Polytech Grenoble - RICM3

Projet en langage C

Du 28 mai au 4 juin 2013

Objectifs du projet

- Ecriture de code en langage C
  $\rightarrow$ renforcer la pratique de ce langage

- Structure de mini-projet :
  - travail en équipe : quadrinômes
  - temps plein (sur une semaine)
  - autonomie

- Evaluation : soutenance/démonstration du logiciel

Organisation

Planning :

- Présentation : mardi 28/05 matin : F018
- Compléments sur le C (listes) : mercredi matin 8h30 F018
- Présentation gestion de version (git/svn) : mercredi 9h45 F018
- Salles 28/05 - 31/051 : 104+201+202 (tp) et 116+117+118 (td)
- Salles lundi 3/06 : 214+216+204 (tp) et 116+117+118(td)
- Soutenances mardi 4 juin après-midi : 241 (tp en 211+213)

Encadrement :

P. Waille (+ aide L. Mounier)

page Web (placard électronique) :

`http://www-ufrima.imag.fr/INTRANET/placard/INFO/ricm3/PROJET_C`

Soutenances/Démonstration

- Déroulement :
  - démonstration : $\sim$ 15 mn
  - questions : $\sim$ 10 mn

- Objectifs : montrer ce qui fonctionne
  - préparer des jeux d'essai
  - prévoir le déroulement de la présentation
  - travailler le discours

Thème du projet : compression sans pertes

→ Au menu :

- Algorithme LZW (Lampel-Ziv-Welch)
  - programes de codage et décodage
  - réalisation d'un dictionnaire
  - lecture/écriture de fichiers binaires
  - compression/décompression "à la volée"

- Expérimentation sur divers formats de données :
  textes, programmes (source et binaire), images, etc.

- Attention : incompatibilité entre fichiers non textuels et utilisation de 0 marque de fin de chaîne de caractères (0 : valeur d'octet légale).

Notion de compression sans perte

- Une méthode de représentation de données dans un code C1.
- Une méthode de représentation des mêmes données dans un autre code C2.
- La transformation C1 vers C2 est intégralement réversible : le passage de C1 à C2 ne perd aucune information sur les données.
- Le codage en C2 occupe moins de mémoire que le codage en C1.

Dictionnaire

- Les données à représenter sont décomposées en une séquence d'éléments d'un dictionnaire.
- A chaque élément du dictionnaire est associé un code.
- La séquence des éléments est remplacée par la séquence des codes des éléments.
- Technique 1 (**Huffman**) : choisir la longueur des codes des éléments du dictionnaire en fonction de leur fréquence d'apparition dans le fichier à compresser.
- Technique 2 (**Lampel Ziv**) : construire dynamiquement un dictionnaire adapté au contenu du fichier à compresser.

Approche Huffman Statique

- Les données sont transcodées octet par octet.
- Le code associé à chaque valeur possible d'octet est choisi après analyse préalable du contenu du fichier à compresser.
- Le codage repose sur un arbre binaire pondéré par le nombre d'occurences de chaque valeur possible d'octet dans le fichier.
- L'arbre, nécessaire pour le décodage en réception, est inclus dans le fichier compressé.
- Variante adaptative : construction dynamique incrémentale de l'arbre (arbre non transmis mais suboptimal).

Approche Lampel Ziv Welch

Principe : encoder des séquences de un ou plusieurs octets.

- Le **dictionnaire** est **construit dynamiquement** de la même manière par l'émetteur et par le récepteur : il n'est **pas transmis** dans le fichier compressé.
- Le dictionnaire initial contient **toutes** les séquences **mono-octet**.
- Les **séquences** multi-octets dont le **préfixe** appartient déjà au dictionnaire y sont **ajoutées dynamiquement**.
- Les éléments du dictionnaire sont numérotés et tous les **numéros** codés sur le **même nombre de bits**.
- Le nombre de bits est incrémenté dynamiquement quand la taille du dictionnaire atteint une nouvelle puissance de 2.

Lampel Ziv Welch (1984)

Welch améliore les algorithmes de Lampel et Ziv : LZ77 et LZ78.
LZW : breveté, brevet tombé depuis dans le domaine public (2004).

Décomposition de toute séquence en seq_préfixe + seq_mono.
Exemple : "lion" = "lio" (préfixe) + "n" (mono)

Compression : chaque séquence est codée par un doublet (p,f).
p : numéro de préfixe (mono- ou multi- octet)
f : numéro de suffixe (mono- octet)

Recherche et codage de la plus grande séquence dont le préfixe p soit déjà présent dans le dictionnaire, puis ajout de la séquence pf dans le dictionnaire.

Technique d'apprentissage : efficace sur répétition de longues chaînes.

Exemple : chaînes débutant par " l"

258	32+'l'	Le_lion et_le rat
259	258+'e'	
260	259+' '	Il faut, autant qu'on peut, obliger tout_le_monde :
		On a souvent besoin d'un plus petit que soi.
		De cette vérité deux fables feront foi,
261	258+'a'	Tant_la chose en preuves abonde.
262	259+'s'	Entre_les pattes d'un_lion
263	258+'i'	
264	258+"'"	Un rat sortit de terre assez à_l'étourdie.
		Le Roi des animaux, en cette occasion,

258	259	260	261	262	263	264
_l	_le	_le	_la	_les	_li	_l'

265	258+'u'	Montra ce qu'il était, et lui donna la vie.
		Ce bienfait ne fut pas perdu.
		Quelqu'un aurait-il jamais cru
266	263+'o'	Qu'un lion d'un rat eût affaire ?
		Cependant il advint qu'au sortir des forêts
267	266+'n'	Ce lion fut pris dans des rets,
268	260+'p'	Dont ses rugissements ne le purent défaire.
		Sire rat accourut, et fit tant par ses dents
		Qu'une maille rongée emporta tout l'ouvrage.
269	264+'o'	Patience et longueur de temps ...

258	259	260	261	262	263
l	le	le	la	les	li

264	265	266	267	268	269
l'	lu	lio	lion	le p	l'o

Exemple : chaînes débutant par "on"

Le lion et le rat

Il faut, autant qu'on peut, obliger tout le monde :
On a souvent besoin d'un plus petit que soi.
De cette vérité deux Fables feront foi,
Tant la chose en preuves abonde.
Entre les pattes d'un lion
Un rat sortit de terre assez à l'étourdie.
Le roi des animaux, en cette occasion,

Montra ce qu'il était, et lui donna la vie.
Ce bienfait ne fut pas perdu.
Quelqu'un aurait-il jamais cru
Qu'un lion d'un rat eût affaire ?
Cependant il advint qu'au sortir des forêts
Ce lion fut pris dans des rets,
Dont ses rugissements ne le purent défaire.
Sire rat accourut, et fit tant par ses dents
Qu'une maille rongée emporta tout l'ouvrage.
Patience et longueur de temps
Font plus que force ni que rage.

Gains et pertes du codage LZW

- LZW utilise un dictionnaire de $2^{(8+p)}$ éléments.
- Une séquence normale de L octets occupe 8L bits.
- Une séquence lzw de $L > 1$ octets **absente** du dictionnaire ($\simeq$ L séquences mono) occupe L fois (8+p) bits : **expansion**.
- Une séquence lzw de L octets **trouvée** dans le dictionnaire occupe (8+p) bits : **compression**.

Long	Dic	$\Delta \rightarrow$ %	p=1	p=2	p=3	p=4
$L > 1$	$\notin$	$\frac{p+8}{8}$	+12,5	+25	+37,5	+50
L=1						
L=2		$\frac{p+8}{8L}$	-43,7	-37,5	-31,2	-25
L=3	$\in$		-62,5	-58,3	-54,1	-50
L=4			-71,8	-68,7	-54,1	-66,6

Algorithmes et Structures de Données

Algorithme de codage

lexique :

E : fichier d'entrée ; S : fichier de sortie

D : un dictionnaire (ens. de chaînes de caractères identifiées par un index)

w : chaîne de caractère ; a : un caractère

algo :

$D \leftarrow$ ens. de toutes les chaînes de longueur 1 ; $w \leftarrow$[1er caractère de E]

tantque la fin de E n'est pas atteinte

$a \leftarrow$ caractère suivant de E

si $w.a$ est dans D **alors**

$w \leftarrow w.a$ { recherche d'une plus longue sous-chaîne }

sinon

{w est la plus longue sous-chaîne présente dans D}

écrire sur S l'index associé à w dans D

$D \leftarrow D \cup \{w.a\}$

$w \leftarrow [a]$

écrire sur S l'index associé à w

Algorithme de décodage

lexique :

S : fichier d'entrée ; E : fichier de sortie

D : un dictionnaire (ens. de chaînes de caractères identifiées par un index)

i, i' : index dans D

w, w' : chaînes de caractère ; a : un caractère

algo :

$D \leftarrow$ ens. de toutes les chaînes de longueur 1

$i \leftarrow$ 1er code de S ; $a \leftarrow$ chaîne d'index i dans D ; $w \leftarrow [a]$

écrire w sur E

tantque la fin de S n'est pas atteinte

$i' \leftarrow$ code suivant de S ; $w' \leftarrow$ chaîne d'index i' dans D

écrire w' sur E

$a \leftarrow$ 1er caractère de w'

$D \leftarrow D \cup \{w.a\}$

$i \leftarrow i'$

Un problème potentiel (1)

Séquence d'entrée : `xxxza ...`

Codage :

lecture de `x` ; lecture de `x` ; chaîne courante = `x`

$\rightarrow$ écriture du code de `x` (120) ; ajout de `xx` (259) dans D

lecture de `x` ; lecture de `z` ; chaîne courante = `xx`

$\rightarrow$ écriture du code de `xx` (259) ; ajout de `xxz` (260) dans D

lecture de `a` ; chaîne courante = `z`

$\rightarrow$ écriture du code de `z` (122) ; ajout de `za` (261) dans D

Contenu du fichier codé : 120.259.122 ... (x.xx.z. ...)

Un problème potentiel (2)

Fichier codé : `120.259.122 ...`

Décodage :

lecture de 120 ; caractère courant = `x`

→ écriture de `x` (120)

lecture de 259

→ **code non présent dans D !**

⇒ il s'agit nécess. du car. courant concaténé au dernier code lu : `xx`

→ écriture de `xx` ; ajout de ajout de `xx` (259) dans D

lecture de 122 ; caractère courant = `z`

→ écriture de `z` ; ajout de `xxz` (260) dans D

Fichier décodé : `xxxz ...`

Algorithme de décodage (version corrigée)

$D \leftarrow$ ens. de toutes les chaînes de longueur 1
$i \leftarrow$ 1er code de S ; $a \leftarrow$ chaîne d'index i dans D ; $w \leftarrow [a]$
écrire w sur E
tantque la fin de S n'est pas atteinte
    $i' \leftarrow$ code suivant de S
    si $i' \notin D$ **alors**
        $w' \leftarrow$ chaîne d'index i dans D
        $w' \leftarrow w'.a$
    sinon
        $w' \leftarrow$ chaîne d'index i' dans D
    écrire w' sur E
    $a \leftarrow$ 1er caractère de w'
    $D \leftarrow D \cup \{w.a\}$
    $i \leftarrow i'$

Le dictionnaire

→ Mémorise des couples (Séquence d'octets, Code)

Exemple d'interface :

- `void Initialiser()`
  initialise un dictionnaire avec toutes les monoséquences
- `void Inserer (Code prefixe, Code mono, Code *code)`
  ajoute la séquence d'octets `prefixe.mono`, affecte son `code`
- `char *CodeVersChaine (Code code, int *longueur)`
  renvoie séquence et affecte la longueur associée à `code`
- `Code SéquenceVersCode (char *séquence, int longueur)`
  renvoie le code associé à `séquence`

Propriétés :

- une séquence n'est insérée qu'après tous ses préfixes
- pas de suppressions ...

Exemples de structures de données possibles

1. tableau de séquences d'octets
   - recherche séquentielle ; insertion en queue
   - peu efficace, mais **facile à programmer !**
   - version triée : recherche dichotomique
2. arbre binaire de recherche (de séquences d'octets)
   - améliore le coût de la recherche
3. table de hachage
   - hachage fermé ou ouvert (listes de collisions)
   - choix d'une "bonne" fonction de hachage ?
4. dictionnaire arborescent
   - partage de nombreux préfixes communs
   - recherche et insertion "efficaces"

Le travail demandé

A faire impérativement !

- version de “base” du codage LZW
  - version “simple” du dictionnaire
  - fichiers sources et résultat dans des fichiers Ascii
- évaluation sur divers formats de données
  (HTML, source de programme, binaire, images, etc.)
- comparaison des **taux de compression** obtenus

Diverses extensions à la carte . . .

- lecture/écriture binaire
- versions plus efficaces du dictionnaire
  (arbres ternaires, table de hachage, etc.)
- adaptation de la taille du code (nbre de bits) en fonction du contenu du dictionnaire
- ré-initialisation du dictionnaire (si plein, si taux de compression stagne, . . .)
- Move-To-Front ?
- évaluation de ces diverses extensions

Conseils et compléments

Les ressources fournies

(dans le placard électronique . . .)

- un binaire qui fonctionne, avec traces d'exécutions
  → pour comprendre l'algorithme !
- une implémentation de dictionnaire arborescent
  → pour démarrer et comme base de comparaison
- des liens vers des pages Web
  → pour des compléments d'explication . . .
- une *forge logicielle*
  → pour la gestion de version (SVN)
- une assistance en salles de TP et/ou par mail
  → pour répondre à toutes vos questions . . .

Organisation du travail (suggestions)

- prendre le temps de **comprendre** le problème, les ressources fournies ...
- Quadrinômes + structure du projet
  $\Rightarrow$ **répartition** du travail (ex : équipes de 2)
- Bien définir les **interfaces** entre les différents modules (en-tête de fonction, formats de fichiers)
- Compiler et **tester** séparément chaque module !
- Intégrez et sauvergarder régulièrement des versions "stables" (mêmes incomplètes)
- Garder du temps pour :
  - faire des mesures expérimentales
  - préparer la soutenance !

Notions complémentaires : fichiers

- Manipulés par leur nom dans l'interpréteur de commandes
- Accès en C via un descripteur : structure contenant les informations sur l'accès en cours
- Utilisation :
  - Ouverture → obtention d'un nouveau descripteur
    ```
    FILE *fopen(char *nom, char *mode);
    ```
  - Accès / Test
    ```
    int fscanf(FILE *f, char *format, ...);
    int fprintf(FILE *f, char *format, ...);
    int feof(FILE *f);
    ```
  - Fermeture → libération des ressources
    ```
    int fclose(FILE *f);
    ```

Exemple : lecture et affichage d'un fichier

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    FILE *fichier; char c;

    if (argc > 1) {
        fichier = fopen(argv[1], "r");
        if (fichier != NULL) {
            fscanf(fichier, "%c", &c);
            while (!feof(fichier)) {
                printf("%c", c);
                fscanf(fichier, "%c", &c);
            }
        }
    }
    return 0;
}
```

Infos utiles sur les fichiers

- Mode : "r" ou "w", mais d'autres existent
- En cas d'erreur d'ouverture, utiliser `perror`

```
void perror(char *message);
```

- Descripteurs ouverts par défaut :
  - *stdin* : entrée standard (clavier)
  - *stdout* : sortie standard (écran)
  - *stderr* : sortie d'erreur standard (écran)

Exemple : ecriture dans un fichier

```
#include <stdio.h>
int main() {
    FILE *fichier; char c;

    fichier = fopen("toto.txt", "w");
    if (fichier != NULL) {
        scanf("%c", &c);
        while (!feof(stdin)) {
            fprintf(fichier, "%c", c);
            scanf("%c", &c);
        }
    } else {
        perror("Erreur à l'ouverture");
    }
    return 0;
}
```

Formats et caractères accentués

Nombreux formats de représentation des caractères spécifiques à chaque langue (accents et cédilles en français). Exemple : codage du caractère ê dans quatre formats répandus :

- iso-latin1 (norme de codage sur 8 bits) : 234.
- utf-8 (norme actuelle de codage sur 1 ou plusieurs octets) : 0xc3 0xaa
- HTML/WEB (commandes sous forme de séquence de caractères ASCII) : "ê"
- LaTeX (traitement de texte scientifique) : "ê"

Quelques commandes sont utiles pour adapter le format des documents à compresser ou observer le résultat de la compression.

Commandes od, recode, djpeg

od -t x1 -c fichier. Octal dump affiche le contenu n octets par n octets. **-t** x1 = hexa, 1 octet.
-c = 1 octet sous forme de caractère (si affichable).

recode Latin-1..UTF-8 liste_fichiers (latin → utf) Modifie en place les fichiers (pas de fichier résultat).
Exemples : "UTF-8", "Latin-1" ou "ISO_8859-1", "h1" ou "h2" ou "h3" ou "h4" (versions de HTML) , -d "LaTeX".

djpeg -colors 256 -bmp f.jpeg > f.bmp
jpeg (compressé) → bitmap (forme non compressée).
Pour 1 octet/pixel : palette de couleurs sur 8 bits (-colors 256) ou niveaux de gris (-greyscale).