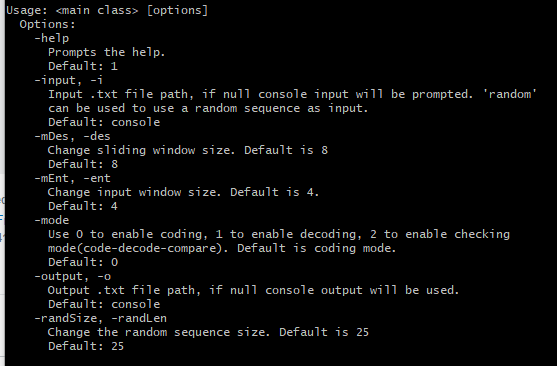
Exercici Compressió LZ-77

**Execució**

En aquest exercici hem afegit l’ús de JCommander, de forma que l’ús de paràmetres és molt més senzill.

Si executem LzDecoder.jar amb el paràmetre -help 1 observarem els possibles paràmetres que aquest admet.



L’ús del programa és simple, si no s’afegeixen paràmetres per defecte funcionarà a través de la consola, però tenim l’opció de modificar variables, i activar els diferents modes.

**Funcionament**

Codificació

Independentment de com s’ha entrat la seqüència, la codificació d’ella segueix el mateix esquema. Entenem com a codificació la transformació d’una seqüència vàlida de zeros i uns a una codificació vàlida de compressió LZ-77.

Per la realització de la codificació total d’aquesta seqüència ens ajudem d’un *while* principal situat en el *main*. Aquest s’executarà sempre i quant tinguem més input a codificar o el buffer d’entrada no tingui la mateixa mida que la mida d’aquest buffer passada per paràmetre abans de l’execució del programa. Mentre aquestes dues condicions es compleixin, anirem cridant la funció *searchBuffer()* amb els arguments de l’execució del programa i un *String* ‘code’ on anem inserim el resultat de la codificació.

La funció *searchBuffer()* realitza tota la feina de la codificació i també gira al voltant d’un *while*. Aquest també té dues condicions: que la mida del buffer lliscant i el buffer d’entrada no siguin iguals a les mides passades en els *args[]*. A dins d’aquest *while*, per tant, mirem quina de les dos condicions s’ha complert: la mida del buffer d’entrada és mes petita o més gran? La mida del buffer lliscant és mes petita o més gran? Cada una d’aquestes dos preguntes és un *if* a dins del *while*.

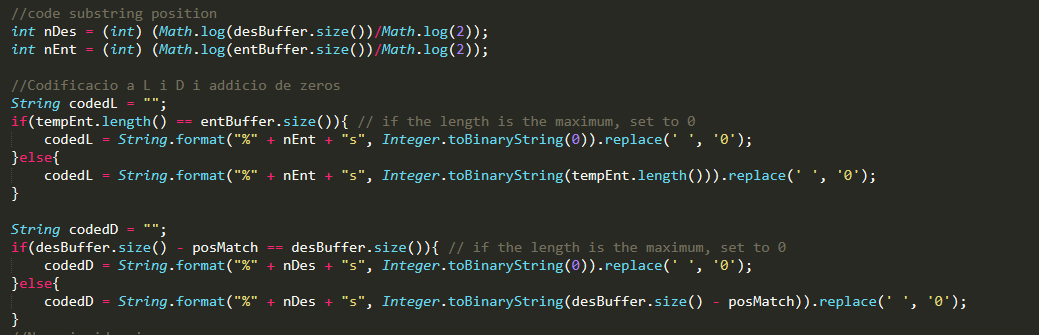
En el cas de tenir *entBuffer* més petit que la mida preestablerta, mirem si tenim més input a modificar. De ser així, vol dir que hem de treure el primer dígit d’*inputData* i afegir-lo al buffer d’entrada. Si pel contrari, la mida d’*entBuffer* és més gran que la preestablerta, mirem si es tracta de la fi de la codificació. Per saber-ho, mirem si el buffer lliscant té la mida adequada. De ser així, buidem completament *entBuffer* i retornem el codi. Si no es compleix la condició, no realitzem res més.

En el cas de tenir *desBuffer* més petit que la mida preestablerta, traiem el primer dígit d’*entBuffer* i l’afegim a aquest buffer lliscant.

Ara que tenim els dos buffers amb els dígits i mida que toquen, hem de començar a ‘codificar’ en sí: mirar si hi ha coincidències de combinacions d’*entBuffer* a *desBuffer*. Prèviament convertim els dos buffers en *Strings* amb la funció *getStringBuffer()*. La comprovació de si hi ha una coincidència es realitza amb el mètode d’*String* *contains()*, i s’obté la posició d’aquest *match* amb *indexOf()*. Si es troba una coincidència, es para l’execució del for: només ‘ens interessa’ un *match* (el primer).

Abans de tractar aquest *match*, fem el càlcul de *codedL* i *codedD*, corresponents als dígits de la codificació (és a dir, la notació (L,D) on L és la posició en el buffer lliscant i D el número de dígits coincidents).

Per a codificar els valors L i D, simplement fem una transformació de un enter a binari. Com que no utilitzem el 0 per a comptar les posicions, el valor corresponent a 2^n es codifica al valor 0.



En aquest moment pot ser que tinguem la posició d’un *match* o que no hi hagi hagut un *match* (posició = -1). En aquest últim cas, simplement traiem el primer dígit d’*entBuffer* i l’afegim a l’*String* de codificació. Si hem tingut un *match*, s’afegeix a l’*String* de codificació el càlcul de *codedL* i *codedD*. Ara, hem de treure tants dígits del buffer lliscant com la mida del *String* coincidència.

Finalment, mirem si ‘ja hem acabat’ per a realitzar la correcte codificació dels últims dígits, ja que pot ser que *inputData* estigui buit però encara hi hagi dígits en *entBuffer* i *desBuffer*. Per tant, si *inputData* està buit, traiem tants dígits de *entBuffer* com la mida de la resta de la mida preestablerta del buffer lliscant amb la mida actual d’aquest buffer. A continuació, mirem si queden elements al buffer d’entrada que no es ‘codifiquen’, els quals s’afegeixen tal qual a l’*String* de codificació.

Atenció: en tot el programa, cada vegada que eliminem o afegim elements el els buffers, fem la crida al mètode *trimToSize()* per assegurar que la mida dels buffers que llegim sigui la correcte.

Descodificació

La descodificació de l’algoritme LZ-77 segueix els següents passos:

1. Treure el primer element de l’*String* codificat (és a dir, la primera *String* abans del primer espai) corresponent a la finestra lliscant.
2. Ara, per cada *String* partida per cada espai, fem:
   1. Miro la mida. Si és d’1, vol dir que estem a la part final de la descodificació. Per tant, l’afegeixo tal qual al *String* descodificat, trec el primer element de *desBuffer* i afegeixo aquest nou element al final d’ell. Retorno la descodificació
   2. Si no té mida 1, (és L) obtenim ‘la D’ extraient el següent element de la seqüència a descodificar. Ara tenim elem1 (L) i elem2 (D). Convertim els dos a binari. Si L o D és zero, hem de convertir aquest número a 2mida del dígit, ja que utilitzem els ‘zeros’ dels números binaris per indicar el número 2mida del dígit, és a dir, 00 = 4, 000 = 8, etc.
   3. Ara que tenim L ben codificat, traiem tants dígits com L del *desBuffer*. La posició d’on hem de treure L dígits l’obtenim amb mida del buffer-L + i. Els afegim a l’*String* de descodificació i retornem.

**Proves i resultats**

Per a comprovar que el nostre codi funciona correctamente hem afegit un mode ‘check’ al nostre programa, que el que fa és codificar la seqüència proporcionada, i seguidament descodificar-la, per a comprovar que la original i la resultant de la descodificació són idèntiques.

Per a fer proves amb diferents valors de mEnt i mDes hem fet servir seqüències binàries aleatòries de 15000 bits.

La seqüència aleatòria és diferent per a cada test (no fem servir mai una seqüència repetida, però al ser llargues i aleatories la distribució de símbols és pràcticament quasi igual).

Per executar el nostre programa per a fer les proves només fa falta executar:

***java -jar LzDecoder.jar -i random -mode 2 -randSize 15000 -mEnt x -mDes y***

on x i y són els valors a testejar.

Aquests són els valors de compressió que hem obtingut:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **mDes**  **mEnt** | **8** | **32** | **64** | **256** | **512** | **1024** | **2048** |
| **4** | 0,368:1 | 0,423:1 | 0,399:1 | 0,337:1 | 0,315:1 | 0,300:1 | 0,296:1 |
| **8** | 0,336:1 | 0,499:1 | 0,544:1 | 0,576:1 | 0,569:1 | 0,550:1 | 0,537:1 |
| **32** | - | 0,425:1 | 0,475:1 | 0,557:1 | 0,589:1 | 0,622:1 | 0,663:1 |
| **64** | - | - | 0,442:1 | 0,517:1 | 0,555:1 | 0,591:1 | 0,632:1 |
| **256** | - | - | - | 0,472:1 | 0,503:1 | 0,542:1 | 0,578:1 |
| **512** | - | - | - | - | 0,486:1 | 0,523:1 | 0,559:1 |
| **1024** | - | - | - | - | - | 0,506:1 | 0,550:1 |

Com es pot observar, els valors de compressió sempre estàn per sota de 1, això significa que realment no aconseguim comprimir les dades, sinó el contrari, utilitzem més informació per a emmagatzemar les mateixes dades.

El valor màxim de compressió està en 0.663:1 amb valors de mEnt=32 i mDes=2048.

El motiu pel que el nostre compressor no aconsegueix comprimir les dades pot ser un dels següents:

* El format en que guardem la seqüència codificada no és el més òptim, per a cada “chunk” dades codificades hi afegim un símbol “ “, el qual tenim en compte a l’hora de calcular la mida de les dades codificades, ja que per a poder llegir-ho ens és necessàri. Això pot augmentar la mida del codi resultant significativament.
* Hi ha casos de seqüències que podríem tenir en compte, com és el cas de tenir un buffer ple de 0s o ple de 1s, aquest cas fa que l’ús d’un buffer per a observar patrons vistos anteriorment perdi sentit, ja que deixa de ser eficient. Es pot fer servir una tècnica per a que, quan es troba aquesta situació, s’inserti un bit invertit a la seqüència evitant el problema en qüestió. (tant el codificador com el decodificador ho hauran de tenir en compte).

Un dels usos comuns de LZ77 és l’addició de Huffman, ja que els dos algoritmes es poden combinar perfectament, aconseguint millor compressió.