Filière MP : ENS de Cachan, Lyon, Rennes et Paris

Page de garde du rapport de TIPE - Session 2014

NOM : BLANCHÉ Prénoms : Alexandre, Antoine

Lycée : Camille Guérin (86)

Classe : MP*

Ville : Poitiers

Concours auxquels vous êtes admissible dans la banque MP inter-ENS :

(les indiquer par une croix inscrite dans les cases ci-dessous)

ENS Cachan	MP - option MP		MP - option MPI	
	Informatique			
ENS Lyon	MP - option MP		MP - option MPI	
	Informatique - option M	X	Informatique - option P	
ENS Rennes	MP - option MP		MP - option MPI	
	Informatique	X		
ENS Paris	MP - option MP		MP - option MPI	
	Informatique			

Matière dominante du TIPE (la sélectionner d'une croix inscrite dans l'une des cases ci-dessous) :

Informatique [X] Mathématiques [ ] Physique [ ]

Titre du TIPE : La transformée de Burrows-Wheeler

Nombre de pages (à indiquer dans les cases ci-dessous) :

Texte		Illustrations		Bibliographie		
T	7	I	2	B	1	Code : 33

Résumé (ou descriptif succinct) du TIPE (6 lignes, maximum) :

Etude de la transformée de Burrows-Wheeler.
Programmation de la transformée et de plusieurs algorithmes de compression sans perte.
Comparaison des divers algorithmes, et des différentes combinaisons possibles.

A Poitiers, le 12/06/14

Signature du (de la) candidat(e)

Signature du professeur responsable de la classe préparatoire dans la discipline

Cachet de l'établissement

LYCEE CAMILLE GUERIN – POITIERS – Le C.P.E.,

BLANCHÉ
Alexandre

Travaux d'Initiative Personnelle Encadrés

La Transformée de Burrows-Wheeler

Introduction

Notre société actuelle, entièrement tournée vers les technologies du numérique, impose de nombreux transferts d'informations. Nous avons donc étudié des techniques de compression de données, de type « sans perte », qui conservent l'intégralité de l'information présente dans un fichier, par le biais de transformations bijectives.
Nous nous sommes intéressés à la transformée de Burrows-Wheeler, un algorithme de « pré-compression », qui modifie une chaîne afin d'améliorer l'efficacité d'autres algorithmes.
Notre objectif était donc d'implémenter rigoureusement et efficacement cette transformée, puis de programmer d'autres algorithmes de compression courants qui peuvent lui être appliqués, et analyser son impact en comparant diverses critères.

Ce TIPE a été mené conjointement avec Guillaume Ménard.

I. Présentation de la transformée

a) Principe de base

b) Intérêt de la transformée

II. Présentation des algorithmes

a) MTF, RLE, Huffman

b) Subtilités d'implémentation

c) L'implémentation du tri

III. Comparaison des algorithmes

a) En temps de calcul

b) En taux de compression

c) Remarques

I. Présentation de la transformée de Burrows-Wheeler

a) Principe de base

La transformée de Burrows-Wheeler est un algorithme inventé en 1994 par M. Burrows et D.J. Wheeler. Cet algorithme ne compresse pas en tant que tel, mais réorganise les caractères d'une chaîne afin d'améliorer le taux de compression d'autres algorithmes de compression sans perte qui lui seraient appliqués. Cette transformation est utilisée notamment dans la compression bzip2 ou en séquençage d'ADN.

Traitons l'exemple « BLAN-MENA » :

On crée la liste de toutes les rotations :	On les trie par ordre lexicographique :	On ajoute la chaîne initiale et on note sa position :
ABLAN-MEN	-MENABLAN	-MENABLA**N**
NABLAN-ME	ABLAN-MEN	ABLAN-ME**N**
ENABLAN-M	AN-MENABL	AN-MENAB**L**
MENABLAN-	ENABLAN-M	BLAN-MEN**A** (4)
-MENABLAN	LAN-MENAB	ENABLAN-**M**
N-MENABLA	MENABLAN-	LAN-MENA**B**
AN-MENABL	N-MENABLA	MENABLAN-
LAN-MENAB	NABLAN-ME	N-MENABL**A**
		NABLAN-M**E**

Enfin, on extrait le dernier caractère de chaque rotation triée, et on adjoint la position de la chaîne initiale. On obtient ainsi la chaîne : **4NNLAMB-AE**

Pour le décodage : on part de la chaîne « 4NNLAMB-AE ».

On trie les couples (caractère, position de ce caractère dans la chaîne) par ordre lexicographique, comme ceci :

(- ,7) ; (A,4) ; (A,8) ; (B,6) ; (E,9) ; (L,3) ; (M,5) ; (N,1) ; (N,2)

Puis on débute à la position donnée (ici 4) et on parcourt d'indice en indice :

Le couple 4 est (B, 6), il pointe vers le couple 6 (L, 3), qui pointe vers le couple 3 (A, 8), et ainsi de suite jusqu'à retomber sur la position initiale, 4.

(4) B →(6) L →(3) A →(8) N →(1) - →(7) M →(5) E →(9) N →(2) A →(4) stop

On renvoie bien la chaîne initiale : **BLAN-MENA**

b) Intérêt de la transformée

Le principal intérêt de cette réorganisation est d'augmenter la probabilité d'avoir une succession de plusieurs caractères identiques :

```
bw "moi et toi et lui";;
- : string = "11iitt oou itmee l"
```

Ici, on a plusieurs fois le motif « _et ». Les rotations correspondantes seront successives après l'étape de tri :

…

_et_lui_moi_et_to**i**

_et_toi_et_lui_mo**i**

…

On a donc des caractères « i », qui précèdent le motif « _et », côte-à-côte dans la chaîne finale.

Exemple :

```
bw "Il_peut_être_démontré_que_dès_que_le_module_de_zn_est_strictement_plus_
grand_que_2_(zn_étant_sous_forme_algébrique,_quand_xn2_+_yn2>_22),_la_suite
_diverge_vers_l'infini,_et_donc_c_est_en_dehors_de_l'ensemble_de_Mandelbrot
._Cela_nous_permet_d'arrêter_le_calcul_pour_les_points_ayant_un_module_stri
ctement_supérieur_à_2_et_qui_sont_donc_en_dehors_de_l'ensemble_de_Mandelbro
t._Pour_les_points_de_l'ensemble_de_Mandelbrot,_c'est-à-
dire_les_nombres_complexes_c_pour_lesquels_zn_ne_tend_pas_vers_l'infini,_le
_calcul_n'arrivera_jamais_à_terme,_donc_il_doit_être_arrêté_après_un_certai
n_nombre_d'itérations_déterminé_par_le_programme._Il_en_résulte_que_l'image
_affichée_n'est_qu'une_approximation_du_vrai_ensemble.";;

- : string =
"37nndlllcnllldu_2_tei)ieàtetet2_n_n_2__.____22>eà..eee.eeeées,cseenst eeees
ssenneltt,neestclicn,2sscaeeeess,,rererrrenlennsaédsltsslcet,déestnitteateà
stesud+esrsnttrll_mrtmcc_jrruMMMty__p''_pmr_mmmmmmélll_nn_n____i_iill__nnn_
_____nnn-
_____oo___mulmudddgmnrllrllllturddduurléltlngdddCdddussssstt___t_'''mmtvvtp
tvvcxrllll'_'_m__iplafnn_aro_leecnnaufrrr_'xa''ffoomttrdaou'rd_____iIuIue
_eeeaab__bbbu__u____paeupiaieeeeoormreerra__éo__yxueeozuziezoooeaaaaau_iiii
___oeeeeoeaeaoiiimmrppdnncidddismfhhrrrpPpsnr____m_____a_pau__i____s_eauuuu
evggétbtibettébreoeepbbbpaaaooeeept_rrteetrrnuurreueeèèalinnnn__eeeee___é_o
sooeensnsensnniuréilcc_ê_éaaêêssnnnêiqdqqqqqqqqsccdds__'soooeoloeii___eo_a_
__(-__drtnrghdtpr_dr__r"
```

On remarque ici de nombreux motifs de caractères identiques successifs dans la chaîne codée.
En jaune ceux de plus de 4 caractères.

II. Présentation des algorithmes

a) MTF, RLE, Huffman

i. MTF

L'algorithme Move-to-front est très souvent utilisé pour accompagner la transformée de Burrows-Wheeler. Il ne compresse pas, mais change la valeur des caractères de la chaîne afin de rendre certains plus communs, typiquement les premiers de l'alphabet.

Son déroulement est le suivant :

La chaîne est notée s.

- On forme une liste l recueillant les caractères utilisés (ici le code ASCII) :

l = [/000 ; /001 ; /002 ; … ; 'a' ; 'b' ; 'c' ; … ; 'ÿ']

- On crée une chaîne vide s' de même longueur que s.

- Pour chaque caractère *ch* de s:

On cherche la position de *ch* dans l, et on place dans s' le caractère correspondant.
Ex : Si s.[29] = 'a' et que 'a' est en position 35 dans l à cette étape, alors s'.[29] reçoit '#', le caractère n°35 en ASCII.
Puis on supprime *ch* de l, et on l'ajoute en tête.

Ainsi, on a remplacé les caractères du texte par d'autres, qui apparaîtront plus souvent, donc plus économes avec un codage entropique de type Huffman.

ii. RLE

Run-Length-Encoding est un algorithme très simple et couramment utilisé de compression sans perte. Il remplace une séquence de plusieurs caractères identiques par la longueur de la séquence, suivie du caractère.

```
rle "aaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbccddddddfghijjjj";;
- : string = "16a16b2c6dfghi4j"
```

Cet algorithme est parfaitement adapté à la transformée de Burrows-Wheeler, qui crée de nombreuses répétitions.

La principale difficulté dans l'implémentation venait de la présence éventuelle de tous les caractères ASCII dans la chaîne, d'où l'emploi d'un caractère d'échappement, annonçant un passage en mode chiffre/mode lettre, caractère qu'il fallait également gérer. Nous sommes donc arrivés à une version universelle, qui accepte tout type de chaînes.

iii. Huffman

Les codages de Huffman sont tels que le nombre de bits sur lequel un caractère est codé dépend de son nombre d'occurrences. Un caractère courant sera codé sur un nombre restreint de bits.
Huffman, optimal du point de vue de la longueur, est un sujet très vaste que je ne développerai pas ici. Notre transformée peut cependant améliorer son efficacité.

b) Subtilités d'implémentation

i. Suffix-Array

L'idée du Suffix-Array est due à U. Manber et G. Myers en 1990.

Cela consiste à remplacer un tableau de chaînes par un tableau d'entiers, chacun représentant le point de départ du suffixe (ou de la rotation).

BABCA → 1
ABCA → 2
BCA → 3
CA → 4
A → 5

Cette méthode a permis de résoudre le problème de la complexité spatiale quadratique que nous avions naïvement au départ.

On trie donc le tableau [|1 ; 2 ; 3 ; 4 ; 5|] en [|5 ; 2 ; 1 ; 3 ; 4|], qui représente le Suffix-Array des rotations triées.

ii. EOF

Burrows et Wheeler donnent une méthode d'amélioration du tri, utilisant un caractère *EOF* (End-Of-File), que l'on notera $.

Ce caractère ne doit pas apparaître dans le reste de la chaîne, et possède une valeur quelconque dans l'ordre lexicographique, que nous avons prise strictement supérieure à toutes les autres. Trier les rotations d'une chaîne S revient à trier les suffixes de la concaténation de S avec l'*EOF* :

Cette technique enlève les cas d'égalités qui empêchent de trier simplement les suffixes sans *EOF*. Le temps de calcul est donc expérimentalement divisé par 2.

BABCA$	→	ABCA$	→	ABCA$B	→	chaîne renvoyée :
ABCA$	tri	A$		A$BABC		**3BC$ABA**
BCA$		BABCA$		BABCA$ (3)		
CA$		BCA$		BCA$BA		
A$		CA$		CA$BAB		
$		$		$BABCA		

La position de l'*EOF* coïncidant avec la position transmise, nous avons considéré un *EOF* fictif, non physiquement présent, résolvant la recherche d'un caractère libre.

Nous renvoyons donc : **3BCABA**

<u>À la décompression :</u>

Nous trions les caractères, en augmentant leur position de 1 après la position donnée (ici 3), pour simuler l'existence de l'*EOF*:

(A,**4**) ; (A,**6**) ; (B,1) ; (B,**5**) ; (C,2) ; ($,3)

Nous décodons jusqu'à ce que le pointeur nous envoie vers la pseudo-case contenant l'*EOF* (ici 6) :
(3) B → (1) A → (4) B → (5) C → (2) A → (6) : **BABCA**

La technique de l'*EOF* n'a aucune contrainte et accélère grandement l'étape de tri.

c) L'implémentation du tri

L'étape de tri étant de loin la plus lente, Burrows et Wheeler en proposent une implémentation efficace mais complexe. Nous avons donc préféré une méthode plus simple, imaginée par les chercheurs U. Manber et G. Myers, ayant pour but de remplacer le Quicksort naïf.

Radix Sort

Le principe du tri Radix est simple : on trie les chaînes caractère par caractère, ce qui mène à former des classes d'équivalence, regroupant les chaînes égales selon leurs p premiers caractères, au bout de la p^e étape.

Tri selon caractère :		1		2		3
DFGEA…	→	**A**SQER…	→	A**B**PTR…	→	AB**J**HY…
ASQER…		**A**BJHY…		A**B**JHY…		AB**P**TR…
ABJHY…		**A**BPTR…		A**S**QER…		ASQER…
KJHFT…		**D**FGEA…		DFGEA…		DFGEA…
ABPTR…		**K**JHFT…		KJHFT…		KJHFT…

Une implémentation astucieuse donne une complexité en pire cas O(nk) si l'on arrête le tri au bout de k caractères pour une chaîne de taille n.

Méthode de Manber et Myers

Cette méthode exploite le caractère cyclique des chaînes à trier.
On effectue un Radix de k caractères. Les chaînes d'une même classe d'équivalence sont égales selon leurs k premiers caractères. On souhaite donc les trier selon leurs caractères k+1 à 2k. Ces caractères sont justement les préfixes de rotations déjà pré-triées. Il suffit donc de parcourir le tableau des chaînes pré-trié et replacer les éléments de la classe dans l'ordre de leurs « jumeaux » décalés de k caractères.

Exemple :

(1) AAAABAC		**(3) AA**BACAA
(2) AAABACA		(2) **AA**ABACA
(3) AABACAA	→	(1) **AA**AABAC
(4) ABACAAA	Radix avec k=2	**(4) AB**ACAAA
(5) BACAAAA		(6) **AC**AAAAB
(6) ACAAAAB		**(5) BA**CAAAA
(7) CAAAABA		(7) **CA**AAABA

Ici il n'y a qu'une classe, qui contient les chaînes (3),(2),(1). On souhaite les trier selon leurs successeurs, en vert, représentant respectivement les rotations (5), (4), (3), en bleu.
Ces rotations ont déjà été pré-triées dans l'ordre (3) – (4) – (5).
On place donc l'antécédent de (3) en premier : (1) ; puis celui de (4) et enfin celui de (5), respectivement (2) et (3).

(1) AAAABAC
(2) AAABACA
(3) AABACAA
(4) ABACAAA
(6) ACAAAAB
(5) BACAAAA
(7) CAAAABA

Il n'y a plus d'égalité : on a terminé.
Sinon, on continue : à la p^e étape, les 2^pk premiers caractères sont triés.

Complexité totale : O(nk + n log ((n-k)/k)) en pire cas.

Une étude de fonction donne k=1 optimal, soit une complexité en **O(n log n)**.

Cette complexité est asymptotiquement meilleure que le Quicksort en $O(n^2 \log n)$ en pire cas à cause des comparaisons linéaires et non pas unitaires.
En pratique, la méthode de Manber et Myers est 1,4 fois plus lente que le Quicksort, sur plusieurs tests.
En pire cas, à savoir une chaîne d'un unique caractère répété, type ''aaaaaaaaaa…'', l'algorithme est infiniment meilleur que le Quicksort, ce qui respecte l'étude de complexité menée plus haut.

III. Comparaison des algorithmes

Nos résultats de comparaisons sont disponibles en annexe.

a) En temps et en espace

Les algorithmes utilisés ont tous une complexité spatiale linéaire en la taille de la chaîne.

Les algorithmes hors BW sont en général assez rapides, de l'ordre de quelques dizaines de secondes pour des fichiers de grande taille, à savoir plusieurs méga-octets. La complexité temporelle des algorithmes est linéaire, hormis Burrows-Wheeler, dont le décodage est en O(n log n) (O(n) avec un Radix Sort), et le codage en O(n log n), avec cependant une constante bien plus élevée.

Seul l'algorithme de codage de Burrows-Wheeler est particulièrement lent, le décodage étant lui très rapide. BW étant dix à vingt fois plus lent que les autres, son utilisation en devient assez délicate…

b) En taux de compression

Globalement, la transformée améliore le taux de compression des autres algorithmes, parfois de manière importante (Texte_1200 : sans 38% ; avec 73%). L'efficacité en compression augmente avec la taille des fichiers, dépassant largement celle des algorithmes seuls (JPEG_637 : 30% ; JPEG_29000 : 52%).

La composition la plus efficace en moyenne est : **BW – RLE – MTF – Huffman**, quasiment à égalité avec BW – MTF – RLE – H, suivie de BW – MTF – H, très proche cependant.

Les compositions sans BW sont parfois10% moins efficaces pour une vitesse multipliée par 10, ce qui peut sérieusement remettre en question l'intérêt de la transformée.

c) Remarques

- Sur plusieurs fichiers et combinaisons de composition, itérer les compressions ne permettait pas de compresser davantage…

Nombre de compressions (.bw.rle.mtf.huf_bis)	0	1	2	3	4
Taille (octet)	31 162 263	**7 983 343**	8 424 981	9 004 761	9 637 421

- Une modification de la chaîne transformée par Burrows-Wheeler rend la décompression incohérente.

- Nous aurions pu améliorer l'efficacité du RLE en codant le « compteur » avec des caractères peu utilisés, tels les derniers du code ASCII, dans une base numérique plus grande. Cependant, cela n'a aucun impact sur des fichiers quelconques, et cela rend Huffman moins efficace.

- Nous avons également compressé et décompressé des images.

Conclusion

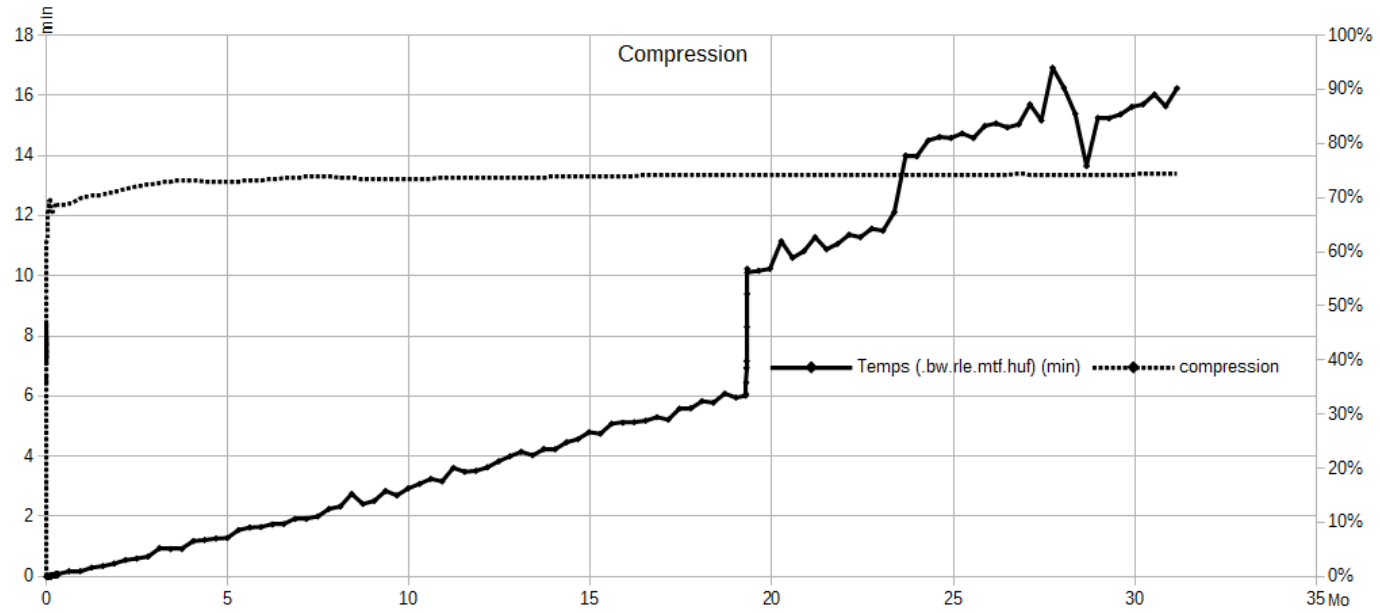
La transformée de Burrows-Wheeler est globalement efficace, et permet d'améliorer, parfois de manière très importante, les taux de compression des algorithmes que nous avons étudiés.

En revanche, le principal écueil reste le temps de calcul. Cette difficulté, déjà rencontrée par Burrows et Wheeler, est très handicapante pour un usage répandu. Comme précisé en annexe, la transformation dure une dizaine de minutes pour des fichiers de 25 Mo.

Une utilisation qui pourrait être préconisée, serait de tenter toutes les combinaisons possibles de compositions des algorithmes, puis de choisir la plus efficace pour chaque fichier à compresser. En effet, l'étape BW étant largement prédominante sur les autres, le temps de calcul des autres combinaisons est négligeable.

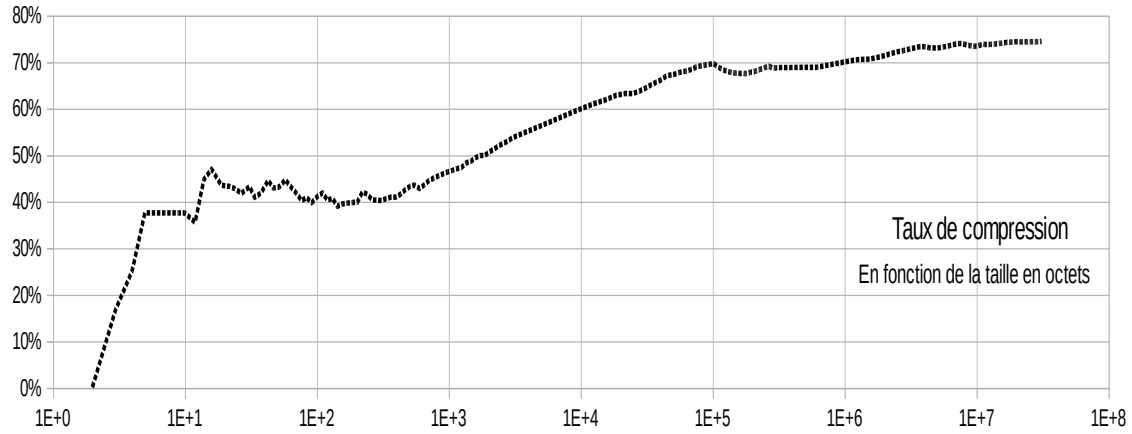
ANNEXE

ANNEXE 1



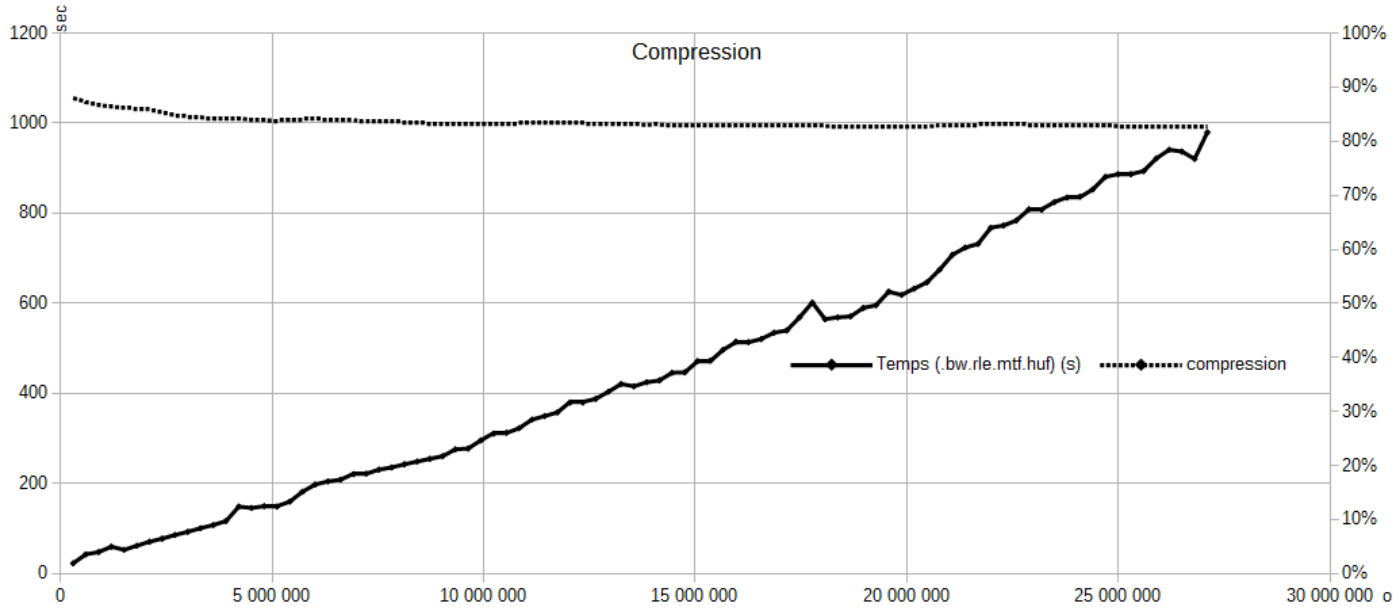
Temps de calcul en fonction de la taille des fichiers.

Le pic brutal autour de 18Mo est dû à la présence d'une longue répétition dans le texte. Le tri, à l'époque le Quicksort, était donc dans une configuration proche du pire cas.



Taux de compression en fonction de la taille des fichiers.
Sur ce fichier de grande taille, on tend vers une compression de 75%.

ANNEXE 2



Sur un autre fichier, nous avons une progression quasiment linéaire du temps de calcul, car le fichier ne possédait alors pas de répétition majeure.

	JPEG_29000.bmp (30 108 677)			PNG_29000.bmp (30 108 677)			Texte_29000.txt (31 162 263)			Moyenne		
	Taux	Durée		Taux	Durée		Taux	Durée		Taux	Durée	
		Étape	Totale		Étape	Totale		Étape	Totale		Étape	Totale
.bw	-0%	648	648	-0%	1 717	1 717	-0%	1 569	1 569	-0%	1 311	1 311
.bw.mtf	-0%	18	666	-0%	9	1 726	-0%	11	1 580	-0%	13	1 324
.bw.mtf.huf	**54%**	15	681	81%	10	1 736	71%	13	1 593	69%	13	1 337
.bw.mtf.rle	6%	46	712	58%	20	1 746	42%	29	1 609	35%	32	1 356
.bw.mtf.rle.huf	51%	16	728	81%	7	1 753	72%	9	1 618	68%	11	1 366
.bw.rle	6%	45	693	64%	17	1 734	46%	28	1 597	38%	30	1 341
.bw.rle.huf	17%	19	712	69%	8	1 742	67%	11	1 608	51%	13	1 354
.bw.rle.mtf	6%	21	714	64%	8	1 742	46%	11	1 608	38%	13	1 355
.bw.rle.mtf.huf	52%	14	728	**82%**	6	1 748	**74%**	8	1 616	**70%**	9	1 364
.huf	8%	21	**21**	5%	21	**21**	44%	16	**16**	19%	19	**19**
.mtf	0%	71	71	0%	42	42	0%	37	37	0%	50	50
.mtf.huf	31%	18	89	67%	13	55	39%	19	56	46%	17	67
.mtf.rle	2%	47	47	37%	29	29	-0%	52	52	13%	43	43
.mtf.rle.huf	28%	19	66	62%	11	40	37%	17	69	42%	16	58
.rle	9%	43	43	50%	22	**22**	1%	56	56	20%	40	40
.rle.huf	21%	19	62	59%	10	32	44%	17	73	41%	15	56
.rle.mtf	9%	74	117	50%	44	66	1%	33	89	20%	50	91
.rle.mtf.huf	28%	17	134	64%	10	76	39%	16	105	44%	14	105

Ne figurent ici que 3 exemples. Nous avons cependant effectué de nombreuses compressions sur d'autres fichiers. Ces exemples de grande taille (30Mo), nous ont paru être un bon bilan.
La taille des fichiers est indiquée entre parenthèses, en octets. Les taux de - 0% sont normaux, puisque BW et MTF effectuent seulement un pré-traitement, sans compresser.

ANNEXE 3

CODE COMPLET

N.B : Voici le code de notre programme, en langage OCaml, version 4.00.1. Nous n'avons pas transmis l'intégralité du code, car de nombreux programmes ont été optimisés et sont donc devenus obsolètes. Ici figure la version finale de chaque programme.
Bien évidemment, le code compile et fonctionne correctement.

Tous les programmes ont été écrits par mon binôme Guillaume Ménard et moi-même, à l'exception de « mlbmpperso.ml », qui a été écrit par Marc Lorenzi, enseignant en Mathématiques et Informatique en CPGE. Il nous a permis de convertir des fichiers .bmp (images bitmap) pour les rendre exploitables en Caml.

Burrows-Wheeler_30.ml

```
(*********************************************)
(*       La transformée de Burrows-Wheeler      *)
(*********************************************)
  (* Par Guillaume Ménard & Alexandre Blanché *)

               (* Version 30 *)


#use "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/Radix MM 5.ml";;
#use "C:/Users/antoine/TD Caml/TIPE/Radix MM 5.ml";;
open String;;

(* Transforme un Char en String *)
   let string_of_char = String.make 1;;

(* Renvoie le chiffre correspondant pour un caractère *)
   let digit_of_char c = int_of_char c - int_of_char '0';;

(* Précise si un caractère est un chiffre *)
   let is_int_char c = let v=digit_of_char c in v>=0 && v<=9;;

(* Fonction modulo restreinte et, espérons, plus rapide *)
   let (//) a b =
      if a = b then
         0
      else if a = -1 then
         b-1
      else a;;

(* Relation de comparaison des chaines circulaires *)
   let rec compare_bien_aux s i1 i2 fin_i1 s_length fini =
      let s1=s.[i1] and s2=s.[i2] in
         if s1=s2 then
            begin
               if i1=fin_i1 && fini then
                  0
               else
                  compare_bien_aux s ((i1+1) // s_length) ((i2+1) // s_length) fin_i1
s_length true
            end
         else if s1<s2 then -1
         else (*if s1>s2 then*) 1;;

   let compare_bien s s_length i1 i2 = compare_bien_aux s i1 i2 i1 s_length false;;
```

```
(* Renvoie la liste des entiers de 0 à n *)
   let rec range_aux n l =
      if n=0 then 0::l
      else range_aux (n-1) (n::l);;
   let range n = range_aux n [];;

(* Sors la liste des rotations de la chaîne, avec la chaîne de base. *)
   let switch_all_new s =
      let s_length = length s in
      List.fast_sort (compare_bien s s_length) (range (s_length -1));;

(* Cherche la position de la chaine de départ *)
   let rec pos_depart_aux l pos =
      match l with
      |[]-> failwith "pos_depart_aux"
      |h::t-> if h=0 then pos else pos_depart_aux t (pos+1);;
   let pos_depart l = pos_depart_aux l 1;;

(* Recolle les caractères formant la chaine de rendu, on prend donc les nombres qui
précèdent ceux triés. *)
(*
   s : chaîne en question
   n : modulo (longueur de s)
   l : liste d'entiers représentant les permutations
   d : longueur de la chaîne à joindre au début de rep
   rep : chaîne de traitement et sortie
   i : indice de placement dans rep
*)
   let rec fusion_caractere_opti_aux s n l rep i =
      match l with
      |[] -> rep
      |h::t -> rep.[i] <- s.[(h-1) // n];
               fusion_caractere_opti_aux s n t rep (i+1);;

   let fusion_caractere_opti s n l d =
               fusion_caractere_opti_aux s n l (make (d+List.length l) ' ') d;;

(* BW qui double le premier caractère si c'est un non-chiffre, met un "a" sinon. *)
   let bw_opti s =
      if s = "" then
         ""
      else
         let permut_triee_list = switch_all_new s in
         let n = length s in
         let first_char = s.[(List.hd (permut_triee_list) -1) // n] in


         let debut = (string_of_int (pos_depart permut_triee_list)) ^
            (if is_int_char first_char then "a" else string_of_char first_char) in
         let d = length debut in

         let chaine = fusion_caractere_opti s n permut_triee_list d in
         for i=0 to d-1 do
            chaine.[i] <- debut.[i]
         done;

         chaine;;


(*BW qui ajoute l'EOF *)
(* Renvoie la liste des caractères de 0 à n *)
   let rec range_char_aux n l =
      if n=0 then (char_of_int 0)::l
      else range_char_aux (n-1) ((char_of_int n)::l);;
   let range_char n = range_char_aux n [];;
```

```
(* enlève c de la liste l, sachant qu'il apparait au plus une fois *)
  let rec supprime c l =
      match l with
      |[]->[]
      |h::t->
         if h=c then t
         else h::(supprime c t);;

(* Renvoie le plus petit caractère pas dans la chaine, s'il n'existe pas, il renvoie le
premier caractère *)
   let rec eof_aux s i rep =
      if i>=String.length s then
         rep
      else
         eof_aux s (i+1) (supprime s.[i] rep);;
   let eof s =
      match eof_aux s 0 (range_char 255) with
      |[]->s.[0]
      |h::t->h;;

   let eof2 s =
      let candidats = Array.make 256 true in
      for i=0 to String.length s -1 do
         candidats.(int_of_char s.[i])<-false;
      done;
      let c = ref 0 in
      while not candidats.(!c) && !c<255 do
         c := !c+1;
      done;
         char_of_int !c;;



   let bw_eof s =
      if s = "" then
         ""
      else
         let s_eof = s^string_of_char (eof2 s) in
         let permut_triee_list = switch_all_new (s_eof) in
         let n = length s_eof in
         let first_char = s_eof.[(List.hd (permut_triee_list) -1) // n] in


         let debut = (string_of_int (pos_depart permut_triee_list)) ^
            (if is_int_char first_char then "a" else string_of_char first_char) in
         let d = length debut in

         let chaine = fusion_caractere_opti s_eof n permut_triee_list d in
         for i=0 to d-1 do
            chaine.[i] <- debut.[i]
         done;

         chaine;;


(* Décodage *)

(* On regarde sur combien de caractère s'étend le premier nombre *)
let longueur_chiffres text =
   let c = ref 0 and l= length text in
      while !c<l && is_int_char text.[!c] do
         c:=!c+1
      done; !c;;
```

```
(* Prend une chaine, et met chaque couple (caractère, position) dans un tableau *)
   let tboftxt txt =
      let rep= Array.make (length txt) ('a',0) in
         for i=length txt -1 downto 0 do
            rep.(i)<-(txt.[i],i)
         done;
         rep;;

   (* Vide Tableau dans texte, lgth est le nombre de répétition à faire, indice la
position du caractère dans le tableau et i celui dans la chaine. *)
   let rec ajoute_opti_aux texte tableau indice i lgth=
      if i >= lgth then
         texte
      else
         let (c,rg) = tableau.(indice) in texte.[i] <- c;
         ajoute_opti_aux texte tableau rg (i+1) lgth;;

   let ajoute_opti tableau debut =
      let lgth = Array.length tableau in
      ajoute_opti_aux (make (lgth) ' ') tableau debut 0 lgth;;

   let compare_valeurs a b = if a=b then 0 else if a<b then -1 else 1;;

   let bw_inv_opti txt =
      if txt = "" then
         ""
      else
         let longpos = longueur_chiffres txt in
         let longtexte = length txt -longpos-1 in
            let origin = tboftxt (sub txt (longpos+1) longtexte) and
            debut = int_of_string (sub txt 0 longpos) -1 in
               Array.fast_sort compare_valeurs origin;
               ajoute_opti origin debut;;

(* Pour la version avec EOF, on s'arrête à l'avant-dernier caractère *)
   let ajoute_opti_eof tableau debut =
      let lgth = Array.length tableau in
      ajoute_opti_aux (make (lgth-1) ' ') tableau debut 0 (lgth-1);;
   let bw_eof_inv txt =
      if txt = "" then
         ""
      else
         let longpos = longueur_chiffres txt in
         let longtexte = length txt -longpos-1 in
            let origin = tboftxt (sub txt (longpos+1) longtexte) and
            debut = int_of_string (sub txt 0 longpos) -1 in
               Array.fast_sort compare_valeurs origin;
               ajoute_opti_eof origin debut;;


(* Codage RLE: Run-Length Encoding *)

(* Compte le nombre de caractères a identiques à la suite dans txt à partir de la
position pos.
On suppose que le premier caractère testé est bien a, on sort donc 1 si le suivant
varie. *)
let rec nb_suite_carac_aux a txt pos fin nb=
   if pos < fin && txt.[pos]=a then
      nb_suite_carac_aux a txt (pos+1) fin (nb+1)
   else
      nb;;
let   nb_suite_carac a txt pos = nb_suite_carac_aux a txt pos (length txt) 0;;
```

```
(* Principe du compteur en lettres:
On compte en base 26: a=0, b=1, ... , z=25.
Les "décimales" les moins prépondérantes sont à gauche, les "dizaines" sont à droite.
Ex: 26*2-1 = zb , 26*2 = ac , 26*2+1 = bc *)
let rec compteur_lettres_aux n rep=
   match (n/26,n mod 26) with
   (q,r)->
      let alphabet = "abcdefghijklmnopqrstuvwxyz" in
      let a = (make 1 alphabet.[r]) in
         if q=0 then rep^a else compteur_lettres_aux q (rep^a);;
let compteur_lettres n = compteur_lettres_aux n "";;


(* Méthode: Premier passage sans rien toucher, en incrémentant simplement une variable
qui définira la longueur de la chaîne compressée. Puis deuxième passage en plaçant
cette fois les caractères dans une chaîne de longueur adéquate. *)
   let rec premier_passage txt pos mode i =
      if pos >= length txt then
         i
      else
         let n = nb_suite_carac txt.[pos] txt pos in
            let newmode = not (is_int_char txt.[pos]) in
               let long_caractere = if txt.[pos]<>'@' then 1 else 2 in
                  let long_compteur =
                     if n=1 && txt.[pos]<>'@' then
                        0
                     else if newmode then
                        length (string_of_int n)
                     else
                        length (compteur_lettres n)
                  in
                     if newmode=mode then
                        premier_passage txt (pos+n) newmode (i+long_compteur +
long_caractere)
                     else
                        premier_passage txt (pos+n) newmode (i+1+long_compteur +
long_caractere);;



(* Place la chaîne s1, caractère par caractère, dans s2 à partir de debut. *)
   let place s1 s2 debut =
      let n = length s1 in
      for i=0 to n-1 do
         s2.[debut+i] <- s1.[i]
      done;;

(* Nouvel algorithme de RLE

@ caractère de saut: lorsqu'on le rencontre UNE FOIS, on passe de mode lettres (true) à
mode chiffre (false), ou l'inverse, et lorsqu'on le rencontre DEUX FOIS, on considère
le caractère @ sans changer de mode.
 mode = true : les caractères sont des lettres, et leur compteur est un nombre
 mode = false : les caractères sont des chiffres et leur compteur une lettre

   newmode = le mode utilisé pour l'étape en cours

   Pour le @: si on change de mode, on met @+la suite
            mais si on rencontre le vrai caractère @ dans le texte:
         on le double, donc "abc@de" sera changé en abc@@de, et "ab@1de" en ab@@@1@de
         mais "ab@@@de" sera bien changé en ab3@@de.
 *)
```

```
   let rec rle_opti_aux txt pos mode rep i =
      if pos >= length txt then
         rep
      else
         let n = nb_suite_carac txt.[pos] txt pos in
            let newmode = not (is_int_char txt.[pos]) in
               let caractere = if txt.[pos]<>'@' then make 1 txt.[pos] else "@@" in
                  let compteur =
                     if n=1 && txt.[pos]<>'@' then
                        ""
                     else if newmode then
                        string_of_int n
                     else
                        compteur_lettres n
                  in

                  let s = (if newmode=mode then "" else "@")^compteur^caractere in
                  place s rep i;
                  rle_opti_aux txt (pos+n) newmode rep (i+length s);;

let rle_opti txt =
   (* Le fait de mettre true dès le début permet de mettre un @ dès le premier
caractère si besoin. *)
   let n = premier_passage txt 0 true 0 in
      rle_opti_aux txt 0 true (make n ' ') 0;;



(* Fonction inverse: décodage du RLE *)


(* Associe à une lettre sa position dans l'alphabet, en partant de a=0 *)
let lettre_int a pos = int_of_char a -97;;

(* Inverse de compteur_lettres *)
let rec compteur_lettres_inv str pos =
   if pos = length str-1 then
      lettre_int str.[pos] 0
   else
      (lettre_int str.[pos] 0)+26*compteur_lettres_inv str (pos+1);;

(* Renvoie le couple: (valeur du compteur du caractère testé, longueur du compteur).
On ne teste jamais sur le dernier caractère de txt, donc mettre [pos+1] ne pose pas
problème. *)
let rec compteur txt pos origine mode =
      if is_int_char txt.[pos+1] = mode then
                     compteur txt (pos+1) origine mode
      else
                  let l = pos-origine+1 in
                     let chaine = sub txt origine l in
                        if mode then (int_of_string chaine , l)
                                else (compteur_lettres_inv chaine 0 , l);;


(* mode<>is_int_char a : cela signifie que si l'on est en mode chiffre et que l'on voit
une lettre, ou que l'on est en mode lettre et que l'on voit un chiffre, alors tout est
normal (il s'agit du compteur).
Principe de chgt:
      chgt = true : changement de mode,  non prise en compte du caracrère (@)
      chgt = false : pas de changement ou caractère @ pris en compte
   (chgt<>mode = "newmode") *)
```

```
(* Détermine la longueur de la chaine finale *)
   let rec premier_passage_inv txt pos mode i =
      if pos >= length txt then i
      else
         let a = txt.[pos] in
            let chgt = pos<>length txt-1 && a='@' && txt.[pos+1]<>'@' in
               let (chaine,longueur) =
                  if mode <> is_int_char a || a='@' then
                        if not chgt then if a<>'@' then (make 1 a,0)
                                                   else ("@",1)
                        else ("",0)
                  else
                     match compteur txt pos pos mode with (nombre,long)->
                        let long2 = long + (if txt.[pos+long]='@' then 1 else 0) in
                           let b = txt.[pos+long2] in
                              (make nombre b,long2)
               in
                  premier_passage_inv txt (pos+longueur+1) (chgt<>mode) (i+length
chaine);;


(* Décompresse effectivement *)
let rec rle_inv_opti_aux txt pos mode rep i =
   if pos >= length txt then rep
   else
      let a = txt.[pos] in
         let chgt = pos<>length txt-1 && a='@' && txt.[pos+1]<>'@' in
            let (chaine,longueur) =
               if mode <> is_int_char a || a='@' then
                     if not chgt then if a<>'@' then (make 1 a,0)
                                                else ("@",1)
                     else ("",0)
               else
                  match compteur txt pos pos mode with (nombre,long)->
                     let long2 = long + (if txt.[pos+long]='@' then 1 else 0) in
                        let b = txt.[pos+long2] in
                           (make nombre b,long2)
            in

         place chaine rep i;
         rle_inv_opti_aux txt (pos+longueur+1) (chgt<>mode) rep (i+length chaine);;

(* Décompression de RLE dans une chaine "vide" de taille bien choisie *)
let rle_inv_opti txt =
   let n = premier_passage_inv txt 0 true 0 in
      rle_inv_opti_aux txt 0 true (make n ' ') 0;;




(* BW avec EOF fictif *)
   let rec compare_bien_eoff_aux s i1 i2 s_length =
      if i1 = s_length || i2 = s_length then
         i1-i2 (* On a un EOF fictif : quand on arrive en fin de chaine, on est plus
grand que tous les autres, on ne trie donc que les suffixes. *)
      else
         let s1=s.[i1] and s2=s.[i2] in
            if s1=s2 then
               compare_bien_eoff_aux s (i1+1) (i2+1) s_length
            else if s1<s2 then -1
            else (*if s1>s2 then*) 1;;

   let compare_bien_eoff s s_length i1 i2 = compare_bien_eoff_aux s i1 i2 s_length;;
```

```
   (* Recolle les caractères formant la chaine de rendu, on prend donc les nombres qui
précèdent ceux triés. *)
(*
   s : chaîne en question
   n : modulo (longueur s)
   l : liste d'entiers représentant les permutations
   d : longueur de la chaîne à joindre au début de rep
   rep : chaîne de traitement et sortie
   i : indice de placement dans rep
*)
   (* À la place d'insérer le dernier caractère, c'est-à-dire qu'on a la chaine qui
devrait se terminer par l'EOF, on saute cette étape, car on est censé ajouter l'EOF
invisible et on met ce caractère tout à la fin ensuite, car il correspond à la chaine
qui commençait pas l'EOF *)



   let rec fusion_caractere_eoff_aux s n l rep i =
      match l with
      |[] -> rep.[i]<-s.[String.length s -1]; rep
      |h::t ->
         let j = (h-1) // n in
         rep.[i] <- s.[j];
         (* Si on tombe sur l'eoff, on ne rajoute rien, donc on fait +1-1 pour revenir
à l'endroit de l'écriture *)
         fusion_caractere_eoff_aux s n t rep (i+1+if j = String.length s -1 then -1
else 0);;

   let fusion_caractere_eoff s n l d =
               fusion_caractere_eoff_aux s n l (make (d+List.length l) ' ') d;;


   (* Sors la liste des rotations de la chaîne, avec la chaîne de base. *)
   let switch_all_eoff s =
      let s_length = length s in
      List.fast_sort (compare_bien_eoff s s_length) (range (s_length -1));;

   let bw_eoff s =
      if s = "" then
         ""
      else
         let permut_triee_list = switch_all_eoff s in
         let n = length s in

         let debut = (string_of_int (pos_depart permut_triee_list))  in
         let d = length debut +1 in

         let chaine = fusion_caractere_eoff s n permut_triee_list d in
         for i=0 to d-2 do
            chaine.[i] <- debut.[i]
         done;

         let first_char = chaine.[d] in
         chaine.[d-1]<-if is_int_char first_char then 'a' else first_char;

         chaine;;

   (* on rajoute fictivement l'EOF à la position debut en incrémentant de 1 tous ceux
qui sont après *)
   let tboftxt_eoff txt debut =
      let rep= Array.make (length txt) ('a',0) in
         for i=length txt -1 downto 0 do
            rep.(i)<-(txt.[i],i+if i>=debut then 1 else 0)
         done;
         rep;;
```

```
   let bw_eoff_inv txt =
      if txt = "" then
         ""
      else
         let longpos = longueur_chiffres txt in
         let longtexte = length txt -longpos-1 in
         let debut = int_of_string (sub txt 0 longpos) -1 in
         let origin = tboftxt_eoff (sub txt (longpos+1) longtexte) debut in
            Array.fast_sort compare_valeurs origin;
            ajoute_opti origin debut;;


let switch_all_radix s = tri_radix_mm s;;
let bw_radix s =
      if s = "" then
         ""
      else
         let permut_triee_list = switch_all_radix s in
         let n = length s in
         let first_char = s.[(List.hd (permut_triee_list) -1) // n] in


         let debut = (string_of_int (pos_depart permut_triee_list)) ^
            (if is_int_char first_char then "a" else string_of_char first_char) in
         let d = length debut in

         let chaine = fusion_caractere_opti s n permut_triee_list d in
         for i=0 to d-1 do
            chaine.[i] <- debut.[i]
         done;

         chaine;;

let switch_all_eoff_radix s = tri_radix_mm_eoff s ;;
let bw_eoff_radix s =
      if s = "" then
         ""
      else
         let permut_triee_list = switch_all_eoff_radix s in
         let n = length s in

         let debut = (string_of_int (pos_depart permut_triee_list))  in
         let d = length debut +1 in

         let chaine = fusion_caractere_eoff s n permut_triee_list d in
         for i=0 to d-2 do
            chaine.[i] <- debut.[i]
         done;

         let first_char = chaine.[d] in
         chaine.[d-1]<-if is_int_char first_char then 'a' else first_char;

         chaine;;

(* Inventaire des fonctions utiles *)
   let bw=bw_opti;;
   let bw_inv=bw_inv_opti;;
   let rle=rle_opti;;
   let rle_inv=rle_inv_opti;;

(* Vérification de la bijectivité *)
   let bij_bw s = bw_inv (bw s) = s;;
   let bij_bw_eoff s = bw_eoff_inv (bw_eoff s) = s;;
   let bij_bw_eoff_radix s = bw_inv (bw_eoff_radix s) = s;;
      let bij_rle s = rle_inv (rle s) = s;;
```

LZ77_2.ml

```
(*********************************************)
(*                    LZ77                   *)
(*********************************************)
  (* Par Guillaume Ménard & Alexandre Blanché *)


let string_of_char c = String.make 1 c;;
let string_of_int_perso c = string_of_char (char_of_int c);;

(* Comme sub, mais si on sort de la zone de recherche, ça marche. *)
let sub_perso texte pos long =
   let n=String.length texte in
      if pos>=n then
         ""
      else
         String.sub texte (max pos 0) (min long (n-pos));;

(* Vérifie que le posRième caractère de requete est le même que le posCième de cible,
et continue de vérifier si c'est le cas. *)
let rec verif_identique_aux requete posR cible posC =
   if posR>=String.length requete then
      true (* Si on a déjà parcouru tout requete, c'est qu'il était dedans *)
   else if posC>=String.length cible then
      false (* Si on a déjà parcouru tout cible et qu'il reste requete, c'est qu'il
n'était pas dedans *)
   else
      (requete.[posR]=cible.[posC]) && verif_identique_aux requete (posR+1) cible
(posC+1);;

(* Vérifie que la chaine requete se trouve bien en posCième position de cible *)
let verif_identique requete cible posC = verif_identique_aux requete 0 cible posC;;

(* On se fixe une position maximale (pos) et minimale (pos_min), et tant qu'on a pas
requete "identique" à cible à la position pos, on le diminue jusqu'à être à min *)
let rec chercheFin requete cible pos pos_min =
   if pos<pos_min then
      -1 (* Renvoie -1 si on n'a pas trouvé *)
   else if verif_identique requete cible pos then
      pos (* Renvoie la position à laquelle on a trouvé requete dans cible *)
   else
      chercheFin requete cible (pos-1) pos_min;;

(* Donne la position de la plus grande sous-chaine (début conservé) de requete dans
cible entre pos et pos_min ainsi que sa longueur pour que cette chaine soit entièrement
entre pos_min et pos *)
let rec cherchepluslong requete cible pos pos_min =
   if requete="" then
      (-1,0)
   else
      match chercheFin requete cible (pos-String.length requete) pos_min with (* On ne
recherche que dans la zone autorisée *)
      | -1 -> cherchepluslong (sub_perso requete 0 (String.length requete -1)) cible
pos pos_min
      |a -> (a,String.length requete);;

(* cherchepluslongter "abc" "bcabc" 2 0;;*)
```

```
(* Fait LZ77, en mettant 0C si C n'est pas trouvé dans le buffer, et PL la position et
la longueur de la chaine trouvée dans le buffer *)
let rec lz77_aux texte buffer fenetre pos =
   if pos>=String.length texte then
      ""
   else
      match chercheplus long (sub_perso texte pos fenetre) texte pos (max 0 (pos -
buffer)) with
      |(_,0)->(string_of_int_perso 0)^(sub_perso texte pos 1)^(lz77_aux texte buffer
fenetre (pos+1)) (* Si aucune sous-chaine de la fenêtre n'est trouvée, il renvoie 00C
et augmente pos de 1 *)
      |(a,b)->(string_of_int_perso (pos-a))^(string_of_int_perso b)^(lz77_aux texte
buffer fenetre (pos+b));; (* indique la distance à gauche du caractère trouvé, sa
longueur et augmente pos de la longueur trouvée  *)

(* Choix du buffer de 255 et d'une fenêtre de 31 ; On conserve les 255 premiers
caractères, on encode à partir de la suite *)
let lz77_encode texte = (sub_perso texte 0 255)^(lz77_aux texte 255 255 255);;
let lz77_encode texte = (sub_perso texte 0 255)^(lz77_aux texte 255 31 255);;
let lz77_encode texte = (sub_perso texte 0 63)^(lz77_aux texte 255 31 63);;


(* Décode : 0C -> C, PL -> L caractères à partir du Pième en partant de la fin de
debut*)
let rec lz77_decode_aux debut texte pos =
   if pos>=String.length texte then
      debut (* Quand on a fini, on renvoit ce qui est décodé. *)
   else if texte.[pos]=(char_of_int 0) then
      lz77_decode_aux (debut^(string_of_char texte.[pos+1])) texte (pos+2) (*Quand on a
deux zéros successifs, on affiche le caractère suivant, et on avance de 3 *)
   else
      lz77_decode_aux (debut^(sub_perso debut (String.length debut -(int_of_char
texte.[pos])) (int_of_char texte.[pos+1]))) texte (pos+2) (* On ajoute les "L"
caractères suivant la position du "P"ième en partant de la fin et on avance de 2 *);;

(* Conserve le début (buffer) et transforme le reste *)
let lz77_decode texte =
   lz77_decode_aux (sub_perso texte 0 255) texte 255;;


let lz77_test_aux texte = lz77_decode (lz77_encode texte);;
Random.self_init();;
Random.int 20;;
let cara_alea ()=string_of_int_perso (Random.int 255);;
let rec texte_alea_aux n rep =
   if n=0 then rep
   else texte_alea_aux (n-1) ((cara_alea ())^rep);;
let texte_alea () =  texte_alea_aux (Random.int 5000) "";;

let rec lz77_test nb =
   let texte=(texte_alea ()) in
      if nb=0 then ""
      else if lz77_test_aux texte <> texte then texte
      else lz77_test (nb-1);;

(* Renvoie les différentes longueurs *)
let lz77_compare texte =(String.length texte,String.length (lz77_encode texte));;

let lz77_encode_var texte lgth = (sub_perso texte 0 255)^(lz77_aux texte 255 lgth
255);;
let lz77_compare_var texte lgth =(String.length texte,String.length (lz77_encode_var
texte lgth));;
```

MtF.ml

```
(* Renvoie la liste des caractères de 0 à n *)
   let rec range_char_aux n l =
      if n=0 then (char_of_int 0)::l
      else range_char_aux (n-1) ((char_of_int n)::l);;
   let range_char n = range_char_aux n [];;

(* Cherche le caractère a dans l, le met au début, en indiquant sa position. *)
   let rec met_en_tete_et_position_aux l a =
      match l with
      |[]->failwith "met_en_tete_et_position_aux"
      |h::t ->
         if a = h then
            (t,0)
         else
            let (l2,pos) = met_en_tete_et_position_aux t a in
            (h::l2,pos+1);;
   let met_en_tete_et_position l a =
      let (l2,pos) = met_en_tete_et_position_aux l a in
      (a::l2,pos);;

(* Écrit dans rep les différentes positions des caractères à partir du nième, comme
dans un move to front *)
   let rec mtf_aux texte l n rep =
      if n<String.length texte then
         let (l2,pos) = met_en_tete_et_position l texte.[n] in
         rep.[n]<-char_of_int pos;
         mtf_aux texte l2 (n+1) rep
      else
         rep;;

   let mtf texte =
      mtf_aux texte (range_char 255) 0 texte;;


(* Cherche le nieme caractère dans l, le met au début, en indiquant lequel c'est. *)
   let rec met_en_tete_et_caractere_aux l n =
      match l with
      |[]->failwith "met_en_tete_et_caractere_aux"
      |h::t ->
         if n = 0 then
            (t,h)
         else
            let (l2,car) = met_en_tete_et_caractere_aux t (n-1) in
            (h::l2,car);;
   let met_en_tete_et_caractere l n =
      let (l2,car) = met_en_tete_et_caractere_aux l n in
      (car::l2,car);;

(* Fait l'inverse de move_to_front : n est la position (caractère dans chaine) regardée
*)
   let rec mtf_inv_aux texte l n rep =
      if n<String.length texte then
         let (l2,pos) = met_en_tete_et_caractere l (int_of_char texte.[n]) in
         rep.[n]<- pos;
         mtf_inv_aux texte l2 (n+1) rep
      else
         rep;;
   let mtf_inv texte =
            mtf_inv_aux texte (range_char 255) 0 texte;;
```

mlbmpperso.ml

```
(*----------------------------------------------------------------------*)
(* Lecture/écriture d'images BMP 24 bits *)

(* Marc Lorenzi *)
(* 20/09/2008 *)
(*----------------------------------------------------------------------*)
#load "graphics.cma";;
open Graphics;;

(*----------------------------------------------------------------------*)
(* word = 2 bytes, dword = 4 bytes *)
(* Aucune différence en Caml *)

type word  = int;;
type dword = int;;

(*----------------------------------------------------------------------*)
(* En-tête de fichier BMP : Bitmap File Header *)

type bitmapFileHeader = {
  bfType      : string;
  bfSize      : dword;
  bfReserved1 : word;
  bfReserved2 : word;
  bfOffBits   : dword;
};;

(*----------------------------------------------------------------------*)
(* En-tête de fichier BMP : Bitmap Info Header *)

type bitmapInfoHeader = {
  biSize          : dword;
  biWidth         : dword;
  biHeight        : dword;
  biPlanes        : word;
  biBitCount      : word;
  biCompression   : dword;
  biSizeImage     : dword;
  biXPelsPerMeter : dword;
  biYPelsPerMeter : dword;
  biClrUsed       : dword;
  biClrImportant  : dword;
};;

(*----------------------------------------------------------------------*)
(* Lecture/écriture des deux premiers octets du fichier (normalement "BM") *)

let read_type channel =
  let s = "  " in
  s.[0] <- input_char channel;
  s.[1] <- input_char channel;
  s;;

let write_type channel =
  output_char channel 'B';
  output_char channel 'M';;

(*----------------------------------------------------------------------*)
(* Lecture/Ecriture d'un Word et d'un DWord.*)
(* Dans un fichier BMP, les octets les moins significatifs dont stockés
en premier (little endian) *)
```

```
let read_dword channel =
  let a = input_byte channel in
  let b = input_byte channel in
  let c = input_byte channel in
  let d = input_byte channel in
  (d lsl 24) lor (c lsl 16) lor (b lsl 8) lor a;;

let write_dword channel x =
  let a = x lsr 24
  and b = (x lsr 16) land 255
  and c = (x lsr 8) land 255
  and d = x land 255 in
  output_byte channel d;
  output_byte channel c;
  output_byte channel b;
  output_byte channel a;;

let read_word channel =
  let a = input_byte channel in
  let b = input_byte channel in
  (b lsl 8) lor a;;

let write_word channel x =
  let c = (x lsr 8) land 255
  and d = x land 255 in
  output_byte channel d;
  output_byte channel c;;

(*----------------------------------------------------------------------*)
(* Lecture/Ecriture du BitmapFileHeader *)

let read_file_header channel =
  let t = read_type channel in
  let sz = read_dword channel in
  let r1 = read_word channel in
  let r2 = read_word channel in
  let off = read_dword channel in
  {
    bfType = t;
    bfSize = sz;
    bfReserved1 = r1;
    bfReserved2 = r2;
    bfOffBits = off;
  };;

let write_file_header channel fh =
  write_type channel;
  write_dword channel fh.bfSize;
  write_word channel fh.bfReserved1;
  write_word channel fh.bfReserved2;
  write_dword channel fh.bfOffBits;;


(*----------------------------------------------------------------------*)
(* Lecture/Ecriture du BitmapInfoHeader *)

let read_info_header channel =
  let sz = read_dword channel in
  let w = read_dword channel in
  let h = read_dword channel in
  let pl = read_word channel in
  let bc = read_word channel in
  let compr = read_dword channel in
  let szim = read_dword channel in
  let xpm = read_dword channel in
  let ypm = read_dword channel in
  let clru = read_dword channel in
```

```
  let clri = read_dword channel in
  {
    biSize = sz;
    biWidth = w;
    biHeight = h;
    biPlanes = pl;
    biBitCount = bc;
    biCompression = compr;
    biSizeImage = szim;
    biXPelsPerMeter= xpm;
    biYPelsPerMeter= ypm;
    biClrUsed = clru;
    biClrImportant = clri;
  };;

let write_info_header channel ih =
  write_dword channel ih.biSize;
  write_dword channel ih.biWidth;
  write_dword channel ih.biHeight;
  write_word channel ih.biPlanes;
  write_word channel ih.biBitCount;
  write_dword channel ih.biCompression;
  write_dword channel ih.biSizeImage;
  write_dword channel ih.biXPelsPerMeter;
  write_dword channel ih.biYPelsPerMeter;
  write_dword channel ih.biClrUsed;
  write_dword channel ih.biClrImportant;;

(*----------------------------------------------------------------------*)
(* Lecture des pixels. Renvoie une matrice m de dimensions w x h où
(w, h) est la taille de l'image. m.(i).(j) contient le triplet (r,g,b)
des composantes couleurs du pixel situé àux coordonnées (j, i) de l'image.
La coordonnée 0,0 est en général le coin inférieur gauche, à moins que le
champ biHeight du bitmapInfoHeader soit une valeur négative. Ce cas n'est
pas traité ici.
Unique subtilité : le nombre d'octets dans une "ligne" du fichier image
doit être un multiple de 4. Si le nombre réel de pixels ne vérifie pas cette
condition, on complète par des zéros.*)

(* Multiple de 4 immédiatement supérieur ou égal à w *)
let offset w =
  let r = (3 * w) mod 4 in
  if r = 0 then 0
  else 4 - r;;

(* val read_pixels : bitmapFileHeader -> bimapInfoHeader -> in_channel -> unit *)

type pixel = int * int * int;;
type pixel_matrix = pixel array array;;

let read_pixels fh ih channel =
  let w = ih.biWidth
  and h = ih.biHeight in
  let offs = offset w in
  let m = Array.make_matrix w h (0,0,0) in
  for j = 0 to h - 1 do
    for i = 0 to w - 1 do
      let b = input_byte channel in
      let g = input_byte channel in
      let r = input_byte channel in
      m.(i).(j) <- (r, g, b)
    done;
    for i = 1 to offs do
        let _ = input_byte channel in ()
    done
  done;
  (m:pixel_matrix);;
```

```
(* val write_pixels : out_channel -> (int*int*int) array array -> unit *)

let write_pixels channel (m:pixel_matrix) =
  let w = Array.length m
  and h = Array.length m.(0) in
  let offs = offset w in
  for j = 0 to h - 1 do
    for i = 0 to w - 1 do
      let r, g, b = m.(i).(j) in
      output_byte channel b;
      output_byte channel g;
      output_byte channel r;
    done;
    for i = 1 to offs do
        output_byte channel 0
    done
  done;;

(*----------------------------------------------------------------------*)
(* Lecture d'une image BMP *)

(* val read_bmp : string -> (int*int*int) array array *)

let read_bmp filename =
  let channel = open_in_bin filename in
  let fh = read_file_header channel in
  let ih = read_info_header channel in
  let m = read_pixels fh ih channel in
  close_in channel;
  m;;

(*----------------------------------------------------------------------*)
(* Affichage sur la sortie standard des en-têtes d'une image *)

let print_data s x =
  print_string s;
  print_int x;
  print_string "\n";;

let print_headers filename =
  let channel = open_in_bin filename in
  let fh = read_file_header channel in
  let ih = read_info_header channel in
  close_in channel;
  print_string "BitmapFileHeader...\n";
  print_string("=> bfType          : " ^ fh.bfType ^ "\n");
  print_data "=> bfSize          : " fh.bfSize;
  print_data "=> bfReserved1     : " fh.bfReserved1;
  print_data "=> bfReserved2     : " fh.bfReserved2;
  print_data "=> bfOffBits       : " fh.bfOffBits;
  print_newline();
  print_string  "bitmapInfoHeader...\n";
  print_data "=> biSize          : " ih.biSize;
  print_data "=> biWidth         : " ih.biWidth;
  print_data "=> biHeight        : " ih.biHeight;
  print_data "=> biPlanes        : " ih.biPlanes;
  print_data "=> biBitCount      : " ih.biBitCount;
  print_data "=> biCompression   : " ih.biCompression;
  print_data "=> biSizeImage     : " ih.biSizeImage;
  print_data "=> biXPelsPerMeter : " ih.biXPelsPerMeter;
  print_data "=> biYPelsPerMeter : " ih.biYPelsPerMeter;
  print_data "=> biClrUsed       : " ih.biClrUsed;
  print_data "=> biClrImportant  : " ih.biClrImportant;
  print_newline();;

(* Création des en-têtes de fichier BMP pour l'écriture *)
```

```
let make_file_header w h =
  let off = offset w in
  {
    bfType = "BM";
    bfSize = (w + off) * h * 3 + 54;
    bfReserved1  = 0;
    bfReserved2 = 0;
    bfOffBits = 54;
  };;

let make_info_header w h =
  let off = offset w in
  {
    biSize = 40;
    biWidth = w;
    biHeight  = h;
    biPlanes  = 1;
    biBitCount = 24;
    biCompression = 0;
    biSizeImage = (w + off) * h * 3;
    biXPelsPerMeter = 0;
    biYPelsPerMeter = 0;
    biClrUsed = 0;
    biClrImportant = 0;
  };;

(* val write_bmp : string -> (int*int*int) array array -> unit *)

let write_bmp filename m =
  let channel = open_out_bin filename in
  let w = Array.length m
  and h = Array.length m.(0) in
  let fh = make_file_header w h
  and ih = make_info_header w h in
  write_file_header channel fh;
  write_info_header channel ih;
  write_pixels channel m;
  close_out channel;;

(*----------------------------------------------------------------------*)
(* Afficher l'image dans une fenêtre graphique *)
(* Un appui sur une touche ferme la fenêtre *)
(* !! Sous Windows, le fait de cliquer sur la case de fermeture de
la fenêtre graphique N'ARRETE PAS LE PROGRAMME. Ennuis en perspective,
donc. Conclusion : appuyer sur une touche pour sortir proprement du
programme. *)

let show_me (m:pixel_matrix) =
  let w = Array.length m
  and h = Array.length m.(0) in
  let s = " " ^ (string_of_int (w + 20)) ^ "x" ^
          (string_of_int (h + 45)) ^ "+0+0" in

  open_graph s;
  let window_title = "Pixel Matrix Viewer (" ^
                     (string_of_int w) ^ "x" ^
                     (string_of_int h) ^ ")" in
  set_window_title (window_title);
  (*clear_graph();
  auto_synchronize false;*)
  for j = 0 to h - 1 do
    for i = 0 to w - 1 do
      let (r, g, b) = m.(i).(j) in
        set_color (rgb r g b);
        plot (i + 5) (j + 5)
    done
  done;
```

```
  (*synchronize();*)
  let _ = read_key() in close_graph();;

(*----------------------------------------------------------------------*)
(* Petits utilitaires *)

(* Dimensions d'une matrice de pixels *)

let dimensions (m:pixel_matrix) =
  (Array.length m, Array.length m.(0));;

(* Créer une matrice de pixels de taille w x h *)

let create_pixel_matrix w h =
  let m = Array.create_matrix w h (0, 0, 0) in
  (m:pixel_matrix);;

(* Copier une matrice de pixels *)

let copy_pixel_matrix (m:pixel_matrix) =
  let (w, h) = dimensions m in
  let m1 = create_pixel_matrix w h in
  for i = 0 to w - 1 do
    for j = 0 to h - 1 do
      m1.(i).(j) <- m.(i).(j)
    done
  done;
  (m1:pixel_matrix);;

(* Négatif d'une matrice de pixels *)

let negative m =
  let (w, h) = dimensions m in
  let m1 = create_pixel_matrix w h in
  for i = 0 to w - 1 do
    for j = 0 to h - 1 do
      let (r, g, b) = m.(i).(j) in
      m1.(i).(j) <- (255 - r, 255 - g, 255 - b)
    done
  done;
  m1;;

(* Conversion en niveaux de gris *)

let gray_levels m =
  let (w, h) = dimensions m in
  let m1 = create_pixel_matrix w h in
  for i = 0 to w - 1 do
    for j = 0 to h - 1 do
      let (r, g, b) = m.(i).(j) in
      let c = (r + g + b) / 3 in
      m1.(i).(j) <- (c, c, c)
    done
  done;
  m1;;

(* Symétrie gauche <-> droite *)
let mirror_left_right m =
  let (w, h) = dimensions m in
  let m1 = create_pixel_matrix w h in
  for j = 0 to h - 1 do
    for i = 0 to (w - 1) / 2 do
      m1.(i).(j) <- m.(w - i - 1).(j);
      m1.(w - i - 1).(j) <- m.(i).(j)
    done
  done;
  m1;;
```

```
(* Symétrie haut <-> bas *)

let mirror_up_down m =
  let (w, h) = dimensions m in
  let m1 = create_pixel_matrix w h in
  for i = 0 to w - 1 do
    for j = 0 to (h - 1) / 2 do
      m1.(i).(j) <- m.(i).(h - j - 1);
      m1.(i).(h - j - 1) <- m.(i).(j)
    done
  done;
  m1;;

(*----------------------------------------------------------------------*)
(* Tests *)

let run_tests() =
  let image = "ara.bmp" in
  print_headers image;

  let m = read_bmp image in
  show_me m;

  let m1 = negative m in
  write_bmp "__img1__.bmp" m1;
  show_me m1;

  let m1 = mirror_left_right m in
  write_bmp "__img2__.bmp" m1;
  show_me m1;

  let m1 = gray_levels m in
  write_bmp "__img4__.bmp" m1;
  show_me m1;

  let m1 = negative (gray_levels m) in
  write_bmp "__img5__.bmp" m1;
  show_me m1;

  show_me m;;

(*run_tests();;*)
```

Compression_bmp_5.ml

```
(************************************************************************)
(*                     Compression de fichiers bmp                      *)
(************************************************************************)
              (* Par Guillaume Ménard & Alexandre Blanché *)

                              (* Version 5 *)

(* mlbmpperso.ml : programme créé par Marc Lorenzi, lecture et écriture de fichiers
.bmp *)
(*#use "C:/Users/antoine/TD Caml/TIPE/mlbmpperso.ml";;
#use "C:/Users/antoine/TD Caml/TIPE/Burrows-Wheeler_22.ml";;*)


#use "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/mlbmpperso.ml";;
#use "C:/Users/antoine/TD Caml/TIPE/mlbmpperso.ml";;
#use "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/Burrows-Wheeler_30.ml";;
#use "C:/Users/antoine/TD Caml/TIPE/Burrows-Wheeler_30.ml";;
#use "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/huffman_perso3.ml";;
#use "C:/Users/antoine/TD Caml/TIPE/huffman_perso3.ml";;
#use "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/MtF.ml";;
#use "C:/Users/antoine/TD Caml/TIPE/MtF.ml";;
open String;;



(* Compression du fichier *)


let compr_bmp filename =
   let t = read_bmp filename in
   let (longueur,largeur) = (Array.length t.(0), Array.length t) in
   let n = longueur * largeur in
      (* Insertion des caractéristiques du fichier:
         "(longueur)&(le reste)" *)
   let nbchiffres a = int_of_float (log (float_of_int a )/.log 10.)+1 in
   let b = nbchiffres longueur in
   let s = make (b+1+3*n) ' ' in

   let ch_long = string_of_int longueur in


   (* Ajoute la longueur *)
   for i=0 to b-1 do
      s.[i] <- ch_long.[i]
   done;
   s.[b] <- '&';

   (* Insertion des caractères de chaque pixel dans la chaîne *)
   let h = b+1 in

   for i=0 to largeur-1 do
      for j=0 to longueur-1 do
         let k = longueur * i + j and (r,g,b) = t.(i).(j) in
         s.[h+k] <- char_of_int r;
         s.[h+k+n] <- char_of_int g;
         s.[h+k+2*n] <- char_of_int b
      done;
   done;

   let rep = rle (bw s) in
   let oc = open_out_bin (filename^".AG") in
  output_value oc rep;
  close_out oc;

   rep;;
```

```
(* On ne rajoute pas un @ mais un &, on espère qu'il n'y ait pas de @, il n'est pas
question d'en rajouter sciemment ! *)

(* Lit dans un fichier écrit avec output_value *)
let lire ic =
let obj = ref "" in
try
   while true do
   obj := !obj^(input_value ic);
   done ;!obj
with _ -> close_in_noerr ic ;!obj;;


(* Décompression *)


let decompr_bmp filename =
   (* On met la chaine dans s *)
   let f = open_in_bin filename in let s=lire f in close_in f;

   let chaine = bw_inv (rle_inv s) in
   (* recherche de la longueur *)
   let j = ref 0 in
   while chaine.[(!j)]<>'&' do
      j:=!j+1
   done;

   let longueur = int_of_string (sub chaine 0 !j) in

   let n = ((length chaine) - !j - 1)/3 in
      (* tombe juste! *)
   let m = Array.make_matrix (n/longueur) longueur (0,0,0) in

(* Remplissage *)

   let k = (!j)+1 in

   for x=0 to (n/longueur)-1 do
   for y=0 to longueur-1 do

      let g = x*longueur + y in
      m.(x).(y) <- (int_of_char chaine.[k+g] ,
                    int_of_char chaine.[k+n+g],
                    int_of_char chaine.[k+2*n+g])

   done;
   done;
   (*let lg = length filename in
   if lg>3 && sub filename (lg-3) 3=".AG" then
      write_bmp (sub filename 0 (lg-3)) m
   else*)
      write_bmp (filename^".bmp") m;;
```

```
(* Compression avec Huffman *)
let compr_bmp_huff filename =
   let t = read_bmp filename in
   let (longueur,largeur) = (Array.length t.(0), Array.length t) in
   let n = longueur * largeur in
      (* Insertion des caractéristiques du fichier:
         "(longueur)&(le reste)" *)
   let nbchiffres a = int_of_float (log (float_of_int a )/.log 10.)+1 in
   let b = nbchiffres longueur in
   let s = make (b+1+3*n) ' ' in

   let ch_long = string_of_int longueur in

   (* Ajoute la longueur *)
   for i=0 to b-1 do
      s.[i] <- ch_long.[i]
   done;
   s.[b] <- '&';

   (* Insertion des caractères de chaque pixel dans la chaîne *)
   let h = b+1 in

   for i=0 to largeur-1 do
      for j=0 to longueur-1 do
         let k = longueur * i + j and (r,g,b) = t.(i).(j) in
         s.[h+k] <- char_of_int r;
         s.[h+k+n] <- char_of_int g;
         s.[h+k+2*n] <- char_of_int b
      done;
   done;

   let rep = codage s in
   let oc = open_out_bin (filename^".huf") in
  output_value oc rep;
  close_out oc;;
```

CompressionGenerale.ml

```
#load "unix.cma" ;;
open String;;
#use "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/Compression_bmp_5.ml";;
#use "C:/Users/antoine/TD Caml/TIPE/Compression_bmp_5.ml";;

let repertoire = "C:/Users/antoine/TD Caml/TIPE/FichiersTest/";;
let repertoire = "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/mlbmp/";;

(* Lit dans un fichier écrit avec output_value *)
let lire ic =
let obj = ref "" in
try
   while true do
   obj := !obj^(input_value ic);
   done ;!obj
with _ -> close_in_noerr ic ;!obj;;

(* Applique la fonction "fonction" au fichier repertoire^fichier, l'enregistre en lui
rajoutant extension, et précise dans le fichier_de_sauvegarde le temps qu'il a pris *)
let compression fonction repertoi fichier extension fichier_de_sauvegarde =
   (* On ouvre le fichier source *)
   let f = open_in_bin (repertoi^fichier) in let s=lire f in close_in f;

   (* On compresse*)
   let t = Unix.time () in let rep = fonction s in let d = Unix.time () -. t in

   (* On enregistre le fichier *)
   let oc = open_out_bin (repertoi^fichier^extension) in output_value oc rep; close_out
oc;

   (* On enregistre le temps mis *)
   let oc2 = open_out_gen [Open_append;Open_creat] 222 fichier_de_sauvegarde in
output_string oc2 (fichier^" -> "^fichier^extension^" : "^(string_of_float d)^"s ;
"^(string_of_int (String.length rep))^"octets\n"); close_out oc2;;

let compression_invisible fonction repertoi fichier extension fichier_de_sauvegarde =
   (* On ouvre le fichier source *)
   let f = open_in_bin (repertoi^fichier) in let s=lire f in close_in f;

   (* On compresse *)
   let t = Unix.time () in let rep = fonction s in let d = Unix.time () -. t in

   (* On n'enregistre pas le fichier *)


   (* On enregistre le temps mis *)
   let oc2 = open_out_gen [Open_append;Open_creat] 222 fichier_de_sauvegarde in
output_string oc2 (fichier^" -> "^fichier^extension^" : "^(string_of_float d)^"s ;
"^(string_of_int (String.length rep))^"octets\n"); close_out oc2;;

(* Transforme un string en output_value *)
let string_to_output_value s repertoire_et_fichier_output =
   let oc = open_out_bin repertoire_et_fichier_output in output_value oc s; close_out
oc;;

let fichier_to_output_value repertoire_et_fichier_fichier =
   let f = open_in repertoire_et_fichier_fichier in
   let s = input_line f in
   let oc = open_out_bin (repertoire_et_fichier_fichier^".oc") in
   output_value oc s; close_out oc ;   close_in f;;
```

```
(* Transforme un bmp en output_value en rajoutant la longueur de l'image en début de
chaine *)
let bmp_to_output_value image repertoire_et_fichier_output =
   let t = read_bmp image in
   let (longueur,largeur) = (Array.length t.(0), Array.length t) in
   let n = longueur * largeur in
      (* Insertion des caractéristiques du fichier:
         "(longueur)&(le reste)" *)
   let nbchiffres a = int_of_float (log (float_of_int a )/.log 10.)+1 in
   let b = nbchiffres longueur in
   let s = String.make (b+1+3*n) ' ' in

   let ch_long = string_of_int longueur in

   (* Ajoute la longueur *)
   for i=0 to b-1 do
      s.[i] <- ch_long.[i]
   done;
   s.[b] <- '&';

   (* Insertion des caractères de chaque pixel dans la chaîne *)
   let h = b+1 in

   for i=0 to largeur-1 do
      for j=0 to longueur-1 do
         let k = longueur * i + j and (r,g,b) = t.(i).(j) in
         s.[h+k] <- char_of_int r;
         s.[h+k+n] <- char_of_int g;
         s.[h+k+2*n] <- char_of_int b
      done;
   done;
   let oc = open_out_bin repertoire_et_fichier_output in output_value oc s; close_out
oc;;

let input_value_to_bmp repertoire_et_fichier_input image =
   let f = open_in_bin repertoire_et_fichier_input in let chaine=lire f in close_in f;

   (* recherche de la longueur *)
   let j = ref 0 in
   while chaine.[(!j)]<>'&' do
      j:=!j+1
   done;

   let longueur = int_of_string (sub chaine 0 !j) in

   let n = ((length chaine) - !j - 1)/3 in
      (* tombe juste! *)
   let m = Array.make_matrix (n/longueur) longueur (0,0,0) in

   (* Remplissage *)

   let k = (!j)+1 in

   for x=0 to (n/longueur)-1 do
   for y=0 to longueur-1 do

      let g = x*longueur + y in
      m.(x).(y) <- (int_of_char chaine.[k+g] ,
                    int_of_char chaine.[k+n+g],
                    int_of_char chaine.[k+2*n+g])

   done;
   done;
   write_bmp image m;;

   let partiel fonction long s = fonction (String.sub s 0 long);;
   let placebo s = s;;
```

fonctions.ml

```
#use "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/CompressionGenerale.ml";;
#use "C:/Users/antoine/TD Caml/TIPE/CompressionGenerale.ml";;
let repertoire = "C:/Users/antoine/TD Caml/TIPE/mlbmp/Test Format_1200/";;
let repertoire = "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/mlbmp/Test
Format_1200/";;
let fichier_de_sauvegarde = repertoire^"log.txt";;

(* Applique les compressions à une liste de (fonction, fichier, extension) *)
let rec compress = function
   |[]->()
   |(fonction,fichier,extension)::t-> compression fonction repertoire fichier extension
fichier_de_sauvegarde;compress t;;
let rec compress_repertoire repertoi = function
   |[]->()
   |(fonction,fichier,extension)::t-> compression fonction repertoi fichier extension
(repertoi^"log.txt");compress_repertoire repertoi t;;
let rec compress_repertoire_invisible repertoi = function
   |[]->()
   |(fonction,fichier,extension)::t-> compression_invisible fonction repertoi fichier
extension (repertoi^"log.txt");compress_repertoire_invisible repertoi t;;

let l_todo_1200_sans_huf =
[(bw,"JPEG_1200.bmp.oc",".bw");...;(rle_inv,"Texte_1200.txt.oc.bw.rle",".rle_inv")];;
let l_todo_1200_huf =
[(huf,"JPEG_1200.bmp.oc",".huf");...;(huf,"Texte_1200.txt.oc.mtf.rle",".huf");(huf,"Tex
te_1200.txt.oc.bw.mtf.rle",".huf")];;

(* Découpe de manière régulière en n un fichier te de longueur long *)
let rec construit_l_todo_partiel te long k n =
   if n<k then
      []
   else
      (partiel bw (long*k/n),te, string_of_int k ^".bw")::(construit_l_todo_partiel te
long (k+1) n);;
let l_todo_partiel_29000=construit_l_todo_partiel "Texte_29000.txt.oc" 31162263 1 100;;
(* 31162263 est la longueur de la chaine, qu'on découpe en 100 *)


(* Test le découpage en ne faisant rien, pour vérifier que c'est instantanné, placebo
est la fonction identité. *)
let rec construit_l_todo_partiel_1200placeb te long k n =
   if n<k then
      []
   else
      (partiel placebo (long*k/n),te, string_of_int k^".pl")::
(construit_l_todo_partiel_1200placeb te long (k+1) n);;
let l_todo_partiel_29000pl=construit_l_todo_partiel_1200placeb "Texte_29000.txt.oc"
31162263 1 100;;

(* Ici, on compresse en concervant bw, puis rle, puis mtf, puis huf (en l'oubliant) du
découpage de Texte_29000 *)
(*BW_eoff en premier *)
let rec construit_29000_bw_eoff te long k n =
   if n<k then
      []
   else
      (partiel bw_eoff  (long*k/n),te, string_of_int k
^".bw_eoff")::(construit_29000_bw_eoff te long (k+1) n);;
compress_repertoire "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/mlbmp/Decoupage/"
(construit_29000_bw_eoff "Texte_29000.txt.oc" 31162263 1 100);;
(*RLE ensuite *)
let rec construit_29000_bw_rle k n =
   if n<k then
      []
```

```
   else
      (rle,"Texte_29000.txt.oc"^(string_of_int k)^".bw_eoff",
".rle")::(construit_29000_bw_rle (k+1) n);;
compress_repertoire "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/mlbmp/Decoupage/"
(construit_29000_bw_rle 1 100);;
(* MTF ensuite *)
let rec construit_29000_bw_rle_mtf k n =
   if n<k then
      []
   else
      (mtf,"Texte_29000.txt.oc"^(string_of_int k)^".bw_eoff.rle",
".mtf")::(construit_29000_bw_rle_mtf (k+1) n);;
compress_repertoire "C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/mlbmp/Decoupage/"
(construit_29000_bw_rle_mtf 1 100);;
(* Huffman ensuite, qu'on oublie *)
let rec construit_29000_bw_rle_mtf_huf k n =
   if n<k then
      []
   else
      (huf,"Texte_29000.txt.oc"^(string_of_int k)^".bw_eoff.rle.mtf",
".huf")::(construit_29000_bw_rle_mtf_huf (k+1) n);;
compress_repertoire_invisible
"C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/mlbmp/Decoupage/"
(construit_29000_bw_rle_mtf_huf 1 100);;


(* Et maintenant, on compose plusieurs fois la série bw rle mtf huf_bis *)
let l_compose =
[(bw_eoff,"Texte_29000.txt.oc",".bw_eoff");(rle,"Texte_29000.txt.oc.bw_eoff",".rle");(m
tf,"Texte_29000.txt.oc.bw_eoff.rle",".mtf");(huf_bis,"Texte_29000.txt.oc.bw_eoff.rle.mt
f",".huf_bis");(bw_eoff,"Texte_29000.txt.oc.bw_eoff.rle.mtf.huf_bis",".bw_eoff");(rle,"
Texte_29000.txt.oc.bw_eoff.rle.mtf.huf_bis.bw_eoff",".rle");...;(huf_bis,"Texte_29000.t
xt.oc.bw_eoff.rle.mtf.huf_bis.bw_eoff.rle.mtf.huf_bis.bw_eoff.rle.mtf.huf_bis.bw_eoff.r
le.mtf",".huf_bis")];;


(* Comme on n'a pas un début de courbe convaincant, on le fait, mais on n'affiche pas
les fichiers qui vont nous embêter, on compose tout d'un coup... *)
let tout s =
   huf (mtf (rle (bw_eoff s)));;

(* max indique la découpe maximale (1=entier, 100=un centième du fichier) du fichier
prise, comme si le fichier ne faisait que long/max au total, ça permet d'avoir le
fichier de départ découper en 10000, plutôt que d'avoir un décalage dû aux entiers...
*)
let rec construit_29000_tout te long k n max =
   if n<k then
      []
   else
      (partiel tout  (long*k/n/max),te, string_of_int k ^"_max_" ^ string_of_int max ^
".tout")::(construit_29000_tout te long (k+1) n max);;

let rec construit_29000_tout_loga te long k pas =
   if long<k then
      []
   else
      (partiel tout  (int_of_float k),te, string_of_int (int_of_float k)
^"_loga(taille).tout")::(construit_29000_tout_loga te long (k*.pas+.1.) pas);;

compress_repertoire_invisible
"C:/Users/Public/guillaume/Lycee/CG/MPE/TIPE/mlbmp/Decoupage/"
(construit_29000_tout_loga "Texte_29000.txt.oc" 3116. 1. (exp((log 3116.)/.100.)));;
```

huffman_perso3.ml

```
type poids = int;;
type caractere = int;;
type arbre = Feuille of caractere * poids | Arbre of arbre * poids * arbre;;

let histogramme s =
   let h = Array.make 256 0 in
      for i=0 to String.length s -1 do
         let pos = int_of_char s.[i] in
         h.(pos) <- h.(pos) + 1
      done; h;;

let poids h =
  match h with
  |Feuille (_, p) -> p
  |Arbre (_, p, _) -> p;;

let (<<) a1 a2 = (poids a1)-(poids a2);;

let faire_arbre h =
   let tb = Array.make 256 (Feuille(0,0)) in
      for i=0 to 255 do
         tb.(i) <- Feuille(i,h.(i))
      done;
      Array.sort (<<) tb;
      let j = ref 0 in
         while poids tb.(!j) = 0 do
            j:= !j+1;
         done;
         for k= !j to 254 do
            let pd=poids tb.(k) + poids tb.(k+1) in
               tb.(k+1)<-Arbre(tb.(k), pd, tb.(k+1));
               let i= ref (k+1) in
                  while !i<255 && pd > poids tb.(!i+1) do
                     let tmp = tb.(!i) in
                        tb.(!i) <- tb.(!i + 1);
                        tb.(!i+1) <- tmp;
                        i:= !i+1;
                  done;
         done; tb.(255);;

let rec creer_codes t tbcodes q =
   match t with
   |Feuille (x, _) -> tbcodes.(x)<-q
  |Arbre (t1, _, t2) -> creer_codes t1 tbcodes (q^"0");creer_codes t2 tbcodes (q^"1");;

let codage_complexite_n2 s =
   let tbcodes = Array.make 256 "" in
      creer_codes (faire_arbre (histogramme s)) tbcodes "";
      let rep = ref "" in
         for i = 0 to String.length s - 1 do
            rep := !rep ^ tbcodes.(int_of_char s.[i]);
         done;
         !rep;;

let premiere_passe s tbcodes =
   let lgth=ref 0 in
      for i = 0 to String.length s - 1 do
         lgth := !lgth + (String.length tbcodes.(int_of_char s.[i]));
      done;
      !lgth;;
```

```
let codage s =
   let tbcodes = Array.make 256 "" in
      creer_codes (faire_arbre (histogramme s)) tbcodes "";
      let rep = String.make (premiere_passe s tbcodes) ' ' and j=ref 0 in
         for i = 0 to String.length s - 1 do
            let ajout=tbcodes.(int_of_char s.[i]) in
            let lg_ajout = String.length ajout in
            for k=0 to lg_ajout-1 do
               rep.[!j+k] <- ajout.[k];
            done;
            j:=!j+lg_ajout
         done;
         rep;;

let rec decodage_aux s t t0 i =
   match t0 with
   |Feuille (x, _) ->
      (String.make 1 (char_of_int x))^decodage_aux s t t i
   |Arbre (t1, _, t2) ->
      if i=String.length s then
         ""
      else
         if s.[i]='0' then
            decodage_aux s t t1 (i+1)
         else
            decodage_aux s t t2 (i+1);;

let decodage s t=
   match t with
   |Feuille (x, p) -> String.make p (char_of_int x)
   |_ -> decodage_aux s t t 0;;

let efficacite s = (string_of_int (8 * String.length s)) ^" --> "^ (string_of_int
(String.length (codage s)));;

(* Exemple d'un arbre de hauteur 256... *)
let range () =
   let h = Array.make 256 0 in
   let a=ref 0 and b =ref 1 and c= ref 0 in
      for i=0 to 255 do
         h.(i) <- !b;
         c:= !b; b:= !a+ !b; a:= !c;
      done; h;;

let huf=codage;;
let huf_inv=decodage;;

let rec huit_vers_un_aux s p n rep=
   if n<8 then
      if p+n<String.length s then
         huit_vers_un_aux s p (n+1) (2*rep+int_of_string (String.make 1 s.[p+n]))
      else
         (* à la fin, il complète la chaine par des 0 pour finir l'octet *)
         huit_vers_un_aux s p (n+1) (2*rep)
   else
      rep;;
let huit_vers_un s p = huit_vers_un_aux s p 0 0;;

(* Encode huffman sur 1 octet, donc renvoie les caractères à la place d'une série de 0
et de 1 *)
let huf_bis s =
   let s2 = huf s in
   let rep = String.make ((String.length s2+7)/8) ' ' in
   for i=0 to (String.length s2+7)/8 -1 do
      rep.[i] <- char_of_int (huit_vers_un s2 (8*i))
   done;
   rep;;
```

Radix MM 5.ml

```
(*********************************************)
(*    Radix et méthode de Manber et Myers      *)
(*********************************************)
               (* Version 5 *)

open String;;


(* Place les indices 0 à n-1, représentant les caractères, dans la table de hachage tab
*)
let place s tab =
   for i=0 to length s-1 do
      let k = int_of_char s.[i] in
      tab.(k) <- i::tab.(k)
   done;;

(* Place les éléments de la table dans le tableau t *)
let rec vide_aux t a i = function
   |[] -> i (* i = indice auquel au peut insérer *)
   |h::q -> (* t.(i) <- [|(h,(true,a,a));(h,(true,a,a))|]; *)
      (**)  t.(i).(0) <- (h,(true,a,a));
            t.(i).(1) <- (h,(true,a,a)); (**) (* plus rapide! *)
            vide_aux t a (i+1) q;;

(* Vide la table et crée le tableau classes *)
let vide nb_char tab t =
   let i = ref 0
   and classes = Array.make_matrix (max (nb_char*nb_char) (Array.length t)) 2 0 in
   for a=0 to nb_char-1 do
      (* classes.(a) <- [|!i;!i|]; *)
      (**) classes.(a).(0) <- !i ;
           classes.(a).(1) <- !i ; (**)
      (* début de la classe du caractère a (éventuellement vide!) *)
      i := vide_aux t a (!i) (tab.(a))
   done;
   t,classes;;

(* Tri radix : Tri des chaînes selon leur premier caractère
On place les indices des rotations dans la table de hachage, puis on vide cette table
dans un tableau t, qui se trouve ainsi trié selon le premier caractère de chaque chaîne
*)
let radix_1 nb_char s =
   let n = length s +nb_char-256 in
   let tab = Array.make nb_char [] in
   place s tab;
   if nb_char=257 then tab.(256) <- [n-1] (* else () *);
(* On rajoute l'eoff dans la classe 257 *)
   let t = Array.make_matrix n 2 (0,(true,0,0)) in
   vide nb_char tab t;;


(*** Méthode de Manber et Myers ***)


(* variables :

- t : tableau que l'on trie, la première colonne est la référence, la deuxième est
modifiée au cours du parcourt ; à la fin de l'étape, on recopie la deuxième sur la
première
t.(i).(_) = (a,(b,c,d)), la rotation en position i :
a : numéro de la rotation (type int)
b : booléen "ce terme est dans une classe, donc n'est pas trié" (type bool)
c : indice de la classe (type int)
d : indice complémentaire pour distinguer deux termes issus d'une même classe après une
étape (type int)
```

```
- classes : tableau des classes, contient n cases, majoration (grossière) du nombre
maximal de classes. On aurait pu se limiter à n/2...
Colonne 0 : Indice du début de chaque classe
Colonne 1 : Pointeur qui est dirigé vers la destination de la prochaine insertion dans
chaque classe (sujet au changement donc à réinitialiser entre chaque étape)

- reste : tableau à 1 case, donnant le nombre de chaînes non triées. On arrête si ce
nombre tombe à 0

- pos :  tableau donnant la position de chaque rotation dans t : pos.(a) = i ssi
t.(i).(0) = a,_
*)

(* Premier renommage : initialisation de t par détection des caractères déjà triés
(ceux qui sont seuls dans leur classe *)
let initialise nb_char t n classes reste = (* ask "initialise"; *)
   let i = ref 0 in
   while !i<>nb_char do
      (* ask ((make 1 (char_of_int !i)) ^"\n");
      ask ((string_of_int (classes.(!i).(0)))^"\n"); *)

      let k = classes.(!i).(0) (* vraie valeur *) in
      if k >= n then
         i := nb_char
      else
         let a,(b,c,_) = t.(k).(0) in (* 0 ou 1 arbitrairement *)

         if c<>(!i) then (* les classes !i à c-1 sont vides *)
               i:= c
         else (* test de solitude *)
            if k<>(n-1) && (let _,(b',c',_) = t.(k+1).(0) in b' && c'=c)
            || k<>0 && (let _,(bb',cc',_) = t.(k-1).(0) in bb' && cc'=c)
               then (* il n'est pas seul... *)
                  incr i
               else (* il est seul DONC il est bien trié! (tri selon le premier
caractère) *)
                  begin
                     (* t.(k) <- [|(a,(false,300+(!i),0));(a,(false,300+(!i),0))|]; *)
                     (**)
                     t.(k).(0) <- a,(false,300+(!i),0);
                     t.(k).(1) <- a,(false,300+(!i),0);
                     (**)
                     reste.(0) <- reste.(0) -1;
                     incr i
                  end
   done;;
(* complexité O(1) *)

(* Renvoie 2^p *)
let pow p = int_of_float (2. ** (float_of_int p));;

(* Modulo modifié, ne renvoie que des entiers positifs *)
let mod' a b = let c = a mod b in
               if c<0 then c+b else c;;


(* Parcourt le tableau t en regardant pour chaque rotation a si l'antécédent a' de a
est dans une classe. Si oui, on place a' là où le pointeur nous l'indique. A l'issu de
cette étape p, la colonne 1 de t est triée selon 2^(p+1) caractères. *)
let parcourt n p pos t classes = (* ask "parcourt"; *)
   (* print_int (Array.length classes); *)
   for i=0 to n-1 do
      let a,(_,c,_) = t.(i).(0) in (* valeur réelle! *)
```

```
      (**
      print_string "i = "; print_int i; print_newline ();
      print_string "a = "; print_int a; print_newline ();
      print_string "classe : "; print_int c; print_newline ();
      **)

      (*if Array.length classes<257 || a - pow p>=0 then (* Si on a l'eoff
(nb_char=257, on ne revient pas sur la fin de la chaine, on est forcément bien
rangé...) *) (* Finalement, au pire on revient, ce n'est pas bien grave... *)*)

            let a',(b',c',_) = t.(pos.(mod' (a - pow p) n)).(0) in (* <- antécédent *)

            (**
            print_string "antécédent = "; print_int a'; print_newline ();
            print_string "classe : "; if b' then print_string "oui" else print_string
"non"; print_newline ();

            print_matrix t "t"; ask "check.";
            **)

            if b' (* l'antécédent est dans une classe *) then
               begin
                  t.(classes.(c').(1)).(1) <- (a',(true,c',c)); (* on place où le
pointeur nous l'indique *)
                  classes.(c').(1) <- classes.(c').(1) +1 (* on redirige le pointeur
vers la case suivante *)
               end;
               (* NE PAS modifier pos ici!!! Ce sera fait dans renomme *)
   done;;


(* Crée le tableau pos *)
let posit t n = (*ask "posit";*)
   let p = Array.make n 0 in
   for i=0 to n-1 do
      let a,_ = t.(i).(0) in
      p.(a) <- i
   done;
   p;;   (* pos.(a) = la position de a dans t, colonne 0 *)



(* Fonction de renommage des classes : on crée des nouvelles classes numérotées
0,1,2..., les dernières du tableau classes ne sont dons pas utilisées. On change la
classe des éléments "vivants", càd dans une classe, et on réinitialise le tableau
classes avec les indices des nouvelles classes. *)
let renomme n t pos classes reste = (* ask "renomme"; *)
   let last = ref (false,0,0) in (* Référence à la dernière case vue *)

   let actuelle = ref (-1) in


   for i=0 to n-1 do


      let a,(b,c,d) = t.(i).(1) in (* valeur actualisée par parcourt *)
      pos.(a) <- i; (* actualisation de la position *)

      (* -> renommage des classes! Actualisation, càd copie de la colonne 0 sur la
colonne 1 *)
      let (bb',cc',dd') = !last in

      if b (* ssi a,_ a été modifié lors de parcourt et donc est dans une classe *)
then
         if bb' && cc'=c && dd'=d (* i<>0 inutile ici grâce au !last, que l'on a
d'ailleurs déjà évalué *)
            then (* il n'est pas seul : le précédent est de sa classe *)
```

```
                begin

                   t.(i).(0) <- a,(true,!actuelle,0);
                   t.(i).(1) <- a,(true,!actuelle,0);

                   (* on lui donne la nouvelle classe (!actuelle) de son prédécesseur *)
                   last := (true (* car b = true *),c,d)
                end


          else if i<>n-1 && (let _,(b',c',d') = t.(i+1).(1) in b' && c'=c && d'=d)
             then (* il n'est pas seul : le suivant est de sa classe *)
                begin
                   actuelle := 1+ !actuelle ;

                   classes.(!actuelle).(0) <- i;
                   classes.(!actuelle).(1) <- i;

                   t.(i).(0) <- a,(true,!actuelle,0);
                   t.(i).(1) <- a,(true,!actuelle,0);
                   last := (true,c,d)
                end

          else (* il est seul *)
                begin

                   t.(i).(0) <- a,(false, 600 + reste.(0) ,0);
                   t.(i).(1) <- a,(false, 600 + reste.(0) ,0);
                   (* on donne un nom original à l'élément décédé. Il sera le seul à le
porter. *)


                   (* Attention! On change last pour garder une trace de ce qui fut à
cette place *)
                   last := (true,c,d);
                   reste.(0) <- reste.(0) -1;
                end
       else
             (* Attention! Même si l'élément n'est pas dans une classe, il faut penser à
renommer last! *)
          last := (false,c,d)

   done;;


(* Fonction finale : on effectue un radix pour k=1 puis on itère parcourt et renomme
jusqu'à ce que l'on dépasse de la taille des chaînes ou que chacune soit triée. *)

let ultimate_radix_mm_aux nb_char s =
   let t,classes = radix_1 nb_char s in
   let n = length s +nb_char-256 in
   let reste = [|n|] in
   initialise nb_char t n classes reste;

   let p = ref 0
   and pos = posit t n in
   while reste.(0)<>0 do

      (* 2^p caractères triés lorsqu'on COMMENCE l'étape p *)
      parcourt n !p pos t classes;
      incr p;
      if pow !p (* <- nombre de caractères triés à cette étape du calcul *) < n then
         renomme n t pos classes reste
      else reste.(0) <- 0
   done;
   t;;
```

```
let ultimate_radix_mm = ultimate_radix_mm_aux 256;;
let ultimate_radix_mm_eoff = ultimate_radix_mm_aux 257;;

(* Complexité : O(n*log n) pour une chaîne de longueur n *)


(* Affiche le résultat sous forme de rotations *)
let interprete t s =
   let n = length s in
   print_string ("\n"^"Rotations triées pour: \""^s^"\"\n");
   for i=0 to n-1 do
      let a,_ = t.(i).(1) in
      print_string ("("^(string_of_int a)^")"^(if n>10 && a<10 then "  " else " ")^(sub
s a (n-a))^(sub s 0 a));
      print_string "\n"
   done;
   print_string "\n";;

(* Fonction qui affiche concrètement le résultat du tri *)
let test s =
   let t = ultimate_radix_mm s in
   interprete t s;;


(* Fonction de vérification de la correction du tri : attention, complexité
désastreuse! *)
let verif s =
   let t = ultimate_radix_mm s in
   let n = length s in
   let bool_t = Array.make n false in
   for i=0 to n-1 do
      bool_t.(i) <- true
   done;

   for i=0 to n-1 do
      if not bool_t.(i) then failwith "erreur : doublons"
   done;

   let extrait a = (sub s a (n-a))^(sub s 0 a) in

   for i=1 to n-1 do
      let r2,_ = t.(i).(1)
      and r1,_ = t.(i-1).(1) in
      let s2 = extrait r2 and s1 = extrait r1 in
      if s1 > s2 then failwith "erreur : tableau non trié"
   done;
   print_string "==> Tri réussi!\n";;

let rec tri_radix_aux t i rep =
   if i<0 then
      rep
   else
      let (a,_)=t.(i).(1) in
      tri_radix_aux t (i-1) (a::rep);;
let tri_radix_general nb_char s =
   let t = ultimate_radix_mm_aux nb_char s in
   tri_radix_aux t (Array.length t -1-nb_char+256) [];;

let tri_radix_mm = tri_radix_general 256;;
let tri_radix_mm_eoff = tri_radix_general 257;;
```

ANNEXE 4

Bibliographie :

WIKIPÉDIA, Transformée de Burrows−Wheeler, [En ligne], https://fr.wikipedia.org/wiki/Burrows−Wheeler_Transform (Page consultée le 19/05/2014)
WIKIPÉDIA, LZ77 et LZ78, [En ligne], http://fr.wikipedia.org/wiki/LZ77_et_LZ78 (Page consultée le 19/05/2014)
BURROWS Michael, WHEELER David John, A Block Sorting Lossless Data Compression Algorithm, Cambridge : University of Cambridge, 10 Mai 1994, 24 pages
FELDSPAR Antaeus, An Explanation of the Deflate Algorithm, [En ligne], http://www.zlib.net/feldspar.html (Page consultée le 19/05/2014)
MICROSOFT, LZ77 Compression Algorithm, [En ligne], http://msdn.microsoft.com/en−us/library/ee916854.aspx (Page consultée le 19/05/2014)
WIKIPÉDIA, Run−length encoding, [En ligne], https://fr.wikipedia.org/wiki/Run−length_encoding (Page consultée le 19/05/2014)
Faiseur, Les algorithmes de compression − Partie 1 − La théorie, [En ligne], http://www.asmforum.net/t107−les−algorithmes−de−compression−partie−1−la−theorie (Page consultée le 19/05/2014)
WIKIPÉDIA, Radix Sort, [En ligne], https://en.wikipedia.org/wiki/Radix_sort (Page consultée le 19/05/2014)

Guillaume Ménard et moi-même souhaiterions remercier Marc Lorenzi, pour son aide et son soutien tout au long de ce TIPE.