DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

INTRODUCCIÓN A LA PROGRAMACIÓN

Tarea de Laboratorio nro. 1

Profesor: Cristián Sepúlveda S.

Estudiantes: Bryan Alcarruz, Boris Troncoso.

19 de julio de 2025

Índice

1.- Introducción

En este informe abordaremos la solución a un problema logístico que enfrenta la empresa logistica UPS, la cual se encarga del retiro de encomiendas por medio terrestre desde las ubicaciones de sus clientes. La idea central es lograr una asignación eficiente de clientes a bodegas, y luego planificar recorridos óptimos para los camiones que parten desde cada una de estas bodegas.

El escenario tiene varias condiciones que se deben cumplir: cada camión parte y termina su recorrido en la misma bodega, solo puede salir un camión por bodega, y cada cliente debe ser atendido por una única bodega. Además, se busca que la cantidad de clientes que atiende cada camión sea lo más equilibrada posible, permitiendo como máximo una diferencia de un cliente entre ellos.

Este problema se relaciona directamente con el clásico "Problema del Viajante de Comercio" (TSP por sus siglas en inglés), que busca encontrar el camino más corto para visitar un grupo de lugares una sola vez y volver al punto de inicio. Aunque suena sencillo, resolverlo de forma exacta es muy costoso en términos computacionales cuando se tienen muchos puntos, por eso suele resolverse con métodos heurísticos, que buscan soluciones buenas (aunque no necesariamente perfectas) en un tiempo razonable.

Aquí utilizaremos una estrategia sencilla basada en funciones auxiliares, ciclos y estructuras básicas, que primero reparte a los clientes entre las bodegas considerando la cercanía y el equilibrio, y luego construye para cada camión un recorrido que minimice la distancia total a recorrer. La implementación se hizo con datos simulados para representar un caso práctico, y al final se muestran los resultados y algunas conclusiones generales.

2.- Método

Para resolver el problema propuesto, se desarrolló un programa en Python que combina estructuras básicas (condicionales, ciclos, funciones) con librerías externas como numpy y matplotlib. Se realizó un procesamiento que permite asignar clientes a bodegas de manera equilibrada y luego determinar una ruta de reparto eficiente en función de las distancias más cortas entre puntos. El desarrollo del algoritmo se dividió en los siguientes pasos:

1.- Lectura de archivos: Se definió una función que permite leer archivos .txt con coordenadas de puntos (bodegas y clientes), generando listas de pares [x, y] que representan sus ubicaciones en un plano. Esta función recorre cada línea del archivo, separa los valores y los guarda como listas dentro de una lista principal.

2.- Visualización inicial: Se graficaron los puntos en el plano para tener una primera vista de la distribución espacial. Los clientes se representaron con puntos azules, y las bodegas con cuadrados rojos. Para esto se utilizó la librería matplotlib.

3.- Asignación de clientes a bodegas: Cada cliente se asignó a la bodega más cercana, respetando la condición de equilibrio: todos los camiones deben atender una cantidad similar de clientes, con una diferencia máxima de uno. Para esto, se calculó un límite de clientes por bodega usando división entera y módulo en función del total de clientes y bodegas. Luego, para cada cliente, se ordenaron las bodegas por distancia creciente, y se asignó el cliente a la primera bodega disponible que aún no alcanzará su límite.

4.- Cálculo de rutas: Una vez realizada la asignación, se determinó la ruta que debía seguir cada camión para visitar a los clientes. Cada camión parte desde su bodega, visita el cliente más cercano que aún no ha sido atendido, y repite este proceso hasta recorrer todos los puntos asignados para volver finalmente a la bodega

5.- Visualización final de resultados: Se graficaron las asignaciones utilizando colores distintos para cada bodega, lo que permitió verificar visualmente la distribución de los clientes. Luego, se trazaron las rutas generadas por cada camión mediante líneas conectadas entre los puntos visitados, comenzando y terminando en la bodega correspondiente. Esta representación final permitió confirmar que tanto la asignación como la planificación de las rutas se realizaron correctamente.

3.- Algoritmos Propuestos

A continuación se presentan los algoritmos propuestos para resolver el problema. El primero se encarga de asignar cada cliente a la bodega más cercana, respetando un límite de clientes por bodega.

```
asignar_clientes(NUM clientes, NUM bodegas): NUM[][][]
    NUM asignacion ← []
    NUM asignados_por_bodega ← []
    NUM i ← 1
    WHILE i ≤ length(bodegas) DO
        add(asignacion, [], 0)
        add(asignados_por_bodega, 0, 0)
        i ← i + 1

    NUM cantidad_clientes ← length(clientes)
    NUM cantidad_bodegas ← length(bodegas)
    NUM base ← cantidad_clientes // cantidad_bodegas
    NUM extra ← cantidad_clientes % cantidad_bodegas

    NUM limites ← []
    i ← 1
    WHILE i ≤ cantidad_bodegas DO
        IF i ≤ extra THEN
            add(limites, base + 1, 0)
        ELSE
            add(limites, base, 0)
        i ← i + 1

    NUM cliente_actual ← 1
    WHILE cliente_actual ≤ length(clientes) DO
        NUM distancias_cliente ← []
        NUM j ← 1
        WHILE j ≤ length(bodegas) DO
            NUM d ← distancia(clientes[cliente_actual], bodegas[j])
            add(distancias_cliente, [d, j], 0)
            j ← j + 1

        metodo_burbuja(distancias_cliente)

        NUM k ← 1
        BOOL asignado ← FALSO
        WHILE k ≤ length(distancias_cliente) AND asignado = FALSO DO
            NUM indice ← distancias_cliente[k][2]
            IF asignados_por_bodega[indice] < limites[indice] THEN
                add(asignacion[indice], clientes[cliente_actual], 0)
                asignados_por_bodega[indice] ← asignados_por_bodega[indice] + 1
                asignado ← VERDADERO
            k ← k + 1

        cliente_actual ← cliente_actual + 1

    RETURN asignacion
```

La siguiente función determina la ruta de reparto desde cada bodega, conectando los puntos en función de la menor distancia restante.

```
recorrido_optimo(NUM bodega[2], NUM clientes): NUM[][]
    NUM clientes_restantes ← metodo_copiar(clientes)
    NUM recorrido ← [bodega]
    NUM actual ← bodega

    WHILE length(clientes_restantes) > 0 DO
        NUM menor_distancia ← distancia(actual, clientes_restantes[1])
        NUM mas_cercano ← clientes_restantes[1]
        NUM i ← 2
        WHILE i ≤ length(clientes_restantes) DO
            NUM d ← distancia(actual, clientes_restantes[i])
            IF d < menor_distancia THEN
                menor_distancia ← d
                mas_cercano ← clientes_restantes[i]
            i ← i + 1
        add(recorrido, mas_cercano, 0)

        NUM j ← 1
        WHILE j ≤ length(clientes_restantes) DO
            IF clientes_restantes[j][1] = mas_cercano[1] AND clientes_restantes[j][2] = mas_cercano[2] THEN
                delete_index(clientes_restantes, j)
                j ← length(clientes_restantes) + 1
            ELSE
                j ← j + 1

        actual ← mas_cercano

    RETURN recorrido
```

Finalmente, se incluye una función auxiliar para calcular las distancias entre los clientes y una bodega específica,

```
distancias_por_bodega(NUM clientes, NUM bodega[2]): NUM[]
    NUM distancias ← []
    NUM i ← 1
    WHILE i ≤ length(clientes) DO
        NUM d ← distancia(clientes[i], bodega)
        add(distancias, d, 0)
        i ← i + 1
    RETURN distancias
```

4.- Resultados

Ejemplo 1:

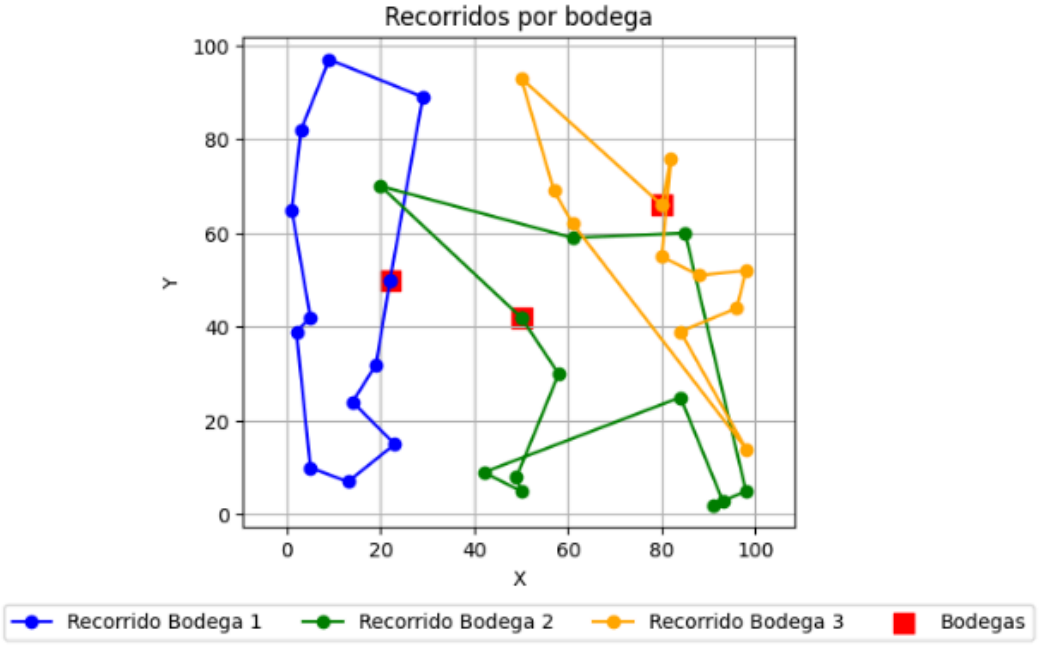

Figura 1: Recorridos de reparto para el archivo clientes_1.txt y bodegas_1.txt.

Ejemplo 2:

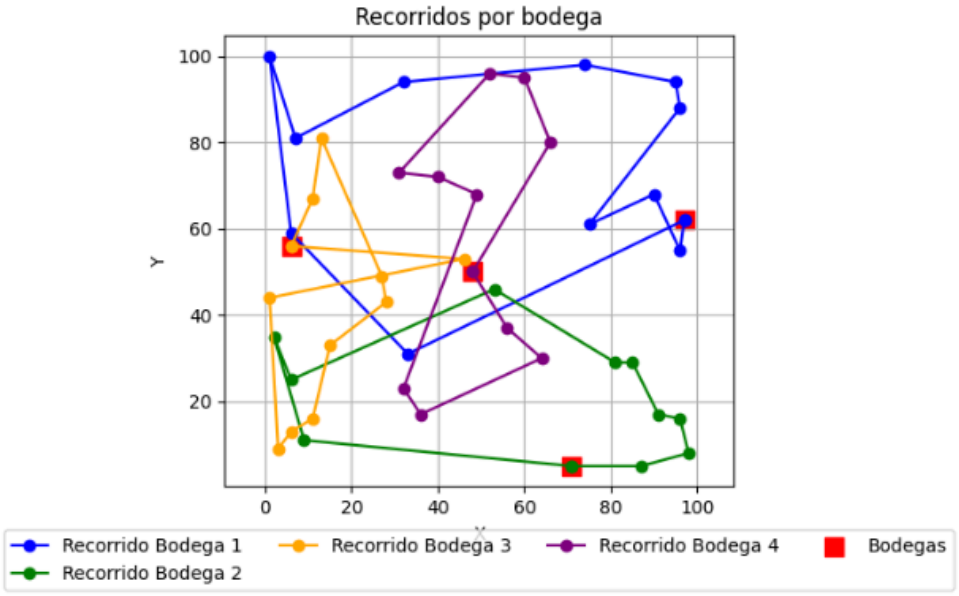

Figura 2: Recorridos de reparto para el archivo clientes_2.txt y bodegas_2.txt.

Ejemplo 3:

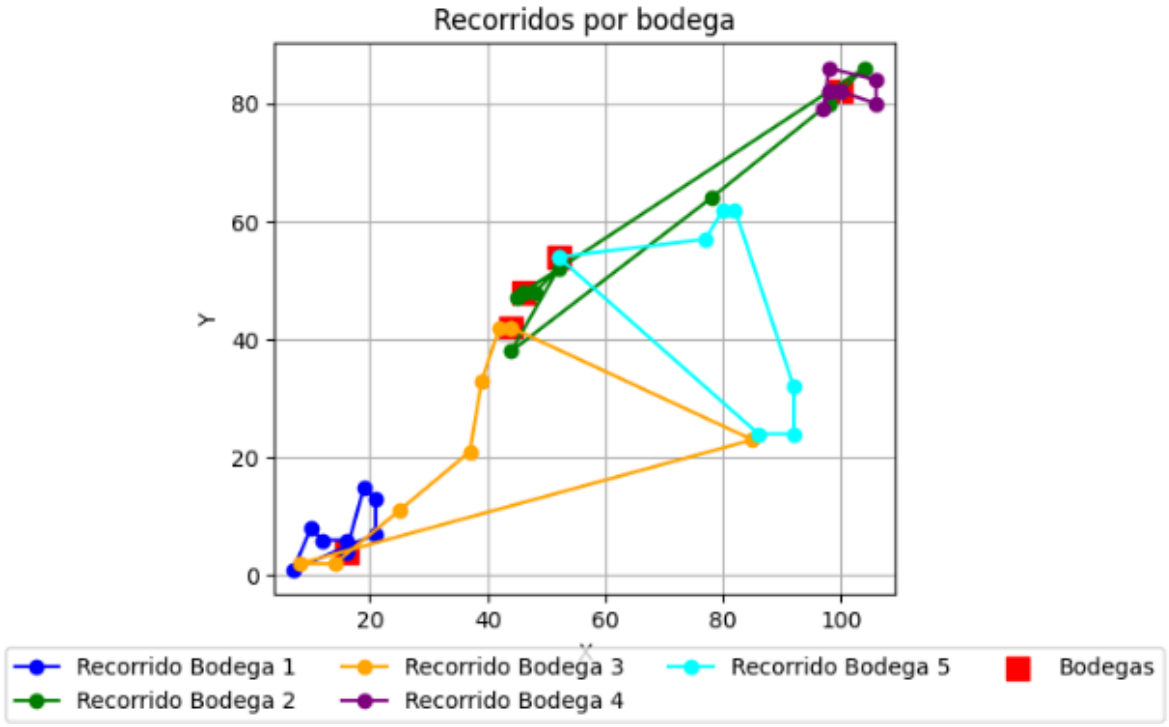

Figura 3: Recorridos de reparto para el archivo clientes_3.txt y bodegas_3.txt.

Ejemplo 4:

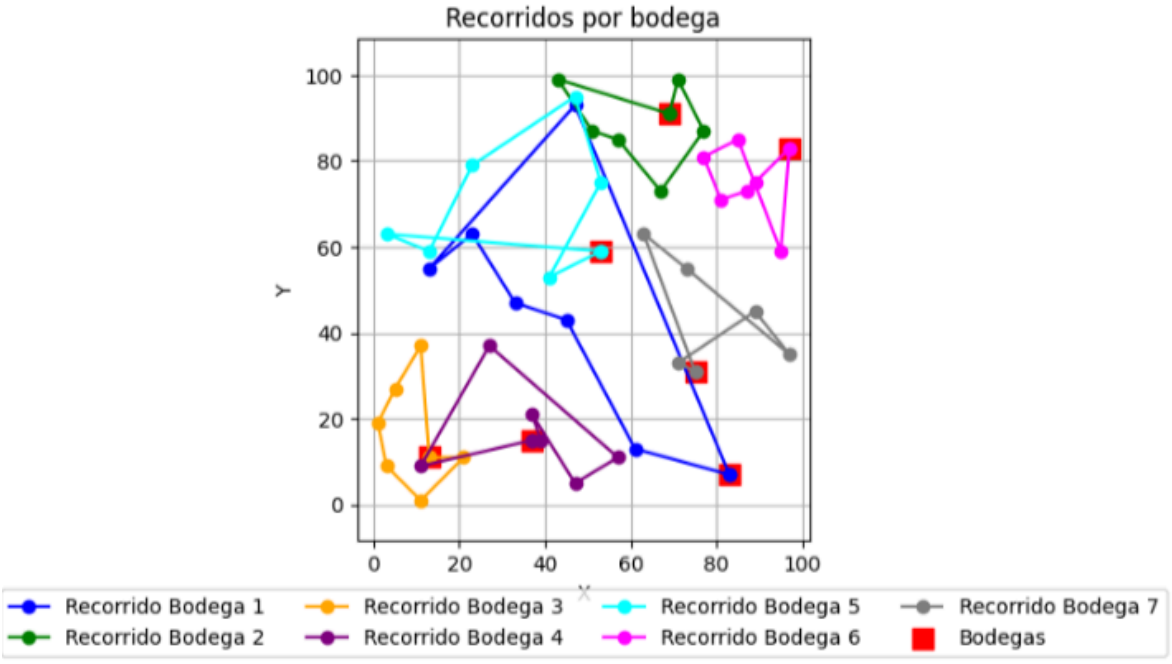

Figura 4: Recorridos de reparto para el archivo clientes_4.txt y bodegas_4.txt.

Conclusión

Observando los resultados, podemos concluir que el algoritmo cumple el objetivo de asignar a cada bodega la misma cantidad de clientes o con una diferencia de máximo un cliente, por lo que el algoritmo utilizado el cual define la ruta cumple, y nos da una buena aproximación.

A lo largo de esta tarea se abordó un problema clásico de optimización logística relacionado con la asignación de clientes a bodegas y el diseño de rutas eficientes para su recolección. Se implementaron algoritmos que permitieron dividir equitativamente los clientes entre las bodegas disponibles, respetando las restricciones del enunciado, como la diferencia máxima de un cliente entre camiones y la asignación exclusiva de cada cliente a una única bodega.

Para resolver este problema, se recurrió a métodos heurísticos simples pero efectivos, como el uso de la distancia euclidiana para determinar cercanía, una distribución balanceada de carga entre bodegas y un recorrido con el objetivo de encontrar el punto más cercano para minimizar la distancia total recorrida por cada camión. Si bien no se garantiza una solución óptima global, los resultados obtenidos son consistentes y razonables, considerando el objetivo de eficiencia computacional y simplicidad del enfoque.

El desarrollo de este ejercicio permitió aplicar de manera práctica conceptos de programación estructurada, uso de estructuras de datos como listas anidadas, y lógica algorítmica aplicada a problemas reales. Además, se fortaleció la capacidad de transformar pseudocódigo a código funcional en Python, lo que nos permite ver la implementación de la propuesta hecha en pseudocódigo.

Si bien las rutas generadas no garantizan ser las más cortas posibles, el método implementado entrega resultados coherentes y funcionales para el propósito del ejercicio. Esto se debe a que el algoritmo utilizado prioriza siempre el siguiente cliente más cercano, sin evaluar todas las posibles combinaciones de recorrido. Para mejorar el resultado, se podría implementar una búsqueda exhaustiva que compare todos los recorridos posibles y seleccione el más eficiente. Sin embargo, esto implicaría un código considerablemente más complejo y un mayor tiempo de ejecución, especialmente a medida que aumenta el número de clientes.