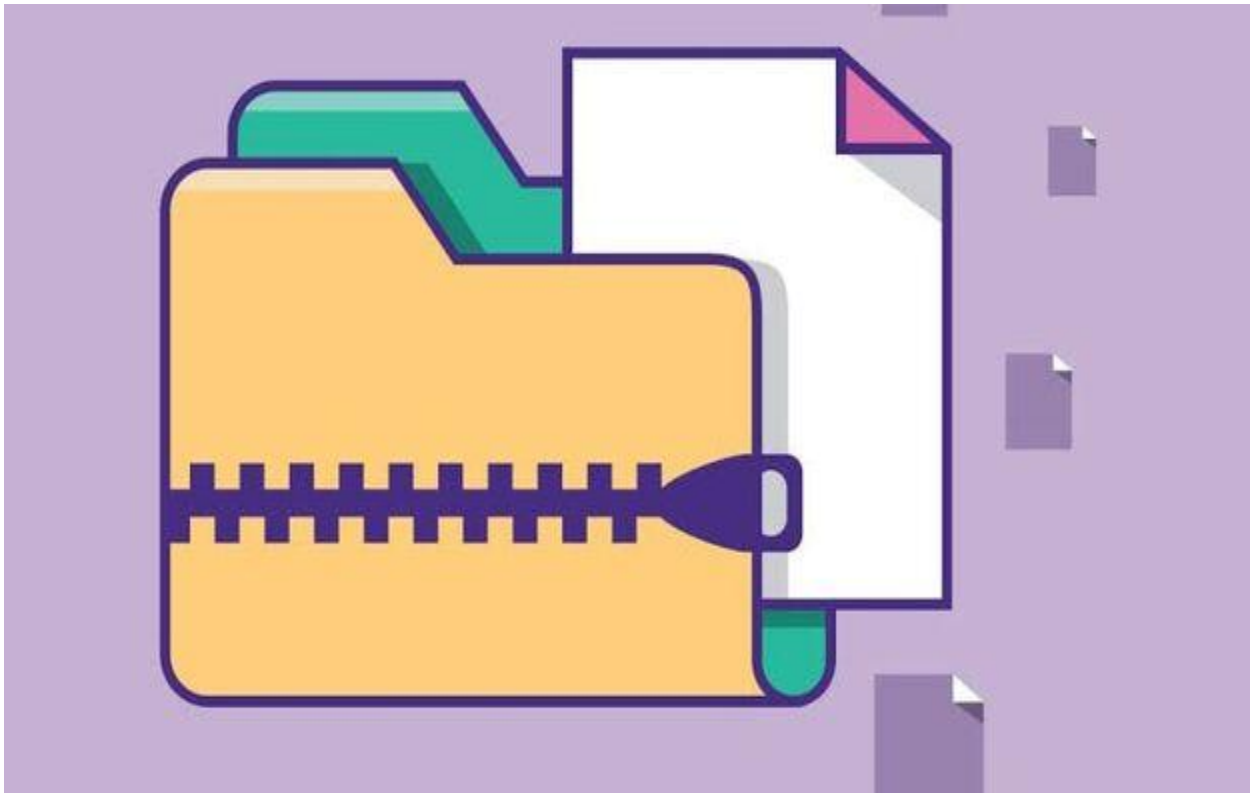
COMPRESSION



INDICE

OBJETIVO DE LA PRÁCTICA

La compresión es una técnica muy utilizada en diferentes ámbitos de la vida, pero en la informática es muy útil, ya que nos permite codificar la información que necesitamos guardar de forma que ocupa menos espacio y por lo tanto, menos memoria.

El hecho de que necesitemos codificar la información se debe a que los humanos la almacenamos de manera que nos sea fácil de interpretar y accesible, pero no de la manera más óptima. Comprimiendo la información, lo que estamos haciendo es almacenarla sin perder detalle pero ocupando una menor cantidad de espacio.

Además, la compresion sirve para tener una mayor privacidad en tus archivos y que las transferencias de estos sean más rápidas y eficientes.

Para ello, hay diferentes métodos pero en esta práctica solo utilizaremos tres:

- Run-Length encoding
- Algoritmo de Lempel-Ziv-Welch
- Codigos Huffman

Run-Length encoding

Run-Length encoding es un algoritmo de compresión que básicamente lee un elemento y comprueba cuantas veces se repite hasta llegar al siguiente elemento diferente. Un ejemplo sería algo así:

AAABBCCCCC

Esta sería la version sin comprimir. En este caso, una cadena de letras en la que se repiten 3 A seguidas, 2B y 5C.

La forma adaptada de comprimirlo que se nos propone en la prática sería la siguiente:

A, A, 1, B, B, 0, C, C, 3

Debemos tener en cuenta que el máximo de repeticiones que el programa debe aceptar son 257. Si después de esa cantidad se sigue repitiendo el mismo elemento empezamos de nuevo desde el elemento 257 hacia adelante hasta volver a llegar a 257 y así hasta acabar con los elementos repetidos. No obstante, debemos tener en cuenta que se representaría de la siguiente manera. Imaginemos que tenemos 260 veces repetido el caracter A:

A, A, 255, A, A, 1

Para obtener este resultado he hecho lo siguiente:

Primero declaramos dos variables del tipo int: currentByte, es decir, el byte actual que utilizando un InputStream lo leemos y con un OutputStream lo escribimos y un contador que guardará las veces que se repite el número en cuestión.

Mientras el número actual no sea -1, (de ser así el fichero se habría acabado) declararemos una variable llamada nextByte para leer el siguiente elemento y entraremos en un while con una serie de condiciones:

Si el número siguiente al que estamos leyendo es diferente del actual, entonces escribiremos el actual y continuaremos. Si el número siguiente es igual al actual, incrementaremos el contador a no ser que este, restándole 1, sea igual a 255, ya que recordemos que en ese caso tendremos que escribirlo junto al número repetido en dos ocasiones y volver a ponerlo a 0.

Si el contador es mayor que 0 una vez que los números actual y siguiente no son iguales, deberemos escribir el número actual dos veces y luego el contador -1.

Ahora que tenemos el algoritmo de compresión necesitaremos el algoritmo de descompresión.

DESCOMPRESION

Para descomprimir utilizaremos un algoritmo bastante similar al de comprimir. Declararemos una variable del tipo int que guardará el número actual que estamos leyendo y crearemos un while que se cumplirá siempre y cuando el número que estamos leyendo no es igual a -1, es decir, siempre que no se haya acabado el fichero.

Después estableceremos las siguientes condiciones:

Si currentByte es diferente de nextByte, escribimos el currentByte y ahora currentByte pasa a valer lo que vale nextByte.

Cuando currentByte sea igual que nextByte, crearemos una variable que será un contador que determinará cuantas veces tenemos que reescribir el número repetido.

Este contador será la suma de 2 más el siguiente número del fichero, es decir, lo que vendria a ser el contador en el archivo comprimido. De esta manera tendremos el número exacto de veces que se repite el número, y despues solo habrá que generar un bucle que dure tanto como el contador de repeticiones que hemos declarado.

Por ejemplo, si recibimos esto:

2, 2, 3

Esto quiere decir que en el archivo descomprimido se repite 5 veces el número 2.

Por lo tanto, si entramos en el bucle comparando los dos primeros números cuando estos son iguales significa que el siguiente número que leemos es el número de repeticiones restándole las dos veces que se encuentra el número ya escrito. Si generamos un for que escriba el currentByte hasta llegar a la suma del contador más 2, habremos escrito el número las veces que toca.

Una vez hecho esto, saltamos al byte siguiente y volvemos a comprobar que no sea -1 y continuamos así hasta acabar con el fichero.

LZW

El algoritmo de compresión Lempel-Ziv-Welch, simplificado así como la versión que necesitamos implementar en la práctica, se trata de una especie de diccionario que utiliza prefijos para guardar toda la información de una manera más eficiente. Cada entrada de este diccionario consta del índice y del elemento en cuestión

Básicamente recorremos el fichero y si vemos un elemento nuevo que no existía en el diccionario lo guardamos con un índice 0.

Para aclararlo utilizaremos el siguiente ejemplo:

«AABABCAEABDFFAAEFABDF»

Esto comprimido con LZW sería así:

0, A | 1, B | 2, C | 1, E | 2, D | 0, F | 6, A | 4, F | 5, F

Básicamente utilizamos índices que nos sirven para hacer referencia a los valores o símbolos que vamos guardando. Es importante mencionar que debemos limpiar el diccionario una vez que este llegue a las 255 entradas.

De esta forma, leemos el primer valor. Es una A, no la tenemos guardada así que creamos una entrada con el indice 0 y el valor que le pertenece, en este caso A. Despues lo siguiente que tenemos es otra A, que la tenemos registrada y una B. Entonces, haremos referencia a la entrada 1 y escribiremos 1,B, ya que la entrada uno tiene la letra A.

Se trata de reproducir un algoritmo que repita este proceso continuamente y para ello, yo he hecho lo siguiente:

Creamos un mapa que tenga como clave una String y como valor un prefijo.

Tambien declaramos una variable int (prefix counter) que guardará el número de la entrada en el diccionario y una String vacía (s).

Guardamos en un int el byte que estamos leyendo y lo guardamos en su versión de char en la string vacía que habremos declarado anteriormente (s).

Ahora, mientras se cumpla la condición de que el fichero no se haya acabado, declararemos las siguientes condiciones:

Siempre que nuestro diccionario no contenga “s” crearemos una nueva entrada en la que guardaremos “s” y el prefijo. Si nuestro diccionario contiene 255 entradas lo vaciaremos y escribiremos la nueva.

Una vez hemos hecho esto, guardaremos en una string llamada “pre” el prefijo de la entrada utilizando el método substring y capturando de esta manera el prefijo, el cual si no existe será 0.

Básicamente, si el elemento no existe se le asigna el prefijo 0, y si existe, se le da una referencia a la entrada en la que se guardó el patrón.

DESCOMPRESION

Para descomprimir declaramos una variable llamada offset que mirará el índice (en caso de que haya) y el encodedByte representa el byte codificado.Se crea una lista llamada prefixes para almacenar los prefijos decodificados y un HashMap llamado knownPrefixes para mantener un registro de los prefijos conocidos.

Ahora introducimos un bucle while que se ejecutará hasta el final del archivo y dentro de este se comprueba el valor del offset para determinar si el encodedByte tiene un prefijo en la tabla o no.

Si el offset es mayor que 0, se recupera el prefijo correspondiente de la lista prefixes utilizando el índice offset - 1.

Si el offset es igual a 0, significa que el encodedByte no tiene un prefijo en la tabla y creamos un nuevo prefijo vacío. Se forma un nuevo prefijo combinando el prefijo de la tabla (o el prefijo vacío) y el encodedByte.

Esto se realiza creando una nueva lista newPrefix, copiando los elementos del prefijo existente y agregando el encodedByte como un nuevo elemento.

Recorremos el nuevo prefijo y se escriben los bytes en el OutputStream os. Esto restaura la secuencia de datos original.Si el nuevo prefijo formado no está presente en la tabla (knownPrefixes), se agrega a la tabla y se añade a la lista prefixes para su uso posterior en la descompresión.Se leen los siguientes bytes del InputStream is para continuar el ciclo y obtener el siguiente offset y encodedByte.