PROBLEMA

Un repartidor de una empresa de distribución de bebidas tiene que visitar a todos sus clientes cada día. Pero, al comenzar su jornada de trabajo, no conoce qué cantidad de bebidas tiene que servir a cada cliente, por lo que no puede planificar una ruta óptima para visitarlos a todos. Por tanto, nuestro repartidor decide llevar a cabo la siguiente estrategia:

- El camión parte del almacén con la máxima carga permitida rumbo a su cliente más próximo.
- El repartidor descarga las cajas de bebidas que le pide el cliente. Si no tiene suficientes cajas en el camión, le entrega todas las que tiene. Este cliente terminará de ser servido en algún otro momento a lo largo del día, cuando la estrategia de reparto vuelva a llevar al repartidor hasta él.
- Después de servir a un cliente:
    - Si quedan bebidas en el camión, el repartidor consulta su sistema de navegación basado en GPS para conocer la ruta que le lleva hasta su cliente más próximo pendiente de ser servido.
    - Si no quedan bebidas en el camión, vuelve al almacén por el camino más corto y otra vez carga el camión completamente.
- Después de cargar el camión, el repartidor consulta su sistema de navegación y se va por el camino más corto a visitar al cliente pendiente de ser servido más próximo

Implementa un subprograma que calcule y devuelva la distancia total recorrida en un día por nuestro repartidor, a partir de lo siguiente:

1. Grafo representado mediante matriz de costes con las distancias de los caminos directos entre los clientes y entre ellos y la central.
2. Capacidad máxima del camión (cantidad de cajas de bebidas).
3. Asumiremos que existe una función int Pedido() que devuelve el número de cajas que quedan por servir al cliente en el que se encuentra el repartidor. (No hay que implementarla, solo se utiliza)

Nota importante: Es absolutamente necesario definir todos los tipos de datos implicados en la resolución del problema, así como los prototipos de las operaciones utilizadas de los TADs conocidos y también los prototipos de los algoritmos de grafos utilizados de los estudiados en la asignatura.

RESOLUCIÓN

Antes de empezar con el código, para resolver el problema te piden un esbozo de la solución, una especie de mapa conceptual que guiará al profesor a través de tu código para que se entienda mejor y sea mucho más legible.
Recuerda: en EDNL y sobre todo si el corrector es De La Huerta, la legibilidad es lo más importante. Comentar tu código y que sea conceptualmente sencillo de entender es la diferencia entre suspender y aprobar esta asignatura.

Letra

1 - Como partimos desde el almacén, establecemos el almacén como nuestra situación actual, salimos con el camión cargado con toda su capacidad, y no tendremos en cuenta el almacén para el reparto ya que no es un cliente.
2 - Sacamos con el Algoritmo de Floyd la matriz de costes mínimos de ir desde cualquier punto hasta cualquier otro, porque nos será necesario para por ejemplo ir a recargar el camión cuando se nos acabe la mercancía desde cualquier punto.
3 - El cliente próximo se buscará en la matriz que devuelve Floyd, siendo este el que esté más cerca desde nuestra situación actual y además no se haya terminado su pedido.
4 - Añadimos a la distancia total recorrida el coste de ir hasta el próximo cliente y establecemos como situación actual la del cliente al que vamos a llevar mercancía.
5 - Si tenemos más mercancía de la que nos pide el cliente, simplemente restamos lo que nos pide de las existencias de nuestro camión y marcamos que ese cliente ya ha sido repartido.
   5.1 - Si el pedido es mayor o igual que la cantidad de producto que transportamos, las existencias de nuestro camión se quedan a cero y volvemos al almacén a recargar, esto es sumar a la distancia recorrida la distancia desde la situación actual hasta el almacén y volver a llenar el camión entero.
6 - Desde el punto donde nos encontremos después del paso 5 o 5.1 volvemos a buscar el cliente próximo y repetimos desde el paso 3 hasta que hayamos repartido a todos los clientes.

Código

```
#include "alg_grafoPMC.h"

typedef float tCoste;

int Pedido();

//Prototipo del algoritmo de Floyd
//matriz<tCoste> Floyd(const GrafoP<tCoste>& G, matriz<typename GrafoP<tCoste>::vertice>& P)

GrafoP<tCoste>::vertice clienteprox(const vector<tCoste>& Costes, const vector<bool>& ClientesR
epartidos)
{
    int i;
    tCoste min;
    GrafoP<tCoste>::vertice cliente;

    min = GrafoP<tCoste>::INFINITO;
    for(i=0; i<= Costes.size()-1; i++)
    {
        if (Costes[i]<min && !ClientesRepartidos[i])
```

```
        {
            min = Costes[i];
            cliente = i;
        }
    }

    return(cliente);
}

bool RepartoTerminado (const vector<bool>& ClientesRepartidos)
{
    int i;
    bool resultado;

    while(resultado && i <= ClientesRepartidos.size()-1)
    {
        resultado = ClientesRepartidos[i];
        i++;
    }

    return(resultado);
}

tCoste algoritmo(const GrafoP<tCoste>& Distancias_directas, int capacidad, const GrafoP<tCoste>
::vertice& almacen)
{
    int cantidad_restante;
    tCoste distancia_total;
    GrafoP<tCoste>::vertice proximo_cliente, situacion_actual;
    matriz<tCoste> DistanciasMinimas;
    matriz<GrafoP<tCoste>::vertice> P;
    vector<bool> ClientesRepartidos(Distancias_directas.numVert(), false);

    distancia_total = 0;
    //Cargamos el camión entero
    cantidad_restante = capacidad;
    //No tenemos en cuenta al almacén como cliente pendiente de reparto
    ClientesRepartidos[almacen] = true;
    //Partimos desde el almacén
    situacion_actual = almacen;

    //Creamos una matriz con los costes mínimos de ir desde cada sitio hasta cada sitio
    DistanciasMinimas = Floyd(Distancias_directas, P);

    while (!RepartoTerminado(ClientesRepartidos))
    {
        //Buscamos el próximo cliente (el más cercano y que no esté terminado su pedido)
        proximo_cliente = clienteprox(DistanciasMinimas[situacion_actual], ClientesRepartidos);
```

```
        //Nos movemos hasta el próximo cliente
        distancia_total += DistanciasMinimas[situacion_actual][proximo_cliente];
        situacion_actual = proximo_cliente;
        //Le damos lo que nos pide o si es más de lo que tenemos le damos todo
        if( Pedido() < cantidad_restante)
        {
            //Descargamos lo que nos piden
            cantidad_restante -= Pedido();
            //Hemos terminado con este cliente
            ClientesRepartidos[situacion_actual] = true;
        }
        else
        {
            //Vaciamos el camión con todo lo que nos queda para el cliente
            cantidad_restante = 0;
            //Vamos hasta el almacén
            distancia_total += DistanciasMinimas[situacion_actual][almacen];
            situacion_actual = almacen;
            //Recargamos el camión
            cantidad_restante = capacidad;
        }
    }

    return(distancia_total);
}
```

Disculpad la ampliación de los márgenes pero es que si no lo hacía, era mucho más difícil de entender.
Simple y llanamente he pasado a C++ lo que he dicho en la letra. No tiene mucho misterio, absolutamente nada. De hecho, creo que el ejercicio se calificó como perfecto por la sencillez en su resolución.
Sin embargo os voy a dar un par de pautas para la resolución de este y todos los ejercicios de EDNL, tips que he aprendido después de dedicarle muchísimas horas a esta asignatura. (Mis compañeros de curso lo sabrán bien, disculpad la infinidad de correos por el foro).

Tips:

1. **Divide y vencerás**. Hay muchos ejercicios en EDNL que parecen en un primer momento, inabordables. Intimidan y agobian, pero nada más lejos de la realidad, lo único que hay que hacer es "cortarlos en trocitos". Poniendo el ejemplo de este problema, a priori me pareció inabarcable, y mucho más en medio de un examen. Pero empiezas a pensar, y la única complicación que tiene es encontrar el siguiente cliente y saber si has terminado o no el reparto. Lo más complejo lo hace el algoritmo de Floyd y la función Pedido que solo tienes que utilizar. Pues bien: aprovecha al máximo la modularidad. Haz todas las pequeñas funciones que necesites para quitarte de en medio todas las complicaciones posibles. Verás como así resuelta mucho más sencillo cualquier problema.
2. **Quita del algoritmo principal todo lo que puedas**. Siguiendo la línea del primer punto. Así ganarás legibilidad. Mira por ejemplo la función RepartoTerminado. Es una absoluta gilipollez. Un bucle, fin. ¿Pero a que es mucho más claro leer en el bucle de la función principal "mientras que el reparto no se haya terminado"? Esa es la clave, eso es lo valioso, una de las cosas que más valora De la Huerta. Lo mismo con cliente_prox. El hilo principal del programa es tan fácil de leer sustituyendo esos bucles por funciones que merece la pena aislarlos. Da pereza, sí, pero se gana mucho.

3. **No te vayas por las ramas con los nombres de las variables.** Sencillez, claridad y sin dejar lugar a ambigüedades. Olvídate de usar letras sueltas o acrónimos. Si tienes una variable que guarda la distancia total, llámala distancia_total y no DT o dist_tot ni nada de eso. Recuerda: legibilidad.
4. **Primero declara, luego define**. En asignaturas como POO te van a dar mucha caña con que la declaración y la definición de una variable en C++ deben estar unidas, pero aquí olvídate de eso. Buscamos la legibilidad. Hazlo como en IP, declara todas las variables en un parrafito y en el siguiente empiezas a darle valor o dentro de los for o como veas, pero declara y luego define, nunca hagas las dos cosas a la vez en EDNL.
5. **While mejor que for**. De la Huerta odia que se utilicen fors cuando podría utrilizarse perfectamente un while. No te la juegues, usa whiles siempre que no tengas que usar un contador o algo así.
6. **Menor o igual mejor que menor.** En cuanto a los bucles, en la condición de iteración De la Huerta prefiere que se utilicen menor o igual a menores. Dice que es más lógico. Yo creo que es una tontería, pero oye, lo que sea por sacar nota. Es decir, si en un while tienes como condición de iteración que i<10, pon mejor que i<=9.
7. **Ten muy claros los algoritmos de grafos que enseñan.** Esto es primordial. Si utilizas Floyd cuando te valía Dijkstra estás fuera. Ten claro lo que hacen, sus costes, ventajas y desventajas. Para esto no hay más consejo que estudiar.

Y ya está. Que no te asuste lo que digan por ahí. EDNL es perfectamente aprobable, simplemente hay que echarle horas y estudiar, como en todas las asignaturas.

¡Mucho ánimo!