L'algoritme LZSS es un algoritme de compresió de text sense perdues, aquest algoritme es basa en les repeticions del propi text que s'esta comprimit, les quals utilitza per a reduir el tamany del fitxer codificant punters de 2 bytes en lloc de tota la repetició del text.

Quan l'algoritme es troba iterant per un text, aquest comprova si la part on es trova es una repetició per on ja s'ha iterat anteriorment. Si es aixi, es codifica un "punter compost" el qual indica quants de caracters s'han de retrocedir per a trobar la coincidencia i quina es la longitud d'aquesta. D'aquesta manera quan es vulgui descomprimir simplement s'aurà d'interpretar el punter.

Ara be, necessitem un lloc on buscar si el caracter per on ens trovem actualment es una coincidencia, a aquesta estructura de dades li direm Sliding Window, on hi guardarem els ultims 4096 caracters del text juntament amb les seves posicions dintre d'aquest, el numero 4096 no es un numero a l'atzar, l'he escollit perque l'interval 0-4095 pot ser codificat emprant només 12 bits, valor que ens indicara quants de caracters hem de retrocedir en el text, recordem que el punter compost es codificat amb 16 bits, qual cosa ens deixa 4 bits per a codificar la longitud de la coincidencia, que podra ser de fins a 16 caracters. Aixo ens deixa amb possibles coincidencies de fins a 16 caracters que es poden trovar 4095 caracters anteriors a la posició actual. Aquests nombres no son arbitraris, els he definit teninint en compte el tenir coincidencies lo suficientment llargues mentre tinc un gran rang on buscar coincidencies.

L'estructura de dades que he fet servir per a la Sliding Window es un HashMap de Pairs<Integer,Character>, el fet de fer servir Pair es simple d'explicar, necessito emagatzemar cada caracter amb la seva posicio al string. El fet d'utilitzar un hashmap es degut a la seva alta eficiencia en operacions de busqueda-inserció, la Sliding window es una estructura on realitzo moltes operacions de busqueda i per tan no m'importava ocupar una mica mes de memoria ram per tal de tenir una millor eficiencia, encara que haguessa pogut implementar l'algoritme amb nomes un array de pairs.

L'ultim punt a tractar son els flags, que son els flags? L'algoritme quan vol descomprimir un fitxer necessita saver si el caracter que esta tractar es un caracter raw (un caracter no codificat i no un punter) o un punter.

Per a això l'algoritme utilitza un bit de flag el qual es trova just avans de cada element del string (ja sigui raw char o punter) si el bit es 1, indica que el següent element es un punter, sino, es un caracter sense codificar.

Això pero planteja un problema, no podem escriure caracters de 9 bits o punters de 17 ja que fer-ho trencaria la paritat del string, llavors una solucio seria utilitzar el primer bit de cada byte per a codificar el flag, pero aixo ens faria perdre molta resolucio de codificació, a part de fer-nos perdre 128 caracters ASCII que no podriem representar. Com ho he solucionat? cada 8 elements del string (raw chars o punters) he codificat un byte on els seus 8 bits representen els flags dels següents 8 elements del string. D'aquesta manera puc codificar els flags sense perdre resolució.

ALTRES DESICIONS

Tambe he decidit no codificar amb punters coincidencies mes petites o iguals a 2 caracters, ja que si la coincidencia es de 1 caracter, el punter ocupa mes que el caracter sense codificar, i si es de 2, el punter ocupa el mateix, el qual no comprimeix el String pero si que consumeix temps de processament.

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

Feina de recerca:

Vaig començar la meva investigació directament a internet, el primer que vaig buscar era que eren la familia dels algoritmes LZ, ja que els noms de 3 dels algoritmes que hem implementat en aquest projecte són molt pareguts, despres vaig concretar mes i vaig investigar LZSS, descubrint que no era tan diferent del LZW i en que es diferenciaven.

Un cop apres això, vaig buscar implementacións practiques del meu algoritme en altres llenguatges que coneguessa mes que java, en concret en c++ i en python, i vaig observar quines tectiques de programacio i quines estructures de dades feien sevir per a implementar el meu algoritme; hem vaig fixar sobretot amb la utilització d'un hashmap per a la implementacio de la sliding window i de la forma de codificar els punters compostos caracteristics del meu algoritme.

Finalment vaig trovar una pagina web on un programador explicava tot el process que havia seguit per a implementar aquest mateix algoritme amb java, encara que interessant i util, molt del material que ell emprava no m'era d'utilitat ja que, per exemple, els bits de flag no els implementava com jo volia implementar-los, sino d'una forma que, encara que mes eficien, era molt mes complexa de programar; motiu pel qual vaig decidir pensar una implementació alternativa (un bit de flags cada 8 elements) la qual ha resultat bastant satisfactoria.

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

Resultats:

L'algoritme LZSS es un algoritme molt poderos pel que fa a la compresió de textos sempre i quant:

-El text a comprimir sigui com a minim de la mida de la sliding window

-El text a comprimit tingui un minim de repeticions

Si es tracta per exemple de un text aleatori, es provable que l'algoritme no nomes no comprimeixi el fitxer sino que el faigue més gran (degut a l'aparició dels bytes de flags).

El mateix passa amb un text massa curt.

Pel que fa als problemes que he tingut jo, tractar amb dades binaries ha sigut dificil, mes quant no sempre tenia el control bit a bit de les dades amb les que tractava. He tingut molts de problemes amb els metodes d'escriptura i lectura de java i amb els charsets que aquest utilitzaven per llegir i escriure a disc.

Vull fer molt d'emfasi amb el tema dels charsets ja que, si el programa te algun error a l'hora de comprimir i descomprimir, es molt provable que sigue degut al requeriment d'utilitzar un charset diferent per a cada algoritme, degut a que tots tractem les dades de forma diferent. Una dificultat afetgida que sense dubte, crec que no es adecuada per al projecte, ja que segons el meu punt de vista, la dificultat del projecte auria de residir en la implementació dels respectius algoritmes i en la coordinacio d'un grup de treball.