Le développement d'Internet entraîne d'importants transferts de données. Il est donc nécessaire de développer des méthodes de compression de données performantes et fluides sur un flux de données. On s'intéresse ici à un procédé de compression sans perte, nécessaire pour retrouver l'exact original. Ce projet vise à répondre à cette demande en mettant en œuvre l'algorithme de Mike Burrow et David Wheeler.

Présentation du projet

Le projet se sépare en deux parties : la compression et la décompression, exactes inverses l'une de l'autre.

COMPRESSION	DECOMPRESSION
Tri selon l'ordre des contextes	Décompression
Conversion	Inversion de la permutation
Permutation	Inversion de la Conversion (Reconstitution)
Compression	Réciproque du tri

Du point de vue de l'utilisateur, deux programmes sont disponibles, l'un pour la compression, l'autre pour la décompression. Les deux peuvent être exécutés sur une ligne de commande comme indiqué ci-dessous

	Classe	Arguments	Résultat	Exemple
Compression	Cmp	- Taille des paquets - Nom du Fichier à compresser	Fichier avec l'extension .cmp (ex : fichier.doc.cmp)	java Cmp 5000 fichier.doc
Décompression	Dcmp	- Nom du Fichier à décompresser sans l'extension .cmp	Fichier avec l'extension .rec (ex : fichier.doc.rec)	java Dcmp fichier.doc

N.B. – Il existe aussi la possibilité d'exécuter « pas à pas » le projet en l'appliquant à un fichier nommé exemple dans le répertoire courant.

Il suffit d'enchaîner les instructions suivantes :

- java TriPrefixe
- java Conversion
- java Permutation
- java Compression
- java Decompression
- java InversionPermutation
- java Reconstitution
- java Inversion

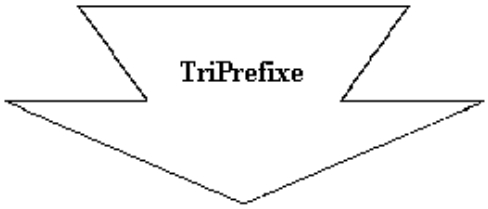
Structures des données

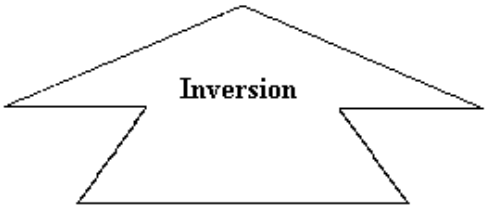
Toutes les entrées sont des flux d'octets entrants (« InputStream ») et toutes les sorties sont des flux d'octets sortant « OutputStream ».

On utilise des « pipes » pour pouvoir enchaîner les différentes étapes du tri, c'est-à-dire relier la sortie d'un sous-programme à l'entrée du sous-programme suivant.

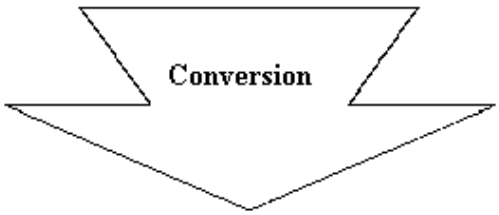
Les tableaux qui suivant mettent en évidence la structure du flux d'octets à chaque étape. En descendant, on part du texte original pour aller au texte compressé, c'est la chaîne de compression. En remontant, on a le contraire, c'est la chaîne de décompression.

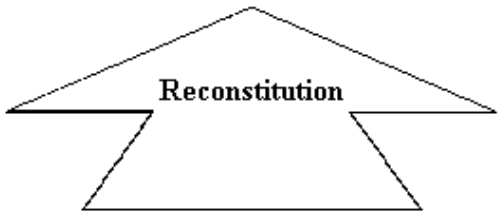
	octets	données
	Tous les octets	Texte original
	Indicateur de fin de flux	-1



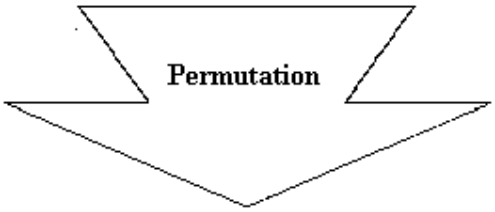


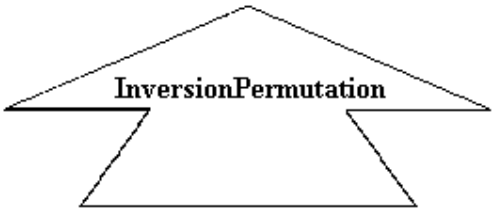
	octets	données	
N fois	1	Taille n de l'alphabet	
	2 … n+1	Alphabet	
	n+2 … n+4	Octet de poids fort … Octet de poids faible	Taille du texte
	n+5 … n+7	Octet de poids fort … Octet de poids faible	Place du dernier caractère après le tri selon les contextes.
	…	Texte trié	
	Indicateur de fin de flux	-1	



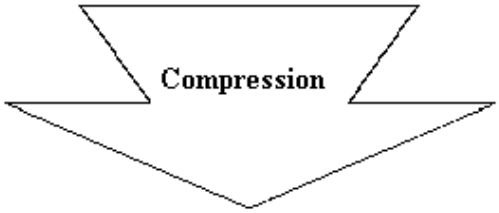


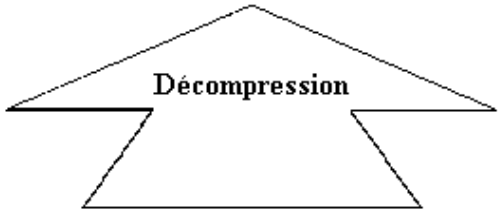
	octets	données	
N fois	1	Taille n de l'alphabet	
	2 … n+1	Alphabet	
	n+2 … n+4	Octet de poids fort … Octet de poids faible	Taille du texte
	n+5 … n+7	Octet de poids fort … Octet de poids faible	Place du dernier caractère après le tri selon les contextes.
	…	Texte converti en entiers	
	Dernier octet	-1	





	octets	Données	
N fois	1	Taille n de l'alphabet	
	2 … n+1	Alphabet	
	n+2 … n+5	Octet de poids fort … Octet de poids faible	Taille du texte non compressé
	n+5 … n+7	Octet de poids fort … Octet de poids faible	Place du dernier caractère après le tri selon les contextes.
	n+8 … 2n+7	Classement du caractère « 0 » … Classement du charactère « n-1 »	
	…	Texte	
	Dernier octet	-1	





	octets	données	
N fois	1	Taille de l'alphabet	
	2 … n+1	Alphabet	
	n+2 … n+5	Octet de poids fort … Octet de poids faible	Taille du texte non compressé
	n+5 … n+7	Octet de poids fort … Octet de poids faible	Place du dernier caractère après le tri selon les contextes.
	n+8 … 2n+7	Classement du caractère « 0 » … Classement du charactère « n-1 »	
		Texte compressé	
	Dernier octet	-1	

Algorithmes

On passe en revue dans cette partie toute les classes et méthodes du programme pour préciser la manière dont elles s'organisent.

Tri (Utilisée uniquement dans la classe TriPrefixe)

Pour réaliser le tri selon les contextes en « nlog(n) », on utilise un arbre qui permet d'écrire la permutation sigma de l'énoncé.

Sur chaque nœud de l'arbre, on met la liste des indices à trier et le début et la fin de la places qu'ils occuperont dans la permutation. Le rang indique le nombre de caractères identiques dans les contextes précédents. Lorsque cette liste est un singleton, la place est bien définie et l'objet est placé. Sinon, on utilise le premier indice de la liste comme pivot, et on sépare la liste des indices en trois listes(qui forment les nœuds des trois arbres fils) selon la méthode de Bentley et Sedgewick.

TriPrefixe

Arguments nécessaire à la création d'une instance de cette classe :

- Flux d'octets entrant (InputStream)
- Flux d'octets sortant (OutputStream)
- Taille des paquets (int)

TrierPrefixe (méthode principale) : Réalise la première opération de la compression

Argument	Néant
Instructions	- Construction de tableaux d'entiers de la taille d'un paquet à partir du flux entrant (un tableau correspond à un paquet) - Puis pour chaque tableau crée : - Lecture de l'alphabet du tableau. - Ajout de zéro à l'alphabet si nécessaire. - Ecriture de la taille de l'alphabet sur un octet - Ecriture de l'alphabet sur le flux sortant. - Ecriture de la taille du paquet sur le flux. - Tri du texte selon les contextes à l'aide la méthode tri de la classe Tri. - Ecriture du texte trié sur le flux d'octet sortant
Résultat renvoyé	Néant

Conversion

Arguments nécessaires à la création d'une instance de cette classe :

- Flux d'octets entrant (InputStream)
- Flux d'octets sortant (OutputStream)

ExtraitAlphabet (Non utilisée): Extraction de l'alphabet du flux entrant

Argument	n : Taille de l'alphabet
Instructions	- Lire l'alphabet dans le flux entrant. - Stocker l'alphabet dans la variable Alphabet - Ecrire l'alphabet dans le flux sortant

Résultat renvoyé	String Alphabet : L'alphabet du texte à compresser

Convertit (méthode principale) : Convertit le flux de caractère en flux d'entier

Argument	Néant
Instructions	- Pour chaque paquet : - Lecture et recopie de la taille de l'alphabet sur le premier octet. - Lecture et recopie de l'alphabet en lui-même - Lecture et recopie de la taille du fichier sur trois octets du flux entrant. - Recopie de l'emplacement du dernier caractère (avant le tri) dans le paquet, codé sur 3 octets - Réalisation de la deuxième étape du tri, conformément à l'énoncé.
Résultat renvoyé	Néant

LettreEnPremier : déplace en tête un caractère dans une chaîne de caractère

Argument	String a : chaîne de caractère à modifier Int j : emplacement de la lettre à déplacer dans la chaîne
Instructions	Place la jième lettre en tête
Résultat renvoyé	Renvoie la chaîne de caractère en argument après avoir déplacé en tête la jième lettre

Permutation

Arguments nécessaires à la création d'une instance de cette classe :
- Flux d'octets entrant (InputStream)
- Flux d'octets sortant (OutputStream)

recopieAlphabet : Recopie l'alphabet sur le flux sortant

Argument	Int n : taille de l'alphabet
Instructions	- Lecture de l'alphabet dans le flux entrant. - Ecriture de l'alphabet dans le flux sortant.
Résultat renvoyé	Néant

Permute (méthode principale) : Calcule la fréquence de chaque caractère dans un paquet et remplace chaque caractère par son classement dans l'ordre décroissant des fréquences.

Argument	Néant
Instructions	- Pour chaque paquet : - Recopie de l'en-tête du paquet jusqu'à la position du dernier caractère avant tri. Récupération de la taille du paquet. - Calcul de la fréquence de chaque caractère dans la paquet avec ExtraitFrequence. - Tri du tableau des fréquences obtenu avec la méthode Trie - Ecriture de la permutation obtenue (le 1er octet écrit correspond au classement du caractère 0, et le dernier octet correspond au classement du caractère n-1, n étant la taille de l'alphabet utilisé - Remplacement de chaque caractère par son classement, et écriture du tableau obtenu sur le flux sortant

Résultat renvoyé	Néant

ExtraitFrequence

Argument	Tab : tableau d'entier dont on veut trouver la fréquence n : taille de l'alphabet
Instructions	- Ecrit dans la première colonne du tableau l'indice correspondant à la ligne du tableau - Ecrit dans la deuxième colonne la fréquence de l'indice.
Résultat renvoyé	- Tableau à 2 colonne : la première indique l'entier de l'alphabet et la deuxième la fréquence de l'entier correspondant.

Trie

Argument	- Tableau à deux colonnes : une pour les indices, l'autre pour les fréquences à trier
Instructions	- Quicksort sur la seconde colonne. - Extraction de la première colonne.
Résultat renvoyé	- Tableau unidimensionnel contenant les indices de la première colonne réordonnés selon l'ordre des fréquences

Trie2 : Fonction récursive auxiliaire de trie qui exécute le trie propement dit.

Transpose : Fonction auxiliaire de trie2 qui transpose deux fréquences avec leurs indices sur un tableau à deux colonnes.

Compression

Arguments nécessaires à la création d'une instance de cette classe :

- Flux d'octets entrant (InputStream)
- Flux d'octets sortant (OutputStream)

recopieAlphabet : Recopie l'alphabet sur le flux sortant

Argument	n : taille de l'alphabet
Instructions	- Lecture de l'alphabet dans le flux entrant. - Ecriture de l'alphabet dans le flux sortant.
Résultat renvoyé	Néant

Compresse (méthode principale) : Recopie la tête du flux puis réalise la compression proprement dite

Argument	Néant
Instructions	- Pour chaque paquet : - Recopie de l'en-tête du fichier jusqu'à la permutation. Récupération de la taille du paquet. - Réalisation de la 3ème phase de la compression conformément à l'énoncé.
Résultat renvoyé	Néant

Initialise : Crée un tableau de 8 zéros

Argument	Néant
Instructions	- Création d'un tableau de 8 entiers et affectation a tous ces entiers de la valeur 0.
Résultat renvoyé	Tableau de huit entiers valant 0.

Ajoute : Ajoute un bit ou un tableau de bits au flux d'octet sortant.

Argument	Baj : tableau d'entier	Baj : entier
Instructions	- Applique l'autre fonction ajoute à chacun des entiers	- Incrémente la variable globale p qui indique le numéro du bit à écrire dans l'octet enCours. Lorsque p=7, p retourne à 0 et on écrit l'octet enCours dans le flux de sortie avant de le réinitialiser à 0. - Réévaluation de la variable globale enCours, l'octet en cours d'écriture
Résultat renvoyé	Néant	

Décompression

Arguments nécessaires à la création d'une instance de cette classe :

- Flux d'octets entrant (InputStream)
- Flux d'octets sortant (OutputStream)

RecopieAlphabet : Recopie l'alphabet sur le flux sortant

Argument	n : taille de l'alphabet
Instructions	- Lecture de l'alphabet dans le flux entrant. - Ecriture de l'alphabet dans le flux sortant.
Résultat renvoyé	Néant

Decompresse (Méthode Principale):

Argument	Néant
Instructions	- Pour chaque paquet - Recopie de l'en-tête du fichier jusqu'à la permutation. Récupération de la taille du paquet (Attention : il s'agit de la taille du paquet décompressée) - Applique DecompressePaquet au paquets.
Résultat renvoyé	Néant

DecompressePaquet :

Argument	Taille : taille du paquet
Instructions	- Décompression d'un paquet conformément à l'énoncé du sujet. On utilise la méthode litEntierSuivant qui permet ainsi de décompresser caractère par caractère.
Résultat renvoyé	Néant

LitEntierSuivant : Lit l'entier suivant (sous forme compressé) dans le texte compressé

Argument	Néant
Instructions	En utilisant la méthode litbitSuivant, lit des bits dans le flux de données en s'arrêtant quand elle a lu ceux correspondant au code d'un caractère.
Résultat renvoyé	Un tableau d'entier codant un pour un caractère

litbitSuivant

Argument	Néant
Instructions	- Incrémente la variable globale p qui représente la position du bit à lire dans l'octet lu stocké dans la variable globale "lu". Quand p vaut 7 alors p revient à 0 et on lit en nouvel octet du texte compressé. - Stocke le p-ième bit de l'entier lu dans la variable globale "bit".
Résultat renvoyé	Néant

bitsVersEntier

Argument	Int nbre : nombre de bits à convertir
Instructions	- Lits nbre bits et calcule l'entier qu'ils représentent.
Résultat renvoyé	Int e : valeur correspondant aux nbre bits

InversionPermutation

Arguments nécessaires à la création d'une instance de cette classe :
- Flux d'octets entrant (InputStream)
- Flux d'octets sortant (OutputStream)

RecopieAlphabet : voir Permutation

InversePermutation (Méthode Principale): Réalise l'inverse le la méthode permute.

Argument	Néant
Instructions	- Pour chaque paquet : - Recopie l'entête jusqu'à la position du dernier caractère avant tri - Lecture de la permutation - Inversion de la permutation dans le tableau freq - Ecriture du texte en appliquant l'inverse de la permutation
Résultat renvoyé	Néant

Reconstitution

Arguments nécessaires à la création d'une instance de cette classe :
- Flux d'octets entrant (InputStream)
- Flux d'octets sortant (OutputStream)

ExtraitAlphabet : voir conversion

Reconstitue (Méthode Principale): Réalise l'opération inverse de la méthode convertit

Argument	Néant
Instructions	- Lecture de l'alphabet - Lecture et recopie de la taille du paquet - Lecture et recopie de la place du dernier caractère - Exécute l'inverse de la conversion.
Résultat renvoyé	Néant

LettreEnPremier : Voir conversion

Inversion

Arguments nécessaires à la création d'une instance de cette classe :

- Flux d'octets entrant (InputStream)
- Flux d'octets sortant (OutputStream)

Inverser (Méthode Principale): Réalise l'opération inverse de TrierPrefixe.

Argument	Néant
Instructions	- Lecture de la taille du paquet - Lecture du caractère final - Exécution de la méthode Inverse_tri et écriture du résultat sur le flux sortant.
Résultat renvoyé	Néant

InverseTri (Utilisé uniquement dans la classe Inversion)

Argument nécessaire à la création d'une instance de cette classe :

- Tableau d'entiers dont les trois derniers entiers codent la position dans le tableau du dernier caractère avant tri.

Occurrence : Calcule le nombre d'occurrence de chaque caractère dans le texte et pour chaque lettre du texte, le nombre d'occurrence de cette lettre dans le texte qui précède.

Argument	Int long_alphabet : Longueur de l'alphabet Int n : longueur du texte Int d : place du dernier caractère dans le texte
Instructions	- Parcourt du texte pour calculer les occurrences.
Résultat renvoyé	Int [] [] oc : tableau à deux colonnes La première colonne donne le nombre d'occurrence de chaque caractère dans le texte. La deuxième colonne donne le nombre d'occurrence de chaque lettre du texte dans la partie du texte qui précède.

Precedents : Donne le nombre de lettre du texte qui sont avant une lettre donné dans l'ordre de l'alphabet.

Argument	Int long_alphabet : longueur de l'alphabet Int [] [] oc : longueur des occurrences
Instructions	Additionne les occurrences de chaque lettre
Résultat renvoyé	Int [] Pour chaque lettre de l'alphabet, le tableau contient le nombre de lettre du texte qui sont avant dans l'ordre de l'alphabet.

Inv_tri : Réalisation de l'inversion du tri selon les préfixes

Argument	Néant
Instructions	- Réalise l'inverse du tri conformément à l'énoncé du sujet.
Résultat renvoyé	Int [] n_texte : texte non trié.

Agencement des différentes classes

Le problème est d'agencer les différentes méthodes qui réalisent la compression (TriPrefixe, Conversion; Permutation, Compression) ainsi que celles qui réalisent la décompression (Decompression, InversionPermutation, Reconstitution, Inversion).

Java fournit un moyen très simple pour cela. Chacune des classes cités étend la classe Thread.(processus). Ceci permet de lier par des "pipes" le flux sortant d'une classe avec le flux entrant de la suivante, c'est-à-dire de les identifier. Ce sont les méthodes Cmp et Dcmp qui se chargent de créer les bons "pipes" et de lancer chaque processus.

Ce procédé est très souple et permet de s'adapter à des flux d'octets de tous type (entrée/sortie standard, fichier, port de communication…). Il permet aussi d'utiliser un minimum de mémoire, notamment au niveau des classes Compression et Decompression qui écrivent un caractère chaque fois qu'elles en lisent un, sans rien stocker en mémoire.

Performances

Quelques tests permettent d'évaluer la qualité de l'algorithme. Le taux de compression est défini comme la taille du fichier compressé sur la taille du fichier initial.

Impact de la taille des paquets :

Taille des paquets	500	1000	2000	2500	4000
Taux de compression	23,2%	21%	19,8%	18,9%	17,8%

Statistiques réalisées avec la permutation de Huffman sur 5 fichiers Word.

Influence du type du fichier

Type du fichier	Word	Bitmap (N&B)	jpg
Taux de compression	21%	4%	>150%

Statistiques réalisées avec la permutation de Huffman, avec des paquets de taille 500

Le fichier bitmap comprimé à 4% :

Conclusion

Sur le plan des performances :

- Ce programme fonctionne aussi pour comprimer des fichiers comprenant de grandes occurrences. L'exemple extrême est un fichier bitmap de type « bande dessinée ».

- Lorsqu'on augmente la taille des paquets, on améliore la compression (en effet on diminue le nombre d'en-têtes), mais on perd légèrement en rapidité. Il faut donc trouver un compromis. Cependant, le programme n'a pas permis pas de faire des tests satisfaisants pour des paquets de taille supérieure à 4000.

Sur le plan des améliorations possibles du programme :

- Le programme fonctionne mais il est loin d'être optimisé. Par exemple, à de nombreux endroits, on utilise des types entier (int) quand des types octet (byte) suffirait.

- La fonction critique tant du point de vue du temps que du point de vue de la mémoire est le tri initial. En effet le fait d'utiliser des flux et plusieurs processus limite la quantité de mémoire utilisée, sauf lors du tri où l'on crée un arbre assez lourd. Ensuite, le fait que ce tri soit récursif limite la taille de paquet acceptable : En effet, si le paquet est trop grand, le nombre d'appel récursif dépasse la capacité de la pile de java. La limite dépend notablement du système d'exploitation et de l'ordinateur utilisé (environ 5000 sous linux, moins de 1000 sous Windows 98)
- On peut donc améliorer la vitesse du programme en programmant un tri qui se ferait uniquement sur un tableau d'octet. On améliorerait aussi la taille du paquet triable, car le nombre d'appel récursif autorisé semble dépendre de la complexité de la fonction appelée.

- Enfin, le programme est très flexible quand à ses entrées et sorties. Il est donc envisageable de réaliser la compression à la volée de données à transmettre. Bien sûr, il faut que le débit de la ligne de transmission soit faible pour permettre à l'ordinateur qui compresse de le faire à un débit au moins égal à celui de la ligne. Cependant, l'état actuel du programme permet de compresser à un débit d'environ 1ko/s en entrée soit de 2 à 4ko/s en sortie. Ainsi, cela le rend pour le moment utilisable uniquement sur des lignes à très faible débit, par exemple sur une connexion à internet par GSM.