DISEÑO Y DESARROLLO DE ALGORITMOS

TRABAJO PRÁCTICO OBLIGATORIO

2° CUATRIMESTRE 2023 – VIERNES MAÑANA

UADE

Docente: RICARDO ABRAHAM WEHBE

Alumnos:

- Landajo Ramiro, legajo: 1155576
- Bernasconi Sebastian, legajo: 1146056
- Gomez lopez Yancamil, legajo: 1157428

INTRODUCCIÓN

EL PROBLEMA:

Determinación de la instalación de centros de distribución. Una compañía de logística debe analizar y determinar si efectúa la construcción o no de un conjunto de centros de distribución para almacenar y posteriormente transportar la materia prima suministrada por sus clientes. La compañía tiene 50 clientes a quienes compra la materia prima agrícola. Los clientes están localizados en distintas provincias del país. Cada cliente produce una determinada cantidad anual, que es vendida a la compañía. Para efectuar una optimización de sus costos de operación, la compañía está analizando la posible ubicación de ocho centros de distribución situados en diversos puntos del país. Está previsto que cada Centro de Distribución esté localizado sobre una ruta o vía férrea, de manera que pueda enviarse posteriormente la materia prima a los puertos para exportación. Cada centro de distribución tiene un costo anual previsto para su operación. Este costo es independiente del volumen anual de materias primas que administre. Los clientes y los posibles centros de distribución están conectados entre sí, por distintas rutas. Las rutas conectan clientes con otros clientes y con algunos centros de distribución. No todos los clientes están conectados en forma directa con un Centro de distribución, pero sí pueden estar conectados con otros clientes. La distribución es similar a la imagen 1 del apéndice. El costo total por cliente de transporte de materia prima está determinado por el costo mínimo unitario de transportar la materia prima entre dicho cliente al centro de distribución correspondiente, más el costo unitario de transportarla desde dicho centro de distribución al puerto, todo ello multiplicado por el volumen de producción anual del cliente. El costo total anual es la suma de los costos totales por cliente. Se desea encontrar un algoritmo que calcule que centros deben construirse de manera que el costo total anual sea mínimo.

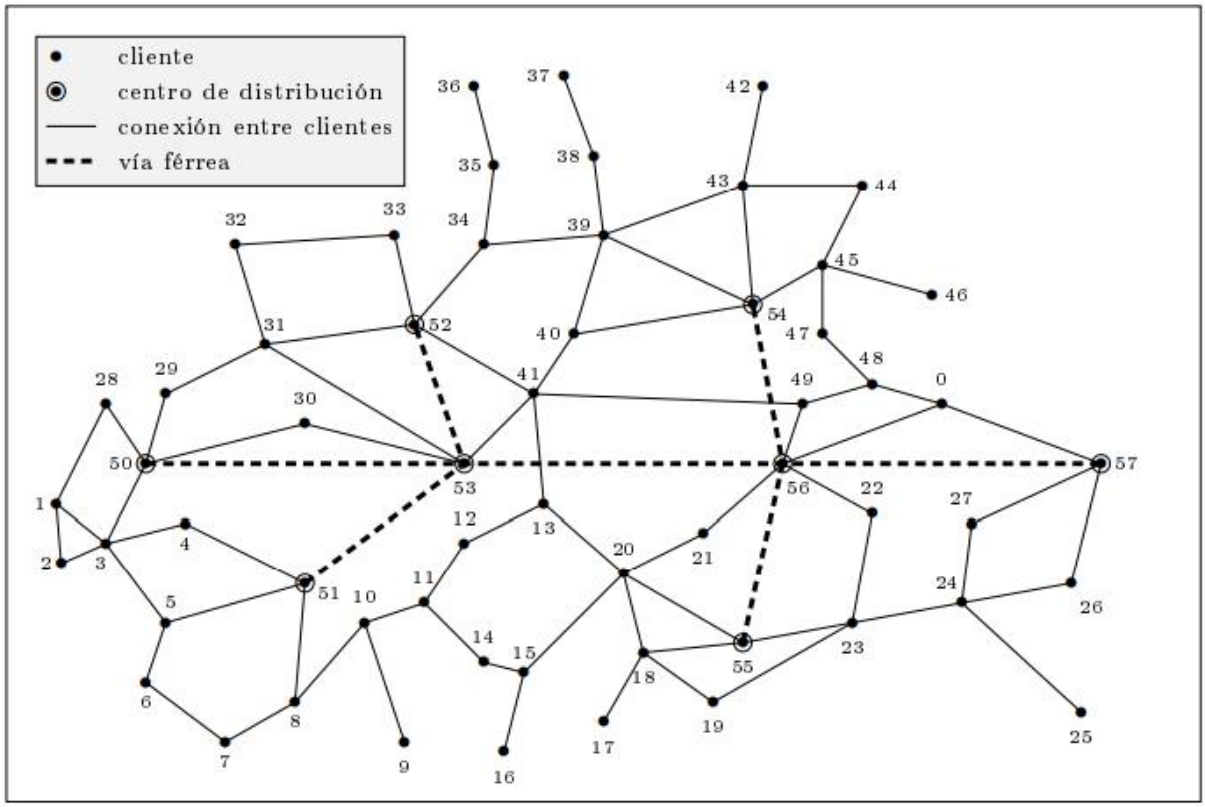

Figura 1: Clientes y centros de distribución. Los clientes están numerados de 0 a 49 y los centros de distribución de 50 a 57.

ESTRATEGIA DE RESOLUCIÓN:

Se empleó una estrategia de Branch and Bound para optimizar el transporte de la mercadería hasta el puerto. Se organizó la información de clientes, centros y el muelle como un grafo donde los clientes, centros y el muelle son nodos y los costos de transporte entre ellos son los pesos de las aristas.

Se utilizó un algoritmo de Dijkstra para calcular los costos de transporte mínimos de cada cliente a cada centro. Con esos datos se realizó el cálculo de los costos según se plantea más arriba en la problemática. La fórmula para el cálculo es la siguiente: **(costo del camino mínimo del cliente al centro) + (costo del camino del centro al muelle) * el volumen de producción del cliente.**

Una vez realizado el cálculo se creó una matriz sobre la cual se efectuaron los cálculos del algoritmo Branch and Bound. Para poder decidir qué centro construir calculamos distintas variables, estas son

- U: Representación del costo pesimista. Se calcula de la siguiente forma**: La suma de los costos mínimos de los centros de distribución construidos + los costos fijos de los centros de distribución construidos.**
- C: Representación del costo optimista. Se calcula de la siguiente forma: **La suma de los costos mínimos de los centros de distribución construidos y los eventuales + los costos fijos de los centros de distribución construidos.**
- Reducción Mínima: Representación de la mínima ganancia que se va a obtener si se construye ese centro. Se calcula de la siguiente forma: **si el costo para el cliente en ese centro NO es el menor de toda la columna la reducción mínima para ese cliente es = 0. En cambio, si el costo para el cliente en ese centro SI ES el menor, la reducción mínima para ese cliente es = la diferencia entre el menor y el segundo menor. Esto es solo sobre los centros construidos y eventuales.**
- Reducción máxima: Representación de la máxima ganancia que se va a obtener si se construye ese centro. Se calcula de la siguiente forma: **si el costo para el cliente en ese centro SI ES el máximo de la columna la reducción máxima para ese cliente es 0. Si todavía no hay ningún centro construido y el de ese centro NO ES el máximo se agarra el máximo de la columna y se hace la diferencia con el del centro a evaluar. Si ya hay un centro construido y el del centro a evaluar no es el máximo Y ES menor al mínimo de los construidos, la reducción máxima es la diferencia entre el menor de los construidos y el del centro actual.**

Al ser un algoritmo branch and bound toma algunas decisiones que en base a estas variables para descartar caminos(también se conoce como **poda**) en los que no llegaría a una solución óptima, ahorrando así mucho tiempo. Las podas que realiza son las siguientes:

- Si C > U, es decir, si lo mínimo que me va a costar(c) es mayor que lo máximo que me va a costar(U) no intento construirlo.
- Si la reducción mínima es mayor al costo fijo, se evita evaluar el camino de NO construir ese centro. Esto se debe a que sí lo minimo que voy a ganar ya me alcanza para cubrir el costo fijo del centro, no me gasto en ver la otra posibilidad
- Si la reducción máxima es menor al costo fijo del centro, se evita evaluar el camino de construir el centro. Esto se debe a que si la máxima ganancia que puedo obtener no me alcanza para cubrir el costo fijo del centro, ni intento construirlo.

Este algoritmo se detiene cuando encuentra valores iguales de U y C debido a que es la solución óptima, esto se debe a que lo máximo y mínimo que me puede costar la construcción de los centros es igual.

PSEUDOCODIGO:

```
Función CalcularU
    Entrada: centrosConstruidos: vector <numeros entero>
    Salida: CostoU : numero entero
    costoU ← 0
    Para i = 0 Hasta cantClientes - 1
        min ← infinito
        Para cada centro en centrosConstruidos
            Si matrizCostos[centro][i] < min Entonces
                min ← matrizCostos[centro][i]
            Fin Si
        Fin Para
        costoU ← costoU + min
    Fin Para
    Para cada centro en centrosConstruidos
        costoU ← costoU + costosFijosCentros[centro]
    Fin Para
    u ← costoU
    Devolver costoU
Fin Función

Función CalcularC
    Entrada: centrosConstruidos:vector de <numeros enteros> ,
centrosEventuales : vector <numeros enteros>
    Salida: CostoC : numero entero
    costoC ← 0
    Para i = 0 Hasta cantClientes - 1
        min ← infinito
        Para cada centro en centrosConstruidos
```

```
            Si matrizCostos[centro][i] < min Entonces
                min ← matrizCostos[centro][i]
            Fin Si
        Fin Para
        Para cada centro en centrosEventuales
            Si matrizCostos[centro][i] < min Entonces
                min ← matrizCostos[centro][i]
            Fin Si
        Fin Para
        costoC ← costoC + min
    Fin Para
    Para cada centro en centrosConstruidos
        costoC ← costoC + costosFijosCentros[centro]
    Fin Para
    Devolver costoC
Fin Función

Función CalcularRedMin
    Entrada: centroEvaluado: numero entero , centrosConsiderados:
vector <numeros enteros>
    Salida: RedMin : numero entero
    redMin ← 0
    Para i = 0 Hasta cantClientes - 1
        valorCentroEvaluado ← matrizCostos[centroEvaluado][i]
        segundoMin ← infinito
        Para cada centro en centrosConsiderados
            Si matrizCostos[centro][i] < valorCentroEvaluado Entonces
                valorCentroEvaluado ← -1
                break
            Sino Si matrizCostos[centro][i] < segundoMin Entonces
                segundoMin ← matrizCostos[centro][i]
            Fin Si
        Fin Para
```

```
            Si matrizCostos[centroEvaluado][i] = valorCentroEvaluado
Entonces
                redMin ← redMin + segundoMin - valorCentroEvaluado
            Fin Si
        Fin Para
        Devolver redMin
    Fin Función

    Función CalcularRedMax(centroEvaluado, centrosConsiderados,
centrosConstruidos)
        Entrada: centroEvaluado: numero entero , centrosConsiderados:
vector <numeros enteros> ,centrosConstruidos: vector <numeros enteros>
        Salida: numero entero
        redMax ← 0
        Si centrosConstruidos = null Entonces
            Para i = 0 Hasta cantClientes - 1
                valorCentroEvaluado ← matrizCostos[centroEvaluado][i]
                max ← infinito negativo
                Para cada centro en centrosConsiderados
                    Si matrizCostos[centro][i] > max Entonces
                        max ← matrizCostos[centro][i]
                    Fin Si
                Fin Para
                Si valorCentroEvaluado < max Entonces
                    redMax ← redMax + max - valorCentroEvaluado
                Fin Si
            Fin Para
        Sino
            Para i = 0 Hasta cantClientes - 1
                valorCentroEvaluado ← matrizCostos[centroEvaluado][i]
                min ← infinito
                Para cada centro en centrosConstruidos
                    Si matrizCostos[centro][i] < min Entonces
                        min ← matrizCostos[centro][i]
```

```
                Fin Si
            Fin Para
            Si valorCentroEvaluado < min Entonces
                redMax ← redMax + min - valorCentroEvaluado
            Fin Si
        Fin Para
    Fin Si
    Devolver redMax
  Fin Función
Fin Algoritmo
```

```
procedimiento QueCentrosConstruyo:
  Entrada: centros : arreglo de numeros enteros
  Salida: nada
  Imprimir("Se construyen los centros: ")
  Para i=0 hasta cantCentros-1:
    Si centros[i] = 1 entonces:
      Imprimir("Centro: " + i)
     fin si
  fin para

  Imprimir("Arreglo x final: ")
  Para i =0 a hasta cantCentros-1:
    Si i =7 entonces:
      Imprimir(centros[i])
    fin si
    Sino entonces:
      Imprimir(centros[i] + ", ")
    fin sino
 fin para

  ImprimirNuevaLinea()
```

Fin procedimiento

```
Procedimiento CentroPorCliente(centros):
entrada: centros: arreglo de numeros enteros
salida: nada
    centrosConstruidosFinales ← Crear nueva Lista de numeros enteros

    Para i=0 hasta cantCentros-1:
        Si centros[i] = 1 entonces:
            Agregar i a centrosConstruidosFinales
        fin si
    fin para

    Para cada cliente de 0 a cantClientes:
        posicionCentroCostoMinimo ← -1
        min ← ValorMuyGrande
        Para cada centro j en centrosConstruidosFinales:
            Si matrizCostos[j][cliente] < min entonces:
                min ← matrizCostos[j][cliente]
                posicionCentroCostoMinimo ← j
            fin si
        fin para
       Imprimir("El cliente " + cliente + " entrega su mercadería al centro: "
+ posicionCentroCostoMinimo +
                 ", con costo " +
matrizCostos[posicionCentroCostoMinimo][cliente])
    fin para

FinProcedimiento
```

Main

```
public class Main {
   cantCentros ← 8;
   cantClientes ← 50;
   volumenProduccionClientes ← arreglo de numeros enteros largo
[cantClientes];
   matrizCostos ← matriz de numeros
enteros[cantCentros][cantClientes];
   costosFijosCentros ← array de numeros enteros[cantCentros];
   costosCentrosAlPuerto ← array de numeros enteros[cantCentros];

CalcularCentrosConstruir()
Entrada: nada
Salida:  arreglo de numeros enteros
       costoAnual ←  -1
       x   ← arreglo de numeros enteros con lenght = cantCentros
      colaP ← cola de prioridad de nodos

       centrosConstruidos ← array de numeros enteros
       centrosConsiderados ← array de numeros enteros

       // loop para agregar los centros a la lista de centros considerados
       Para i=0  hasta cantCentro-1
          agregar i al array centros considerados
       fin para
       centroEvaluado ← 0
       c ←1;
       int u ← Infinito
       int uPoda ← Infinito
       raiz ← Crear nuevo Nodo(x, u, CalcularC(centrosConstruidos,
centrosConsiderados)

       agregar el nodo raiz a la colaP
```

```
Mientras u != c
  nodoEvaluado ← nodo en primera posicion de colaP y se eliminó el nodo de la cola de prioridad
    x ←nodoEvaluado.x ;
    Para i = 0  hasta  cantCentros-1
      si  x[i] = 0 entonces
        entonces centroEvaluado = i;
                  break;
      fin si
    fin para

    Si (nodoEvaluado.c < nodoEvaluado.u)   entonces
      centrosConstruidos ← vacio
      centrosConsiderados← vacio
  fin si
      Para  i = 0  hasta cantCentros-1
        Si x[i] = 1  entonces
          agrego i a centrosConstruidos
          agrego i a centrosConsiderados
        Fin si
        sino si x[i] = 0  entonces
          si  i = centroEvaluado  entonces{
            continuar
          fin si
          agrego i a centrosConsiderados
        fin si
      fin para
      nodoEvaluado.redMin ← CalcularRedMin(centroEvaluado, centrosConsiderados)
      Si nodoEvaluado.redMin > costosFijosCentros[centroEvaluado] entonces
        x[centroEvaluado] ← 1
        centrosConstruidos ← vacio
        centrosConsiderados ← vacio
```

```
                Para   i = 0  hasta cantCentros-1
                    si x[i] = 1 entonces
                        agregar i a  centrosConstruidos
                    fin si
                   Sino Si  x[i] = 0 entonces
                       Si (i = centroEvaluado) entonces
                            continuar;
                       fin si
                        agregar i a centrosConsiderados
                   fin si
                fin para

                int uTemp ← CalcularU(centrosConstruidos);  // calculo u y
c para asignarselo al nuevo nodo
                c ← CalcularC(centrosConstruidos, centrosConsiderados);
                si (uTemp < uPoda) entonces
                    uPoda ← uTemp;
                    u = uTemp;
                    Siguiente ← Crear nuevo Nodo(x, uPoda, c);
                    agregar Siguiente a colaP
                fin si
                sino entonces {
                    u ← uTemp
                   Siguiente ← Crear nuevo Nodo(x, uTemp, c)
                    agrego Siguiente a colaP
                fin sino
                centroEvaluado ← centroEvaluado+1
            fin  para
            sino  entonces {
                centrosConstruidos ← vacio
                centrosConsiderados ← vacio

                para i = 0 hasta cantCentros-1  {
                    si  x[i] = 1  entonces {
```

```
                    agregar i a  centrosConstruidos
                    agregar i a  centrosConsiderados
                fin si
                sino  si x[i] = 0   entonces{
                    si (i = centroEvaluado) entonces {
                        continuar;
                    fin si
                    agregar i a centrosConsiderados
                fin si
            fin para
            nodoEvaluado.redMax ← CalcularRedMax(centroEvaluado,
centrosConsiderados, centrosConstruidos)
            si (nodoEvaluado.redMax <
costosFijosCentros[centroEvaluado])                    entonces {
                x[centroEvaluado] ← -1

             centrosConstruidos ← vacio
             centrosConsiderados ← vacio

                Para  i = 0 hasta cantCentros-1   {
                    si (x[i] == 1) entonces
                         agregar i a centrosConstrudos

                    fin si
                    sino si x[i] = 0)  entonces{
                        si (i = centroEvaluado) entonces
                            continuar;
                        fin si
                        agregar i a centrosConsiderados
                    fin si
                fin para

                int uTemp ← CalcularU(centrosConstruidos)
            c ← CalcularC(centrosConstruidos, centrosConsiderados)
```

```
    si  (uTemp < uPoda)  entonces {
        uPoda ← uTemp;
        u ← uTemp;
        Nodo siguiente ← new Nodo(x, uPoda, c)
        agregar siguiente a colaP
    fin si
    sino
        u ← uTemp;
        Nodo siguiente ← new Nodo(x, uTemp, c);
        agregar siguiente a colaP

    fin sino
    centroEvaluado ← centroEvaluado +1
fin si
sino entonces {
    xConstruyo ← Crear array [cantCentros]
   para   i = 0 hasta  cantCentros-1
         xConstruyo[i] ← x[i];
    fin para
    xConstruyo[centroEvaluado] ← 1
    centrosConstruidos ← vacio
    centrosConsiderados ← vacio
    Para   i = 0  hasta cantCentros -1
        si  (xConstruyo[i] = 1)  entonces
            agregar i a centrosConstruidos
        fin si
        sino si  (xConstruyo[i] == 0)  entonces
            si (i = centroEvaluado) entonces
                continuar;
            fin si
            agregar i a centrosConsiderados
        fin si
    fin para
```

```
int uConstruido ← CalcularU(centrosConstruidos)
c ← CalcularC(centrosConstruidos, centrosConsiderados);
Si (uConstruido < u)  entonces
   uPoda ← uConstruido;
   u ← uConstruido;
   construyo ← crear Nodo(xConstruyo, uPoda, c);
   agregar construyo a colaP
fin si
sino
   u ← uConstruido
   construyo ← crear Nodo(xConstruyo, uConstruido, c)
   agregar construyo a colaP
fin sino
```

```
xNoConstruyo ← crear nuevo array de numeros
enteros[cantCentros];
para ( i = 0 hasta cantCentros-1 ) entonces
   xNoConstruyo[i] ←x[i];
fin para
 xNoConstruyo[centroEvaluado] ← -1
 centrosConstruidos ← vacio
 centrosConsiderados ← vacio

   para ( i = 0 hasta cantCentros-1) {
      si (xNoConstruyo[i] = 1) entonces
         agregar i a centrosConstruidos
      fin si
   sino si(xNoConstruyo[i] = 0)  entoncees{
      si (i = centroEvaluado) entonces{
         continuar;
      fin si
      agregar i a centrosConsiderados
```

```
                    fin si
                fin para

                int uNoConstruido = CalcularU(centrosConstruidos);      //
calculo u y c para asignarselo al nuevo nodo
                c ← CalcularC(centrosConstruidos, centrosConsiderados);
                si (uNoConstruido < u)  entonces{
                    uPoda ← uNoConstruido;
                    u ← uNoConstruido;
                    construyo ← Crear nuevo Nodo(xNoConstruyo, u, c)
                    agregar construyo a colaP
                fin si
                sino {
                    u ← uNoConstruido;
                    noConstruyo ← Crear nuevo Nodo(xNoConstruyo,
uNoConstruido, c);
                    agregar construyo a colaP
                fin sino

                    centroEvaluado← CentroEvaluado+1
              fin sino
            fin sino
          fin si
           sino {
             x[centroEvaluado] ← -1;
              centrosConstruidos ← vacio
              centrosConsiderados ← vacio
               para ( i = 0  hasta cantCentros-1 ) entonces  {
                    si (x[i] = 1)  entonces
                        agregar i a centrosConstruidos
                     fin si
                else si (x[i] = 0){
                    si (i = centroEvaluado) entonces
                        continuar;
```

```
                fin si
                agregar i a centrosConsiderados
            fin si
        fin para

        int uTemp ← CalcularU(centrosConstruidos);
       c ←  CalcularC(centrosConstruidos, centrosConsiderados);
        si (uTemp < u)
            uPoda ←  uTemp;
            u ←  uTemp;
           siguiente ←  Crear nuevo Nodo(x, u, c);
            agregar siguiente a colaP
        fin si

        sino {
            u ← uTemp;
            siguiente ←  Crear nuevo Nodo(x, u, c);
            agregar siguiente a colaP
        fin sino
        centroEvaluado      ← CentroEvaluado+1        ;
    fin sino
  fin mientras

  Devolver x;
fin metodo
```

ANÁLISIS DE COSTOS DEL ALGORITMO:

El costo del algoritmo total sería de O(2^n) donde n es la cantidad de centros. Estamos utilizando Dijkstra, que en este caso tiene una complejidad temporal de O(V^2) donde V es la cantidad de vértices, pero no se tendria mucho en cuenta ya que la complejidad del algoritmo principal de calcular qué centros se van a construir es mayor, siendo esta exponencial. Esto se debe a que, para cada nodo que se analiza dentro del loop while, siempre hay dos opciones, o se construye el centro o no se construye. Si desglosamos todas las ramas que esto generaría 2 opciones por cada centro, ejemplo con 2 centros tendriamos 4 opciones (2^2), donde x valdrian los siguientes valores [1,1], [1,-1], [-1,1], [-1,-1]. Algo que cabe aclarar, es que debido a que estamos utilizando un algoritmo de branch and bound y no backtracking puro, se puede generar una poda de algunas ramas dentro del algoritmo, así reduciendo la cantidad de opciones a considerar, y optimizando el tiempo que tarda en resolver el problema.

CONCLUSIONES:

Aplicando la estrategia anteriormente descrita sobre los datos suministrados por el cliente, se llegó a la conclusión que la configuración más conveniente para el caso es la construcción de los centros 0, 2 y 7. Esto nos daría un costo total anual de 14250.

BIBLIOGRAFIA:

- Implementación de Dijkstra: https://www.geeksforgeeks.org/java-program-for-dijkstras-shortest-path-algorithm-greedy-algo-7/
- Implementacion Colas con Prioridad de Java: https://www.geeksforgeeks.org/priority-queue-class-in-java/