David Max

# DISEÑO DE ALGORITMOS

# DOCUMENTACION – PRACTICA 3

## Proceso del código

Para realizar este proyecto de 'Travelling Salesman Problem', se ha seguido la idea de implementación propuesta en el libro de Dasgupta, concretamente con el método 'branch and bound'.

Lo primero que se hará, será leer el fichero con el método readFile(), el cual nos creará una matriz de distancias de tamaño [nNodos X nNodos], y la inicializará con valores '-1' en todas las posiciones, de tal forma que al terminar de leer las aristas del fichero, en las coordenadas [A][B], y al ser grafos no diridos también [B][A], si existía esa arista en el fichero, se guardará su peso; y si no existía la arista tendrá un valor de '-1'.
Tras esto se creará un ensayo como resultado inicial, que estará adulterado de tal forma que siempre será el peor, ya que tiene un coste de MAX_INT; y un ensayo inicial con todos sus atributos a 0.
Una vez hecho esto, pondremos el primer elemento de la lista de visitados del ensayo a 1, para decir que será el que visitaremos inminentemente (siempre será el origen, nodo 0), y añadimos este nuevo ensayo a nuestra cola.
Esta cola, se tratará de una PriorityQueue, que es un tipo de cola que nos proporciona Java, la cual conforme vas introduciendo elementos, en vez de colocarlos al final de la lista como en las colas normales, los introducira ordenados en base a una propiedad que se ha implementado con un 'CompareTo()' en la clase 'Ensayo', la cual comparará los atributos 'coste' de los ensayos que le entran, y siempre nos devolverá el ensayo con menor coste que tenga almacenado, cuando saquemos un elemento. Esto nos ayudará a encontrar antes una solución, ya que se ordenarán en base al lowerbound que calculemos, y por lo tanto al asegurarnos encontrar un ensayo con un coste 'bajo' y 'pronto' (lo entrecomillo ya que no es una ciencia exacta, puede ser que dependiendo del fichero de entrada, esta estrategia no funcione bien del todo), muchos de los ensayos que se harían con una cola normal nos los saltaremos ya que tendrán un lowerbound mayor al coste de nuestro mejor ensayo, y se ahorrará tiempo.

Ahora comenzará el 'branch and bound', con un 'while' que loopeara mientras queden ensayos en la cola.
Se obtiene el primer ensayo de la cola y lo guardamos en 'actual', y si el coste de actual, que al principio será 0, y conforme avance el algoritmo, esto se sustituira con el 'lowebound' que calcularemos para ensayo; es menor que el coste del 'mejorEnsayo' guardado, estudiaremos ese ensayo.

Para ello primero añadimos el siguiente nodo que le toca al ensayo, a su camino, este nodo estará almacenado en el atributo 'destino' del ensayo. Seguido, se hará un 'for' en el que una variable 'i' tomará el valor de todos los nodos del grafo; y siempre que ese nodo 'i', no sea el mismo que el nodo 'destino', que exista una arista entre 'destino' e 'i', y que el nodo 'i' no este visitado (estas serán los 3 requisitos para expandir un ensayo por un nodo 'i' concreto), seguirá el proceso del código, en caso de no cumplir alguna de las condiciones, se terminará de estudiar ese ensayo.

Si cumplía todo eso, podemos añadir el nodo 'i' al camino de 'actual', y marcar 'i' como visitado en 'actual'. El avanzar por ese camino implica que estudiaremos un nuevo ensayo, y por lo tanto aumentaremos en 1 el acumulador 'index' que utilizaremos al final para mostrar cuantos ensayos se crearon en la ejecución del programa.

Una vez hecho esto, se calculará el heuristico propuesto en el libro de Dasgupta, que se basa en encontrar el arbol de recubrimiento mínimo de los nodos no visitados (realizado con el código de la primera entrega), la arista de menor peso desde el origen hasta ese 'MST', la arista de menor peso desde el destino actual del camino (es decir, su 'último' nodo), y el peso acumulado en el camino ya recorrido.
Sumando todos esos valores, obtendremos a un valor 'mínimo' de recorrido al cual se puede llegar extendiendo el ensayo por esa rama, a este valor lo llamaremos 'lowerbound'.

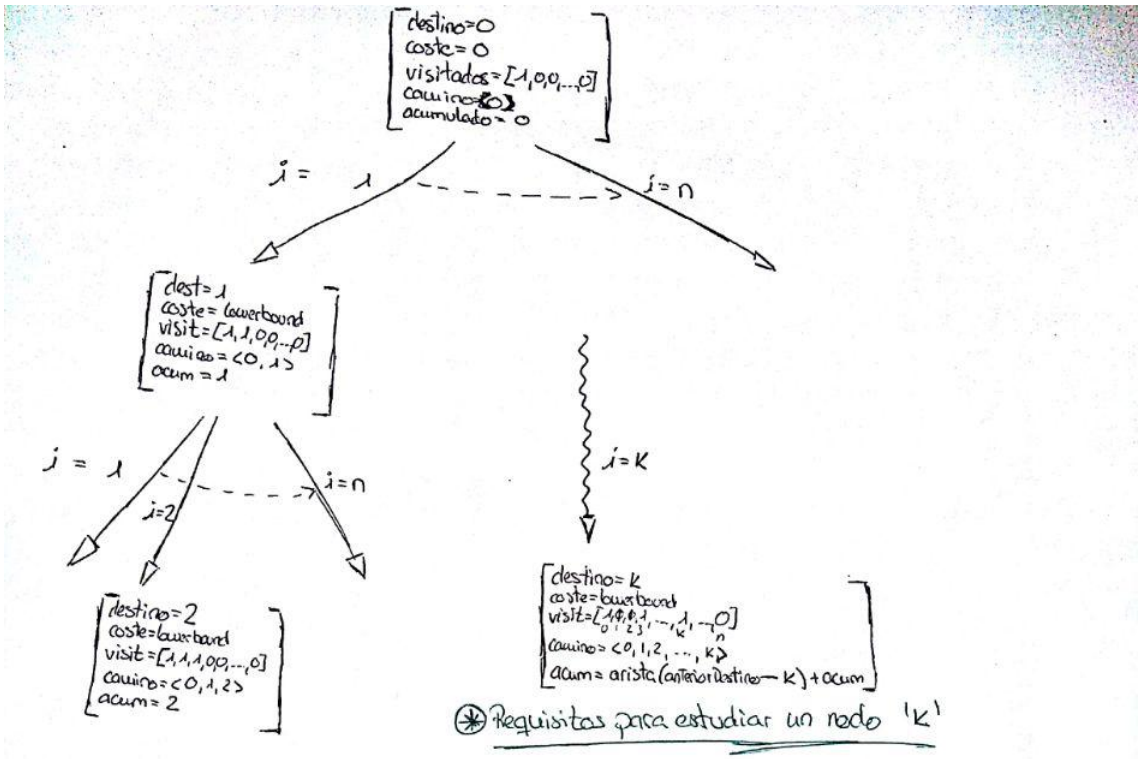
Una vez calculado ese valor, solo nos queda comprobar si el camino al añadirle el nodo 'i', se convertía en posible solución. Esto lo comprobaremos mirando el tamaño del camino del ensayo 'actual', que en caso de tener un tamaño igual a 'nNodos', podría ser una solución.
Una vez comprobado esto, quedará comprobar 2 cosas más. La primera, que el coste del mejor ensayo actual guardado, sea mayor que el peso acumulado en el camino + la arista del ultimo nodo al origen; y segundo, que exista una arista desde el nodo 'i', hasta el nodo inicial 0 (se comprobara con la matriz de distancias). Si el ensayo cumple esas dos condiciones, podemos decir que es la nueva solución óptima, y si no lo es simplemente se desechará.

En caso de que el ensayo, al añadirle el nodo 'i' no tuviera 'nNodos' nodos, tendrá que cumplir un requisito para que vuelva a entrar en la cola a ser estudiado un nivel más a fondo, y este requisito es que el lowebound que hemos calculado para ese ensayo, sea menor que el coste del mejor ensayo actual, ya que eso implicará que tal vez podamos llegar por esa rama a una solución mas óptima.
En ese caso, eliminaremos del el nodo 'i' que hemos añadido al camino de 'actual' ya que entrará como destino del nuevo ensayo y se añadirá después; clonaremos la lista de visitados de 'actual', y clonaremos también el camino una vez eliminado el nodo 'i'.
Ahora, para reestudiar este ensayo, crearemos un nuevo objeto 'Ensayo', el cual se inicializará con un destino 'i', un coste 'lowerbound', una lista de visitados 'v' (clonada del ensayo actual), un camino 'c' (clonado del ensayo actual), y un coste acumulado de camino de 'acumCamino' (calculado para el lowerbound).

Y una vez hecho esto, haya sido posible candidato o no, y tuviera un lowebound aceptable o no, se pondrá la posicion ‘i’ del array de visitados en ‘actual’ a 0, para explorar la rama sin añadir ese elemento, que la ‘i’ pase a ser ‘i+1’, y tengamos un nuevo ensayo.

Cuando la pila se quede vacía podremos asegurar que en ‘mejorEnsayo’ se encontrará el ensayo con el peso más óptimo.

# Árbol de ensayos




⊛ Requisitos para estudiar un nodo 'K'

1) K != destino
2) Que exista la arista (destino – i)
3) Que K no esté visitado.
// Si se cumple esto se añadirá 'K' al camino y se visitará 'K'.
// Se calcula el MST de no visit., nuevoAcum y el lowerbound.
4) Si después de esto, el tamaño del camino == nNodos
   4.1) Si 'mejorEnsayo.coste' > 'nuevoAcum' + arista(destino-origen) => Actualizar 'mejorEnsayo' y existe una arista (destino-origen)
   4.2) Si no cumplía alguna condición se comprueba que el lowerbound sea < 'mejorEnsayo.coste'.
      4.2.1) Si lo es, eliminamos el nodo 'K' del camino, clonamos los visit. y el camino, y utilizamos estos 3 nuevos datos para crear un nuevo ensayo para la cola.
      4.2.2) Si no lo cumple, dejamos de estudiar este ensayo.