L'algorisme LZ78 es un algoritme de compresió de text sense perdues, aquest algoritme es basa en les repeticions del propi text que s'esta comprimint,

les quals utilitza per a reduir el tamany del fitxer codificantho en format; posicio, punter a posició i següent caràcter.

Quan l'algoritme es troba iterant per un text, aquest comprova si la part on es troba és una repetició per on ja s'ha iterat anteriorment.

Si es aixi, es codifica un "punter a la posició " el qual indica a quina posició hem d'anar per a trobar la coincidencia i quina es la longitud d'aquesta.

A més li afegirà un caràcter que serà el seguent de la cadena. D'aquesta manera quan es vulgui descomprimir simplement s'haurà d'interpretar el punter i els

caràcters seguents i ja tindriem la cadena de caràcters no repetida completa.

Ara be, necessitem un lloc on buscar si el caracter per on ens trobem actualment es una coincidencia a aquesta estructura de dades .

L'estructura de dades que he fet servir per al diccionari es un HashMap de <Integer,String>. El fet d'utilitzar un hashmap es degut a la seva alta eficiencia en

operacions de busqueda-inserció, i ja que el meu codi itera i busca constantment les repeticions en el string entrant i les ha d'emmagatzemar,

he cregut que és el més eficient.

Per tant el meu algorisme el que fa explicitament és :

1- Llegir caràcter per caràcter la entrada i anar ubicant aquests caràcters al mapa amb el format: posició al mapa, {referencia al caràcter actual+seguent}

2- Si trobem un caràcter que ja és al mapa, el referenciem amb aquest "punter" i mirem si el seguent està al mapa també ordenat de la manera en la que apareix

en la entrada. Si no hi és, referenciarem amb un 0 el punter. És a dir que si una cadena de caracters de mida n està al mapa hem de referenciar on del

mapa és i afegir-li el següent caràcter que no apareix en el mapa seguidament d'aquesta referència, i al descomprimir-lo aquesta cadena serà de n+1 caràcters.

La qüestió és anar emmagatzemant les repeticions del text en el mapa per a que es comprimeixi el text, pero amb una codificació fàcil de decodificar.

He tingut un problema a l'hora d'escriure el fitxer comprimit ja que no he escollit la millor forma o és una forma no tot lo eficient que podria ser, depenent del cas, son més grans que l'original encara que la compressió sigui la adecuada, i el temps pot ser alt per arxius molt grans.

--------------------------------------------------------------

Feina de recerca:

Vaig començar la meva recerca de l’algorisme a internet, buscant diferents pàgines amb informació sobre la família de LZ i més concretament la familia LZ7, ja que el LZ77 i el LZ78 son molt similars i a l’hora d’entendre-ho em va ser més fàcil unir conceptes.

Un cop apres això, vaig buscar implementacións practiques del meu algoritme en diferents llenguatges de programació tals com c++ i java, per fixar-me en els procediments i quines estructures de dades feien sevir per a implementar l’algorisme; em vaig fixar sobretot amb la utilització d'un hashmap per a la implementacio i una array per a poder iterar d’una manera més senzilla per l’arxiu.

-------------------------------------------------------

Resultats:

He pogut observar que és un algorisme molt poderòs per a arxius binaris i per a textos llargs quan s’utilitza la forma d’escriure adecuada.

Els problemes principals que tenia eren del temps de compressió, ja que al utilitzar dues estructures de dades trigaven molt, encara que vaig llegir que el StringBuilder era barat i no ho ha resultat ser tant, i la necessaria implementació d’una Array va fer que tot s’alentís més.

Un text massa curt no val la pena comprimir-lo ja que mai serà útil.

La manera d’escriure el fitxer, el charset, crec que ha sigut un problema molt important, ja que no he sapigut trobar el més adecuat, com els meus companys, ja que és un tema que mai havia vist ni havia tocat, i em venia de zero i no podia entendre-ho tot el que m’haguès agradat, el meu desconeixement era molt gran.