Universidad de Granada

PRÁCTICA 4

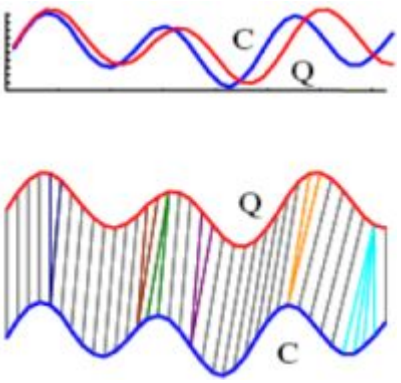
PROGRAMACIÓN DINÁMICA

Antonio Jiménez Rodríguez 2D (SubGrupo D2)

Problema asignado

Ejercicio 3:

En el análisis de señales con aprendizaje automático, es común comparar dos secuencias de datos (números enteros o reales) X= $(x_1, x_2, \dots, x_m)$ e Y= $(y_1, y_2, \dots, y_n)$, para conocer su similitud. Un ejemplo de este caso es el análisis de la marcha y la contabilización de pasos con los datos obtenidos por acelerómetros de teléfonos móviles o pulseras de actividad. El algoritmo Dynamic Time Warping (DTW) es una solución eficiente que nos permite conocer cómo de similares son dos señales de datos X e Y proporcionadas como entrada, buscando una medida de emparejamiento de una señal en otra. Un ejemplo de este emparejamiento se muestra en la siguiente figura, para dos señales Q (rojo) y C (azul).



Para comparar dos señales X= $(x_1, x_2, \dots, x_m)$ e Y= $(y_1, y_2, \dots, y_n)$, el valor DTW[i,j] devuelve la distancia mínima existente entre la señal X= $(x_1, x_2, \dots, x_i)$ y la señal Y= $(y_1, y_2, \dots, y_j)$. Se calcula como:

- DTW[i,j]= |X[i]-Y[j]| + DTW[i-1,j], si la distancia mínima de emparejamiento hasta X[i] e Y[j] (sin contar las posiciones i y j) es DTW[i-1,j]
- DTW[i,j]= |X[i]-Y[j]| + DTW[i,j-1], si la distancia mínima de emparejamiento hasta X[i] e Y[j] (sin contar las posiciones i y j) es DTW[i,j-1]
- DTW[i,j]= |X[i]-Y[j]| + DTW[i-1,j-1], si la distancia mínima de emparejamiento hasta X[i] e Y[j] (sin contar las posiciones i y j) es DTW[i-1,j-1].

Se pide: Implementar un algoritmo de programación dinámica que calcule la distancia DTW entre dos secuencias de números reales.

1. Análisis y notación a utilizar

La distancia que calcula este algoritmo no es una distancia Euclídea comparando un punto de una señal con el correspondiente de la otra. Como vemos en la imagen del enunciado tenemos las siguientes reglas:

- Cada índice de la primera secuencia debe coincidir con uno o más índices de la segunda secuencia y viceversa.
- El primer índice de la primera secuencia debe coincidir con el primer índice de la segunda secuencia (pero no tiene que ser su única coincidencia).
- El último índice de la primera secuencia debe coincidir con el último índice de la otra secuencia (pero no tiene que ser su única coincidencia).

Sabiendo esto vamos a usar la siguiente notación para la explicación de nuestro problema:

- Tenemos una secuencia de datos (representados como gráfica) con n valores, cuyo conjunto de coordenadas lo llamaremos X y lo escribiremos de la siguiente forma: $X = \{x_1, x_2, \dots, x_n\}$, siendo x_i el valor en el eje vertical y siendo i el valor en el eje horizontal.
- De igual forma tenemos una secuencia de datos (representados como gráfica) con m valores, cuyo conjunto de coordenadas lo llamaremos X y lo escribiremos de la siguiente forma: $Y = \{y_1, y_2, \dots, y_m\}$, siendo y_j el valor en el eje vertical y siendo j el valor en el eje horizontal.
- Vamos a denotar a X[i] como el valor x_i y a Y[j] como el valor y_j correspondiente a cada conjunto.
- Llamamos DTW a la matriz resultante de calcular la distancia DTW entre dos secuencias de números reales. Con índices de 0 hasta n y de 0 hasta m.
- Definimos emparejamiento como la unión entre el punto i y j que se producen entre las dos señales para generar la distancia mínima.

Ante este enunciado lo que hacemos es construir a partir de las señales una matriz que nos dará la mínima distancia que existe entre estas dos señales. La matriz la iremos formando como hemos indicado anteriormente, casilla a casilla, calculando cada DTW[i,j]. Con esto obtendremos la diferencia mínima entre las señales que se encuentra en la casilla [n,m].

Como acabamos de ver el planteamiento del problema se asemeja al del cambio de moneda. Este si se puede resolver por Programación Dinámica dado que es posible resolverlo por etapas, se puede plantear como una ecuación de recurrencia y cumple el principio de Optimalidad de Bellman. Por tanto lo resolveremos haciendo un símil con el mismo, y veremos que se cumplen estos requisitos para nuestro problema en las siguientes secciones.

2. Diseño de componentes

- **¿El problema a resolver es un problema de optimización?**

Se trata de un problema de minimización, en concreto se quiere encontrar la distancia mínima existente entre dos señales para saber cómo de similares son estas. Esta distancia se obtiene construyendo la matriz DTW y obtendremos el resultado en la posición [n,m].

- **¿El problema puede resolverse por etapas?**
    - Asumimos que los índices i y j de cada secuencias van de 1 hasta n,m recorriendo los números naturales. En el caso de valer 0 significa que no hay señal.
    - Llamaremos DTW[i,j] al la distancia mínima que hay al comparar una señal que va desde 1 hasta i, con otra que va desde 1 hasta j. Cada una con sus respectivos valores $X = \{x_1, \dots, x_i\}$ y $Y = \{y_1, \dots, y_j\}$.
    - Por tanto el objetivo es conocer .
    - Podemos resolver el problema por etapas para devolver la distancia mínima entre las dos señales, donde en cada etapa podemos calcularemos la distancia de los puntos X[i] con Y[j] y sumarle la comparación anterior mínima.
- **¿El problema puede modelarse mediante una ecuación recurrente?**

Sí, el objetivo sería conocer DTW[n,m] que es la distancia mínima entre una señal de tamaño n y otra de tamaño m, considerando que distancia no es Euclídea y podemos comparar cada punto con cada punto de las señales y un punto puede estar emparejado con varios.

- **Caso base**

  Tendríamos en este caso tres casos base:
    - Que no haya señales, es decir que n = m = 0, por lo tanto la distancia mínima sería 0.
    - Que tengamos una señal pero otra no, es decir, si n != 0 entonces m = 0 y viceversa, en este caso la distancia entre las señales sería **infinita**.

- **Caso general**

$$DTW[i,j] = |X[i] - Y[j]| + \text{mínimo}\{DTW[i-1,j], DTW[i,j-1], DTW[i-1,j-1]\}$$

El caso general supone que la distancia mínima total de comparar dos señales, una de tamaño i y otra de tamaño j, se consigue con la distancia que hay entre los puntos X[i] e Y[j] más el mínimo entre:

  - La suma de las distancia del emparejamiento entre el punto i-1 y j, y las anteriores acumuladas.
  - La suma de las distancias del emparejamiento entre el punto i y j-1, y las anteriores acumuladas.
  - La suma de las distancias del emparejamiento entre el punto i-1 y j-1, y las anteriores acumuladas.

- **¿Se cumple el Principio de Optimalidad de Bellman?**

Si una secuencia {DTW[1][1], … , DTW[n][m] es óptima entonces necesariamente cualquier subsecuencia que haya llevado a esta debe de serlo. Lo vamos a ver de la siguiente forma:

  - El caso base DTW[1][1] es óptimo trivial, ya que se trata de la distancia entre dos puntos, por lo tanto |X[1]-Y[1]| es la distancia mínima y es única para esos dos puntos.
  - Cuando i=1 y j=2 tenemos que DTW[1][2] también es óptimo ya que empareja el punto de X con los dos puntos de Y y la suma de las distancias es la distancia mínima. Pasaría lo mismo para i=2 y j=1.
  - Cuando llegamos a DTW[i][j], este valor también tiene que ser óptimo, dado que el algoritmo ha ido seleccionando los mínimos anteriores.
  - ¿Podría entonces existir una secuencia de decisiones distintas de las tomadas donde la distancia entre las señales sea inferior a DTW[i][j]? Reducción a lo absurdo: Sería imposible, dado que eso implicaría una solución mínima inferior a la proporcionada por la ecuación recurrente, que como hemos visto anteriormente el algoritmo siempre escoge la distancia mínima anterior calculada. Por lo tanto no puede existir una secuencia de emparejamientos menor a la obtenida con distancia mínima DTW[i][j] y queda demostrado que DTW[i][j] es óptimo.

2.1. Algoritmo diseñado

Nuestro objetivo es construir una matriz a partir de 2 secuencias de datos, comparándolas entre si de una forma determinada. El diseño de la memoria lo vamos a hacer de la siguiente forma:

- Los cálculos los vamos a ir almacenando en la matriz DTW_{nxm} donde cada casilla será la comparación del punto X[i] con el punto Y[j].
- La forma de rellenar la rellenar la matriz será por columnas, aunque esto no afecta al resultado final de la misma. Inicializaremos nuestra matriz a un valor "infinito". Asignaremos a la casilla DTW[0,0] el valor 0 y rellenaremos la matriz desde 1 hasta n y desde 1 hasta m de la forma DTW[i,j] = |X[i] - Y[j]| + mínimo{DTW[i-1,j], DTW[i,j-1], DTW[i-1,j-1]}. **La implementación de ese valor "infinito" se podría hacer con el valor -1 y bastaría con comprobar si uno es menor que otro y si es distinto de -1. No se dará jamás el caso que las tres comprobaciones del mínimo valor sean -1.**
- Al empezar a rellenar la matriz por el índice 1, los valores de la fila 1 se consiguen comparando |X[i]-Y[1]|+DTW[i-1,1] ya que los otros valores al estar en la fila 0 serían infinito (ó -1). Es decir habría que escoger entre mínimo{DTW[i-1,1], DTW[i,0], DTW[i-1,0]} = mínimo{DTW[i-1,1], infinito,infinito}. Al igual se haría con la columna 1 que se construiría |X[1]-Y[j]|+DTW[1,j-1].

Una vez completada la matriz tendremos en DTW[n,m] la solución de nuestro problema ya que se ha construido un camino mínimo desde [1,1] (el inicio de las 2 secuencias) hasta [n,m] (el final de las 2 secuencias) acumulando las distancias entre los puntos de estas.

El algoritmo tendrá como entrada:

- X[1..n]: un array con los valores de la primera señal a analizar.
- n: el tamaño de ese primer array.
- Y[1..m]: un array con los valores de la segunda señal a analizar.
- m: el tamaño de ese segundo array.

Y se tendrá como salida:

- s: distancia mínima entre las dos señales de entrada

Así el pseudocódigo del algoritmo será:

```
PROCEDIMIENTO DynamicTimeWarping(DTW[0..n][0..m], X[1..n], Y[1..m])
Para i=0 hasta n, hacer:
    Para j=0 hasta m, hacer:
        DTW[i][j] = INFINITO
    Fin-Para
Fin-Para
DTW[0][0] = 0

Para i=1 hasta n, hacer:
    Para j=1 hasta m, hacer:
        DTW[i][j] = |X[i]-Y[j]| + min{DTW[i-1][j], DTW[i][j-1],DTW[i-1][j-1]}
    Fin-Para
Fin-Para
DEVOLVER DTW[n][m]
```

Además, como tenemos la matriz ya construida podemos aprovechar para devolver como solución los emparejamientos que se producen entre las dos señales para generar la distancia mínima. Esto se resolvería deshaciendo la recurrencia hacia atrás viendo cuales son las decisiones que se han ido tomando para llegar a este valor. Por tanto devolveremos también:

- S[0..1][0..t]: Una matriz resultado, compuesta de 2 filas, donde en la primera fila las coordenadas correspondientes con la secuencia primera y en la última fila las coordenadas correspondientes con la segunda secuencia. Es decir, por cada columna tendremos los puntos de las secuencias que se están emparejando.
- t: el número de emparejamientos.

El pseudocódigo final del algoritmo quedaría así:

```
PROCEDIMIENTO DynamicTimeWarping(DTW[0..n][0..m], X[1..n], Y[1..m])
Para i=0 hasta n, hacer:
    Para j=0 hasta m, hacer:
        DTW[i][j] = INFINITO
    Fin-Para
Fin-Para
DTW[0][0] = 0

Para i=1 hasta n, hacer:
    Para j=1 hasta m, hacer:
        DTW[i][j] = |X[i]-Y[j]| + min{DTW[i-1][j], DTW[i][j-1],DTW[i-1][j-1]}
    Fin-Para
Fin-Para

i = n
j = m
S <- indices de DTW[i][j]
Mientras que i > 1 || j > i, hacer:
    S <- indices del min{DTW[i-1][j], DTW[i][j-1],DTW[i-1][j-1]}
    //Los indices se actualizarían a los del mínimo
Fin-Mientras

DEVOLVER S, DTW[n][m]
```

3. Cálculo de la eficiencia

```
void   DynamicTimeWarping(double **DTW, double *senialX, double *senialY, int n, int m, double &distanciaMinima, int **emparejamientos, int &nEmparejamientos){
  int **aux_emparejamientos, k = n, l = m, h = 0; // O(1)
  nEmparejamientos = 0;           // O(1)

  //Inicializamos la matriz
  for(int i = 0; i <= n; i++)      // O( i(n)+g(n) + h(n)*(g(n)+f(n)+a(n)) ) = O( 1+1 + n (m+1+1) ) = O(n*m)
    for(int j = 0; j <= m; j++)    // O( i(n)+g(n) + h(n)*(g(n)+f(n)+a(n)) ) = O( 1+1 + n (1+1+1) ) = O(m)
      DTW[i][j] = 100000;          // O(1)

  DTW[0][0] = 0;                   // O(1)

  //Contruimos la matriz
  for(int i = 1; i <= n; i++)      // O( i(n)+g(n) + h(n)*(g(n)+f(n)+a(n)) ) = O( 1+1 + n (m+1+1) ) = O(n*m)
    for(int j = 1; j <= m; j++)    // O( i(n)+g(n) + h(n)*(g(n)+f(n)+a(n)) ) = O( 1+1 + n (1+1+1) ) = O(m)
      DTW[i][j] = abs(senialX[i-1]-senialY[j-1]) + minimo(DTW[i-1][j], DTW[i][j-1], DTW[i-1][j-1]);

  //Soluciones
  distanciaMinima = DTW[k][l];     // O(1)

  aux_emparejamientos = new int*[2];  aux_emparejamientos[0] = new int[n*m];  aux_emparejamientos[1] = new int[n*m];   // O(1)


  while(k > 0 && l > 0){ // La eficiencia del bucle depende del número de emparejamientos que haya: O(nEmparejamientos) que no será mayor de n+n
    aux_emparejamientos[0][nEmparejamientos] = k;  aux_emparejamientos[1][nEmparejamientos] = l;  nEmparejamientos++;  // O(1)
    actualizarIndices(k, l, DTW[k-1][l], DTW[k][l-1], DTW[k-1][l-1]);
  }

  //Contruimos correctamente la matriz de emparejamientos ya que estan ordenados del último al primero y muy probablemente con memoria sobrante
  emparejamientos[0] = new int[nEmparejamientos];  emparejamientos[1] = new int[nEmparejamientos];  // O(1)

  for(int i = 0; i < 2; i++){                 // O( i(n)+g(n) + h(n)*(g(n)+f(n)+a(n)) ) = O( 1+1 + 1 (nEmparejamientos+1+1) ) = O(nEmparejamientos)
    for(int j = nEmparejamientos; j > 0; j--){  // O( i(n)+g(n) + h(n)*(g(n)+f(n)+a(n)) ) = O( 1+1 + nEmparejamientos (1+1+1) ) = O(nEmparejamientos)
      emparejamientos[i][h] = aux_emparejamientos[i][j-1];   // O(1)
      h++;  // O(1)
    }
    h=0;    // O(1)
  }

  delete[] aux_emparejamientos[0];  delete[] aux_emparejamientos[1];  delete[] aux_emparejamientos;   // O(1)
}
```

Aquí tenemos el análisis línea a línea del código fuente de nuestro algoritmo. Como vemos tenemos 4 partes con bucles::

- La primera es la encargada de inicializar la matriz, en la que hay dos bucles **for**. Estos se ejecuta un número de n*m veces con operaciones de eficiencia O(1), por lo tanto la eficiencia del propio bucle será de O(n*m)
- La segunda es la que construye la matriz anteriormente inicializada. Como en la primera, también hay dos bucles **for** que se ejecutan un número de veces igual a n*m. Para saber si tienen la misma eficiencia habría que estudiar las dos llamadas a la funciones dentro de los bucles, **abs(...)** que es una función que calcula el valor absoluto de un número con eficiencia O(1) y **mínimo(...)**. Que como podemos ver en la siguiente imagen es también un O(1). Por lo tanto esta parte tendría una eficiencia de O(n*m).

```
2  double minimo(double a, double b, double c){
3    double min = a;      // O(1)
4
5    if(b < min)     // max{ O(1), O(1) } = O(1)
6      min = b;      // O(1)
7    if(c < min)     // max{ O(1), O(1) } = O(1)
8      min = c;      // O(1)
9
10   return min;     // O(1)
11 }
```

- En la tercera parte tenemos un bucle while donde construimos una matriz solución que contiene los emparejamientos producidos. Este se ejecutará un número de nEmparejamientos veces que como mucho será un total de n+m veces. Tendríamos que estudiar también la eficiencia de la función **actualizarIndices(...)** que es usada para elegir el siguiente candidato deshaciendo la función recursiva. Como vemos en la siguiente imagen su orden es O(1).

```
13 void actualizarIndices(int &k, int &l, int a, int b, int c){
14   double min = a;                // O(1)
15   int aux_k = k-1, aux_l = l;    // O(1)
16
17   if(b < min){           // max{ O(1), O(1) } = O(1)
18     min = b;             // O(1)
19     aux_k = k;           // O(1)
20     aux_l = l-1;         // O(1)
21   }
22   if(c < min){           // max{ O(1), O(1) } = O(1)
23     aux_k = k-1;         // O(1)
24     aux_l = l-1;         // O(1)
25   }
26
27   k = aux_k;             // O(1)
28   l = aux_l;             // O(1)
29 }
```

- Por último tenemos dos bucles **for** que escriben la solución de forma ordenada en la matriz de salida. El primer bucle se ejecuta un número constante de veces (2) por lo tanto su eficiencia es O(1). El segundo bucle se ejecuta un número de nEmparejamientos que como hemos visto antes como mucho su valor sería n+m. Por lo tanto la eficiencia total de esta parte es O(n+m).

Podemos concluir entonces que la eficiencia final del algoritmo aplicando límites es de O(n*m).

Respecto a la ecuación recurrente inicial asumiendo que el coste computacional proviene de solucionar una etapa, como en cada etapa lo que vamos es calculando la distancia mínima entre una señal de tamaño i con una señal de tamaño j y DTW[i][j] depende de 3 casos anteriores como ya hemos dicho previamente, la eficiencia del cálculo tiene una ecuación en recurrencias T(n) = 3T(n-1) + 1. Resolviendo esta ecuación nos queda:

1) Pasamos todos los términos de T a la izquierda

$$T(n) - 3T(n-1) = 1$$

2) Resolvemos la parte homogénea de la ecuación

$$T(n) - 3T(n-1) = 0$$
$$t_n - 3\,t_{n-1} = 0$$
$$x^n - 3\,x^{n-1} = 0$$
$$x^{n-1}\,(x - 3) = 0$$
$$p\,(x) = x-3$$

3) Resolvemos la parte no homogénea de la ecuación

$$1 = b_1^n \bullet q_1(n)$$
$$b_1 = 1$$
$$q_1(n) = 1 \rightarrow d_1 = 0$$

4) Construimos el polinomio característico y calculamos t_n

$$p(x) = p_h(x) \bullet (x - b_1)^{d1+1} = (x - 3)*(x - 1)$$

5) Calculamos t_n

$$t_n = c_1\ 3^n\ n^0 + c_2\ 1^n\ n^0 \rightarrow \text{La eficiencia del algoritmo es } O(3^n)$$

Por tanto se gana mucho más en eficiencia utilizando la tabla DTW en lugar de implementar la ecuación en recurrencias, pudiendo observar la característica de Programación Dinámica que hace un uso abusivo de la memoria con tal de mejorar la eficiencia. Con esto se consigue que el algoritmo no recalcule múltiples veces un mismo valor.

4. Detalles de implementación

El lenguaje de programación utilizado para resolver la práctica ha sido C++. Se han tomado las siguientes decisiones de implementación:

- El índice para las señales de entrada va de 0 a n-1 y de 0 a m-1 como se ha hecho en el diseño, por simplicidad para el manejo del lenguaje C++.
- La memoria DTW se ha implementado como una matriz con memoria dinámica de tamaño (n+1)*(m+1). Esto se hace porque los cálculos de las distancias de las señales en la matriz empiezan en la posición 1, la posición 0 son los casos base.
- Los arrays X e Y se han implementado también con memoria dinámica
- El tipo de dato de los vectores y la matriz es **double** ya que estamos trabajando con números reales.
- Los valores de entrada X e Y no tienen ningún requisito previo.

Toda la práctica se ha implementado bajo un único fichero problema3.cpp que puede compilarse y ejecutarse con la orden **make**, gracias al makefile implementado. También se puede ejecutar como **./bin/problema3 ...** con los datos deseados.

Algoritmo main

1. Comprueba los argumentos de entrada, tenemos 3 casos:
   - **1.1. ./bin/problema3 ejemplo** que ejecuta el algoritmo con los datos del ejemplo de esta documentación.
   - **1.2. ./bin/problema3 aleatorio** que ejecuta el algoritmo con datos aleatorios
   - **1.3. ./bin/problema3 n X1 X2 ... Xn m Y1 Y2 ... Ym**
2. Reservar memoria dinámica para DTW, X e Y.

3. Introducir los datos de X e Y. Dependiendo del tipo de ejecución se hará de forma aleatoria, uno a uno o por leyéndolos por pantalla.
4. Aplicar el algoritmo distanciaMinima,emparejamientos = DynamicTimeWarping(DTW[0..n][0..m], X[1..n], Y[1..m])
5. Mostrar distanciaMinima y emparejamientos por salida estándar.
6. Liberar memoria dinámica.

El algoritmo DynamicTimeWarping(DTW[0..n][0..m], X[1..n], Y[1..m]) se ha implementado en una función fuera del main para tener recogido el código del mismo.

Se hace hincapié en que no se han realizado comprobaciones de errores de rango de valores , asumiendo como prerrequisitos de funcionamiento los siguientes:

- n y m deben de ser enteros positivos.
- No se deben introducir para X o para Y más valores del tamaño indicado.

5. Ejemplo

Vamos a ver el funcionamiento del algoritmo con las siguientes entradas:

- n=10
- m=10
- X = {1, 3, 4, 9, 8, 2, 1, 5, 7, 3}
- Y = {1, 6, 2, 3, 0, 9, 4, 3, 6, 3}

Con estos valores obtenemos los siguientes resultados:

```
antonio@antonio: ~/Documentos/UNIVERSIDAD/SEGUNDO CUATRIMESTRE/ALG/PRACTIC...
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
aaaaantonio@antonio:~/Documentos/UNIVERSIDAD/SEGUNDO CUATRIMESTRE/ALG/PRACTICAS/
PRACTICA4/Practica4_Antonio_Jimenez_Rodriguez$ make
g++ -Wall -g -c -o obj/problema4.o src/problema4.cpp -Iinclude
g++ -o bin/problema4 obj/problema4.o -Iinclude
bin/problema4 ejemplo
<<<<<<<<< Ejecucion del Problema 4 con los datos del ejemplo de la documentacion
 >>>>>>>>>

La señal X = {1, 3, 4, 9, 8, 2, 1, 5, 7, 3}
La señal Y = {1, 6, 2, 3, 0, 9, 4, 3, 6, 3}

La matriz DTW solo se puede mostrar bien si el tamaño de las señales es menor de
 10 (también depende del tamaño de la pantalla)

La distancia mínima entre las dos señales es de 15
Los emparejamientos son de la forma {Xi,Yj}: {1, 1}   {2, 2}   {3, 2}   {4, 2}
 {5, 3}   {6, 4}   {6, 5}   {7, 6}   {8, 7}   {9, 8}   {9, 9}   {10, 10}
```

El algoritmo inicializa la siguiente matriz:

	Y[j]	1	3	4	9	8	2	1	5	7	3
X[i]	0	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
6	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
2	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
0	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
9	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
6	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF

Una vez inicializada la matriz habría que ir calculando cada DTW[i][j]:

	Y[j]	1	3	4	9	8	2	1	5	7	3
X[i]	0	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	2	INF	INF	INF	INF	INF	INF	INF	INF
6	INF	5	3	INF	INF	INF	INF	INF	INF	INF	INF
2	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
0	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
9	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
6	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF

DTW[1][1] = |X[1] - Y[1]| - min(DTW[0,1], DTW[1,0], DTW[0,0]) =
= |1-1| - min(INF, INF, 0) = 0

DTW[1][2] = |X[1] - Y[2]| - min(DTW[0,2], DTW[1,1], DTW[0,1]) =
= |1-3| - min(INF, 0, INF) = 2

DTW[2][1] = |X[2] - Y[1]| - min(DTW[1,1], DTW[2,0], DTW[1,0]) =
= |6-1| - min(0, INF, INF) = 5

DTW[2][2] = |X[2] - Y[2]| - min(DTW[1,2], DTW[2,1], DTW[1,1]) =
= |6-3| - min(2, 5, 0) = 3

Continuamos completando la matriz y desde la posición DTW[n][m] deshacemos la ecuación recurrente obteniendo así los emparejamientos. Quedaría de la siguiente forma:

	Y[j]	1	3	4	9	8	2	1	5	7	3
X[i]	0	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	2	5	13	20	21	21	25	31	33
6	INF	5	3	4	7	9	13	18	19	20	23
2	INF	6	4	5	11	13	9	10	13	18	19
3	INF	8	4	5	11	16	10	11	12	16	16
0	INF	9	7	8	14	19	12	11	16	19	19
9	INF	17	13	12	8	9	16	19	15	17	23
4	INF	20	14	12	13	12	11	14	14	17	18
3	INF	22	14	13	18	17	12	13	15	18	17
6	INF	27	17	15	16	18	16	17	14	15	18
3	INF	29	17	16	21	21	17	18	16	18	**15**

Siendo 15 la distancia mínima entre las señales y las casillas en rojo los emparejamientos que se harían en cada punto que como podemos comprobar coincide con la ejecución del programa.