Projet codage et compression

Formation Informatique et Sciences du Numérique

Denis Bouhineau, Vincent Danjean, Guillaume Huard, Cyril Labbé, Anne Rasse, Jean-Marc Vincent, Benjamin Wack

UFR IM^{2}AG
email : Prénom.Nom@univ-grenoble-alpes.fr

UFR IM^{2}AG
UNIVERSITÉ Grenoble Alpes

Formation ISN de professeurs Niveau I

Plan

UNIVERSITÉ Grenoble Alpes

Vue d'ensemble du projet

Objectifs :

- **Analyse et conception d'algorithmes**
  → arbres, files à priorité, itérations, récursion
- **Écriture de programmes** (en langage java)
  → renforcer la pratique, en vrai grandeur, co-développement
- **Validation**
  → tests de correction, analyse de complexité, évaluation de performances

Structure de mini-projet :

- travail en binôme
- temps plein (sur une semaine)
- autonomie

Évaluation : soutenance/démonstration du logiciel

- présentation : $\sim$ 20 mn
- questions du jury : $\sim$ 10 mn

UNIVERSITÉ Grenoble Alpes

Organisation

Planning : TD de 9h à 10h

lundi	**conseils pour les soutenances** [A. Rasse]
mardi	**tests de correction** (unitaires, intégration, non-régression) [V. Danjean]
mercredi	**lectures / écritures bit à bit** [D. Bouhineau]
jeudi	**performance** : taux de compression, temps de compression/décompression [C. Labbé]
vendredi	**ouverture** : lempel-ziv, compression avec perte, autre... [B. Wack]

... puis passage en salle machines.

Encadrement : passage d'un enseignant en salle de TP dans la journée

UNIVERSITÉ Grenoble Alpes

Thème du projet : compression sans pertes

Au menu :

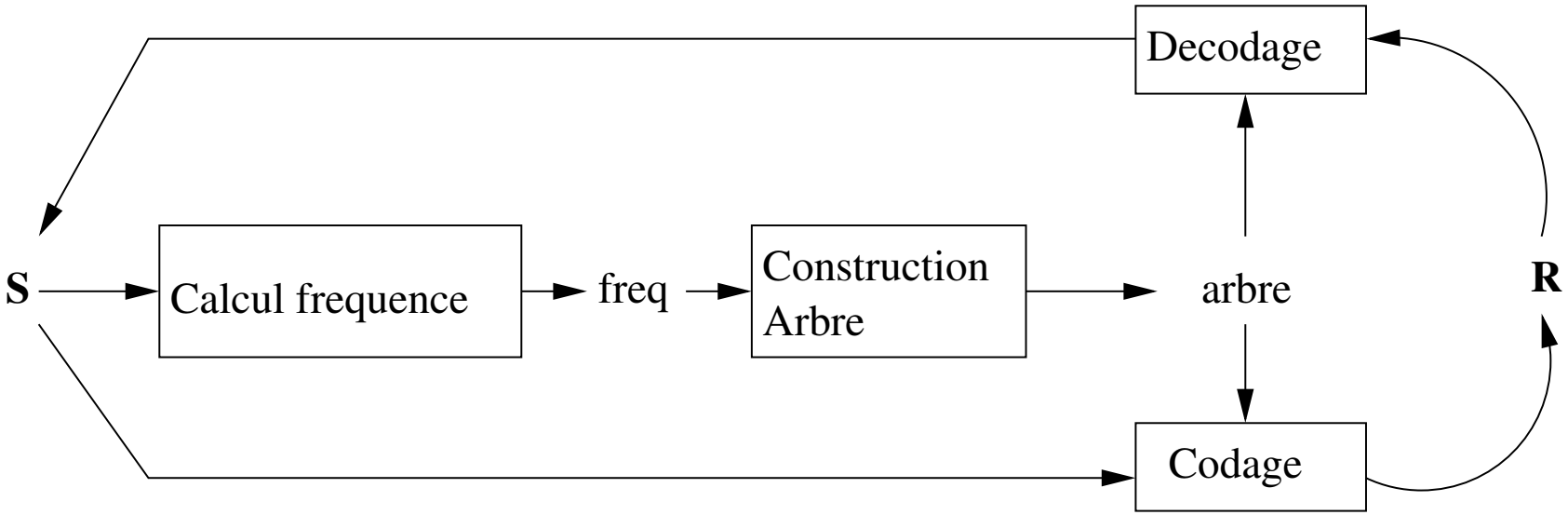
- **Algorithme de Huffman**
  - codage
  - décodage
- **Expérimentation** sur divers formats de données : textes, programmes (source et binaire), images, etc.
- Extensions
  - lecture/écriture de fichiers binaires
  - compression/décompression "à la volée"

UNIVERSITÉ Grenoble Alpes

Plan

Architecture globale

Un ensemble de composants interconnectés :



Communications entre composants :

- échange de structures de données
- fichiers intermédiaires

Calcul de fréquences

Fréquence associée à chaque **symbole** d'entrée possible :

- en prenant 1 octet pour symbole d'entrée : 256 symboles possibles
- table indicée par les valeurs de symbole (ici 0 à 255)

Construction de la **table** :

- parcours préalable de l'ensemble de la source (statique)
- évolution au cours de la compression (dynamique)

Représentation :

- contient indifféremment des fréquences ou un nombre d'occurences
- plusieurs représentations possibles :
  - fréquence associée à chacun des symboles d'entrée
  - séquence de couples (symbole, fréquence)

UNIVERSITÉ Grenoble Alpes

Construction de l'arbre

Arbre binaire :

- **feuilles** étiquetées par les **symboles d'entrée**
- étiquette des nœuds internes indifférente

FAP de couples **(nœud, priorité)** :

- construction des feuilles à partir de la table de fréquences
- ordre des **priorités inverse** de celui des **fréquences**

Représentation :

- nœuds créés dynamiquement qui se référencent les uns les autres
- structure connue, par ex. pour des symboles sur 1 octet :
  - 256 feuilles, 255 nœuds internes
  - tables de 511 nœuds
  - indice $x \in \{0, \ldots, 255\} \iff$ nœud d'étiquette x
  - référence $\iff$ indice dans la table

UNIVERSITÉ Grenoble Alpes

Codage

Table de codes :

- construite à partir de l'arbre (parcours)
- chaque **chemin vers une feuille** $\iff$ code d'un symbole
- possibilité de surévaluer le stockage (256*256*1 octet = 64 Ko)
- attention longueur du code différente pour chaque symbole

Codage :

- parcours de la source, code de chaque symbole écrit dans le résultat
- écriture de caractères '0' et '1' (résultat 8 fois trop gros)

Stockage pour le décodage :

- arbre nécessaire pour le décodage et source non disponible
- stockage de la table de fréquence ou de l'arbre au début du fichier résultat

UNIVERSITÉ Grenoble Alpes

Décodage

Récupération de l'arbre :

- première partie du fichier résultat
- lu tel quel ou reconstruit à partir de la table de fréquences

Décodage :

- parcours de l'arbre au fil de la lecture du fichier codé
- démarrage à la racine
- chaque '0' ou '1' lu est une prise de branche (gauche ou droite)
- à l'arrivée sur une **feuille** :
  - **symbole** de la feuille à écrire en sortie
  - retour à la racine de l'arbre

Plan

Lecture/écriture binaire

Données compressées en binaire (valeurs 0 et 1 codées sur un seul bit)

Problème :

l'élément minimal géré par les entrées/sorties standard est l'octet

Solution : mise en tampon

- lire/écrire les bit en les regroupant dans un tampon (1 octet ou plus)
- n'effectuer les entrées/sorties que sur des tampons complets :
  - en lecture : lire l'équivalent du tampon lorsque celui-ci est vide
  - en écriture : écrire le tampon lorsqu'il est plein
- manipuler individuellement les bits du tampon, ex. :
  - mise à 1 du bit x : `tampon = tampon | (1<<x)`
  - lecture du bit x : `bit = (tampon & (1<<x))>>x`
- attention à bien relire dans l'ordre d'écriture
- attention aux séquences de bits non multiples de la taille du tampon :
  - stocker la longueur de la séquence au début de celle-ci
  - encoder dans la séquence de sous séquences spéciales (échappement)

UNIVERSITÉ Grenoble Alpes

Compression dynamique

Inconvénient de la solution statique :

- deux parcours de la source
- inapplicable sur un flux d'entrée (ex. données provenant du réseau)

Codage dynamique : **table de fréquences construite au fil du codage**

- initialiser `freq` avec une valeur identique pour chaque symbole d'entrée
- pour chaque nouveau symbole lu :
  - coder le symbole à l'aide de la table courante
  - mettre à jour `freq`
  - construire un **nouvel** arbre, puis une nouvelle table de codage

Décodage : même principe (on remplace entrée et lu par sortie et décodé)

Optimisations :

- mise à jour de l'arbre (Faller 73, Gallager 78, Knuth 85, Vitter 87)
- mise à jour de la table de codage en même temps

UNIVERSITÉ Grenoble Alpes

Plan

Attendus minimaux

Version de base du codage de Huffman statique :

- données codées sous forme de caractères '0' et '1'
- arbre ou table de fréquences stocké dans le fichier résultat

Tests de correction :

- un codage suivi d'un décodage donne comme résultat le fichier source
- valable sur diverses sources de données (texte, image, exécutable, . . .)
- tests automatisés

Évaluation de performances :

- taux de compression, avec ou sans arbre/table stocké
- temps d'exécution
- en fonction de la taille/nature du fichier d'entrée
- en fonction de la machine exécutant le programme, de sa charge, . . .
- . . . courbes

UNIVERSITÉ Grenoble Alpes

Quelques conseils . . .

Spécifiez avant de coder

- découpage en classes, **interfaces**
- algorithmes et **structures de données** choisis avant de coder
- **commentaires** dans le code

Testez avant d'intégrer

- **tests unitaires** des différentes fonctions
- traces, outil de mise au point (debugger)

Intégrez régulièrement tout ce qui peut l'être (1 fois/jour ou plus)

Choix algorithmiques **simples** dans un premier temps

UNIVERSITÉ Grenoble Alpes

Plan

À faire aujourd'hui

Appropriation du sujet

- Faire tourner à la main sur de petits exemples
- Préparer des exemples significatifs en prévision des tests

Organisation

- Constituer les binômes
- Planning prévisionnel des tâches à accomplir

À réfléchir avant de commencer : architecture du projet

- Organisation en classes et interfaces
- Choix des structures de données
- Programmes exécutables
- Entrées/sorties de ces exécutables
- Formats des fichiers

À faire aujourd'hui

Appropriation du sujet

- Faire tourner à la main sur de petits exemples
- Préparer des exemples significatifs en prévision des tests

Organisation

- Constituer les binômes
- Planning prévisionnel des tâches à accomplir

À réfléchir avant de commencer : architecture du projet

- Organisation en classes et interfaces
- Choix des structures de données
- Programmes exécutables
- Entrées/sorties de ces exécutables
- Formats des fichiers

Rappels : codage de Huffman

- alphabet d'entrée $\mathcal{A}$ (caractères, pixels, etc.)
  alphabet de sortie $\{0,1\}$ (bit)

- un code de Huffman :
  associe à chaque élément de $\mathcal{A}$ une séquence sur $\{0,1\}$:

$$\mathcal{A} \overset{codage}{\longrightarrow} \{0,1\}^* \qquad \{0,1\}^* \overset{decodage}{\longrightarrow} \mathcal{A}$$

- caractéristiques :
  - code de longueur variable (= code Morse ; $\neq$ code Ascii)
  - propriété de préfixe :
    codage(a_1)=w_1, codage(a_2)=w_2
    $\Rightarrow$ w_1 et w_2 ne sont pas préfixes l'un de l'autre
    ($\rightarrow$ algorithme de décodage efficace)

Application du codage de Huffman

Coder une séquence d'éléments sur $\mathcal{A}$ dans un fichier binaire :

- associer un code "court" aux élements les "plus fréquents"
- $\rightarrow$ **compression** de la séquence initiale

Remarques :

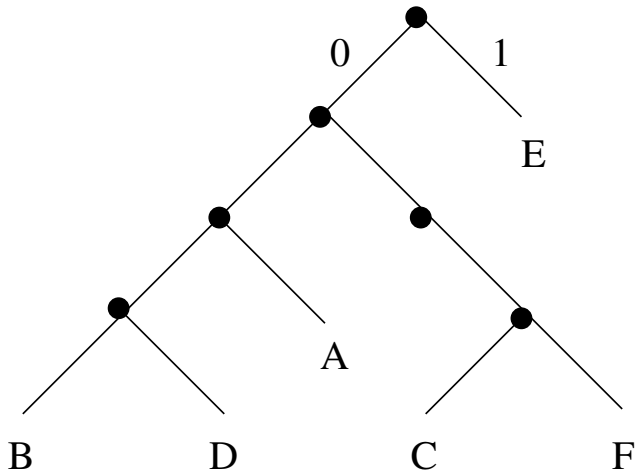
- le code doit être construit **en fonction** du texte initial ...
  - **avant** le codage : Huffman "statique"
  - **pendant** le codage : Huffman "dynamique"
- il doit être connu lors du décodage ...
  (transmis ou recalculé à l'identique)

Arbre de Huffman

Une représentation d'un code de Huffman = arbre de Huffman

- ensemble des feuilles = éléments de $\mathcal{A}$
- accès au fils gauche = "0" ; accès au fils droit = "1"

code(a) = séquence σ des étiquettes du chemin : racine $\xrightarrow{\sigma} a$

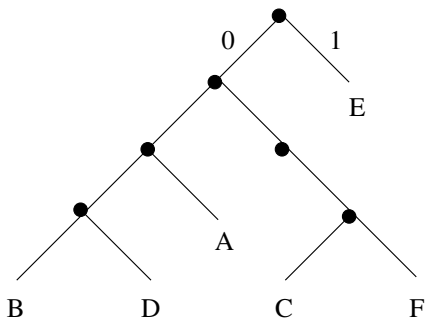


code(E)=1 ; code(A)=001 ; code(C)=0110 ; etc.
$\rightarrow$ la propriété de préfixe est respectée !

Algorithme de Codage

$$S = a_1 \dots a_n \rightarrow R \in \{0,1\}^*$$

Etant donné un arbre de Huffman :



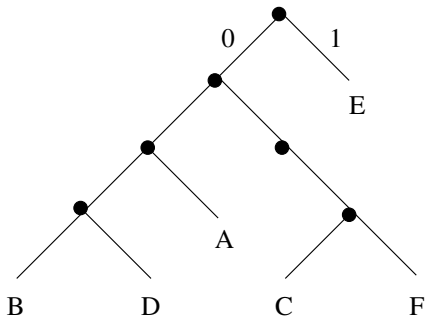
1. construction d'une table de codage `C`
   - parcours en "profondeur d'abord" de l'arbre de Huffman en mémorisant la séquence σ "racine $\xrightarrow{\sigma}$ noeud courant"
   - pour chaque feuille a : `C[a]` $= \sigma$

   Rq : algo récursif ou itératif (avec pile explicite)
2. parcours de S et production de R à l'aide de `C`

Algorithme de Décodage

$$R \in \{0,1\}^* \rightarrow S = a_1 \dots a_n$$

Etant donné un arbre de Huffman :



1. parcours de l'arbre de Huffman selon R
   ("0" : fils gauche ; "1" : fils droit)
2. à chaque feuille a atteinte :
   - écriture de a dans S
   - retour à la racine de l'arbre de Huffman ...

Construction d'un arbre de Huffman ?

- Dépend de la fréquence des éléments de $\mathcal{A}$ dans S :
  fréquence élevée $\Rightarrow$ éléments proches de la racine

- Obtention d'une table de fréquences : `Freq`$[a_i] = f_i$
  (f_i = fréquence de a_i dans S, un réel entre 0 et 1)

- Approche "statique" :
  1. construction de `Freq` par un 1er parcours de S
     (f_i = nombre d'occurences de a_i / $\mid S \mid$)
  2. construction de l'arbre de Huffman (à l'aide de `Freq`)

UNIVERSITÉ Grenoble Alpes

Algorithme informel de construction de l'arbre

On utilise une **file à priorités** (Fap) `F` dont :

- les éléments sont des noeuds de l'arbre
- les priorités sont des réels (fréquence faible $\Rightarrow$ priorité forte)

Algo :

1. insérer les éléments $(a_i, \texttt{Freq}[a_i])$ dans la fap `F`
2. tantque `F` contient au moins 2 éléments
   - extraire deux éléments (n_1, f_1) et (n_2, f_2) de poids minimal de `F`
   - insérer $(n, f_1 + f_2)$ dans `F`, où n est un nouveau noeud de fils gauche n_1 et de fils droit n_2
3. l'élément restant dans `F` est la racine de l'arbre de Huffman