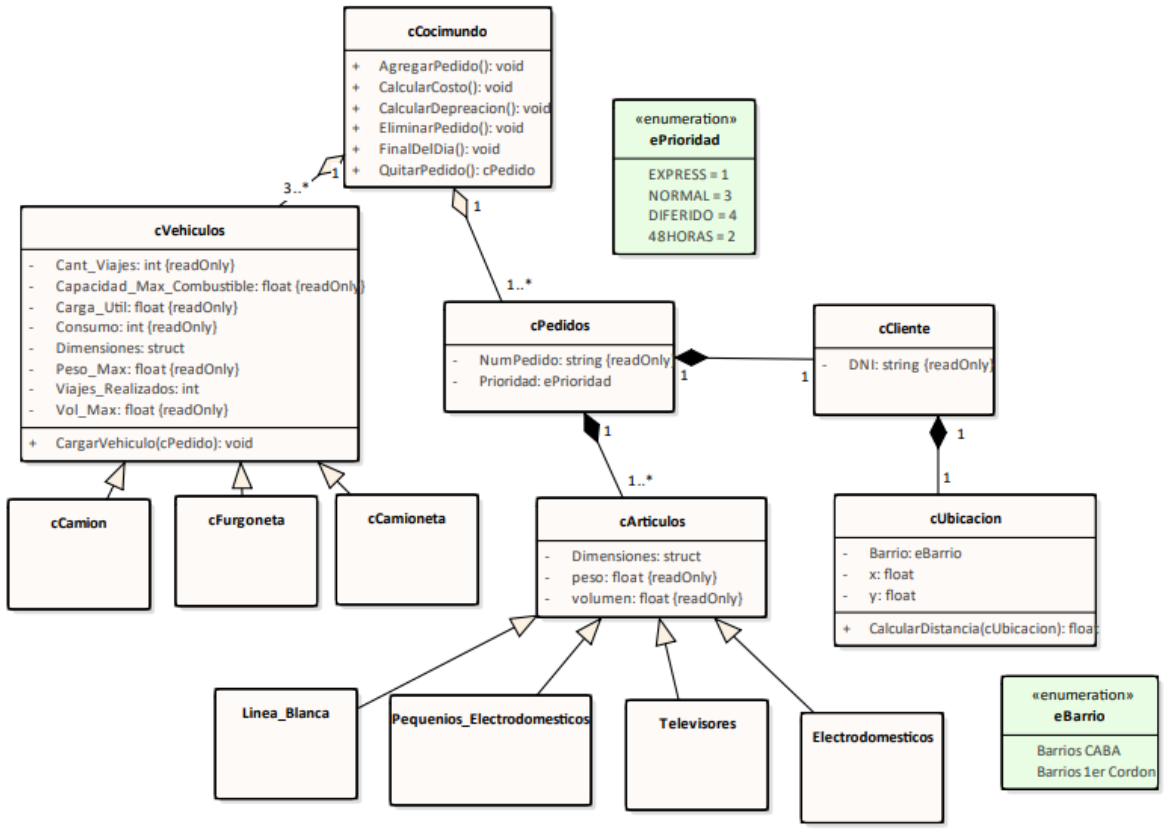
TRABAJO PRÁCTICO LABORATORIO DE PROGRAMACIÓN II



La logística de distribución se basa en el algoritmo greedy de grafos conocido como Dijkstra. Cada nodo tiene una conexión bidireccional con sus adyacentes. Nuestro source es Liniers.
Lo resolveríamos de una manera greedy, entonces nos podría dar malas soluciones en algunos casos, pero en otros podría dar una buena solución.
Lo primero que se chequea es la prioridad. Los clientes compran artículos y pueden optar por pagar pedido express (entrega en 24 horas), pedido normal (entrega en 72 horas) o pedido diferido (96 horas, sin cargo), a los pedidos con prioridad express se les da un numero 1, a los de prioridad normal un 3, y a los de prioridad diferida se les da un 4, indicando así los días restantes para que debían ser entregados. Y por cada día que pase, se les va a restar uno a este número, indicando que queda un día menos de tiempo antes de la entrega.
El plan de entrega se adapta a los productos que tienen prioridad.
A partir de esta división, trabajaríamos el cargado del camión como un problema de la mochila, priorizando que entre las cosas que van juntas y son del mismo cliente, para ahorrar nafta en el camino.

Tenemos lista de pedidos, ordenamos primero por cercanía a Liniers [tenemos una función que saca la distancia a liniers (haciendo la raíz del x al cuadrado y el y al cuadrado), y organizamos la lista segun lo más cercano a liniers primero], y después según prioridad del pedido (hacemos que los pedidos con un 1 (prioridad express) queden primeros, y después los que tienen 3 y 4).

Cargado de pedidos.
Vamos al algoritmo de la mochila (dinámico).
El objetivo es obtener el máximo beneficio de los artículos de la mochila. Cada artículo solo se puede seleccionar una vez
Llenamos primero el vehículo más pequeño.
Hacemos la matriz (o vector) para chequear para llenar este vehículo, y todo lo que entre según el volumen y el peso se carga en una sublista.
Para cargar al vehículo primero vamos a llenar la primer columna y filas con 0´s, y después ir chequeando si el pedido entra, le sumamos a la posición anterior en diagonal, sino, nos quedamos con el máximo entre la posición anterior de arriba (o abajo) o la posición anterior de la izquierda.
Una vez que nos de esto la mejor combinación de pedidos para maximizar la cantidad de pedidos cargados, ponemos esos pedidos en una lista especial, y los quitamos de la lista completa de pedidos. Luego pasamos a cargarlos al vehículo.
Hay un caso especial, que es si el último vehículo que va a salir en el día, en ese caso se cargan los pedidos express si o si, y si entra algo más, se lo intenta cargar.
Pseudocódigo:

Inicio Función Entero Mochila_Dinamica(vector ListaPedidos[], vector ListaVehiculos[])

```
Entero Matriz[N][M] //Matriz dinamica de dimensiones N y M
vector SubListaPedidos //vector dinamico
Entero j, k ➜ 0
Desde j ➜ 0 Hasta j < ListaPedidos.getCantTotal()
    Si (ListaPedidos.getItem(j).getPrioridad() == EXPRESS) //Se arma una
sublista que va a ser la mochila
        SubListaPedidos[k] = ListaPedidos.getItem(j)
        k ➜ k +1
    j ➜ j + 1

// Ahora creamos la mochila
Entero i, w
Desde i ➜ 0 Hasta i < SubListaPedidos.CantArticulos()
    Desde w ➜ 0 Hasta w < ListaVehiculos[].Volumen()
        Si i == 0 o w == 0
            Matriz[i][w] ➜ 0

        Si no si ListaPedidos[i - 1] <= w
            Matriz[i][w] ➜ max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w])
        Si no
            Matriz[i][w] ➜ K[i - 1][w]
        j ➜ j +1
i ➜ i + 1

devolver K[n][W]
```

Fin Función Mochila_Dinamica

Distribución.
Primero llenamos la camioneta más pequeña (no vamos a poner televisores en esta).
Usamos la sublista que obtuvimos del problema de la mochila.
Recorremos toda la lista, chequeando la distancia de un punto (x,y) de un pedido a Liniers, y luego desde ese buscamos el siguiente más cercano, ordenando los pedidos para que queden al fondo los pedidos de los nodos más lejanos, y más al frente los pedidos de los nodos más cercanos.
Recorre la lista y la ordena por cercanía a Liniers, después pasa al más cercano (primera posición en la lista ordenada), y vuelve a acomodar por cercanía a este nuevo nodo, y así hasta terminar la lista.
Esto asegura que una vez elegido un nodo, se van a recorrer los más cercanos a este, antes de ir a los siguientes.

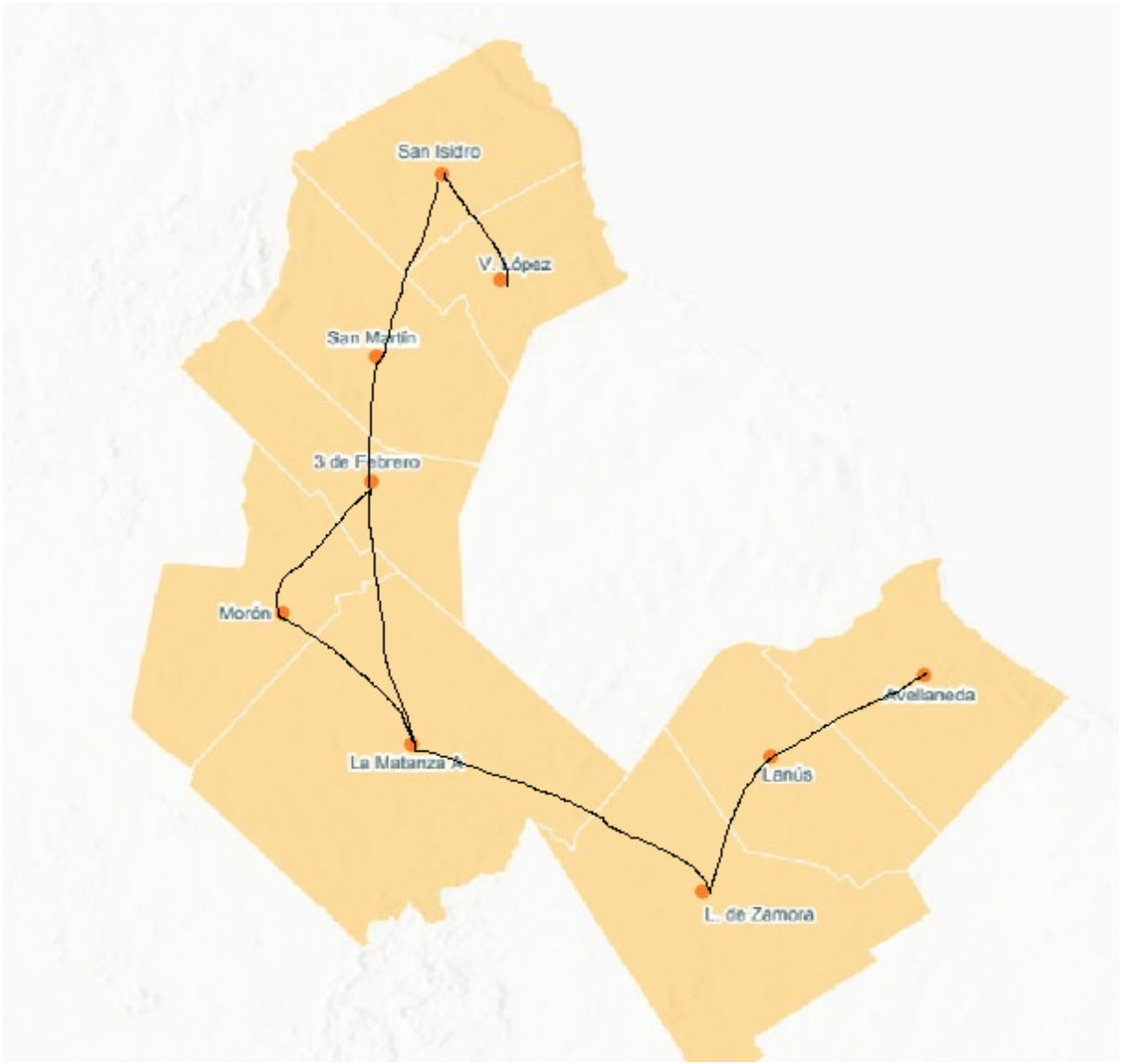
Pseudocódigo:

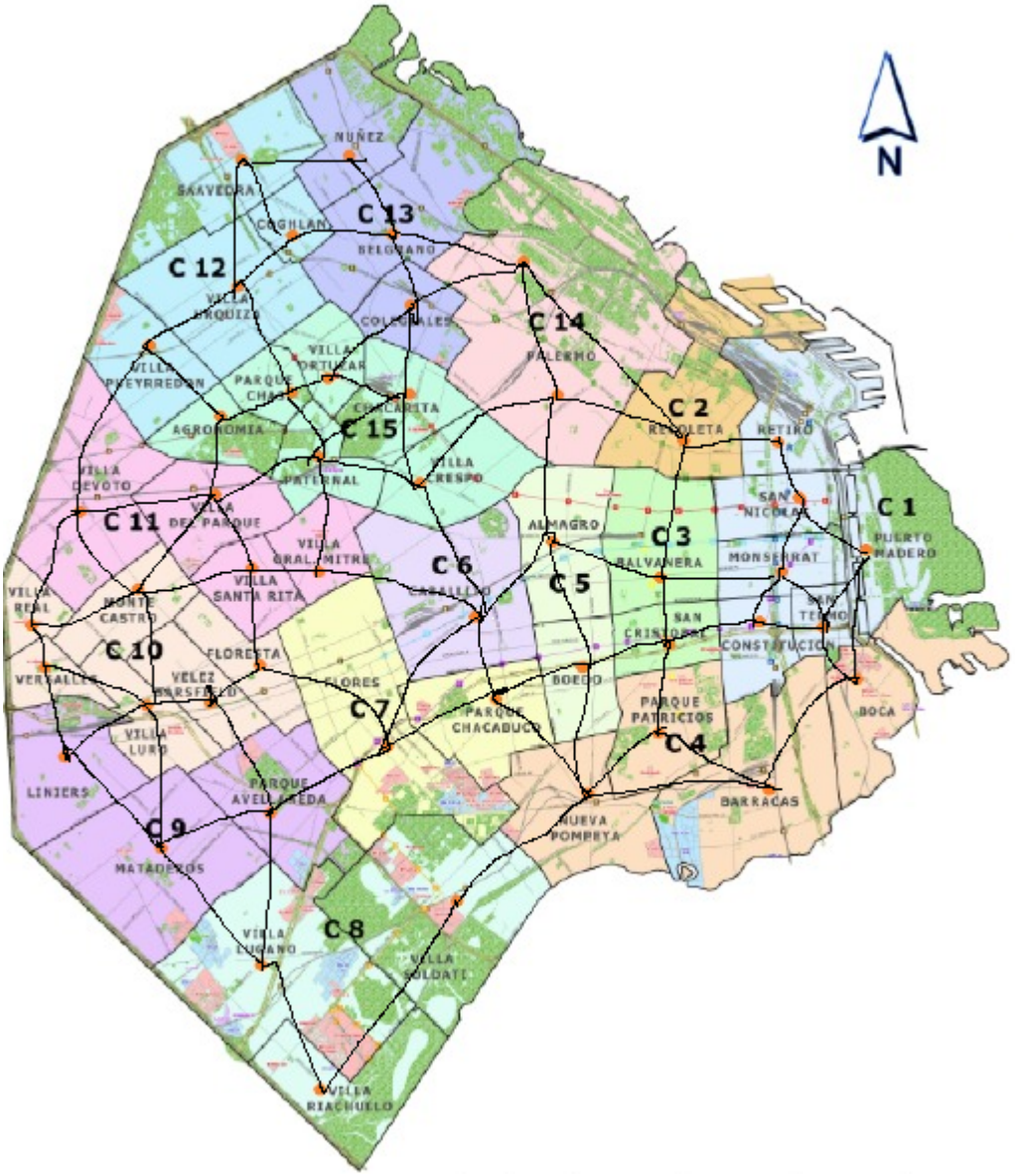
```
Inicio funcion Lista<pedidos> Distribucion(Lista<pedidos> ListaPedidos[],Lista<vehiculos> ListaVehiculso[])
    Lista<pedidos> ListaOrdenada[] //Lista final que va a tener el orden para repartir los pedidos
    entero x, y, pos, cont iguales a 0
    flotante distancia igual a 0
    desde j igual a 0 hasta ListaPedidos.cantidad hacer // bucle que recorre la lista de pedidos desde el primero hasta el anteultimo
        si j es igual a 0
            distancia igual a funciondistancia(Null, ListaPedidos[j]) // la funcion develve la distancia de un punto de un pedido a la de otro, si el primer parametro es NULL da la distancia a liniers
            pos igual a j // guardamos la posicion del nodo mas cercano
            desde i igual a 0 hasta ListaPedidos.cantidad hacer // bucle que recorre desde la ultima posicion hasta
                si distancia es mayor a funciondistancia(Null, ListaPedidos[i]) //compatramos si la distancia del nodo mas cercanos hasta ahjora es mayor que la de este nodo
                    distancia igual a funciondistancia(Null, ListaPedidos[i]) //si lo es nos la guardamos
                    pos es igual a i // y si posicion tambien
                fin si
            fin desde // el de i
            ListaOrdenada agregar(ListaPedidos[pos]) // agregamos el nodo mas cercano a la lista ordenada
            eliminar ListaPedidos[pos] // eliminamos el nodo de la lista vieja
        sino
            distancia igual a 1000 // seteamos la distancia a un numero muy grande para quedarnos con la primer distancia
            desde i igual a 0 hasta ListaPedidos.cantidad hacer
                si distancia es mayor a funciondistancia(ListaOrdenada.ultimo, ListaPedidos[i])
```

distancia igual a funciondistancia(ListaOrdenada.ultimo, ListaPedidos[i])
pos es igual a i
fin si
fin desde // el de i
ListaOrdenada agregar(ListaPedidos[pos])
eliminar ListaPedidos[pos]
fin si
fin desde // el de j
devolver ListaOrdenada // devolvemos la lista de pedidos ordenada de mas cercano a Liniers hasta el mas lejano
fin funcion Distribucion

Recomiendo pegar el pseudocódigo en un block de notas para leerlo, porque aca no se entiende nada.

El algoritmo es greedy, ya que toma la mejor opción en cada decisión, pero por eso puede dar una buena solución, como una muy mala. Este último caso puede pasar cuando dos nodos están cerca de Liniers, y toma el más cercano, pero después hay otros nodos ahora más cercanos a este primer nodo que van siendo agregados alejándonos de este otro nodo cercano a Liniers, lo cual haría que el vehículo reparta hasta algún punto lejano y después tenga que volver hasta el inicio para dejar un paquete.
Esto se puede solucionar si la lista de pedidos es realmente una lista de objetos, cada uno conteniendo pedidos cercanos entre ellos, así los dos nodos iniciales estarían dentro de un solo pedido, mejorando el funcionamiento del algoritmo.


San Isidro
V. López
San Martín
3 de Febrero
Morón
La Matanza
Avellaneda
Lanús
L. de Zamora


N
C 1
C 2
C 3
C 4
C 5
C 6
C 7
C 8
C 9
C 10
C 11
C 12
C 13
C 14
C 15
NUÑEZ
SAAVEDRA
COGHLAN
BELGRANO
VILLA URQUIZA
COLEGIALES
PALERMO
VILLA ORTUZAR
PARQUE CHAS
VILLA PUEYRREDON
AGRONOMIA
CHACARITA
RECOLETA
RETIRO
VILLA CRESPO
PATERNAL
VILLA DEVOTO
VILLA DEL PARQUE
SAN NICOLAS
PUERTO MADERO
ALMAGRO
BALVANERA
MONSERRAT
VILLA GRAL MITRE
VILLA SANTA RITA
VILLA REAL
MONTE CASTRO
SAN CRISTOBAL
CONSTITUCION
SAN TELMO
FLORESTA
VERSALLES
VELEZ SARSFIELD
FLORES
BOEDO
PARQUE PATRICIOS
PARQUE CHACABUCO
BOCA
VILLA LURO
LINIERS
PARQUE AVELLANEDA
BARRACAS
NUEVA POMPEYA
MATADEROS
VILLA LUGANO
VILLA SOLDATI
VILLA RIACHUELO