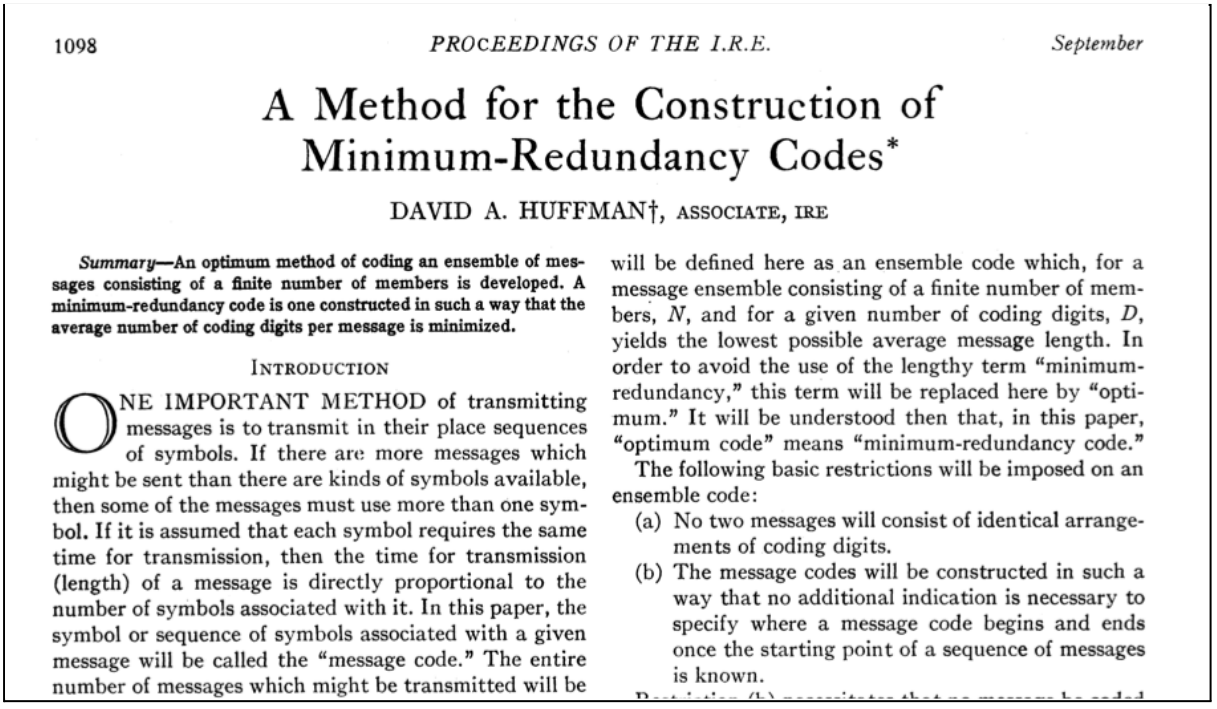
GRETHEN Clémentine
MARTI Victor
N7-1SSN
Groupe MN06

CODAGE DE HUFFMAN
PROGRAMMATION IMPÉRATIVE



1098 PROCEEDINGS OF THE I.R.E. September

A Method for the Construction of Minimum-Redundancy Codes*

DAVID A. HUFFMAN†, ASSOCIATE, IRE

Summary—An optimum method of coding an ensemble of messages consisting of a finite number of members is developed. A minimum-redundancy code is one constructed in such a way that the average number of coding digits per message is minimized.

INTRODUCTION

ONE IMPORTANT METHOD of transmitting messages is to transmit in their place sequences of symbols. If there are more messages which might be sent than there are kinds of symbols available, then some of the messages must use more than one symbol. If it is assumed that each symbol requires the same time for transmission, then the time for transmission (length) of a message is directly proportional to the number of symbols associated with it. In this paper, the symbol or sequence of symbols associated with a given message will be called the "message code." The entire number of messages which might be transmitted will be will be defined here as an ensemble code which, for a message ensemble consisting of a finite number of members, N, and for a given number of coding digits, D, yields the lowest possible average message length. In order to avoid the use of the lengthy term "minimum-redundancy," this term will be replaced here by "optimum." It will be understood then that, in this paper, "optimum code" means "minimum-redundancy code."

The following basic restrictions will be imposed on an ensemble code:

(a) No two messages will consist of identical arrangements of coding digits.
(b) The message codes will be constructed in such a way that no additional indication is necessary to specify where a message code begins and ends once the starting point of a sequence of messages is known.

Source : Huffman (1952), Proceedings of the I.R.E. (Extrait)

15 janvier 2022

0. Résumé

Mots clés : compression - codage de Huffman - octet - fichiers -

Key words : compression - Huffman coding - octet - files

Ce projet a pour objectif d'écrire deux programmes utilisant le codage de Huffman, l'un pour compresser des fichiers, l'autre pour les décompresser.

Dans ce rapport, nous présenterons les difficultés que nous avons rencontrées pour réaliser ce projet, l'organisation et la structure de notre code, la manière dont nous nous sommes répartis le travail ainsi que la démarche pour tester notre code.

Sommaire

I. Introduction

L'optimisation de la transmission de messages volumineux - transmettre facilement, rapidement, sans perte d'informations - reste un challenge.
La compression est une technique qui permet de réduire la taille des messages sans perte d'informations. Ses principaux avantages sont de réduire le volume de stockage des données, le temps de transmission des données, l'utilisation de la bande passante de transmission. Elle permet donc de générer des économies.
En 1952, Huffman, alors étudiant au MIT, propose une méthode statistique originale, à partir de l'idée de recourir à un nouveau codage des caractères.
Le codage de Huffman (1952) est un codage statistique utilisé pour la compression sans perte de données telles que les textes, les images (fichiers JPEG) ou les sons (fichiers MP3). Dans le cas de textes, son principe est de définir un nouveau codage des caractères, codage à taille variable qui tient compte de la fréquence (le nombre d'occurrences) des caractères dans le texte : les caractères dont la fréquence est élevée seront codés sur moins de bits et ceux dont la fréquence est faible sur plus de bits.

Dans ce projet, il s'agissait d'écrire deux programmes, le premier qui compresse des fichiers en utilisant le codage de Huffman et le second qui les décompresse. Nous l'avons réalisé à deux, sur une période de temps allant du 18 novembre au 15 janvier.

Ce rapport rend compte des principales étapes de ce projet. Le plan suivi est le suivant. Nous présentons tout d'abord l'architecture de l'application en modules (2.) en justifiant les principaux choix réalisés (3.), puis nous exposerons les principaux algorithmes et types de données (4.) ainsi que la démarche adoptée pour tester le programme (5.) et les difficultés rencontrées et les solutions retenues (6.). Nous terminerons en présentant notre rétroplanning (7.), le bilan technique de notre projet (8.) et le bilan personnel (9.) avant de conclure (10.).

2. Architecture de l'application en module

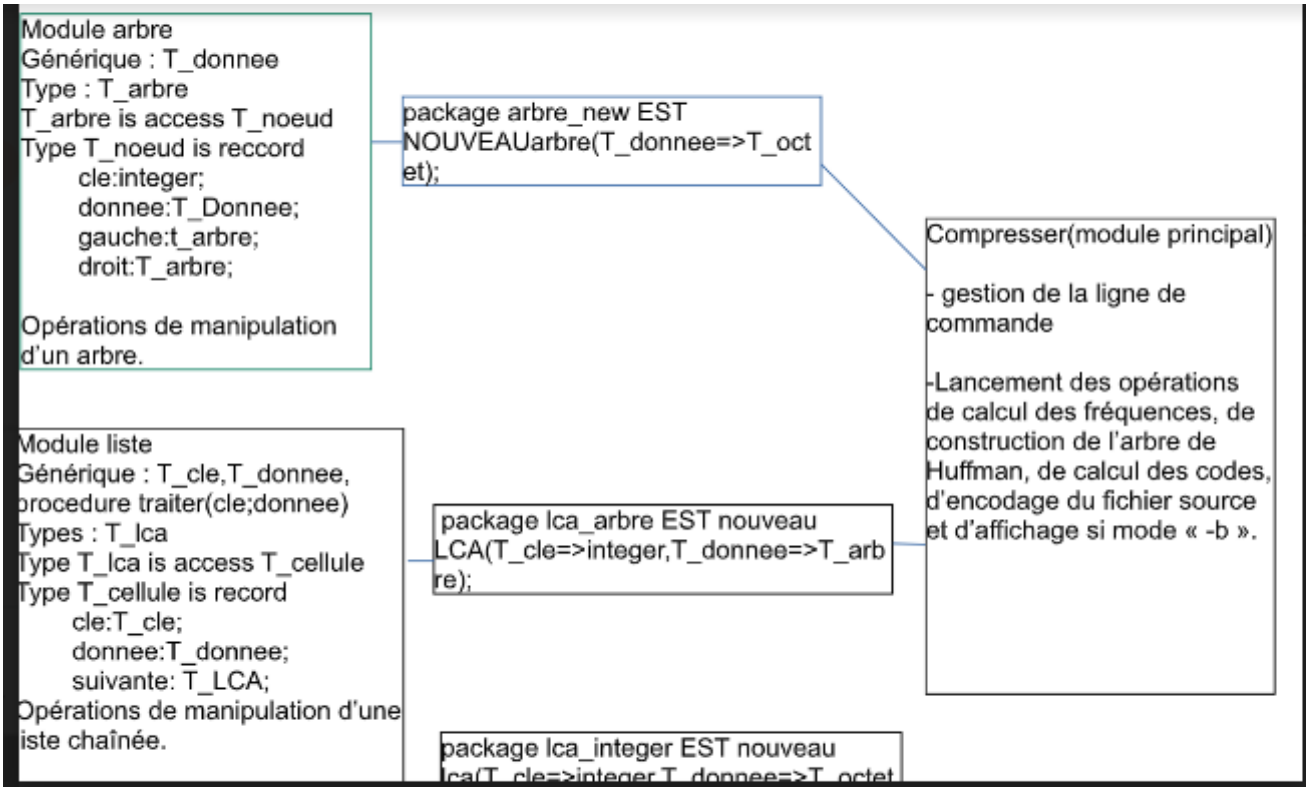


Schéma : organisation des modules pour la compression

Pour la décompression nous n'avons pas utilisé les modules LCA et ARBRE. Mais uniquement des tableaux: le détail se trouve aux pages 7 et 8.

3. Présentation des principaux choix réalisés:

Dans ce projet, nous avions des consignes précises, mais aussi la possibilité de faire des choix.
Nous avons donc adopté les modalités suivantes.

Nous avons un premier module : lca pour implanter les opérations nécessaires à la manipulation des listes chaînées (les clés et les données sont génériques). Pour construire une liste contenant les caractères présents dans un texte (à compresser) et leur fréquence associée, nous utilisons une lca où la clé est un entier (la fréquence) et la donnée l'octet du code ASCII d'un caractère. Nous aurions pu stocker les caractères et leur fréquence sous la forme d'une feuille et donc utiliser une lca où la donnée est un arbre, mais vérifier qu'une feuille est déjà présente dans une lca s'est révélé complexe étant donné que l'égalité n'est pas définie pour un arbre (la fonction « enregistre » dans le module lca devenait donc inutilisable pour un arbre).

Pour la construction de l'arbre de Huffman, on utilise une lca (où la clé est un entier et la donnée un arbre) qui initialement contient des feuilles qui contiennent chacune un caractère du texte et sa fréquence. Cette liste de base sera donc construite à partir de la liste des fréquences. La construction de l'arbre se fait par conséquent au sein de cette liste en fusionnant les arbres de fréquences minimales.

Pour stocker les codes de Huffman des symboles, nous utilisons un tableau de taille 257. Les codes sont des unbounded_string. La dernière case contient le code d'Huffman du caractère de fin : '⁄$'. La case d'indice i contient le code d'Huffman associé au caractère de code ASCII i (s'il est présent dans le texte, sinon elle ne contient rien). La taille (257) permet de stocker le code de '/$' sans « prendre la place » d'un autre caractère. L'utilisation d'un tableau permet de pouvoir chercher le code courant sans avoir à parcourir le tableau depuis le début.

Le parcours_infixe de l'arbre est sous la forme d'un unbounded_string : il suffit de lui concaténer '1' ou '0' selon l'avancée du parcours. Les caractères présents dans l'ordre infixe de l'arbre sont stockés dans un tableau de taille 257 (cela permet de prendre en compte un nombre maximal de caractères ainsi que la présence du symbole de fin) et sont en écriture binaire (type unbounded_string).

Pour écrire l'encodage du texte dans le fichier .hff, nous avons en amont stocké dans une unbounded_string, la concaténation du code binaire des caractères en parcours infixe, le

parcours infixe de l'arbre et le texte encodé grâce à la table de Huffman Le choix de concaténer l'ensemble du texte à écrire dans le fichier avant de l'écrire dans le fichier a permis d'avoir un code plus clair et plus compréhensible.

La procédure affichage est incluse dans le programme principal compresser.adb et a été faite de manière récursive.

Pour la décompression nous avons stocké le parcours infixe ainsi que le texte en code de huffman dans une unbounded_string. Nous avons également écrit en binaire les caractères présents en parcours infixe et nous les avons concaténé pour former un unbounded_string. Nous trouvions que c'était la façon la plus simple de les manipuler.

Nous avons fait le choix de ne pas reconstruire l'arbre de Huffman mais uniquement la table de huffman. En effet, c'est possible de décoder le texte sans l'arbre en comparant puisque le parcours infixe et la suite des caractère permet d'avoir directement les codes de huffman, ils suffit de lire le texte codé et de comparer à la table pour savoir l'ordre des caractères du texte à décompresser.

La table de Huffman est un tableau de taille 257 ou la 257ième case contient le code de /$ et la case i contient le code associé au caractère de code ASCII i.

4. Principaux algorithmes et types de données

Pour ce sujet, nous avons créé plusieurs types, ils ont tous été réalisés pour correspondre à notre manière de coder.

-- Type T_Arbre et ses sous types--

```
type T_noeud;

type T_arbre is access T_noeud;

type T_noeud is record
        cle:integer;
        donnee:T_Donnee;
        gauche:t_arbre;
```

```
        droit:T_arbre;
end record;

-- Type octet --

type T_octet is mod 2**8;

-- Type T_LCA  -liste pour la construction de l'arbre --

type T_cellule;
type T_LCA is access T_cellule;
type T_cellule is record
        cle:T_cle;
        donnee:T_donnee;
        suivante: T_LCA;
end record;
```

Pour compresser nous avons:

```
        package arbre_new is new arbre(T_donnee=>T_octet);

        package lca_arbre is new lca('T_cle=>integer,T_donnee=>T_arbre);

        package lca_integer is new lca(t_cle=>integer,T_donnee=>T_octet);

        Type T_tab2 is array (1..257) of unbounded_string;
```

Pour décompresser nous avons:

```
        type T_tab is array (1..257) of T_octet;

        Type T_tab2 is array (1..257) of unbounded_string;
```

Principaux algorithmes

Voici les principaux algorithmes que nous avons réalisés pour atteindre nos objectifs.

POUR COMPRESSER

- Construction de la liste des fréquence : lca_integer
- conversion en une liste d'arbre
- Construction de l'arbre
- Affichage de l'arbre
- Construction de la signature de l'arbre
- Construction et stockage des codes des symboles
- Encodage du fichier compresser

POUR DECOMPRESSER:

- Décomposer l'entête du texte encodé
- création de la table de Huffman
- affichage des codes de Huffman
- Décoder le texte
- Ecrire le texte correspondant dans un fichier .hff

5. Démarche adoptée pour tester le programme

Pour tester notre programme, nous avons utilisé un exemple connu (voir ci-dessous) et nous l'avons comparé avec les résultats du fichier 'compression.adb'.

Pour commencer, nous avons créé un fichier texte :

--fichiertest.txt--
"exemple de texte :"
"exempte tempete lexeme"

Nous avons affiché l'arbre et la table de Huffman.
Après une étude théorique sur cet exemple (détaillée dans la presentation.pdf) nous avons conclu que l'arbre était correct, ainsi que la table de Huffman. Nous avons également testé la robustesse de notre programme en essayant avec un fichier inexistant et sans fichier. Même si l'arbre est correct, le fichier .hff ne pourrait pas l'être, nous avons alors lancé le programme de décompression et demandé d'afficher ce qu'il y avait en entrée. Nous retrouvions alors effectivement le parcours infixe, les caractères en ordre infixe(avec un doublon) et le texte en code de Huffman. Cette étape de test est détaillée dans la présentation.pdf.

Pour tester la décompression, nous avons testé avec le même fichier test, le fichier retourné (.txt) contenait bien: "exemple de texte :" .
"exempte tempete lexeme"

De plus, la table de huffman reconstruite est la même que celle affichée par compression.
La robustesse de décompression a été testé de la même manière que celle de compression.

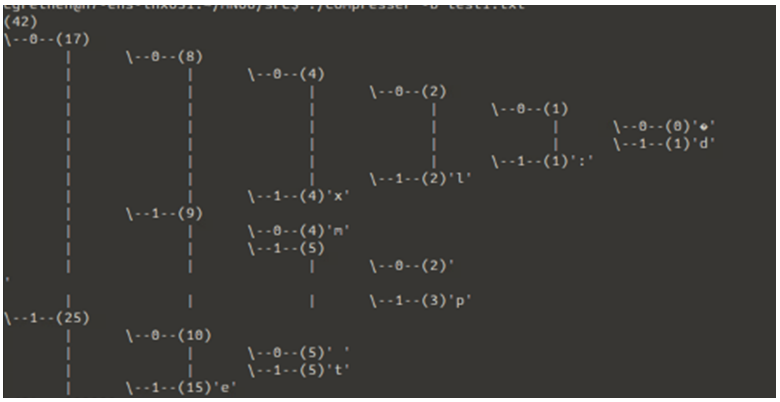


Figure représentant l'arbre de Huffman obtenu et la table de Huffman avec le fichier test.

6.Difficultés rencontrées et solutions adoptées

Durant ce projet, nous avons rencontré plusieurs difficultés. Parfois nous avons dû faire marche arrière, d'autres fois nous avons dû persévérer.

-- Compresser

Le principal problème que nous avons rencontré en début de projet a été la lecture du texte.txt et le stockage des octets. Nous avons levé ce blocage lorsqu'un complément du projet, qui présentait un exemple de lecture de texte, a été envoyé.

Pour la création de l'arbre d'Huffman, nous avons commencé à partir d'une liste non triée (lca). De plus, dans le type de l'arbre, nous avions rajouté un pointeur vers la racine de l'arbre pour pouvoir le parcourir dans les deux sens. Cette manière de faire, bien que réalisable, est difficile à mettre en place et n'apporte que peu de valeur ajoutée.

A la place, nous avons décidé de faire simplement l'arbre à partir d'une liste triée et sans le pointeur vers la racine, ce qui a grandement facilité la réalisation du module arbre.

--Afficher

Pour afficher, nous avons commencé à coder la procédure en itératif, en nous aidant de la fonction 'Taille' du module arbre. Le problème de cette méthode est qu'elle met beaucoup de temps à se mettre en place pour être opérationnelle et nous n'avons pas réussi à obtenir un résultat satisfaisant. Nous avons donc décidé de coder en récursif la fonction afficher, ce qui a permis un code plus léger et efficace.

--Décompresser

Pour décompresser, la difficulté majeure a été la reconstruction de l'arbre de Huffman. Finalement, nous avons décidé de ne pas le reconstruire, ce qui a marché. La limite de cette solution est le manque de l'affichage de l'arbre de Huffman.

7. Rétroplanning et organisation de l'équipe

Notre équipe projet est constituée de deux étudiants, Clémentine et Victor.

Le tableau ci-dessous présente les différentes tâches à réaliser pour mener à bien notre projet, et les dates auxquelles ces tâches ont été effectuées.

Nous avons adopté un mode de coordination des tâches "par ajustement mutuel", c'est-à-dire en échangeant régulièrement de façon informelle sur le projet, mais aussi en dégageant des plages de temps pour avancer en commun sur chaque tâche. Afin d'optimiser le temps passé néanmoins, nous avons aussi opéré une division des tâches et nous nous les sommes réparties.

(C) : tâches réalisées par Clémentine

(V) : tâches réalisées par Victor

(CV) : tâches réalisées conjointement par Clémentine et Victor

Tableau 1. Rétroplanning

Dates/ Tâches	18 nov-fin nov.	début déc-mi-déc.	Mi-déc-fin déc.	vacances	fin déc-15 janv.
étude de la littérature	V - C	V - C			
Raffinage	C	C	C	C	C
Module (arbre-lca)		V - C	C	C	C
test des modules				C	C
Compression		C	C	C	C
Décompression			V	C	C
Affichage					V
présentation					C
manuel					C

démonstration					C
Rédaction du rapport					V-C

Organisation de l'équipe

Réflexion commune sur l'ensemble du sujet.

Clémentine : Rédaction des raffinages, fichier compresser fonctionnel, rédaction des ads et adb de arbre et lca, fichiers test de arbre et de LCA ,décompression fonctionnelle, réalisation des test de compression et décompression, rédaction de la démonstration, du manuel et de la présentation. Rédaction d'une partie du rapport.

Victor : Structure et code non fonctionnel du fichier décompresser, fonction afficher et rédaction du rapport.

8. Bilan technique

Dans l'ensemble, nous sommes satisfaits de ce projet, l'étape la plus importante à nos yeux a été réussie, le programme compresser fonctionne. De plus, nous avons une fonction affiche qui n'est pas parfaitement correcte mais qui est efficace. Et finalement, nous avons un programme décompresser qui a la bonne structure et qui fonctionne sur les exemples.
Du temps supplémentaire nous aurait permis d'améliorer ces points et notamment la robustesse des deux fichiers.

9. Bilan personnel

Nous arrivons au terme de ce rapport, et c'est l'occasion pour nous de communiquer ce que nous retirons de ce travail au niveau individuel, à la fois sur le plan personnel et académique.

Clémentine:

Le projet de Huffman représente le premier travail important et concret de programmation. j'ai pu découvrir en profondeur le langage ADA: notamment la syntaxe utile pour manipuler les fichiers. J'ai également mis en pratique les enseignements dispensés au premier semestre en programmation impérative, je me sens donc plus à l'aise sur la notion de module, d'exception et de généricité.
Grâce à ce projet j'ai également appris à rechercher de manière efficace les ressources nécessaires pour avancer, mais aussi la patience face aux erreurs: j'ai fait l'erreur de coder la compression en une fois (sans étape), j'ai donc perdu beaucoup de temps puisque les erreurs de logique et de syntaxe fut nombreuses. J'ai donc amélioré ma technique sur la décompression.
J'ai passé environ 10 heures sur le raffinage, 18 heures pour la compression, 15 pour la décompression, 3 heures pour la rédaction des supports, 5 heures pour la rédaction des modules et les fichiers de test associés. Le temps passé est la cause d'une mauvaise organisation et des erreurs de code.

Victor :

Le projet "Compression et décompression de données par codage de Huffman" représente le premier travail que je réalise en équipe sur de la programmation, il a donc une importance particulière pour moi. Il s'agissait aussi de mettre en pratique les enseignements dont nous avons bénéficié au premier semestre.
Les enseignements que je retire de ce projet sont nombreux. Sur le plan personnel, j'ai appris la patience (coder est une opération répétitive et fastidieuse), la rigueur (une seule petite erreur génère beaucoup de temps perdu en recherche), l'organisation (partage et coordination des tâches). J'ai appris que les solutions prometteuses ne fonctionnent pas toujours, qu'il faut parfois revenir en arrière.
Au niveau de l'apprentissage académique, ce projet m'a aussi fait progresser. En effet, l'apprentissage réalisé en faisant par soi-même est complémentaire à l'apprentissage

académique. C'est la meilleure manière de s'approprier un sujet et de prendre conscience des difficultés.
Ces enseignements me serviront dans mes projets futurs car je les aborderai différemment, à la lumière de ce que j'ai retenu de ce premier projet.

10. Conclusion

L'objectif de ce projet était d'écrire deux programmes utilisant le codage de Huffman, l'un pour compresser des fichiers, l'autre pour les décompresser. Le code de Huffman reste en effet une référence dans ce domaine. En effet, il est établi selon la probabilité d'apparition des caractères, et attribue un code court aux caractères les plus fréquents, ce qui lui confère son optimalité.
Les résultats que nous avons obtenus sont les suivants : nous avons mis au point un programme opérationnel pour compresser des fichiers textes, et afficher l'arbre de huffman, nous l'avons testé à plusieurs reprises et de plusieurs manières, et nous avons élaboré le programme de décompression, il est construit, mais il reste à finaliser pour qu'ils fonctionne en toutes circonstances.
Le codage de Huffman reste toujours d'actualité 70 ans après sa formalisation, et ce travail nous a permis de prendre conscience de son apport essentiel à la transmission et au stockage de données.

11. Références

AIT-AMEUR Y, CREGUT X, JAFFRES-RUNSER K., (2020-2021), Programmation Impérative, Cours de N7.

HUFFMAN D.A. (1952), A method for the construction of minimum-redundancy codes, *Proceedings of the I.R.E.*, voL.40, 9, pp.1098-1102.