DEVOIR

2

Devoir AP1

13 novembre 2010 - durée 1h00
R. Bourqui, M. Montassier, B. Recur, N. Journet
Le barème est donné à titre indicatif
Internet et documents INTERDITS

2.1 MARCHE A SUIVRE

Ouvrez un terminal et rendez vous dans le répertoire : `/net/Remise/AP1`
Dans ce répertoire a été créé un répertoire portant votre nom. Ouvrez le.
A l'intérieur est présent :

- Une version numérique du devoir au format `pdf`
- le fichier `codeETU.cc` dans lequel vous écrirez l'ensemble des réponses aux questions posées dans ce devoir

Si par mégarde vous effacez l'un des ces deux fichiers lors du devoir, vous pouvez récupérer les fichiers originaux dans le répertoire : `/net/remise/AP1/1_originaux`

2.2 Création d'un algorithme de compression

L'objectif de cet exercice est de mettre en place un code `C++` réalisant une compression sans perte de type RLE [1] sur un message composé uniquement d'entiers (0 ou 1).
Le principe est simple, il faut regrouper les répétitions successives de nombres identiques afin de réduire la taille du message. Par exemple

- le message non compressé 000110000010000 (15 chiffres)
- devient une fois compressé : 3021501140 (10 chiffres)

Le message compressé 3021501140 indique donc que la chaine initiale est composée successivement de trois 0, puis de deux 1 puis de cinq 0 puis de un 1 et enfin de quatre 0.
Ouvrez le fichier `codeETU.cc`. C'est dans ce fichier que vous allez écrire les réponses aux questions qui suivent.

Pour commencer, indiquez vos noms, prénoms, groupe et heure de passage à l'endroit prévu à cet effet.
Tout en haut du fichier `codeETU.cc` remplissez la zone suivante :

```
/*
################################
Nom de l'étudiant :
Prénom de l'étudiant :
groupe :
Heure de passage :
################################
*/
```

Question 1 */ 1 point(s)*
A l'endroit où est indiqué "répondez ici à la question 1" écrivez (sous forme de commentaire évidemment)

- ce que devient le message 010111001000 si on le compresse avec l'algorithme RLE.
- ce que devient le message 513010112030 si on le décompresse avec l'algorithme RLE.

1. Le run-length encoding, appelé en français le codage par plages, est un algorithme de compression de données en informatique. Voir `http://fr.wikipedia.org/wiki/Run-length_encoding` pour plus d'informations

Question 2 */ 1 point(s)*
A l'endroit où est indiqué "répondez ici à la question 2" :
➤ Une constante de type `int` dont le nom est TAILLE et qui vaut 100 a été déclarée
➤ Un tableau dont le nom est `p` et qui est un tableau de TAILLE entiers a été déclaré.
Pour notre exercice, un message correspond donc à un tableau d'entiers composés uniquement de 1 et de 0.
La ligne `int p[taille];` est incorrecte et ne compile pas tel quelle. Corrigez la.
Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.

Question 3 */ 2 point(s)*
A l'endroit où est indiqué "répondez ici à la question 3", écrivez la fonction

```
void afficheMessage(int p[], int size)
```

Cette fonction affiche le message contenu dans la variable `p`. Ce message est de taille `size`.
Ainsi, si `size` vaut 5 et que `p` vaut

1	1	0	0	1			...	

alors il s'affichera à l'écran : $1\ 1\ 0\ 0\ 1$
Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.

Question 4 */ 3 point(s)*
A l'endroit où est indiqué "répondez ici à la question 4", écrivez une fonction

```
int saisirBinaire()
```

qui invite un utilisateur à saisir au clavier soit le chiffre 1 soit le chiffre 0 soit le chiffre -1 et retourne cette valeur.
Si l'utilisateur entre un autre chiffre, l'utilisateur est invité à ressaisir l'un de ces trois chiffres au clavier (et ceci **tant que** la valeur n'est pas correcte).
Cette fonction retourne donc uniquement l'un de ces trois chiffres.
Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.

Question 5 */ 4 point(s)*
A l'endroit où est indiqué "répondez ici à la question 5", écrivez la fonction

```
void initMessageBinaire(int p[], int & size)
```

Cette fonction doit initialiser (avec uniquement des 1 et des 0) un `message` (vide) passé en paramètre (variable `p`).
Ce message est au maximum de dimension `TAILLE`. Si l'utilisateur ne souhaite pas saisir `TAILLE` entiers, il saisit le nombre -1 pour stopper la saisie. Cette fonction initialise la variable `size` au nombre de valeurs correctement saisies par l'utilisateur.
Vous devez bien entendu utiliser la fonction `int saisirBinaire()` pour "forcer" l'utilisateur à ne saisir que des 1, des 0 et -1 s'il souhaite stopper la saisie de nombres.
Si l'utilisateur saisit successivement les chiffres : 1,1,0,34,0,0,1 et -1, le message contenu dans la variable `p` sera `110001` et la variable `size` contiendra la valeur 6.
Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.

Question 6 */ 2 point(s)*
A l'endroit où est indiqué "répondez ici à la question 6", vous est fourni le code de la fonction

```
void compresser(int p[], int size, int pCompresse[],int &sizeCompresse)
```

Cette fonction étudie la valeur de chaque entier composant le message `p` de taille `size` et compresse ce message (selon l'algorithme RLE) et affecte ce message compressé dans la variable `pCompresse`.
A la fin de la fonction, la taille du message compressé est affectée à la variable `sizeCompresse` (passée par référence).
Cette fonction contient une (et une seule) erreur algorithmique qui fait que le résultat fourni n'est pas le bon.

Trouvez l'erreur et corrigez la (pas besoin d'expliquer votre correction).
Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.

Question 7 */ 4 point(s)*
A l'endroit où est indiqué "répondez ici à la question 7", écrivez la fonction

```
void decompresser(int pCompresse[],int sizeCompresse, int pDecompresse[], int
                                    &sizeDecompresse)
```

Cette fonction prend en entrée un message compressé (`pCompresse`) qui est un message compressé de taille `sizeCompresse`.
Cette fonction doit appliquer l'algorithme RLE inverse et décompresser le message dans la variable `pDecompresse`.
A la fin de la fonction, la variable `sizeDecompresse` contiendra la taille du message décompressé.
Si `sizeCompresse` vaut 6 et que le message `pCompresse` vaut :

5	1	1	0	2	1

alors à la fin de la fonction, `sizeDecompresse` vaut 8 et `pDecompresse` vaut :

1	1	1	1	1	0	1	1

Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.

Question 8 */ 2 point(s)*
On calcule un taux de compression de la manière suivante :

$$taux = 100 * \frac{taille\ du\ message\ non\ compresse\ - taille\ du\ message\ compresse}{taille\ du\ message\ non\ compresse}$$

A l'endroit où est indiqué "répondez ici à la question 8", écrivez la fonction

```
float calculCompression(int sizeDecompresse, int sizeCompresse)
```

permettant de retourner le taux de compression de deux tailles de message passées en paramètres (`sizeDecompresse` et `sizeCompresse`).
Si la taille du message non compressé vaut 8 et que la taille de ce message compressé est 5, alors le taux de compression retourné est 37.5
Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.

Question 9 */ 1 point(s)*
A l'endroit où est indiqué "répondez ici à la question 9" expliquez pourquoi, si la chaine vaut $1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0$ et que l'on décide de la compresser selon l'algorithme RLE, le taux de compression vaut alors -100 ?
Vous répondrez bien entendu sous la forme d'un commentaire dans votre code à l'endroit adéquat.

2.3 Codage de césar

Maintenant que nous arrivons à compresser un message, nous allons coder un algorithme permettant de crypter ce message. Pour cela nous allons utiliser une version simplifiée du codage de César.
Le principe est simple. A chaque chiffre du message d'origine est additionné un chiffre d'une clef secrète que seuls l'émetteur et le récepteur connaissent.
Par exemple, si l'on considère la clef de chiffrement et le message non crypté suivants :
Clef de chiffrement :

2	5	3

Message non crypté :

3	1	4	0	2	1	1	0

Alors, le message se crypte de la manière suivante. On place la clef sous le message sur le premier caractère et on additionne les chiffres un à un.

3	1	4	0	2	1	1	0
+	+	+					
2	5	3					
=	=	=					
5	6	7					

On déplace ensuite la clef car tous le chiffres ne sont pas encore cryptés.

3	1	4	0	2	1	1	0
			+	+	+		
			2	5	3		
			=	=	=		
5	6	7	2	7	4		

On déplace ensuite la clef car tous le chiffres ne sont pas encore cryptés.

3	1	4	0	2	1	1	0
						+	+
						2	5
						=	=
5	6	7	2	7	4	3	5

La clef n'est pas déplacée de nouveau car tous les chiffres ont été cryptés.

Question 10 */ **1 point(s)***

A l'endroit où est indiqué "répondez ici à la question 10", écrivez le contenu de la fonction

```
void crypter(int p[], int size, int clef[], int sizeClef)
```

Cette fonction prend en entrée un message `p` de taille `size` et un message `clef` de taille `sizeClef`.
Cette fonction modifie directement le message `p`.
Ainsi, si au départ de cette fonction `p` vaut | 3 | 1 | 4 | 0 | 2 | 1 | 1 | 0 |, avec comme clef | 2 | 5 | 3 | en sortie `p` aura comme valeurs | 5 | 6 | 7 | 2 | 7 | 4 | 3 | 5 |
Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.

Le message crypté se décrypte de la manière suivante. On place la clef sous le message sur le premier caractère et on soustrait les chiffres un à un.

5	6	7	2	7	4	3	5
-	-	-					
2	5	3					
=	=	=					
3	1	4					

On déplace ensuite la clef car tous le chiffres ne sont pas encore décryptés.

5	6	7	2	7	4	3	5
			-	-	-		
			2	5	3		
			=	=	=		
3	1	4	0	2	1		

On déplace ensuite la clef car tous le chiffres ne sont pas encore décryptés.

5	6	7	2	7	4	3	5
						-	-
						2	5
						=	=
3	1	4	0	2	1	1	0

La clef n'est pas déplacée de nouveau car tous les chiffres ont été décryptés.

Question 11 */ 1 point(s)*

A l'endroit où est indiqué "répondez ici à la question 11", écrivez le contenu de la fonction

```
void decrypter(int p[], int size, int clef[], int sizeClef)
```

Cette fonction prend en entrée un message `p` (crypté) de taille `size` et un message `clef` de taille `sizeClef`.
Cette fonction modifie directement le message `p`.
Ainsi, si au départ de cette fonction `p` vaut | 5 | 6 | 7 | 2 | 7 | 4 | 3 | 5 |, avec comme clef | 2 | 5 | 3 | en sortie `p` aura comme valeurs | 3 | 1 | 4 | 0 | 2 | 1 | 1 | 0 |
Vous pouvez tester l'exactitude de votre réponse en décommentant les lignes prévues à cet effet dans le `main`.