Algoritmo

1. Se descargan las dos primeras imágenes de la secuencia, I_i (*even_image*) y I_{i+1} (*odd_image*).

2. Calculamos los **vectores de movimiento** entre las dos imágenes, I_i (*even_image*) y I_{i+1} (*odd_image*).
   Tool: `me`.

3. Generamos una **imagen predicción**, PI_{i+2}, a partir de la imagen I_{i+1} (*odd_image*) y los vectores de movimiento calculados en el paso anterior.
   Tool: `decorrelate`.

4. A partir de la imagen predicción PI_{i+2}, obtenemos el **thumbnail** $Thumbnail(PI_{i+2})$.
   Tool: `mogrify`.

5. Solicitamos al servidor el thumbnail de la siguiente imagen de la secuencia, $Thumbnail(I_{i+2})$.

6. Calculamos las **diferencias** entre el thumbnail de la imagen predicción, $Thumbnail(PI_{i+2})$, y el thumbnail de la siguiente imagen, $Thumbnail(I_{i+2})$, y las ordenamos de mayor a menor.
   El resultado de este proceso nos devuelve la lista de `WOIs` que debemos solicitar al servidor, ordenadas de mayor a menor importancia.
   Tool: `differencesthumbnails`.

   TODO:

   - ~~No incluir en el listado aquellas `WOIs` donde no exista diferencia, para ahorrarnos tener que solicitarlas al servidor.~~
   - Podríamos *cambiar el método* que utilizamos para calcular las diferencias entre los dos thumbnails y realizar un estudio utilizando diferentes métricas:
     - **Y-SSIM**
       - Structural Similarity algorithm applied to luma channel only.
     - **RGB-SSIM**
       - Average of Structural Similarity algorithm applied to R, G, and B channels.
     - **IW-SSIM**
       - Information Content Weighted Structural Similarity algorithm applied to luma channel only.
     - **PSNR-HVS-M**
       - Peak Signal to Noise Ratio taking into account Contrast Sensitivity Function (CSF) and between-coefficient contrast masking of DCT basis functions.

7. Solicitamos al servidor los precintos/`WOIs` (1 precinto = 1 `WOI`) que nos interesan de la siguiente imagen, I_{i+2}, en función del *bitrate* estimado.
   Tool: `woistocache`.

   TODO:

   - Actualmente el valor del *bitrate* es un valor constante que se asigna al inicio de la ejecución del algoritmo. Sería interesante hacer pruebas donde el valor del *bitrate* pueda ir variando durante la ejecución del algoritmo.

8. Comprimimos la imagen predicción PI_{i+2} (obtenida en el paso 3) con la utilidad `kdu_compress`, para convertir de **.pgm** a **.j2c**.
   Los parámetros de compresión deben coincidir con los parámetros de compresión iniciales.

   NOTA:

   - Esta operación debe realizarse en el cliente y puede ser un poco lenta.

9. Convertimos el archivo **.j2c** de la imagen predicción PI_{i+2} a **.cache**.
   Utilizamos la utilidad `woistocache` y una lista que contiene todos los precintos de la imagen. Hay que tener en cuenta que la división de los precintos para toda la imagen sea exacta. En este caso el parámetro `Precincts Selection Mode = 0` para que la selección de los precintos se realice tal y como lo hace Kakadu.

   ```
   woistocache prediction_temp.j2c precincts/xxx.todos.txt WIDTH_RECONS HEIGHT_RECONS ((CL
   ```

10. Con la utilidad `extractcache` extraemos los precinctos de la imagen predicción PI_{i+2} que no están entre los precintos que hemos solicitado en el paso 7 para la imagen I_{i+2}. Los precintos que hemos extraído de la imagen predicción los guardamos en el archivo *temp_aux.cache*.

    ```
    extractcache next_image_j2c_cache prediction_temp.j2c.cache temp_aux.cache
    ```

11. Unimos los precintos que tenemos de la siguiente imagen I_{i+2} con los precintos de la imagen predicción PI_{i+2} que hemos extraído en el paso anterior y los ordenamos.

12. Al solicitar los precintos/`WOIs` de la siguiente imagen I_{i+2} en función de un determinado *bitrate*, puede ocurrir que no se reciban los precintos completos y que solamente se reciban ciertos paquetes. Por este motivo tenemos que revisar entre todos los precintos que hemos recibido cuáles son los que se encuentran incompletos y generar los paquetes vacíos necesarios para completarlos. En este punto tendremos un archivo **.cache** que contiene:

    - los precintos que hemos solicitado de la siguiente imagen I_{i+2},

- los precintos que hemos extraído de la imagen predicción PI_{i+2} que no están entre los nuevos precintos que hemos recibido,
- los paquetes vacíos que hemos ido generando para completar los precintos incompletos que hemos recibido de la siguiente imagen I_{i+2}.

Tool: `cookcache`.

13. Descomprimimos el archivo **.cache** con el code-stream.
Tool: `decodefromcache`.