PROJECTES DE PROGRAMACIÓ

Quadrimestre de tardor, curs 2019/2020

EL COMPRESSOR

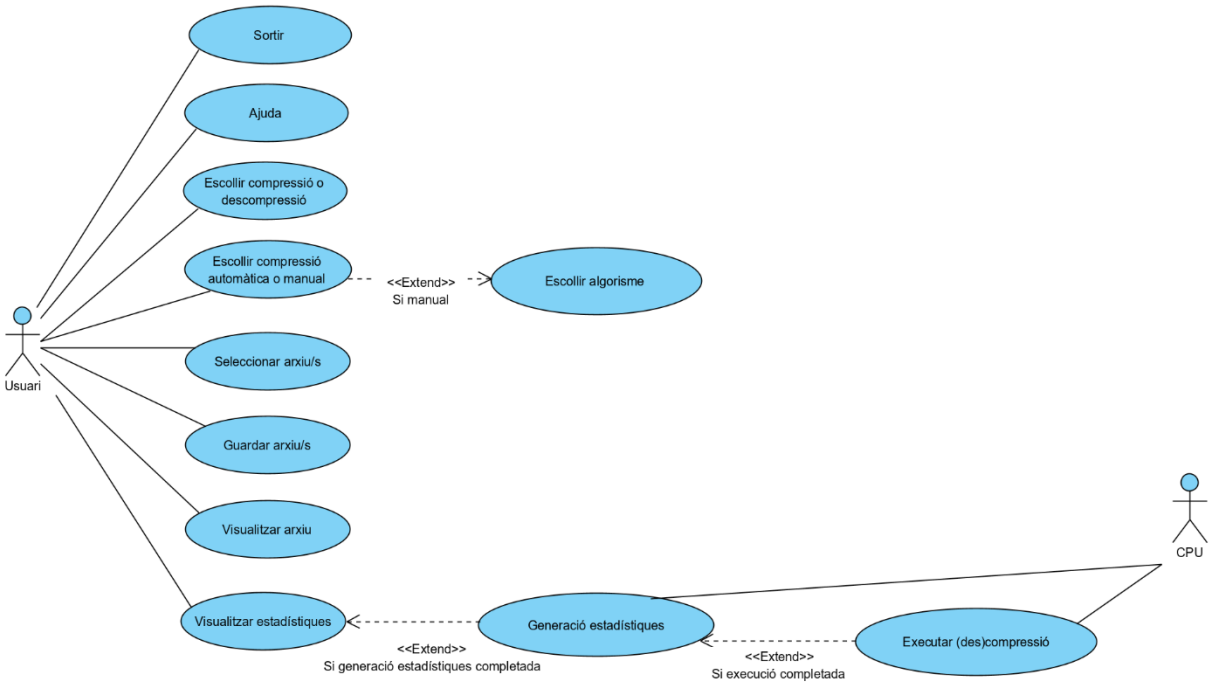
IBARS CUBEL, ALBERT	albert.ibars.cubel@est.fib.upc.edu
MUÑOZ BUSTO, ISAAC	isaac.munoz.busto@est.fib.upc.edu
CLEMENTE MARÍN, DANIEL	daniel.clemente.marin@est.fib.upc.edu
PÉREZ JOSENDE, ALEXANDRE	alexandre.perez.josende@est.fib.upc.edu

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Índex

1. CASOS D'US

1.1. Esquema



1.2. Descripció

Nom cas d'ús:	**Seleccionar arxiu/s**
Descripció:	L'usuari selecciona un o més arxius que vol comprimir o descomprimir
Actors:	Usuari
Precondició:	El programa està obert però no està executant cap compressió ni descompressió
Diàleg típic:	1. L'usuari escull l'opció "Seleccionar arxiu/s" 2. El sistema presenta els directoris de l'usuari 3. L'usuari escull un directori 4. El sistema presenta el contingut del directori seleccionat 5. L'usuari escull un fitxer o una carpeta
Errors o vies alternatives:	Si l'arxiu escollit és d'un format no compatible amb el nostre programa el sistema llença una excepció i torna al punt 4
Postcondició:	L'arxiu escollit està llest per comprimir o descomprimir

Nom cas d'ús:	**Escollir compressió o descompressió**
Descripció:	L'usuari escull quina funció del nostre programa vol fer servir, comprimir o descomprimir
Actors:	Usuari
Precondició:	Haver seleccionat mínim un arxiu
Diàleg típic:	1. L'usuari selecciona el botó "*Comprimir/Descomprimir*" 2. Es desplega un desplegable amb dues opcions: Comprimir i Descomprimir 3. L'usuari selecciona la opció que vol
Errors o vies alternatives:	Si l'arxiu és un arxiu descomprimit i s'escull l'opció descomprimir el sistema llença una excepció i torna al punt 1
Postcondició:	L'arxiu o els arxius seleccionats es preparen per la compressió o la descompressió i apareix un menú amb diverses opcions com la d'escollir si vols compressió manual o automàtica

Nom cas d'ús:	**Escollir compressió automàtica o manual**
Descripció:	L'usuari escull si vol executar la compressió de forma automàtica on el sistema utilitzarà l'algoritme més apropiat pel tipus d'arxiu o si d'altra banda vol escollir ell manualment l'algoritme que s'executarà
Actors:	Usuari
Precondició:	Haver escollit l'opció de "compressió" i l'arxiu o els arxius a comprimir
Diàleg típic:	1. L'usuari selecciona el botó "*Automàtic/Manual*" 2. El sistema li dona a triar les dues opcions: Automàtic o Manual 3. L'usuari selecciona la opció que vol
Postcondició:	El/s fitxer/s estan llestos per comprimir i només fa falta primer el botó "Executar" per iniciar la compressió.

Nom cas d'ús:	**Escollir algoritme**
Descripció:	L'usuari escull l'algoritme que vol fer servir per la compressió
Actors:	Usuari
Precondició:	L'usuari ha escollit l'opció compressió manual
Diàleg típic:	1. L'usuari selecciona el botó "*Escollir algoritme*" 2. El sistema li dona a triar entre els algoritmes implementats 3. L'usuari selecciona l'algoritme que prefereix
Errors o vies alternatives:	Si es un fitxer de text i es selecciona l'algoritme JPEG el sistema llença una excepció i torna al punt 2
Postcondició:	El sistema passa a executar la compressió

Nom cas d'ús:	**Guardar arxiu/s**
Descripció:	L'usuari escull on guarda l'arxiu resultant de la compressió o descompressió
Actors:	Usuari
Precondició:	Haver completat la compressió o la descompressió
Diàleg típic:	1. El sistema presenta els directoris de l'usuari 2. L'usuari escull un directori 3. El sistema guarda el fitxer o la carpeta resultant en aquest directori
Errors o vies alternatives:	Si el directori no és accessible per l'usuari el sistema llença una excepció i torna al punt 1
Postcondició:	El sistema guarda l'arxiu resultant

Nom cas d'ús:	**Visualitzar arxiu**
Descripció:	L'usuari té la possibilitat de visualitzar l'arxiu que vol comprimir o el descomprimit
Actors:	Usuari
Precondició:	Seleccionar un arxiu
Diàleg típic:	1. El sistema presenta una miniatura de l'arxiu seleccionat 2. L'usuari fa click al botó "*Visualització*" 3. El sistema obre l'arxiu per a que l'usuari el pugui visualitzar amb detall
Errors o vies alternatives:	Si l'arxiu no és compatible amb el nostre programa el sistema llença una excepció i torna al punt 1
Postcondició:	Visualització del fitxer per pantalla

Nom cas d'ús:	**Executar (des)compressió**
Descripció:	S'executa la compressió o la descompressió amb l'algoritme adequat
Actors:	CPU
Precondició:	Haver seleccionat un arxiu i si es vol descomprimir o comprimir. En cas de comprimir a ver escollit automàtic o manual i a ver escollit l'algoritme
Diàleg típic:	1. El sistema executa l'algoritme escollit per l'usuari o pel mateix sistema
Postcondició:	Es genera un arxiu resultant

Nom cas d'ús:	**Generació estadístiques**
Descripció:	Es generen les estadístiques de la compressió per a cada arxiu
Actors:	CPU
Precondició:	Haver comprimit un arxiu
Diàleg típic:	1. El sistema genera unes estadístiques i les guarda
Errors o vies alternatives:	Si hi ha algun error en la compressió relacionat amb les estadístiques el sistema llença una excepció i no guarda les estadístiques
Postcondició:	Les estadístiques queden guardades al programa

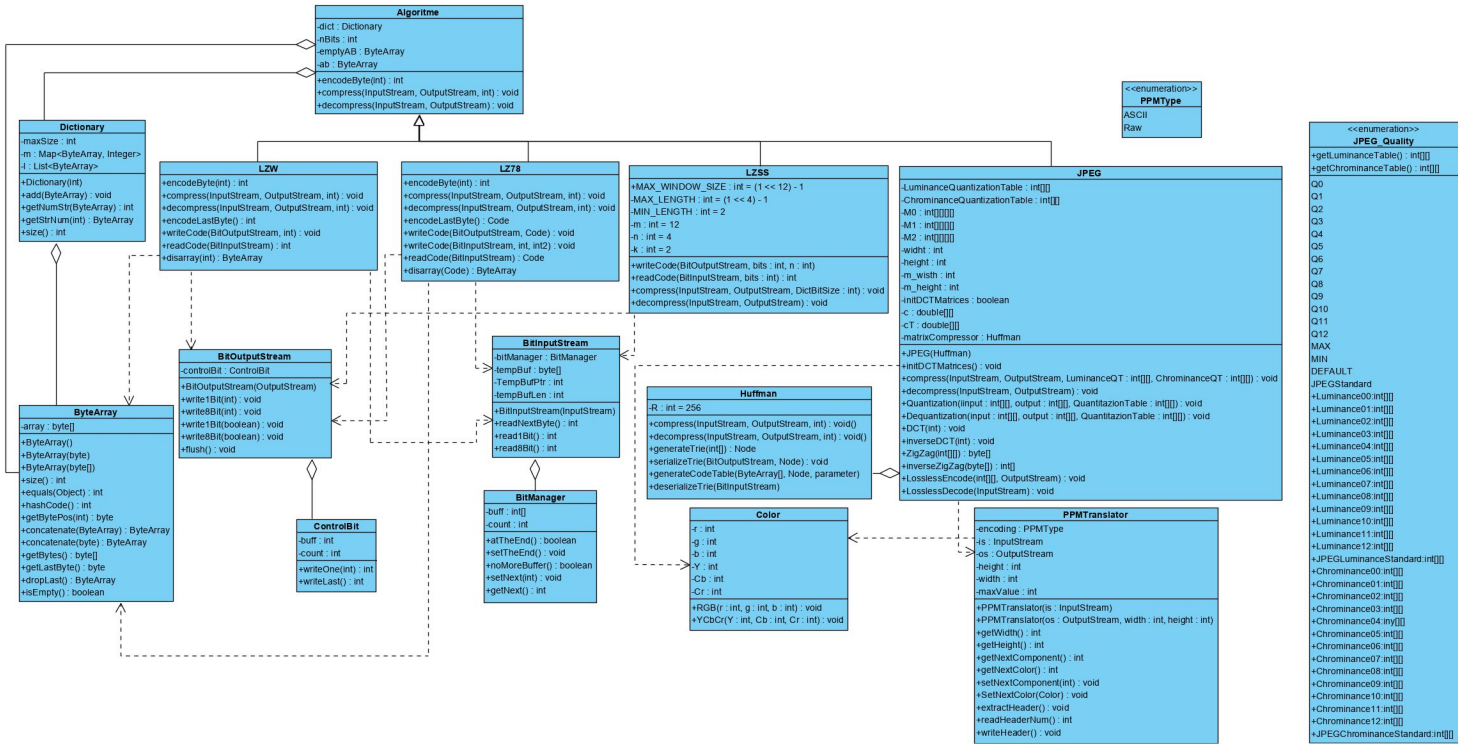
Nom cas d'ús:	**Visualització estadístiques**
Descripció:	L'usuari pot visualitzar les estadístiques resultants d'una compressió
Actors:	Usuari
Precondició:	La CPU ha executat la generació d'estadístiques
Diàleg típic:	1. L'usuari selecciona la opció "*Visualitzar estadístiques*" 2. El sistema mostra per pantalla les estadístiques prèviament guardades
Errors o vies alternatives:	Si no es troben les estadístiques perquè no s'havien guardat el sistema llença una excepció i avisa a l'usuari de que no es disposa d'aquestes estadístiques
Postcondició:	Es mostren per pantalla les estadístiques

Nom cas d'ús:	**Ajuda**
Descripció:	Es disposa d'un document d'ajuda per l'usuari
Actors:	Usuari
Diàleg típic:	1. L'usuari escull l'opció "*Ajuda*" 2. El sistema presenta l'arxiu d'ajuda
Postcondició:	Es visualitza per pantalla l'arxiu d'ajuda

Nom cas d'ús:	**Sortir**
Descripció:	L'usuari vol sortir del programa
Actors:	Usuari
Diàleg típic:	1. L'usuari escull l'opció "*Sortir*" 2. El sistema es tanca
Errors o vies alternatives:	Si el programa es troba executant una compressió o una descompressió pregunta si està segur que vol interrompre-la
Postcondició:	Es tanca el programa

2. DIAGRAMA UML

2.1. Esquema



2.2. Explicació de les classes

Algoritme
Interfície que defineix els atributs i els mètodes que tenen els algoritmes implementats en comú. No l'hem implementat encara perquè hem de polir les coses que tenen en comú. Diccionari i ByteArray són dues classes utilitzades en la majoria dels algoritmes implementats i són atributs d'aquesta classe. Aquesta classe també serà l'encarregada de generar les estadístiques.

LZ78
Implementa l'algoritme LZ78, que consisteix en agafar el fitxer d'entrada i amb un diccionari anar guardant les cadenes de caràcters que es van formant i es van repetint tenint en compte les ja formades mitjançant un diccionari.

LZW
Implementa l'algoritme LZW el qual es basa en l'anterior LZ78, que consisteix en la creació d'un diccionari el qual al principi s'inicialitza amb tots els valors ASCII i es va emplenant de cadenes de caràcters a mesura que es llegeix l'arxiu d'entrada i el qual no s'inclou al fitxer comprimit ja que al descomprimir es pot reconstruir fàcilment.

LZSS
Implementa l'algoritme LZSS que consisteix en anar llegint l'entrada mitjançant una finestra de mida fixa que va buscant coincidències i codifica tuples amb la posició de la coincidència trobada i la mida de la cadena. Es pot codificar com a literal si no troba coincidència ja que ocupa menys com a literal que com a tupla.

JPEG
Implementa l'algoritme JPEG que es basa en la simplificació del color d'uns quants píxels i la posterior compressió amb Huffman del resultat. Al descomprimir l'arxiu es perd qualitat casi imperceptible per l'ull humà.

Huffman
Algoritme cridat per l'algoritme JPEG que s'encarrega de comprimir utilitzant les freqüències de cada símbol mitjançant un arbre, els caràcters més freqüents tenen una codificació més curta.

Dictionary
Diccionari propi de ByteArrays implementat amb un hashmap i una llista.

ByteArray
Estructura formada per un array de bytes.

BitInputStream
Classe encarregada de llegir l'input bit a bit i passar les dades que llegeix a l'algoritme que s'executarà.

BitManager
Classe privada del BitInputStream que controla els fluxos de bits de entrada.

BitOutputStream
Classe encarregada de llegir els resultats dels algoritmes de bit a bit i passar les dades que llegeix a un nou arxiu.

ControlBit
Classe privada del BitOuputStream que controla els fluxos de bits de sortida.

PPMTranslator
Classe que serveix de parser de ppm binaris i ASCII i que escriu en format ppm binari. Permet llegir tan colors com components individuals d'aquests colors i també escriure els colors o les seves components individuals.

Color
Classe que guarda i transforma colors RGB/YCbCr.

JPEG_Quality
Classe encarregada d'enumerar les diferents qualitats disponibles del jpeg i proporcionar els mapejos d'aquestes qualitats a les corresponents matrius de quantificació per la luminància i la crominància.

3. DESCRIPCIÓ DE LES ESTRUCTURES DE DADES

En aquest apartat justificarem l'elecció de les estructures de dades utilitzades per implementar cada algoritme.

Quan esmentem que hem utilitzat un Dictionary per buscar coincidències farem referència a l'estructura formada per un HashMap (que té un ByteArray com a clau i un cert codi com a valor) i una List de ByteArrays (que utilitzarem per buscar una ByteArray associat a un codi determinat).

LZ78

Per a aquest algoritme hem utilitzat les següents estructures de dades:

- **Dictionary**: Hem elegit aquesta estructura per poder buscar coincidències quan comprimim i obtenir el ByteArray codificat quan descomprimim en temps constant O(1). El seu tamany el definirem arbitràriament i influirà en el grau de compressió.
- **ByteArray**: Hem elegit aquesta estructura ja que treballem sobre l'input i l'output a nivell de byte i un vector on cada posició és un byte ens facilita molt la feina quan realitzem operacions (p.e. concatenació o consulta) sobre aquests bytes. El seu tamany serà variable depenent del nombre de bytes que tingui la coincidència.

LZW

Per a aquest algoritme hem utilitzat les mateixes estructures que al LZ78 ´per les mateixes raons ja que aquest és molt similar.

LZSS

Per a aquest algoritme hem utilitzat les següents estructures de dades:

- **ByteArray**: Encara que sigui l'única estructura de dades utilitzada en aquest algoritme farem servir ByteArray per a dues funcionalitats diferents.

  En primer lloc, ens servirà per a representar la **finestra corredissa** on buscarem les coincidències. El fet de implementar-la com a ByteArray ens permetrà fer accessos aleatoris amb cost O(1) i realitzar operacions de inserció i eliminació de conjunts de bytes de manera més òptima. El seu tamany l'assignarem arbitràriament i serà un factor determinant pel que fa al grau de compressió de l'algoritme.

  En segon lloc, ens servirà per a representar el **buffer de coincidències** on anirem guardant la coincidència actual. Obtindrem un gran avantatge en implementar-lo com a ByteArray ja que com treballem sobre l'input i l'output a nivell de byte ens facilitarà molt operacions com la inserció de un nou byte al buffer o com una consulta aleatòria. El seu tamany serà dinàmic ja que anirà variant segons els bytes que formin part de la coincidència actual.

JPEG

Per a aquest algoritme hem utilitzat les següents estructures de dades:

- **Matriu**: Utilitzem matrius per representar la imatge. La imatge la dividirem en submatrius de 8x8. Això comportarà que, quan haguem de seleccionar una posició de la imatge, determinem 4 valors on el dos primers [][] serviran per seleccionar la submatriu 8x8 i els dos últims [][] faran referència a la posició dins la submatriu. Utilitzarem 3

matrius d’aquest tipus (matriu[][][][]) on cadascuna farà referència a una component del color. El fet d’utilitzar matrius farà possible l’accés aleatori a un cost constant. El tamany de les matrius serà estàtic i vindrà directament determinat pel tamany de la imatge.

- **Array de bytes**: Utilitzem arrays de bytes per guardar dades generades per diferents funcionalitats que apareixen durant l’execució de l’algoritme. Hem utilitzat array de bytes en comptes de el tipus ByteArray creat per nosaltres perquè en aquest algorisme no usem cap mètode dels implementats en la classe ByteArray i, per tant, el usar aquesta classe ens generaria dependències innecessàries. D’aquesta manera (en estructurar les dades en un array de bytes) podrem accedir-hi aleatòriament en un cost constant.
- **ByteArrayInputStream**/**ByteArrayOutputStream**: Utilitzem aquesta estructura per traduir el array a stream per poder utilitzar-lo amb les funcions de l’algoritme Huffman (ja que hem de passar un stream per aconseguir que el Huffman sigui agnòstic respecte les dades que li passem).

Huffman

Per a aquest algoritme hem utilitzat les següents estructures de dades:

- **Trie**: Hem utilitzat un trie per estructurar la codificació de les coincidències. El trie estarà format per nodes de tipus Node que contindran informació sobre el caràcter ASCII al que fa referència, la freqüència amb la que apareix aquest codi i quins son els nodes fills. Hem elegit aquesta estructura ja que ens permet realitzar consultes en cost O(m) on m es la longitud del codi. El tamany del trie serà dinàmic i anirà augmentant a mesura que anem tractant la imatge.
- **PriorityQueue**: Hem elegit una PriorityQueue per poder ordenar els arbres segons la freqüència d’aparició en el input. Encara que el cost d’ordenació serà el mateix que si utilitzéssim un vector, hem elegit una PriorityQueue ja que ens sembla la opció més intuïtiva. El seu tamany serà fixe ja que un cop inicialitzada contindrà els nodes arrel de 256 tries (un per cada símbol ASCII).
- **ByteArray/Array de bytes**: Com hem comentat en els apartats anteriors, utilitzarem aquestes dues estructures per emmagatzemar dades de manera seqüencial i de manera que la seva consulta es pugui dur a terme en cost O(1). Utilitzarem ByteArray quan vulguem utilitzar un dels seus mètodes. Altrament, utilitzarem un array de bytes.

4. DESCRIPCIÓ DELS ALGORITMES

El nostre projecta comprimeix i descomprimeix arxius mitjançant diferents algoritmes:

- Algoritme LZ78 *(Lempel-Ziv, 1978)*
- Algoritme LZSS *(Lempel-Ziv-Storer-Szymansk, 1982)*
- Algoritme LZW *(Lempel-Ziv-Welch, 1984)*
- Algoritme JPEG *(Joint Photgraphic Experts Group, 1992)*
- Algoritme Huffman *(David A. Huffman, 1952)*

LZ78

Codi realitzat per **Daniel Clemente.**

L'algorisme LZ78 aconsegueixen compressió substituint les ocurrències repetides de dades per referències a un diccionari que es crea basant en flux de dades d'entrada. Cada entrada del diccionari es de la forma [...] = {index, character}, on index és l'índex d'una entrada de diccionari anterior, i el caràcter s'afegeix a la cadena representada pel diccionari[index]. Per a cada caràcter del flux d'entrada, es busca el diccionari una coincidència: {últim índex coincident, caràcter}.

Si es troba una coincidència, el darrer índex de concordança es defineix en l'índex de l'entrada coincident, i no es produeix res. Si no es troba una coincidència, es crea una nova entrada de diccionari: diccionari [següent índex disponible] = {últim índex de concordança, caràcter} i l'algorisme produeix l'índex de coincidència últim, seguit de caràcter, després es restableix l'últim índex de coincidència = 0 i augmenta el següent índex disponible.

Un cop el diccionari estigui complet, no s'afegeixin més entrades. Quan s'arriba al final del flux d'entrada, l'algoritme produeix l'últim índex coincident.

LZSS

Codi realitzat per **Isaac Muñoz**.

EL LZSS es un algoritme que millora el LZ77 mitjançant un indicador d'un bit per indicar si el següent fragment de dades és un literal o una parella de distància-longitud i l'ús de literals per si una parella de longitud-distància seria més llarga.

On els algorismes LZ77 aconsegueixen compressió substituint les ocurrències repetides de dades per referències a una sola còpia d'aquestes dades existents anteriorment al flux de dades no comprimides.

LZW

Codi realitzat per **Albert Ibars**.

LZW és un algorisme basat en LZ78 que utilitza un diccionari preinicialitzat amb tots els possibles caràcters (símbols) o emulació d'un diccionari preinicialitzat.

La millora principal de LZW és que quan no es troba una coincidència, s'assumeix que el caràcter actual del flux d'entrada és el primer caràcter d'una cadena existent al diccionari (ja que el diccionari s'inicialitza amb tots els caràcters possibles), de manera que només l'última coincidència es produeix un índex (que pot ser l'índex de diccionari preinicialitzat corresponent al caràcter d'entrada anterior (o inicial)).

JPEG

Codi realitzat per **Alexandre Pérez**.

JPEG utilitza una forma de compressió amb pèrdues basada en la transformada de cosinus discreta (DCT). Aquesta operació matemàtica converteix cada fotograma / camp de la font de vídeo del domini espacial (2D) en el domini de freqüència (per exemple un domini de transformació). Un model perceptiu basat en el sistema psicovisual humà descarta informació d'alta freqüència, és a dir, transicions intenses en intensitat i color. Al domini de transformació, el procés de reducció de la informació s'anomena quantització.

El mètode de compressió sol perdre's, és a dir, que es perd alguna informació original de la imatge i no es pot restaurar, afectant possiblement la qualitat de la imatge.

Huffman

Codi realitzat per **Alexandre Pérez**.

La tècnica funciona creant un arbre binari de nodes. Un node pot ser un node de fulla o un node intern. Inicialment, tots els nodes són nodes de fulla, que contenen el símbol en si, el pes (freqüència d'aparició) del símbol i, opcionalment, un enllaç a un node parent que facilita la lectura del codi (a la inversa) a partir d'un node de fulla.

El procés s'inicia amb els nodes de full que contenen les probabilitats del símbol que representen. Aleshores, el procés agafa els dos nodes amb menor probabilitat i crea un nou node intern que té aquests dos nodes com a fills. El pes del nou node s'estableix en la suma del pes dels fills. A continuació, apliquem el procés de nou, al nou node intern i als nodes restants (és a dir, excloem els dos nodes de full), repetim aquest procés fins que només queda un node, que és l'arrel de l'arbre de Huffman.