Compressor i descompressor d'altures de vol

Enric Carpintero Rico

Abril 2025

1 Introducció

Aquest document detalla la implementació del compressor i descompressor de fitxers d'altures de vol demanat per l'assignatura de Compressió de Dades i Imatges de la Facultat d'Informàtica de Barcelona, quadrimestre de primavera 2025.

A excepció de la classe auxiliar `BinTree.hh`, implementada pel professorat del departament de CS de la UPC, tot el codi de l'entrega ha estat desenvolupat per mi.

Per als fitxers de prova proporcionats inicialment a Atenea, `file11111.txt` i `file22222.txt` la ràtio de compressió obtinguda ha estat de 5,78 i 7,97, respectivament. La mida del programa compilat resultant és de 85 kB, mentre que el codi font sense compilar, excloent les llibreries de sistema i les llibreries estàndard, ocupa 31 kB. El temps de compressió pels dos fitxers de prova, als ordinadors de la facultat, és de 2,1 i 1,7 segons, respectivament. El temps de descompressió és 6,7 i 5,7 segons, respectivament.

2 Algorisme

Aquest algorisme està dissenyat únicament per al tipus de fitxers indicats per l'exercici: Cadenes de text pla en ASCII, representant seqüències de nombres naturals, o el nombre -1, separats entre ells per un únic espai o salt de línia. Per tant, aquest algorisme no contempla qualsevol altre tipus de fitxer o dades que no compleixi amb aquestes característiques, i per tant no es pot fer servir com a compressor i descompressor de fitxers de forma general.

L'algorisme de compressió està basat en:

- El fet que tots els elements dels fitxers a comprimir corresponen a nombres naturals.
- Increments/diferències entre elements consecutius.
- Codis de Huffman.

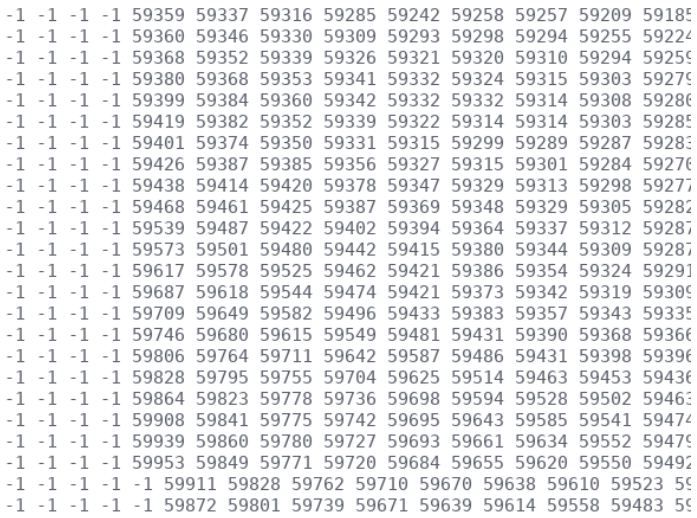

Figure 1: Exemple de seqüència de dades que comprimeix l'algorisme.

En primer pas, es calculen les diferències entre els nombres naturals consecutius, les seqüències de nombres -1 seguits, o la presència dels salts de línia.

A cada ocurrència, se li assigna un codi de longitud de bits variable indicant quin element o elements hem trobat, seguit d'un paràmetre, també de longitud variable, complementant el primer codi.

Finalment, aquestes ocurrències es divideixen en blocs de mida variable, i es codifiquen amb codis de Huffman.

2.1 Codis per diferència sobre el fitxer base

Al text pla original, podem trobar:

Nombres naturals: Per a cada natural que trobem, calculem la diferència de valor respecte a l'anterior nombre natural trobat. Segons el valor absolut d'aquesta diferència, el codifiquem de diferents maneres:

- **Els nombres son iguals:** Codifiquem amb `0b10`.
- **La diferència és 1..16:** Codifiquem amb `0b00` seguit d'un bit pel signe *(0: nou nombre és més gran. 1: nou nombre és més petit)*, seguit per la codificació en binari sense signe de restar 1 al valor absolut de la diferència, amb 4 bits.
- **La diferència és 17..272:** Codifiquem amb `0b010` seguit d'un bit pel signe *(0: nou nombre és més gran. 1: nou nombre és més petit)*, seguit per la codificació en binari sense signe de restar 17 al valor absolut de la diferència, amb 8 bits.

- **La diferència és 273..4368:** Codifiquem amb `0b011` seguit d'un bit pel signe *(0: nou nombre és més gran. 1: nou nombre és més petit)*, seguit per la codificació en binari sense signe de restar 273 al valor absolut de la diferència, amb 12 bits.
- **La diferència és 4369..:** Codifiquem amb `0b1100` seguit de la codificació binària sense signe del nombre amb 31 bits.
- **El primer natural del fitxer** es codifica amb `0b1100` seguit de la codificació binària sense signe del nombre amb 31 bits.

El nombre -1: Es codifica amb `0b111` seguit de seguit per la codificació en binari sense signe de restar 1 al nombre de *-1* seguits trobats de forma no interrompuda per cap natural ni salt de línia, amb 13 bits.

Salt de línia: Es codifica amb `0b1101`.

Els espais entre nombres no apareixen codificats a l'algorisme de compressió. Per cada dos nombres enters seguits, assumim que hi ha un sol caràcter d'espai entre els dos, tret que hi hagi un salt de línia.

2.1.1 Exemple

La seqüència:

$$1031\ 1031\ 1125\ \text{-}1\ \text{-}1\ \text{-}1\ 1117\ \backslash n$$

La codificaríem com a:

- `1100 00000000000000000000010000000111` - Primer nombre que trobem, s'escriu `0b1100` seguit del literal amb 31 bits.
- `10` - Segon nombre, com que és igual a l'anterior, s'escriu `0b01`.
- `010 0 01001101` - Tercer nombre, la diferència entre aquest i l'anterior és de *1125-1031 = 94*, $17 \leq 94 \leq 272$. Per tant, s'escriu `0b010`. La diferència és positiva, per tant, s'escriu `0b0`, i finalment, la resta *94-17 = 77* amb 8 bits.
- `111 0000000000010` - Sèrie de 3 *-1* seguits. S'escriu `0b111` seguit del nombre d'elements, restat 1, amb 13 bits.
- `00 1 0001` - Quart nombre, la diferència entre aquest i l'anterior és de *1117-1125 = -8*, $1 \leq 8 \leq 16$. Per tant, s'escriu `0b00`. La diferència és negativa, per tant, s'escriu `0b1`, i finalment, la resta *8-1 = 7* amb 4 bits.
- `1101` - Un salt de línia s'escriu com a `0b1101`.

`11000000 00000000 00000000 01000000 01111001 00010011 01111000 00000000 10001000 1`*`(0000000)`*

Que, expressat en hexadecimal, veuríem com a

`0x C0 00 00 40 79 13 78 00 88 80`

2.2 Codis de Huffman

Aquesta primera codificació es divideix en blocs de fins a 65536 bytes. Aquests blocs es tornen a codificar fent servir codis de Huffman. Un bloc no inclou necessàriament un nombre sencer de símbols codificats: un mateix símbol pot quedar dividit amb x bits al bloc i, i amb y bits al bloc $i+1$. Sí que és necessari que un bloc inclogui un nombre sencer de bytes, a excepció de l'últim bloc, pel qual el contingut dels bits no utilitzats de l'últim byte és irrellevant.

Tanmateix, els blocs ja codificats amb codis de Huffman no estan confinats a ocupar un nombre sencer de bytes. Un bloc codificat pot començar i acabar a qualsevol bit dins un byte.

Aquests codis de Huffman compleixen les mateixes característiques establertes per l'algorisme DEFLATE[1]:

1. Tots els codis d'una mateixa longitud tenen valors lexicogràficament consecutius, en el mateix ordre que els valors que representen.

2. Els codis més curts precedeixen lexicogràficament als codis més llargs.

Per a cada bloc es genera un arbre de Huffman independent. El diccionari sobre el qual es crea cada arbre de Huffman és el format pels símbols 0..255, amb el pes de cadascun depenent de la seva freqüència al bloc; i el símbol especial 256, de pes 1, que representa el codi de final de bloc.

Cada bloc es codifica amb un llistat de les longituds dels codis de Huffman de cada símbol, per ordre del símbol, seguit del codi de cada símbol del bloc, seguit del codi per al símbol de fi de bloc 256.

La codificació de les longituds dels codis depèn del valor de la longitud:

- **Longituds 1..31:** Es codifica amb la representació en binari sense signe de la longitud amb 5 bits.

- **Longituds 32..:** Es codifica amb `0b00000` seguit de la representació en binari sense signe de restar 31 a la longitud del codi, amb 8 bits.

Per a codificar la fi del fitxer, un cop codificats tots els blocs, s'afegeixen els bits `0b0000000000000`.

3 Anàlisi

La motivació darrera l'algorisme de compressió base gira entorn poder aprofitar els valors numèrics, i no només els bytes purs en format ASCII.

La decisió de codificar els increments o decrements entre valors consecutius prové de la natura de la font de les dades: En tractar-se de les altures sobre nivell del mar d'un avió en vol, és natural que els canvis entre naturals consecutius siguin petits, i que tendeixin a un creixement o decreixement constant, cosa que dificultaria trobar elements repetits per a algorismes de diccionari.

Com podem observar a la Figura 2, el 99% d'enters consecutius es troben a una distància inferior a 128. Per aquest motiu, escollim codificar els increments

Figure 2: Distribució dels valors absoluts de les diferències entre naturals consecutius.

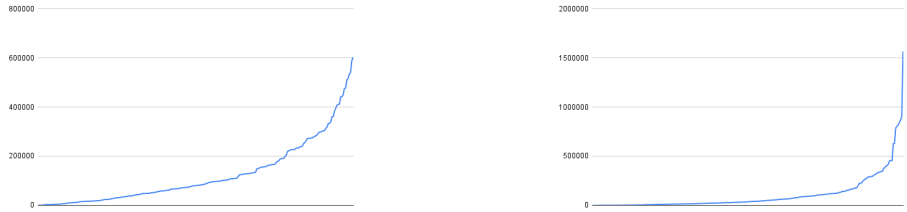

Figure 3: Freqüències absolutes pels bytes 0x00 a 0xFF per les dades codificades només pel primer pas del compressor, ordenades per freqüència.

o decrements petits en la quantitat menor de bits possible, per no malbaratar espai amb *0* redundants.

D'aquesta manera, reduïm la mida de cada nombre natural *(suposant un nombre de 5 dígits seguit d'un espai, 6 bytes = 48 bits)* en...

- 2400% per a naturals iguals *(1˜10% dels casos)*.
- 685% per a diferències menors o iguals a 16 *(55˜85% dels casos)*.
- 400% per a diferències menors o iguals a 272. *(5˜25% dels casos)*.
- 300% per a diferències menors o iguals a 4368. *(<1% dels casos)*.

La compressió de les seqüències de *-1* està destinada a cadenes llargues. Amb només 15 bits, es codifiquen seqüències de fins a 8192 elements. Tot i semblar un nombre excessiu per a seqüències curtes, aquesta codificació aconsegueix reduir el nombre de bits necessaris per a codificar qualsevol nombre de *-1* seguits, fins i tot per a un de sol *(assumint l'espai seguint el nombre com a part de la codificació en text pla del mateix)*. Per a cadenes llargues, com per exemple les que trobem als fitxers de proves, del voltant de 5500 elements seguits, i que signifiquen un 15% del total de nombres enters, aconseguim una reducció del 880000%.

Tal com es pot intuir a l'exemple del punt 2.1.1, o com veiem a la Figura 3, els valors com a bytes de la codificació inicial són molt desproporcionats. Concretament, hi ha una gran abundància dels bytes 0x00, i bytes fets per set

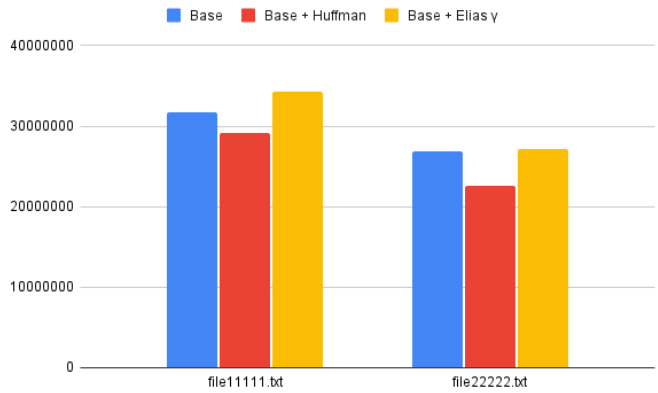

Figure 4: Comparació de les mides de fitxer per a les codificacions base, base + Huffman i base + Elias Gamma.

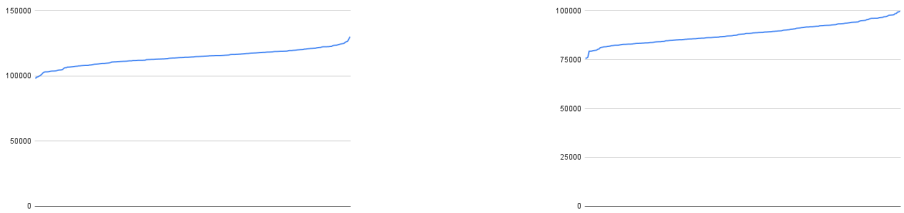

Figure 5: Freqüències absolutes pels bytes 0x00 a 0xFF per les dades codificades completament pel compressor fent servir codis de Huffman, ordenades per freqüència.

bits *0* i un d*1*. Això ens indica que hi ha marge de millora per la ràtio de compressió si tractem de nou aquestes dades.

Després de considerar una codificació amb codis de Huffman o amb *Run Length Encoding* fent servir la codificació Elias Gamma. Després de provar de codificar per segon cop els fitxers de prova, trobem que Huffman aconsegueix una ràtio de compressió al voltant d'1,1 sobre la codificació base, mentre que Elias Gamma empitjora els resultats. Per aquest motiu, decidim que la segona passada es realitzarà amb Huffman.

4 Codi

El codi de la pràctica està escrit en C++. Està dividit en 4 fitxers:

- `main.cpp` - Fitxer principal del codi. Únicament s'encarrega de fer una crida al compressor o descompressor.

- `CompressorPractica.cpp` - Implementa el compressor de fitxers. Consta d'una funció pública, que es crida amb dos paràmetres, strings indicant el camí al fitxer a codificar i el fitxer on desar-lo, que implementa el bucle principal; i una dotzena de funcions privades dedicades a codificar els blocs en codis de Huffman i escriure a disc.
  - `properNumero();` A partir d'un string d'entrada i un iterador, retorna el pròxim nombre enter.
  - `diferenciaAlBloc();` A partir de dos nombres naturals, afegeix al bloc la codificació de la diferència entre els dos.
  - `literalAlBloc();` Afegeix al bloc la codificació d'un nombre natural.
  - `eolAlBloc();` Afegeix al bloc la codificació d'un salt de línia.
  - `negatiusAlBloc();` Afegeix al bloc la codificació d'una seqüència de *-1*.
  - `bitsAlBloc();` Afegeix al bloc una seqüència de bits de la llargària demanada.
  - `bitAlBloc();` Afegeix al bloc un bit.
  - `byteAlBloc();` Funció de suport a l'anterior que afegeix grups de bits al bloc.
  - `endOfFile();` Buida el bloc actual a disc i escriu el símbol de fi de fitxer.
  - `blocAHuffman();` Codifica el bloc actual amb codis de Huffman i els escriu a disc.
  - `longitudArbre();` Donat un arbre binari, genera un vector amb les longituds dels codis de Huffman per a cada símbol.
  - `bitsAlDisc();` Escriu al fitxer de disc una seqüència de bits de la llargària demanada.
  - `bitAlDisc();` Escriu al fitxer de disc un bit.
  - `byteAlDisc();` Escriu al fitxer de disc un byte.
  - `flushDisc();` Escriu a disc la memòria cau que no s'ha escrit encara.
- `DescompressorPractica.cpp` - Implementa el descompressor de fitxers. Consta d'una funció pública, que es crida amb dos paràmetres, strings indicant el camí al fitxer a descodificar i el fitxer on desar-lo, que implementa el bucle principal; i set funcions privades dedicades a descodificar els blocs en codis de Huffman i escriure a disc.
  - `llegeixBitDeDisc();` Retorna el pròxim bit del fitxer de disc.
  - `llegeixBitsDeDisc();` Retorna els pròxims n bits demanats del fitxer de disc.
  - `llegeixByteDeDisc();` Retorna el pròxim byte del fitxer de disc.

- `llegeixBlocDeDisc();` Llegeix i descodifica un bloc codificat en Huffman del fitxer de disc.
- `llegeixBitDelBloc();` Retorna el pròxim bit del bloc.
- `llegeixBitsDelBloc();` Retorna els pròxims n bits demanats del bloc.
- `llegeixByteDelBloc();` Retorna el pròxim byte del bloc.

- `BinTree.hpp` - Classe que implementa els arbres binaris. Aquesta classe ha estat distribuïda pel professorat del Departament de Ciències de la Computació de la UPC[2].

References

[1] L. Peter Deutsch. *DEFLATE Compressed Data Format Specification version 1.3*. URL: `https://datatracker.ietf.org/doc/html/rfc1951`.

[2] J. Petit. *BinTree.hh*. URL: `https://pro2.cs.upc.edu/`.