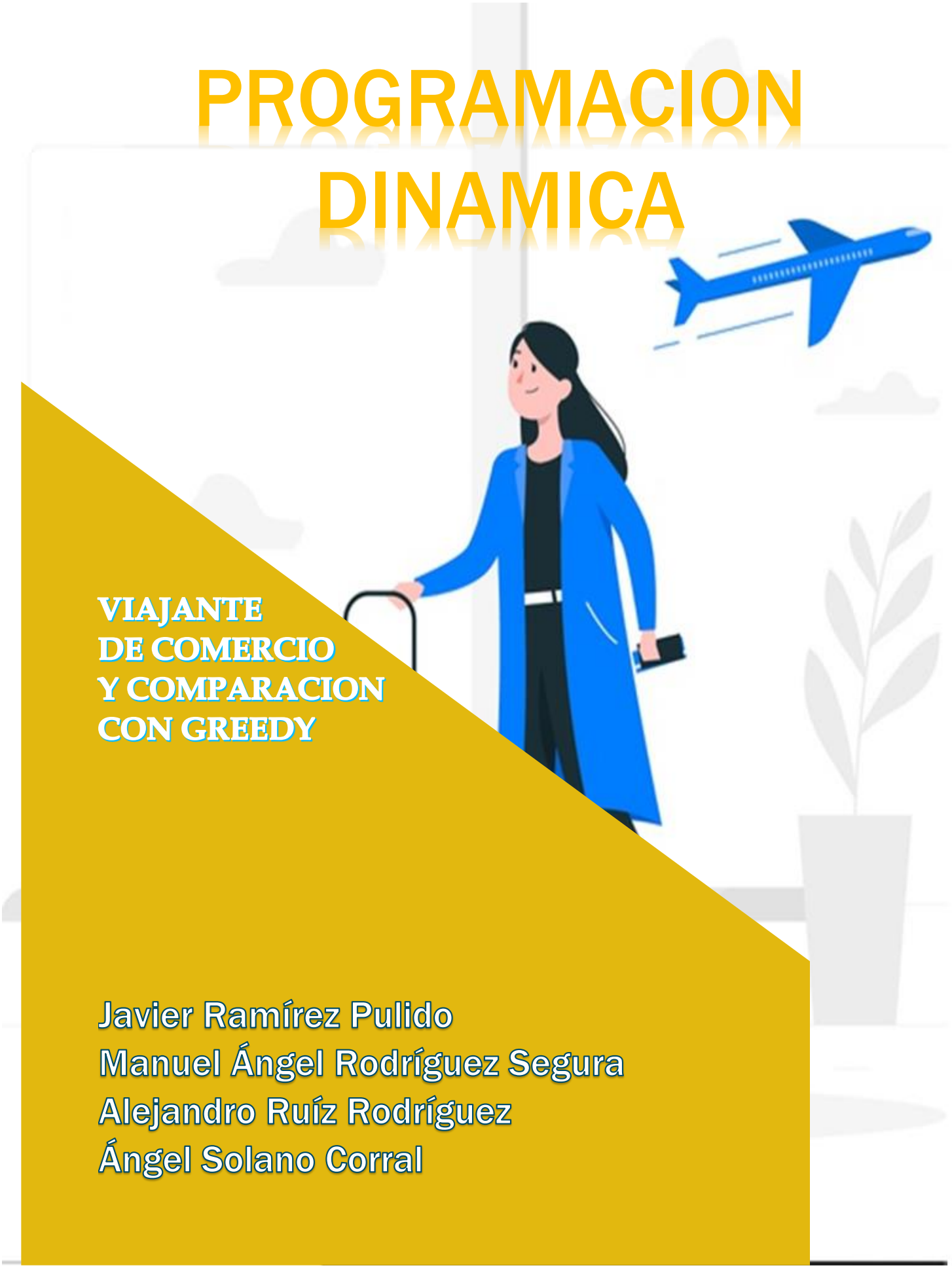
PROGRAMACION DINAMICA



VIAJANTE DE COMERCIO Y COMPARACION CON GREEDY

Javier Ramírez Pulido
Manuel Ángel Rodríguez Segura
Alejandro Ruíz Rodríguez
Ángel Solano Corral

INDICE

DESCRIPCIÓN DEL PROBLEMA

En muchas ocasiones, dado un problema de tamaño n, solo puede obtenerse una caracterización efectiva de su solución basándose en la solución de los subproblemas de tamaño n-1, que permite un desarrollo recursivo. Esto es aplicable al problema del Viajante de Comercios, que tiene como objetivo salir de una ciudad, recorrer todas las restantes sin repetir y volver a la de origen con el camino más corto posible. Las ciudades se conforman por un índice que las identifica individualmente de forma unívoca y unas coordenadas necesarias tanto para su representación como para el cálculo de las distancias entre ellas, pues todas pueden partir hacia todas (grafo conexo, ponderado y completo).

SOLUCION PROPUESTA

En este caso, podemos observar fácilmente que obtener la mejor solución para todos los subproblemas de n-1 tamaño con un desarrollo recursivo que combina todas las soluciones posibles, elige la mejor de ellas en cada nivel. Por ejemplo, con 2 ciudades, únicamente existe un camino disponible (1 -> 2 -> 1), para 3 ciudades existen 2 caminos posibles (1 -> 2 -> 3 -> 1 // 1 -> 3 -> 2 -> 1), para 4 ciudades existirían 6 caminos y para 5 hay 24 alternativas diferentes. El crecimiento de soluciones por cada ciudad que se añade es muy elevado y para cada caso, el número de soluciones posibles es (n-1)!, siendo n el número de ciudades totales.

Nuestra función principal toma como parámetros una ciudad de partida y un conjunto de nodos (índices de ciudades) por las que pasar sin repetir antes de volver a la ciudad de la que se parte.

Por ejemplo, supongamos un caso en el que tengamos 5 ciudades, enumeradas del 1 al 5. Si lo planteamos de tal forma que salga de la ciudad 1, los parámetro para la función principal serían (1, {2,3,4,5})

A continuación, un bucle que hace tantas iteraciones como ciudades haya en el vector de las que quedan por recorrer (4), se encarga de llamar de forma recursiva a la misma función en la que estamos, pero pasando como

parámetros en cada iteración una ciudad distinta del vector de las restantes y el mismo vector de ciudades recibidas, pero habiendo eliminado la nueva ciudad de la que salimos.

En la primera iteración del bucle se pasarían los parámetros (2,{3,4,5}), en la siguiente iteración (3,{2,4,5}), y así pasando como ciudad de origen cada elemento del vector.

Cuando se llega al caso base, que es aquel en el que el vector de ciudades por recorrer pasado como parámetro está vacío, se devuelve la distancia entre esa ciudad de la que se partía esta vez y la ciudad de origen (1).

Si los parámetros son (5,{-}), la función devuelve la distancia directa entre 5 y la ciudad de origen que en nuestro caso es 1.

Tras la llamada a la función recursiva, pero aún dentro del bucle, una vez que el resultado es devuelto, se suma ese valor a la distancia directa entre la ciudad de origen de esa iteración y aquella que se ha pasado como origen de la siguiente recursiva ya calculada. Después, se compara si es la menor obtenida hasta ahora y, de serlo, almacenamos su valor como solución. Si no lo es, no ocurre nada, avanza la iteración del bucle.

Se sumará la distancia entre la ciudad 1 y la 2 más la distancia del camino que parte de la ciudad 2 que es más corto, obtenido con recursividad. Cuando calcula esa distancia, compara con la más pequeña obtenida hasta ahora (que, al ser la primera iteración, compara con un valor incuestionablemente mayor y cumple el requisito de ser menor) y si es menor se la queda como solución. Luego sumaría la distancia entre la ciudad 1 y la 3 más la distancia del camino más corto que tiene la ciudad 3 como el origen. Vuelve a comparar y se la queda si esta es más corta que la anterior. Luego la distancia de la ciudad 1 a la 4 y el camino más corto que sale de 4. Así tantas veces como ciudades haya en el vector de ciudades por recorrer. Este procedimiento se hará en cada nivel. No solo se calcula de la 1 a todas las restantes. También, al llegar al avanzado caso en el que ya vamos por la ciudad 3 y tenemos un vector por pasar de {4,5} se compararía si es más pequeña la suma de la 3 a la 4 con la distancia más corta saliendo de 4 o la suma de la 3 a la 5 con la distancia más corta de 5 al final.

Si el código es correcto, tras todas las recursividades necesarias, terminaría devolviendo la ciudad más corta.

PSEUDOCODIGO

DistanciaYRecorridoMinimo(nodo_inicio, nodos_por_pasar, ciudadesRecogidas, nodo_empece)

Variable global

ENTERO MAX = 2147483647; //mayor valor que puede tomar un entero

Variables locales

ENTERO másCorto = MAX;
ENTERO distancia, elementoBorrado, indiceMasCorto;
VECTOR<*CIUDAD*> comprobaciones;
PAIR <*INT, VECTOR*<*INT*>> solución, aux;

principio
si nodos_por_pasar = Ø entonces

solución.first <- calcularDistanciaCiudades(1, nodo_inicio);
solución.second <- insertar(nodo_inicio);
devuelve solución;

sino

para todo j en nodos_por_pasar hacer

comprobaciones <- insertar(nodo_incio, nodos_por_pasar(j));
elementoBorrado <- nodos_por_pasar(j);
nodos_por_pasar <- borrar(j);

aux <- DistanciaYRecorridoMinimo(elementoBorrado, nodos_por_pasar, ciudadesRecogidas, nodo_empece) ;

distancia:= calcularDistanciaCiudades(comprobaciones) + aux.first;
nodos_por_pasar <- insertar(elementoBorrado);
comprobaciones.vaciar;

si distancia<masCorto entonces

solución <- aux;
solución.first <- distancia;
masCorto <- distancia ;
indiceMasCorto <- nodo_inicio

fsi

fpara;

solución.first <- masCorto;
solución.second <- indiceMasCorto;

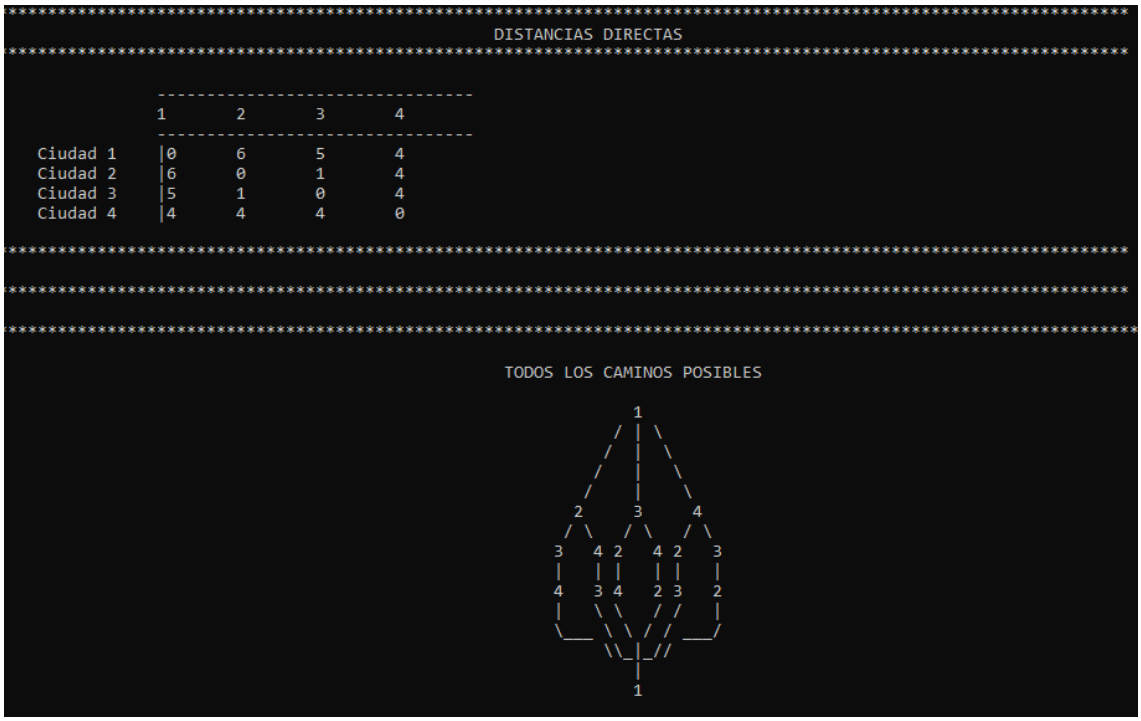
fsi
fsi
devuelve solución;
fin

CASOS DE EJECUCION

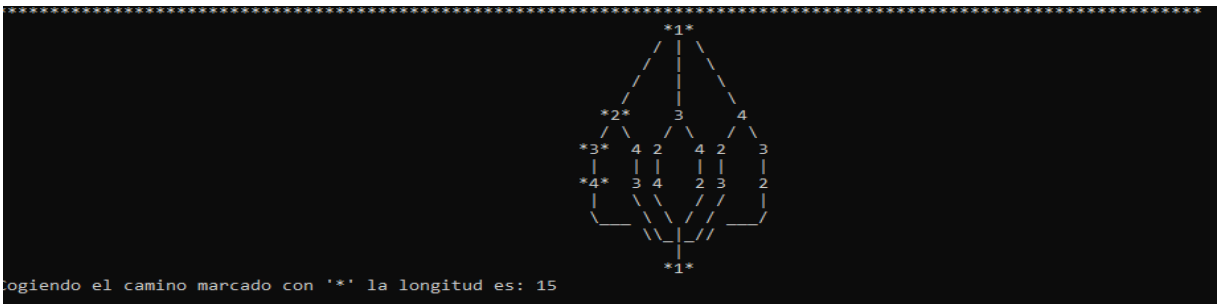
La eficiencia del algoritmo obliga a que el número de ciudades con el que se realizan los casos de ejecución sea muy limitado, pero a continuación podemos ver los resultados y procedimientos que se obtienen de nuestro algoritmo con diferentes cantidades de ciudades.

1.- Nº CIUDADES = 4 (MOSTRANDO CAMINOS POSIBLES)

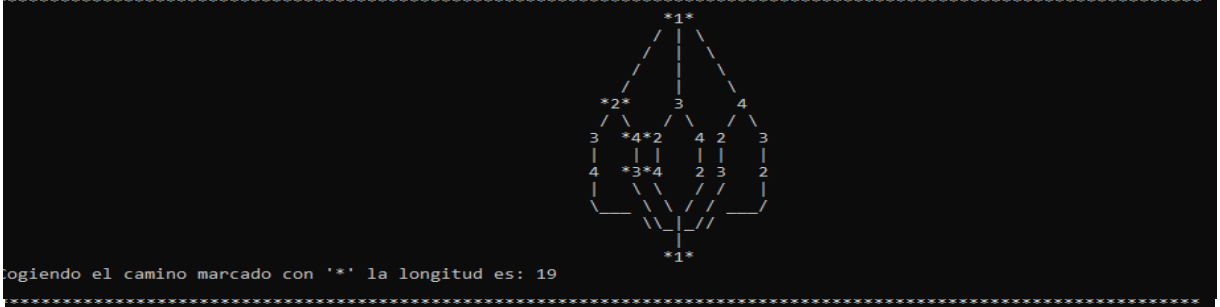
En este caso se ha realizado un programa que realiza el mismo procedimiento para obtener las distancias que en el resto de casos, pero la salida por pantalla está adaptada a poder ver los caminos posibles y cuál de ellos está comprobando en cada momento. En la primera imagen vemos, con 4 ciudades, cuál es el árbol que contiene todos los caminos posibles que parten de la ciudad 1 y, sin repetir ciudad, vuelven a la ciudad 1 pasando por todas las restantes.



En las siguientes capturas se ve qué distancia supondría coger cada uno de los caminos. En eso consiste el algoritmo, la comprobación de combinar las ciudades de todas las formas posibles sin repeticiones y quedarse con el camino más corto. Por ello, dibujar el árbol con todos los caminos de 7 ciudades, por ejemplo, contendría 720 caminos, lo cual es imposible mostrar por pantalla. Esta es la razón por la que observamos las posibilidades con 4 ciudades únicamente para hacerlo más gráfico, pero a partir de este ejemplo las ejecuciones se limitan a mostrar distancias y recorridos.

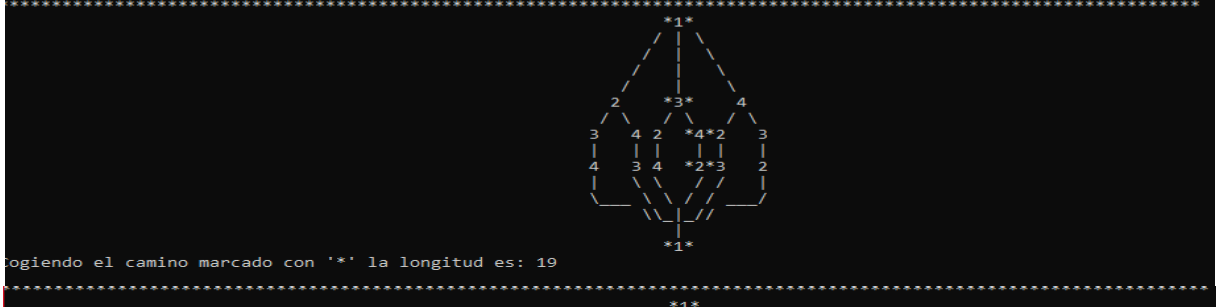


```
Cogiendo el camino marcado con '*' la longitud es: 15
```

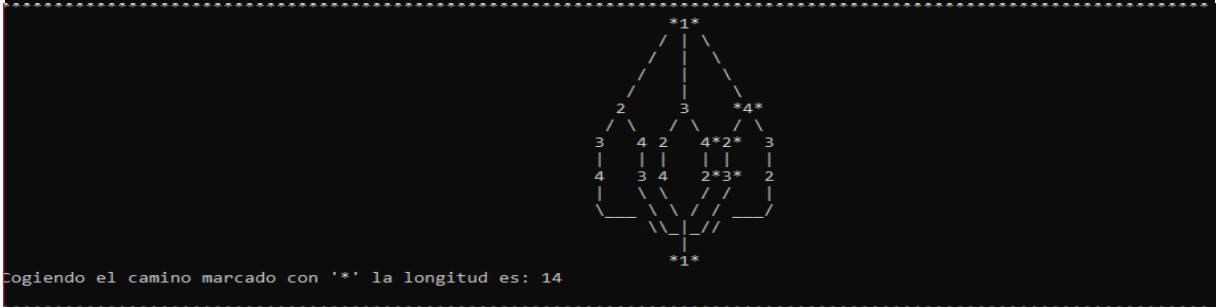


```
Cogiendo el camino marcado con '*' la longitud es: 19
```

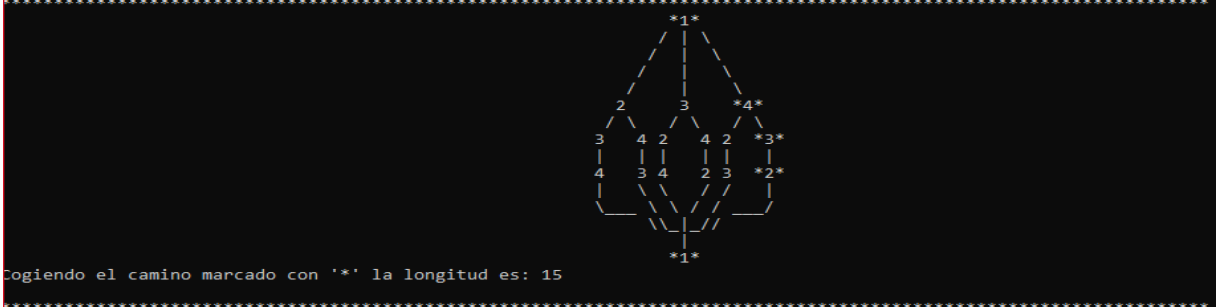
```
Cogiendo el camino marcado con '*' la longitud es: 14
```



```
Cogiendo el camino marcado con '*' la longitud es: 19
```



```
Cogiendo el camino marcado con '*' la longitud es: 14
```



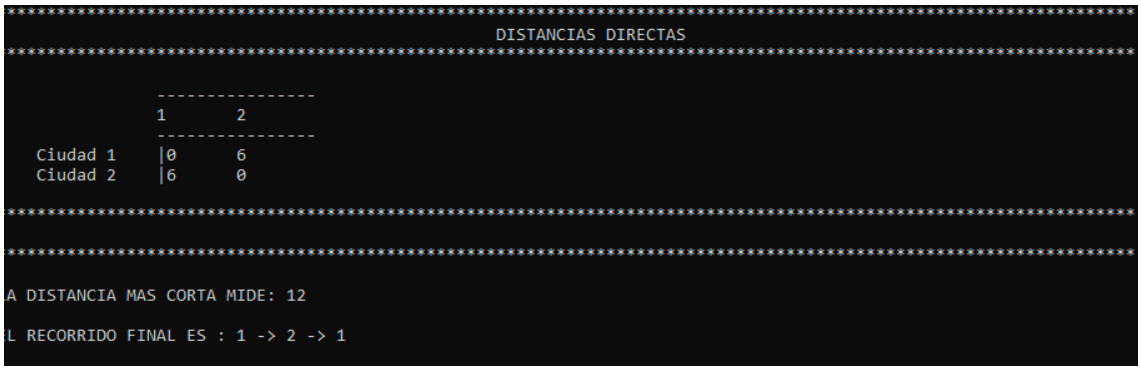
```
Cogiendo el camino marcado con '*' la longitud es: 15
```

```
LA DISTANCIA MAS CORTA MIDE: 14
EL RECORRIDO FINAL ES : 1 -> 3 -> 2 -> 4 -> 1
```

Es fácil observar a raíz de las imágenes anteriores que con pocas ciudades es altamente probable que diferentes caminos resulten en las mismas distancias. Ante estas situaciones de conflicto, el programa se queda con el primer camino obtenido y evita rehacer la secuencia final para obtener mismas distancias.
De entre todas las opciones, la más corta es aquella que partiendo de una pasa por las ciudades 3, 2, 4 y regresa a 1, con una distancia de 14.

2.- Nº CIUDADES = 2

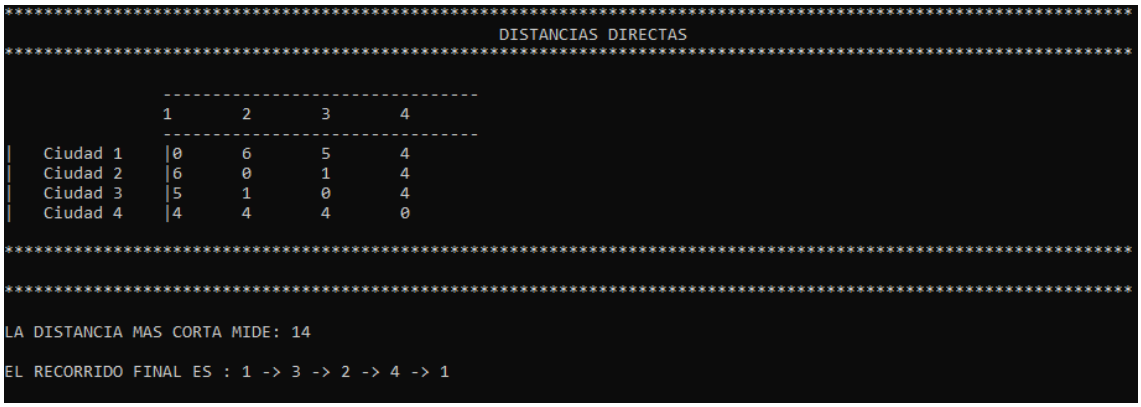
Ahora, comprobamos la ejecución del caso mínimo, un recorrido que solamente pase por una ciudad diferente a esta de la que se parte.



Como era de esperar, el camino directamente va de la ciudad origen a la restante y de vuelta, lo cual tiene una longitud del doble de la distancia de una ciudad a la otra (ida y vuelta).

3.- Nº CIUDADES = 4

Otra ejecución con un fichero que contiene 4 ciudades.



Nº CIUDADES = 4 PERO SALIENDO DE LA CIUDAD 3

Una ventaja del código (aunque poco necesaria) es la flexibilidad que ofrece a la hora de seleccionar la ciudad de partida. Como el camino que busca es mínimo, lo será saliendo de cualquiera de las ciudades, como comprobaremos más tarde, pero puedes probarlo teniendo en cuenta la parte del código mostrada a continuación. El conjunto de funciones involucradas en el cálculo del camino está adaptado a construirse alrededor

de la ciudad de origen y se puede modificar esta cambiando el valor en la variable marcada a continuación. Esta implementación hace fácil que el usuario pueda indicar como parámetro la ciudad que quiere como primera, pero no tiene por qué conocer cuantas ciudades hay en el fichero seleccionado, lo cual podría provocar problemas. (Al modificarlo hay que tener en cuenta que ciudadInicio = índice de la ciudad – 1)

```
/**
* @brief Funcion para mostrar la informacion final por pantalla
* @param solucion pair que contiene tanto la distancia minima como la secuencia de ciudades a seguir
* @param ciudadInicio variable que contiene la ciudad de la que se inicia el recorrido
*/
void ResultadoFinal(pair<int,vector<int> > solucion, int ciudadInicio){

    cout << "\nLA DISTANCIA MAS CORTA MIDE: " << solucion.first << endl ;
    cout << "\nEL RECORRIDO FINAL ES : " ;
    for(int i=solucion.second.size()-1; i>=0; i--)        //Muestra el recorrido al reves porque la recursividad hace que los nodos se almacenen de destino a origen, al reves
        cout << solucion.second[i]+1 << " -> " ;

    cout<< ciudadInicio+1 << "\n";     //Aniade al recorrido final que se muestra por pantalla la vuelta al nodo de inicio

}


int main(int argc, char * argv[]){

    vector<Ciudad> ciudadesRecogidas;                    //vector de ciudades que recogeremos del fichero
    int dimension = obtenerInfoFichero(ciudadesRecogidas, argv[1]); //rellenamos el vector ciudadesRecogidas con las ciudades del fichero y devuelve la cantidad de ciudades que hay
    int ciudadInicio = 2;                                //Variable que indica la ciudad de la que parte el camino (en nuestro caso el camino empieza y termina en la primera)
    vector<int> vectorNodosPaso;                         //Vector que contiene el resto de ciudades por las que pasar
    pair<int,vector<int> > solucion;                     //Pair que contendra la solucion final al problema. En el primer elemento la distancia mas corta y en el segundo el recorrido final
    int **distancias_directas;                           //Creo la matriz de distancias entre ciudades

    //Reserva para la matriz de distancias
    distancias_directas = new int *[dimension];

    for (int i = 0; i < dimension; i++)
```

Con esto simplemente se quiere demostrar que saliendo de cualquiera de las ciudades se obtiene el camino más corto, pero con un recorrido adaptado a las necesidades de salir de otra que no sea la 1.

```
**********************************************************************************************************************
                                                DISTANCIAS DIRECTAS
**********************************************************************************************************************

                ---------------------------------
                1       2       3       4
                ---------------------------------
|   Ciudad 1    |0      6       5       4
|   Ciudad 2    |6      0       1       4
|   Ciudad 3    |5      1       0       4
|   Ciudad 4    |4      4       4       0

**********************************************************************************************************************

**********************************************************************************************************************

LA DISTANCIA MAS CORTA MIDE: 14

EL RECORRIDO FINAL ES : 3 -> 1 -> 4 -> 2 -> 3
```

4.- Nº CIUDADES = 7

Ejemplo como los anteriores, pero con 7 ciudades.

```
**********************************************************************************************************************
                                                DISTANCIAS DIRECTAS
**********************************************************************************************************************

                ---------------------------------------------------------
                1       2       3       4       5       6       7
                ---------------------------------------------------------
|   Ciudad 1    |0      6       5       4       11      8       7
|   Ciudad 2    |6      0       1       4       17      14      13
|   Ciudad 3    |5      1       0       4       17      13      12
|   Ciudad 4    |4      4       4       0       13      11      10
|   Ciudad 5    |11     17      17      13      0       4       6
|   Ciudad 6    |8      14      13      11      4       0       1
|   Ciudad 7    |7      13      12      10      6       1       0

**********************************************************************************************************************

**********************************************************************************************************************

LA DISTANCIA MAS CORTA MIDE: 35

EL RECORRIDO FINAL ES : 1 -> 3 -> 2 -> 4 -> 5 -> 6 -> 7 -> 1
```

Nº CIUDADES = 7 PERO SALIENDO DE LA CIUDAD 3

Y de nuevo salimos de otra ciudad y comprobamos que da la misma distancia.

```
**************************************************************************************************************
                                              DISTANCIAS DIRECTAS
**************************************************************************************************************

              ---------------------------------------------------
              1       2       3       4       5       6       7
              ---------------------------------------------------
|   Ciudad 1    |0      6       5       4       11      8       7
|   Ciudad 2    |6      0       1       4       17      14      13
|   Ciudad 3    |5      1       0       4       17      13      12
|   Ciudad 4    |4      4       4       0       13      11      10
|   Ciudad 5    |11     17      17      13      0       4       6
|   Ciudad 6    |8      14      13      11      4       0       1
|   Ciudad 7    |7      13      12      10      6       1       0

**************************************************************************************************************

**************************************************************************************************************

LA DISTANCIA MAS CORTA MIDE: 35

EL RECORRIDO FINAL ES : 3 -> 1 -> 7 -> 6 -> 5 -> 4 -> 2 -> 3
```

5.- Nº CIUDADES = 9

Una vez comprobado que la distancia es la misma desde cualquier ciudad, volvemos al caso de partir de la 1 y probamos con 9 ciudades.

```
**************************************************************************************************************
                                              DISTANCIAS DIRECTAS
**************************************************************************************************************

              ---------------------------------------------------------------------
              1       2       3       4       5       6       7       8       9
              ---------------------------------------------------------------------
|   Ciudad 1    |0      6       5       4       11      8       7       1       11
|   Ciudad 2    |6      0       1       4       17      14      13      6       17
|   Ciudad 3    |5      1       0       4       17      13      12      6       16
|   Ciudad 4    |4      4       4       0       13      11      10      3       15
|   Ciudad 5    |11     17      17      13      0       4       6       11      8
|   Ciudad 6    |8      14      13      11      4       0       1       8       5
|   Ciudad 7    |7      13      12      10      6       1       0       7       5
|   Ciudad 8    |1      6       6       3       11      8       7       0       12
|   Ciudad 9    |11     17      16      15      8       5       5       12      0

**************************************************************************************************************

**************************************************************************************************************

LA DISTANCIA MAS CORTA MIDE: 44

EL RECORRIDO FINAL ES : 1 -> 3 -> 2 -> 4 -> 8 -> 7 -> 6 -> 5 -> 9 -> 1
```

6.- Nº CIUDADES = 13

En esta última ejecución se probó un fichero de 13 ciudades, pero deja de ser eficiente con cantidades de ciudades grandes. El tiempo obtenido hace incoherente esta opción de cálculo para 13 ciudades, ya que la espera obtenida es superior a los 30 minutos.

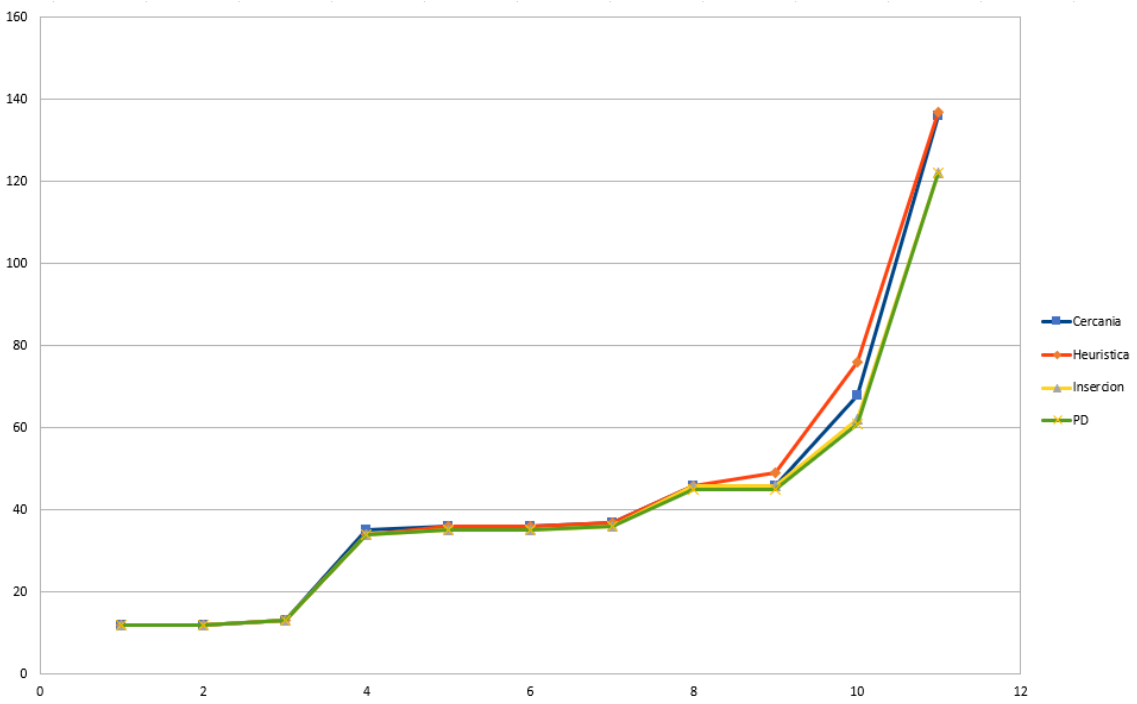
```
C:\Users\manue\OneDrive\Desktop\P4\TSP_PD_TIEMPOS.exe
13  1913.73


--------------------------------
Process exited after 1914 seconds with return value 0
Presione una tecla para continuar . . .
```

COMPARACION SOLUCION Y EFICIENCIAS

Antes de realizar este estudio, debemos aclarar las diferencias que existen entre los algoritmos voraces y los que se basan en técnicas de programación dinámica. Estos primeros siguen una heurística que consiste en elegir la opción óptima en cada paso con la esperanza de llegar a encontrar una solución general óptima sin preocuparse del "futuro" del mismo. Los algoritmos que se basan en PD recurren a métodos que reducen los tiempos de ejecución y que consisten en dividir el problema general en subproblemas y resolver estos una sola vez, guardando sus soluciones en una tabla para su futura utilización.
En el caso del problema del viajante de comercio (TSP), los algoritmos voraces realizados basaban sus heurísticas en ir al vecino más cercano (TSP cercanía), elegir la ciudad que suponía un menor coste en el vector de ciudades (TSP inserción) y una última que se basaba en ver todas las posibilidades de distancias entre una ciudad e ir escogiendo la que era menor (TSP heurística). Como veremos a continuación, no todas estas versiones dan los mismos resultados, pues basan sus algoritmos en técnicas que en determinados momentos toman decisiones que afectan al coste total de la solución. En la resolución del problema por medio de programación dinámica la cosa cambia, pues se trata de realizar todas las combinaciones posibles entre todas las ciudades, teniendo en cuenta el coste que genera ir de una ciudad a otra y a su vez esta con el resto, y así sucesivamente, es decir, como generar un grafo con todas las ciudades y las posibles comunicaciones entre ellas, escogiendo en cada caso la mejor de las mejores opciones. A simple vista podemos entender que se tratará de un proceso costoso en cuanto a eficiencia, pero sin embargo nos va a ofrecer los mejores resultados posibles de entre todos.

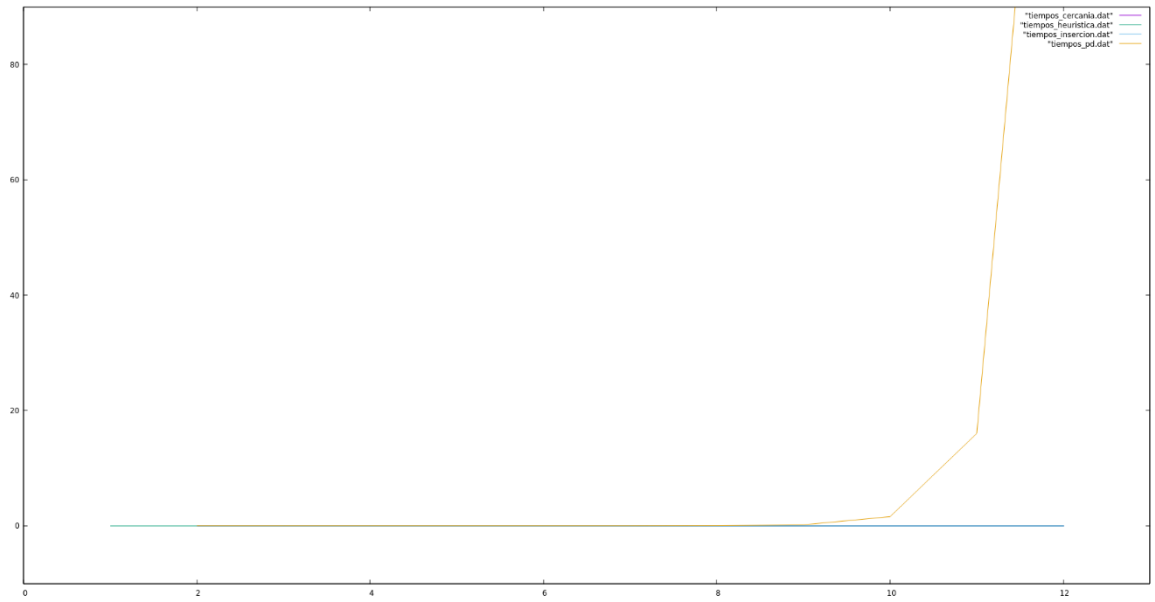
	TSP CERCANIA	TSP HEURISTICA	TSP INSERCION	TSP PD
a2.tsp	12	12	12	12
a3.tsp	12	12	12	12
a4.tsp	13	13	13	13
a5.tsp	35	34	34	34
a6.tsp	36	36	35	35
a7.tsp	36	36	35	35
a8.tsp	37	37	36	36
a9.tsp	46	46	46	45
a10tsp	46	49	46	45
a11.tsp	68	76	62	61
a12.tsp	136	137	122	122

En la siguiente gráfica podemos ver un resumen de las distancias de los algoritmos para los diferentes ficheros .tsp



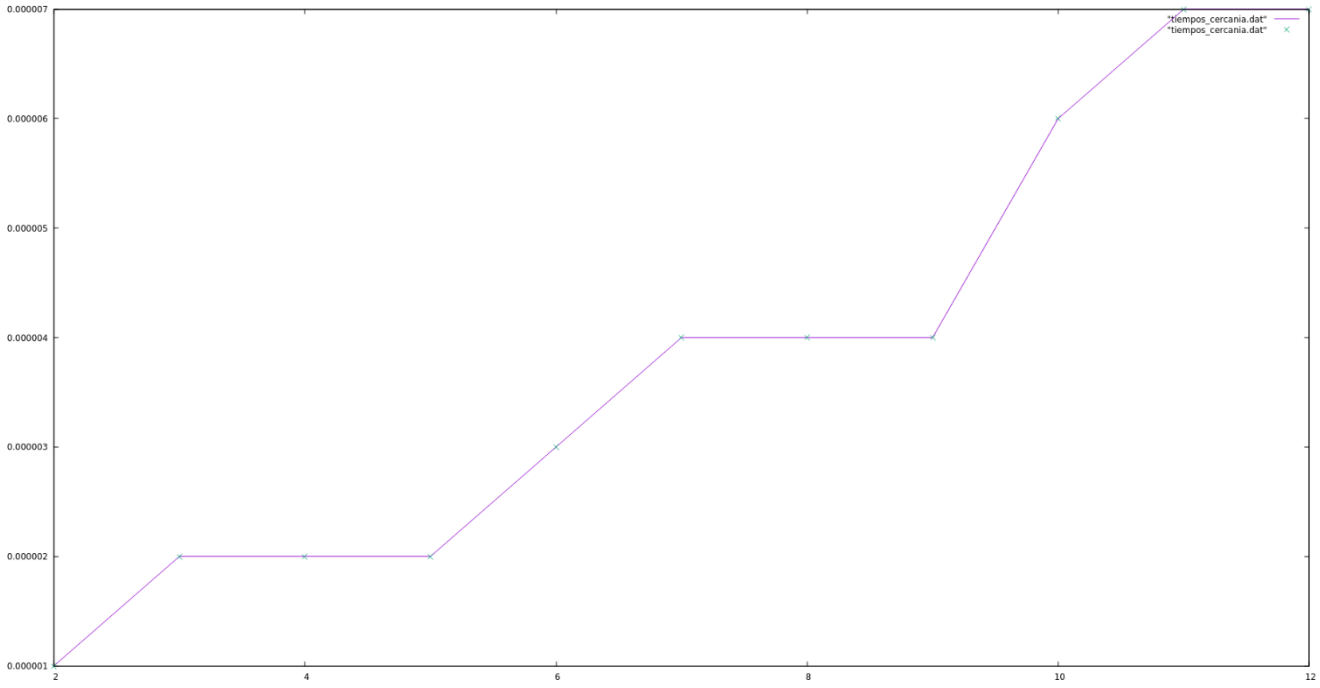
Como hemos podido comprobar en estas dos imágenes, y como era de esperar, el algoritmo PD realizado en esta práctica nos ofrece las mejores distancias, ejecutando todos los algoritmos con ficheros que contienen de 2 a 12 ciudades (a2.tsp, a3.tsp…). Pero que obtenga las mejores soluciones no significa que sean los mejores tiempos, es por ello por lo que vamos a realizar un estudio sobre la diferencia de tiempos que ofrecen los diferentes algoritmos.

En la siguiente gráfica se muestran los tiempos de ejecución de los cuatros algoritmos para los diferentes ficheros .tsp que hemos estado comentando anteriormente.



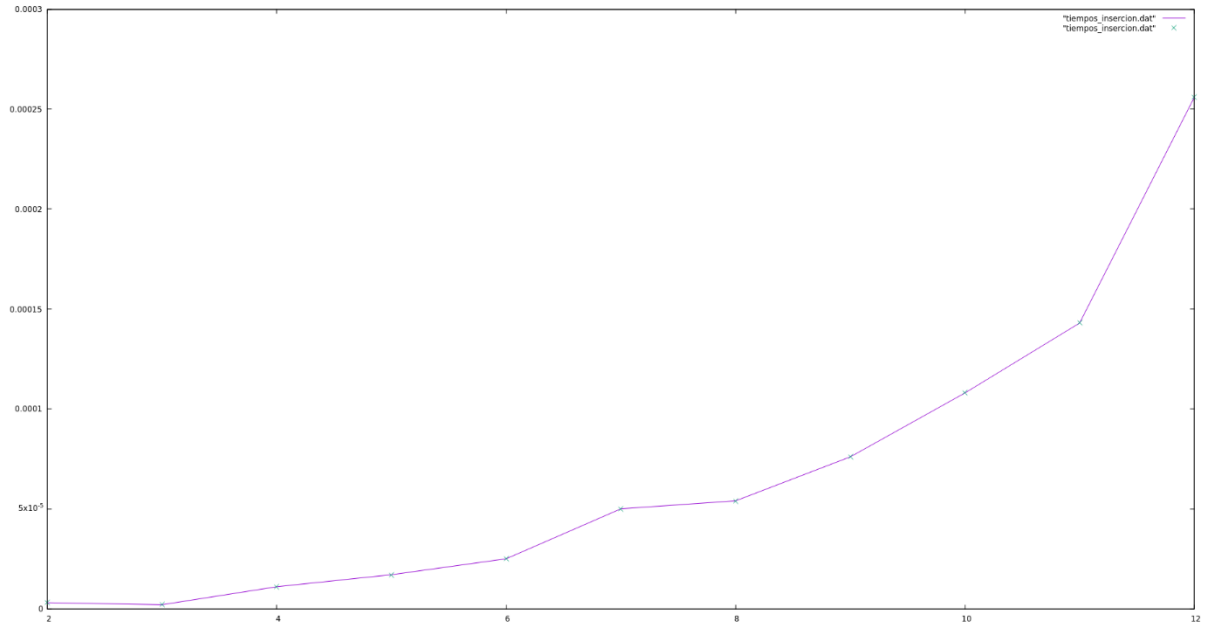
Vemos que el algoritmo de PD ofrece unos tiempos muy elevados conforme el número de ciudades que trata de resolver aumenta. Este algoritmo cuenta con una eficiencia teórica de $O(n*2^n)$ por lo que su crecimiento es más que exponencial.
Vamos a ver las gráficas de los tiempos por separado para entender mejor cada situación.

Gráfica de los tiempos para el algoritmo basado en cercanía:

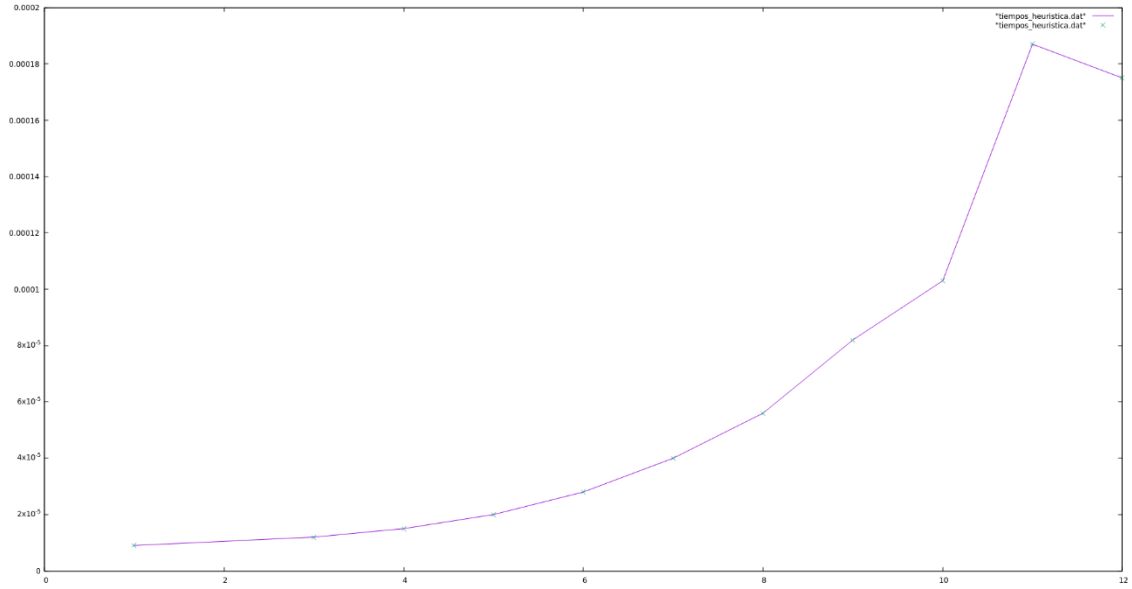


Por cada x que encontramos en la imagen se diferencia un fichero de otro

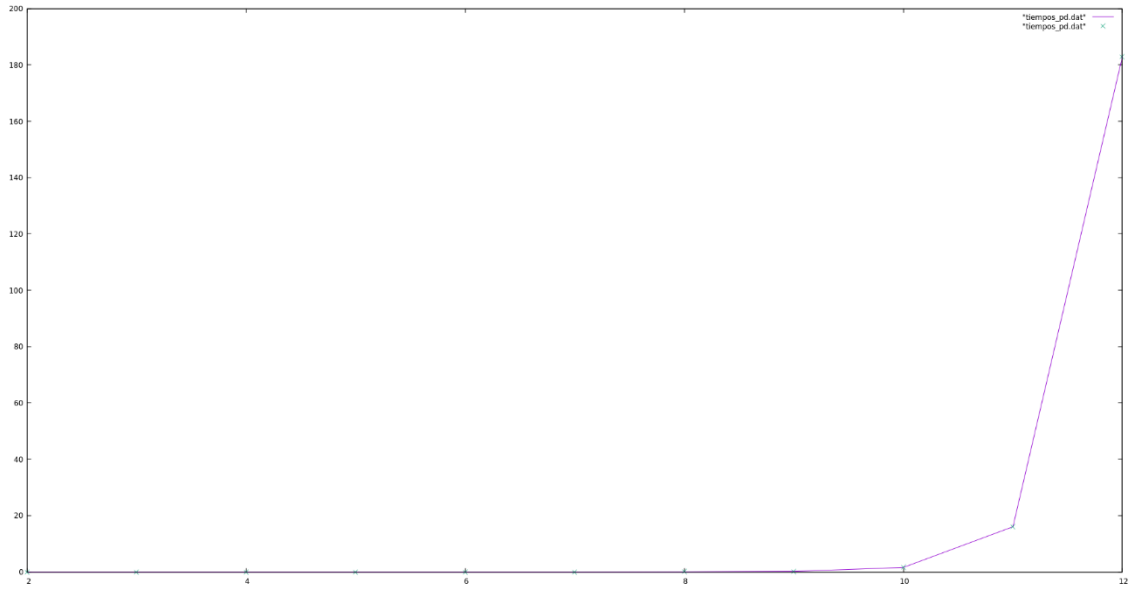
Gráfica de los tiempos para el algoritmo basado en inserción:



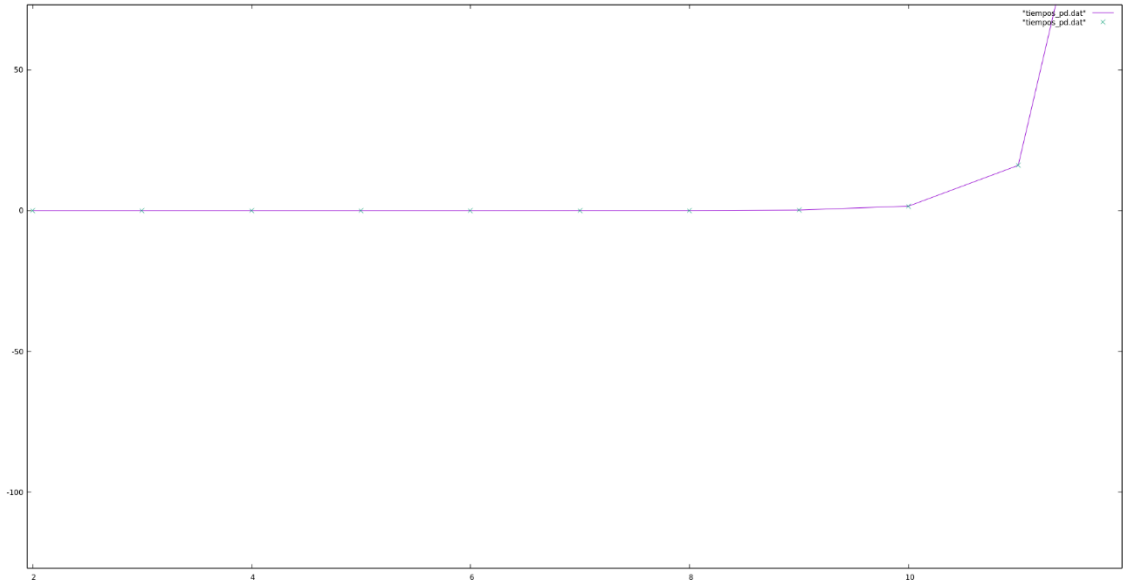
Gráfica de los tiempos para el algoritmo basado en heurística propia:



Gráfica para el caso del algoritmo de programación dinámica:

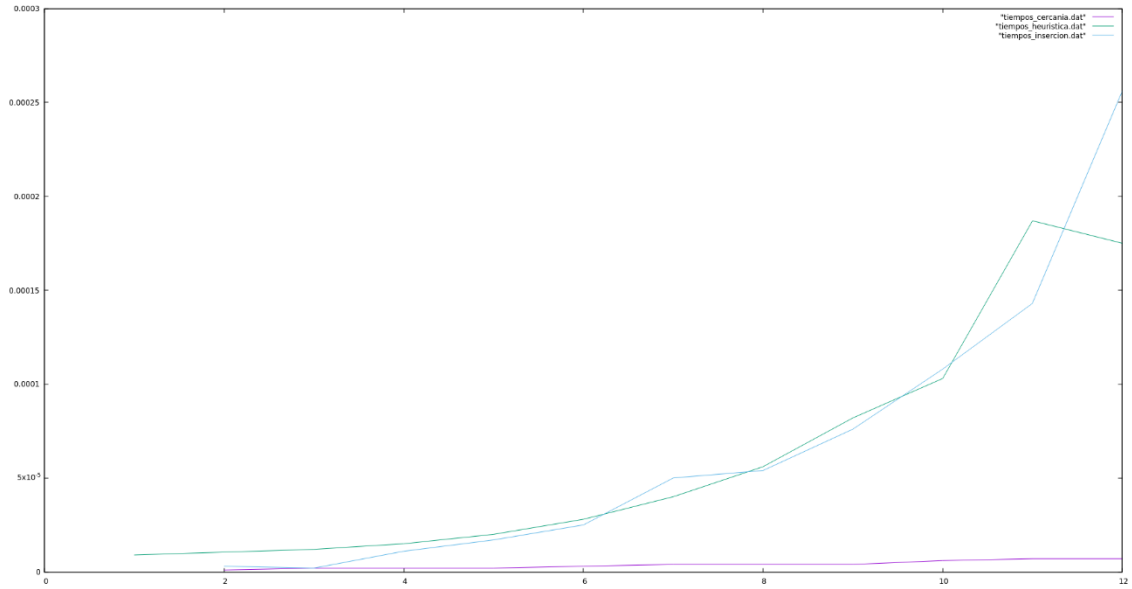


Como los tiempos van a crecer de forma exponencial, no se llega a apreciar con claridad los tiempos que hay cuando se ejecuta el algoritmo para pocas ciudades por lo que vamos a agrandar un poco más esa zona:



Como vemos para un número reducido de ciudades se obtienen valores muy próximos al 0 y apenas diferenciables entre ellos de forma individual.

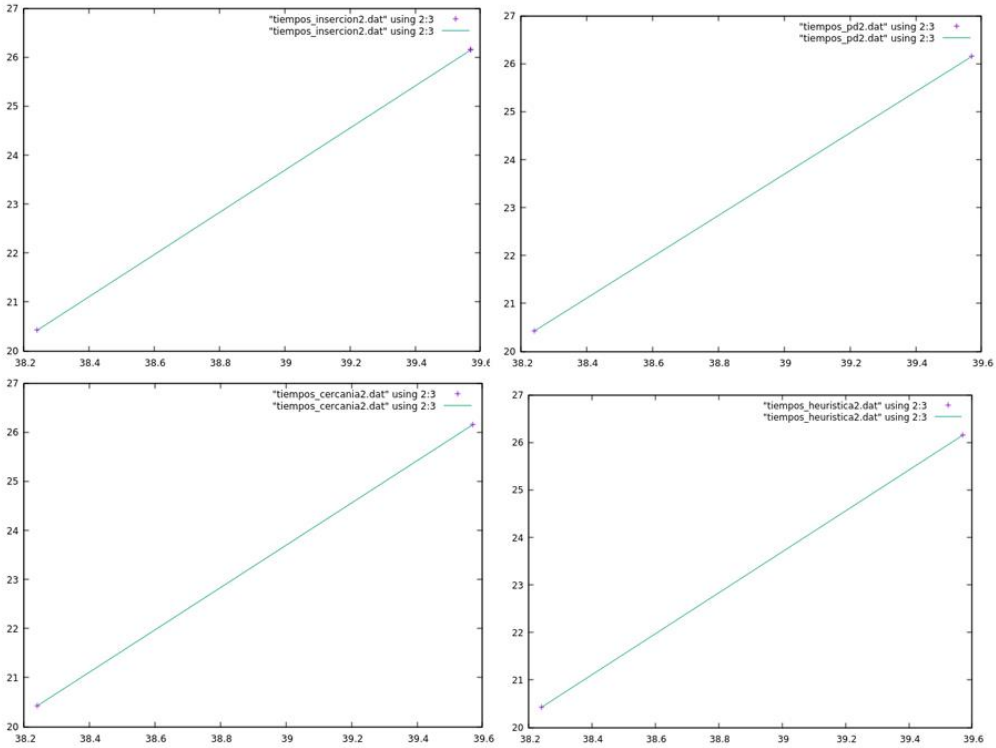
Por último, vamos a comparar las gráficas entre los tres algoritmos voraces para ver sus diferencias:



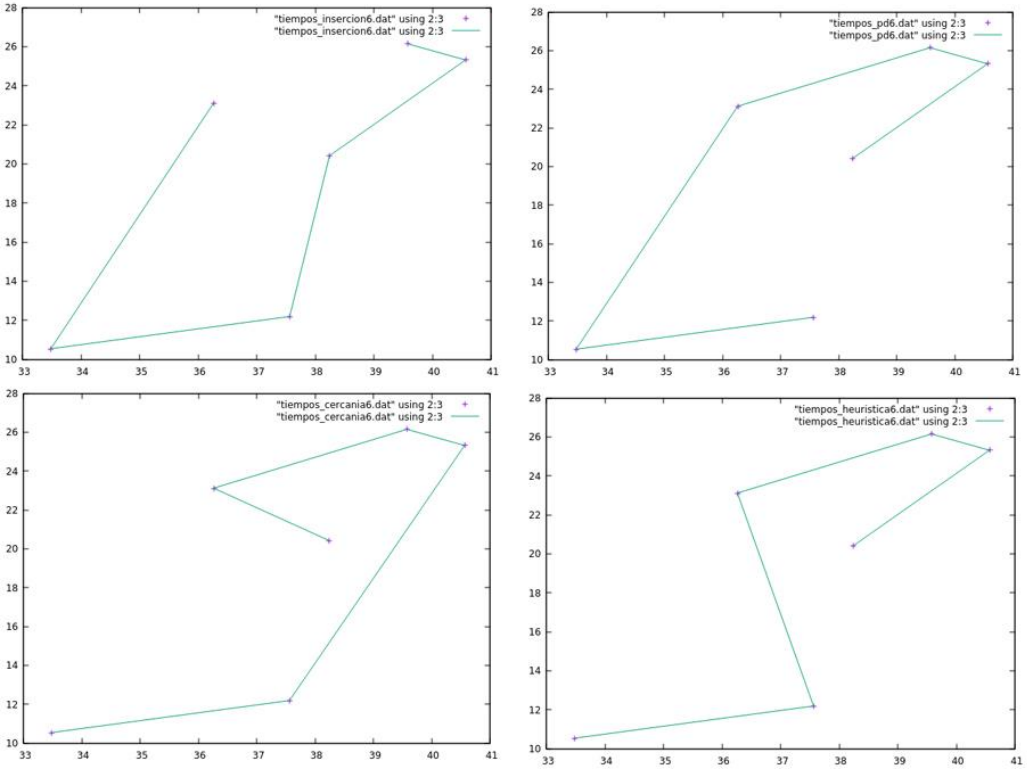
Vemos como se obtienen tiempos similares entre los algoritmos basados en inserción y en la heurística propia, sin embargo, los tiempos se reducen aún más cuando usamos el algoritmo basado en cercanía, pues es el de los tres el que menos comprobaciones realiza en su ejecución, ya que solamente se fija en la ciudad más cercana y la incluye y no tiene en cuenta la distancia que puede suponer tomar el camino por otras ciudades.

COMPARACION DE RECORRIDOS FINALES ENTRE ALGORITMOS

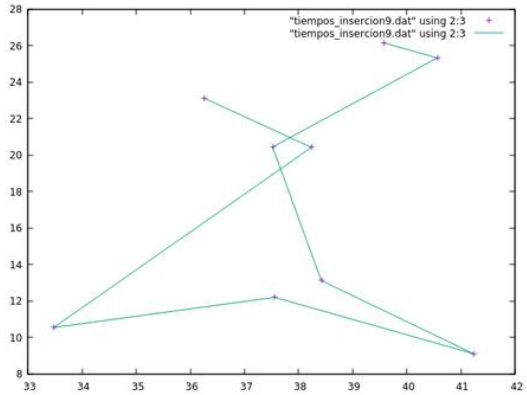
Para 2 ciudades:

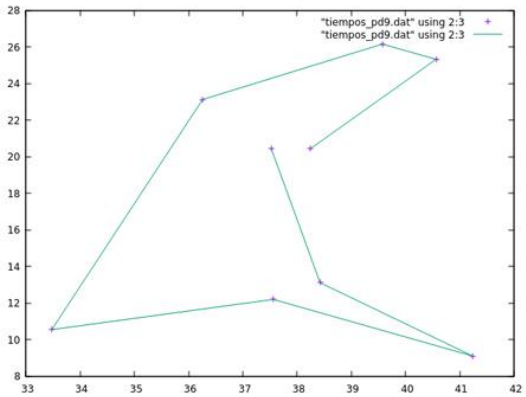


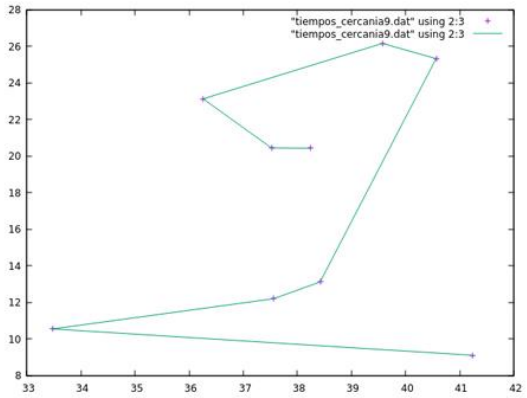
Para 6 ciudades:

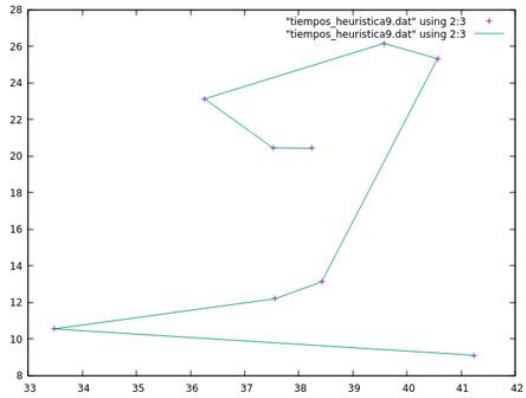


Para 9 ciudades:

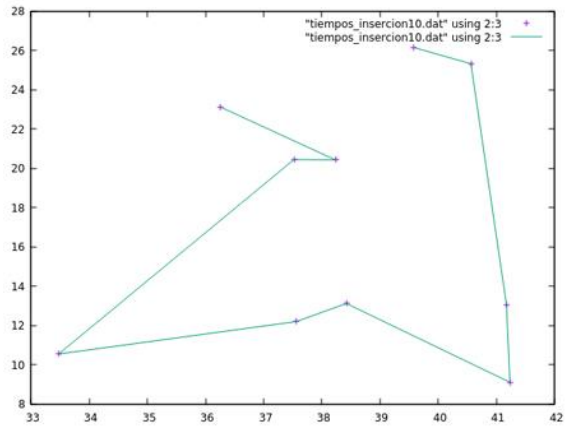


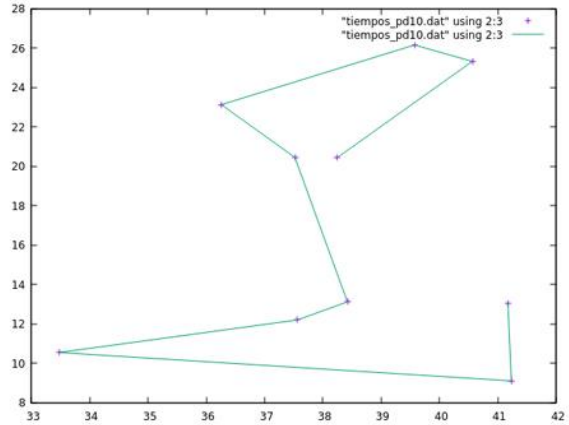


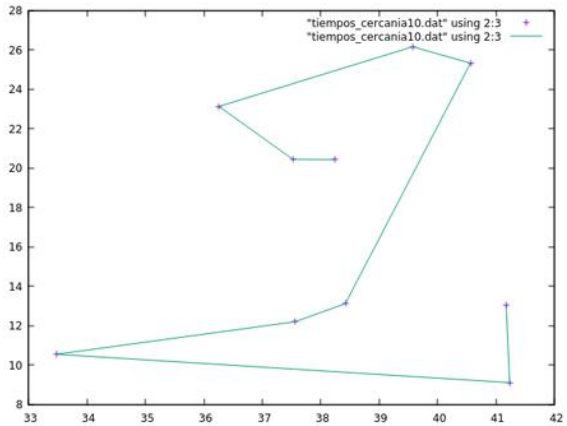


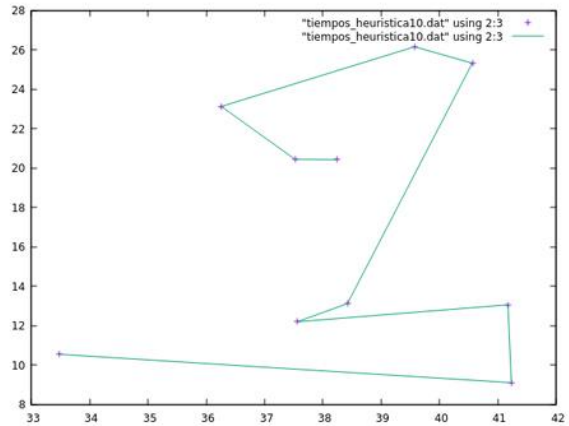


Para 10 ciudades:

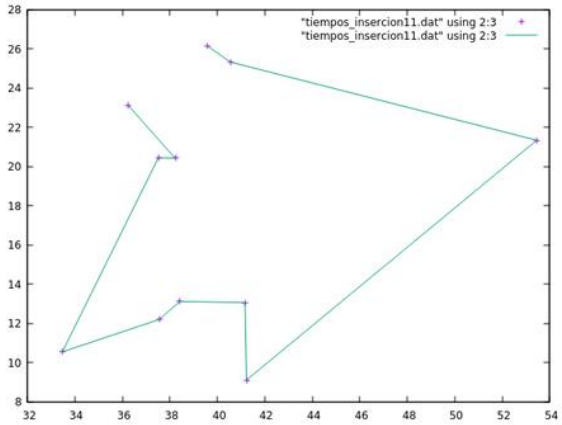


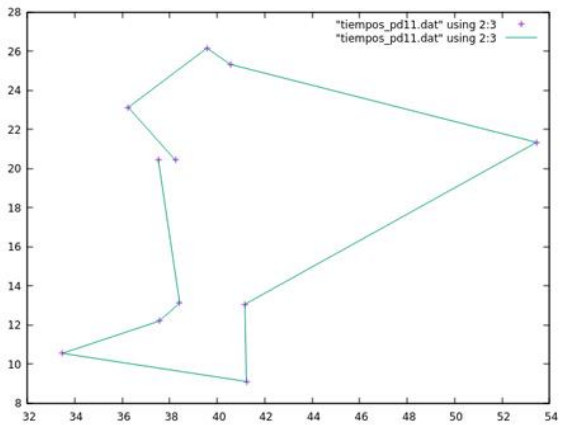


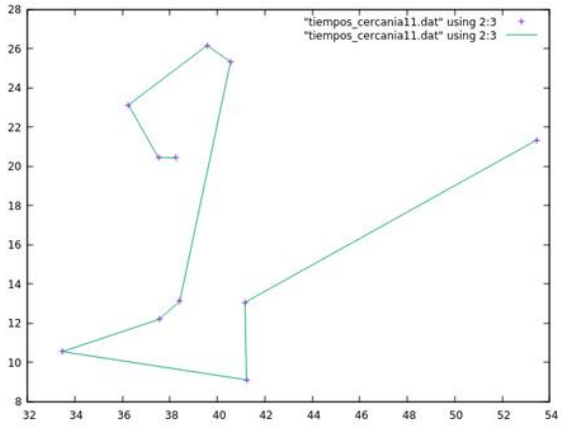


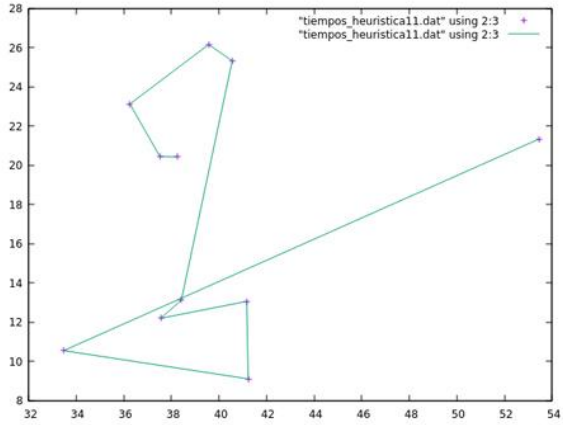


Para 11 ciudades:

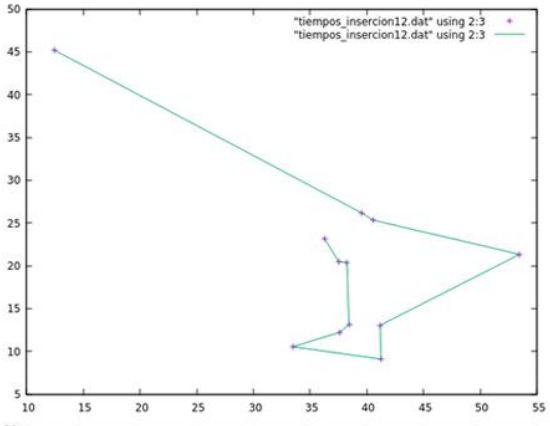


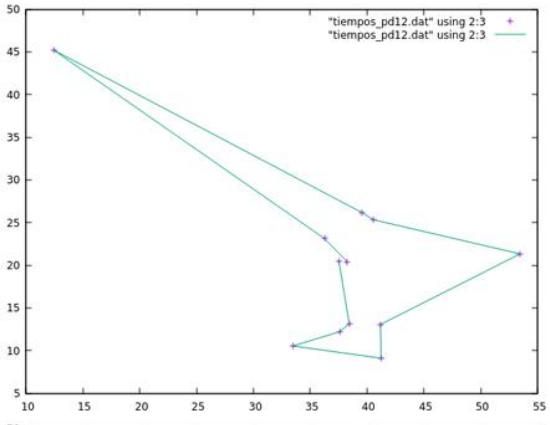


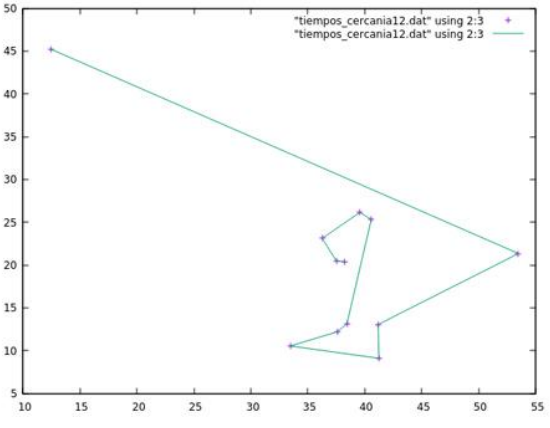


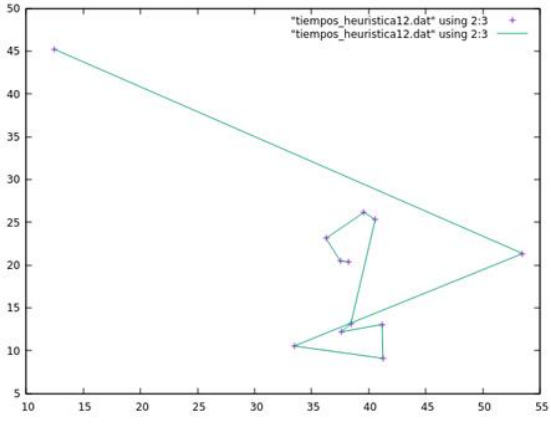


Para 12 ciudades:









RECORDAR QUE EXISTE UNA CARPETA CON TODOS LOS CODIGOS Y UN MAKE QUE HACE POSIBLE SU COMPILACIÓN Y EJECUCIÓN

Línea de compilación para todos: g++ -std=c++11 fichero.cpp -o ejecutable
Línea de ejecución para problema común: ./ejecutable fichero.tsp
Línea para Makefile: make / make clean