users/delta9/NetBeansProjects/PCyTI v2/JavaApplication/bui
Compiling 1 source file to /Users/delta9/NetBeansProjects/PCyTI v2/JavaApplication/
Note: /Users/delta9/NetBeansProjects/PCyTI v2/JavaApplication/src/knn/javaport/main
Note: Recompile with -Xlint:unchecked for details.
compile-single:
run-single:
P
Los puntos limite cercanos a las rectas y=50, X=50 son:
L,M,I,O,Q,S,F,G,
Los puntos limite que conectan a los cuadrantes entre si son:
L,M,E,O,Q,S,F,G,
La ruta es: PRBSGAHTFMKLOIJCDEQN
BUILD SUCCESSFUL (total time: 6 seconds)
```

# CONCLUSIONES

1. El algoritmo es sencillo de implementar.
2. Se puede hacer más eficiente el algoritmo si se trabaja con el cuadrado de las distancias, así se evita aplicar la raíz cuadrada.
3. Subdividir el problema en problemas más pequeños facilita el diseño
4. Si se agregan condiciones el problema se vuelve complejo

# REFERENCIAS

-  RUSS MILLER, LAURENCE BOXER  
**ALGORITHMS SEQUENTIAL & PARALLEL A UNIFIED APPROACH .**  
*Cengage Learning, 2012.*
-  SUMEET DUA AND XIAN DU  
**DATA MINING AND MACHINE LEARNING IN CYBERSECURITY**  
*CRC Press, 2011.*
-  TOM M. MITCHELL  
**MACHINE LEARNING**  
*McGraw Hill, 1997.*
-  ETHEM ALPAYDIN  
**INTRODUCTION TO MACHINE LEARNING**  
*MIT Press, 2004.*
-  BRETT LANTZ  
**MACHINE LEARNING WITH R**  
*Packt Publishing, 2013.*