DOCUMENTACION PROYECTO III
ALGORTIMICA I

INTEGRANTES PROYECTO:

- JESUS CIVICO LOBATO
- IGNACIO PUIG MARGALEF

INTRODUCCIÓN

Comenzando con la introducción de nuestro proyecto, vamos a comentar la traza que hemos seguido (es decir, que hemos hecho), qué resultados hemos obtenido y nuestras conclusiones tras hacer los algoritmos.

TRAZA hasta llegar a la implementación de los algoritmos:

En primer lugar, para comenzar, se nos proporcionaban una serie de instancias en forma de ficheros. **Nuestro objetivo principal debía ser construir las matrices de distancias de las instancias proporcionadas**. Por lo tanto, y teniendo esto en mente, nuestro primer objetivo para ello se basó en hacer una *lectura de todos los ficheros* de la carpeta data (que contiene los ficheros con las instancias), la cual introducimos en la carpeta del proyecto (tal y como se pedía). Para realizar esto, nos hemos ayudado de algunas clases que nos facilita el lenguaje java.

Habiendo leído los 8 ficheros, nos centramos ahora en leer el *contenido* de cada uno de ellos. Observando los ficheros, nos facilitan la información del número de ciudades del fichero, las ciudades enumeradas y las coordenadas x e y de las ciudades (entre otras cosas).

Tal y como íbamos leyendo las ciudades, las íbamos metiendo en una matriz. Esta *matriz* tiene n filas, correspondientes a las n ciudades del fichero que estemos leyendo en ese momento, y 3 columnas (número ciudad, coordenada x, coordenada y).

Teniendo ya una matriz construida con la información necesaria del fichero que estemos leyendo, finalizaremos la primera parte de la traza llegando al objetivo de construir la matriz de distancias.

Para obtener las *matrices de distancias*, hemos recorrido una matriz de tamaño nxn (recordar que n era el número de filas, que era el número de ciudades del fichero actual) y hemos rellenado la matriz calculando la distancia de una ciudad con todas las demás ciudades existentes de la instancia .

NOTA: en nuestro intento de ahorrarnos tiempo de ejecución, tuvimos en cuenta que no iba a ser necesario recorrer las nxn posiciones de la matriz: recorrimos solo los elementos que se situaban por encima de la diagonal principal (m[i][j]), y poniendo la misma distancia en la posición pero esta vez por debajo (m[j][i]). La diagonal principal está llena de ceros.

Llegados a este punto, tendremos nuestra matriz de distancias creada.

Tras esto, pasaremos a nuestro siguiente objetivo: implementar los algoritmos de backtracking (vuelta atras) y divide y venceras, adaptandolos al problema que queremos solucionar, el TSP

El problema del agente viajero, Travel Salesman Problem (TSP) es uno de los problemas clásicos de optimización, donde se considera un agente viajero que debe visitar un conjunto de ciudades considerando:

i) Iniciar y terminar su recorrido en una ciudad;

ii) Cada ciudad debe visitarse una única vez;

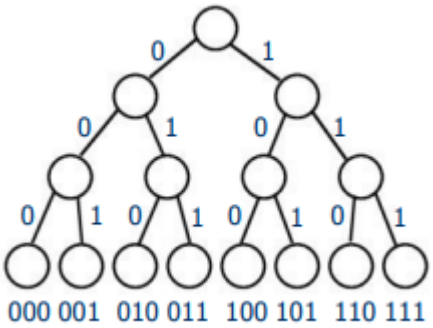
iii) Se deben visitar todas las ciudades;

iv) Encontrar una secuencia de visitas óptima que garantice un tour para el cual la distancia total recorrida sea mínima.
Vamos a hablar de que hemos hecho, cómo y algunas conclusiones a continuación.

ALGORITMOS

- BACKTRACKING

El primer algoritmo usado es *backtracking o vuelta atrás*. Con este algoritmo lograremos algo parecido a lo que buscábamos con el algoritmo exhaustivo o de fuerza bruta: **comprobar todos los caminos posibles**. Para entender bien como se adaptaria al tsp, tenemos que imaginarnos los recorridos en forma de **árbol** tal y como se ve en la siguiente figura:



Como vemos, desde cada nodo (ciudad), salen dos aristas. La arista representada con un 0, sería la arista por la que recorreremos todas las ciudades sin marcar dicha ciudad en ese momento (es decir, la marcaremos más tarde, lo que significa que la distancia será distinta, ya que estamos calculando desde otra ciudad).
La arista representada con un 1 sería el camino por el que marcamos esa ciudad y seguimos recorriendo todas las demás.
De esta manera y, en forma de árbol iremos recorriendo todos los caminos posibles.
¿Pero, y qué más potencial le podemos sacar a esta técnica?
Vamos a hablar de lo que suelen conocer cómo “podar” el árbol.
Esto consiste en aprovechar que tenemos en todo momento la distancia mínima (es decir, un camino mínimo), e ir comprobando cada vez que seleccionemos la siguiente ciudad a ir que la distancia del camino actual no sobrepase la distancia del camino mínimo ya calculado. Es aquí cuando realizaremos la vuelta atras forzada (este camino ya no nos interesa).

Para verlo más claro, vamos a ver un pseudocodigo de una posible implementación:

```
backtracking() {

      si contador igual numero Ciudades y matrizDistancia[posActual][0] > 0 {
         distanciaMin = minimo entre entre distancia minima y el coste total del camino
         devolver distanciaMin;
      }

     Bucle (hasta numeroCiudades) {
        float costeActual = coste que teniamos  + coste de ir a la ciudad actual;

        si costeActual es mayor que la distancia minima {
                devolver distancia minima;
        }

        si ciudad en la que nos encontramos no esta visitada y  matrizDistancia[posActual][i]>0  {
            ciudad actual la ponemos a true;
            distanciaMin = backtracking(matrizDistancia, camino, i, numCiudades, contador + 1,costeActual, distanciaMin)
            ciudad actual la ponemos a false;
        }
     }
     devolver distancia minima;
}
```

EXPERIMENTANDO CON BACKTRACKING

Como hemos mencionado anteriormente, el backtracking va a buscar algo parecido al algoritmo de fuerza bruta: comprobar todas las combinaciones posibles.
Esto, nos generará dos problemas:

- **Un alto tiempo de ejecución**. Esto se puede entender viendo la complejidad que tendría el algoritmo. Para n ciudades, la complejidad en cuanto a tiempo sería de n!. Esto quiere decir que, en el caso de nuestras instancias TSP (siendo la más pequeña Berlin52), el tiempo va a excederse lo suficiente como para que sea muy costoso resolver el problema con este algoritmo. Con Berlín52, el tiempo podríamos verlo calculando: $\text{factorial}(52) = 8.0658175170944E + 67$
  Es por ello que, el cálculo de la distancia mínima para las instancias proporcionadas, se puede demorar horas.

- **Empleo de memoria**. Existen implementaciones de este algoritmo de manera iterativa. No son recomendables en todos los casos. Considerando nuestra implementación recursiva, a causa de las numerosas llamadas a función que vamos a realizar con vuelta atrás, el empleo de pila es elevado.

CONCLUSIONES DEL USO BACKTRACKING EN EL PROBLEMA TSP

Es por ello y, comentando la **conclusión de la resolución de este problema** con el algoritmo de backtracking o vuelta atrás, hemos podido observar:
Es un algoritmo muy eficaz para encontrar soluciones óptimas. Pero, esto lo hará sacrificando mucho tiempo de ejecución y empleando mucha memoria para la resolución de las instancias proporcionadas. Es por ello que analizar los resultados y graficarlos nos es imposible con la capacidad de computación que manejamos. Sabemos que, la forma de ayudar a resolverlo, sería realizar la implementación de este algoritmo adaptado al tsp con la implementación clásica de backtracking. Es por ello que, podemos concluir que es un algoritmo eficaz en cuanto a la precisión de las soluciones que nos de, pero ineficaz debido a su complejidad temporal de N! y a su elevado uso de memoria.