RAPPORT DE MATHEMATIQUES
ALGO DE TRI
(PREMIERE PERIODE)

1 CODES CORRECTEURS

1.1 Code source :

```
//*************************************************************************//
// Nom du programme                | MathsCodesCorrecteurs.cpp             //
//*************************************************************************//
// But                             | Implémenter la gestion des codes      //
//                                 | d'erreurs sur un mot binaire          //
//*************************************************************************//
// Auteurs                         | David Montet / ALexandre Villemaine   //
//*************************************************************************//
// Date de Création / modification | 14/09/04 / 27/09/04                   //
//*************************************************************************//

//*************************************************************************//
// Inclusion de fichiers
//
//*************************************************************************//

#include "stdafx.h"
#include "math.h"

//*************************************************************************//
// Definition des constantes
//
//*************************************************************************//

//constante de taille maximum de tableau
#define TAILLEMAX 256

//*************************************************************************//
// En-tête des fonctions
//
//*************************************************************************//

// Ajout en fin de tableau des bits de parité
// Paramètres en sortie: - tab::tableau       le tableau
//                       - tailleTab::entier  la taille du tableau
int codage(int tab[TAILLEMAX],int *tailleTab);

// Affichage d'un tableau contenant les énumérations possibles
// Paramètres en entrée: - tabMots::tableau le tableau à afficher contenant les
énumérations
//                       - tailleTab::entier   la taille du tableau
//                       - nbBitParite::entier le nombre de bit de parité
// Paramètres en sortie: - nbMots::entier      le nombre de mots dans le tableau
int enumeration(int tailleTab,int nbBitParite,int tabMots[TAILLEMAX][TAILLEMAX],
int *nbMots);

// Vérification de la validité d'un mot
// Paramètres en entrée: - tabMots::tableau    le tableau des énuùérations
//                       - tab::tableau        le tableau contenant
le mot à vérifier
//                       - tailleTab::entier   la taille du tableau
//                       - nbMots::entier      le nombre de mots
dans le tableau
int verification(int tab[TAILLEMAX],int tailleTab, int tabMots[TAILLEMAX]
[TAILLEMAX],int nbMots);
```

```
// Affichage des mots les plus proches du mot recherché
// Paramètres en entrée: - tab::tableau        le tableau contenant le mot
//                                  - tabMots::tableau    le tableau contenant
les énumérations
//                                  - tailleTab::entier   la taille du tableau
//                                  - nbMots::entier      le nombre de mots
dans le tableau
int correction(int tab[TAILLEMAX],int tailleTab, int tabMots[TAILLEMAX]
[TAILLEMAX],int nbMots);

// Retourne la distance de Hamming entre deux mots
// Paramètres en entrée: - tab1,tab2::tableau  les tableaux contenant les mots à
comparer
//                                  - tailleTab::entier   la taille du tableau
int distanceHamming(int tab1[TAILLEMAX],int tab2[TAILLEMAX],int tailleTab);

// Retourne la distance minimale entre les mots du langage
// Paramètres en entrée: - tabMots::tableau    les tableaux contenant ts les
mots du langage
//                                  - nbMots::entier      le nombre de mots
dans le tableau
//                                  - tailleTab::entier   la taille du tableau
int distanceMinimale(int tabMots[TAILLEMAX][TAILLEMAX],int nbMots,int
tailleTab);

/*************************************************************************/
// Programme principal
/*************************************************************************/

int main(int argc, char* argv[])
{
      //taille du mot
      int tailleTab = 4;
      //nombre de mots dans le tableau d'énumération
      int nbMots=0;
      //nombre de bits de parité
      int nbBitParite=4;
      //déclaration d'un mot
      int tab[TAILLEMAX]={0,0,0,0};
      //déclaration d'un message (mot + bits de parité)
      int tab2[TAILLEMAX]={0,0,0,0,0,1,1,0};
      //déclaration de la matrice contenant les mots possibles
      int tabMots[TAILLEMAX][TAILLEMAX];

      //codage des bits de parité de tab
      codage(tab,&tailleTab);

      //affichage du tableau tab
      printf("Affichage de tab:\n");
      for (int i =0;i<tailleTab;i++)
            printf("%d,",tab[i]);
      printf("\n");

      //création de la matrice d'énumeration
      enumeration(tailleTab,nbBitParite,tabMots,&nbMots);

      //affichage de la matrice d'énumération
      printf("affichage de l'enumeration:\n");
      for (int i1 =0;i1<nbMots;i1++)
      {
            for (int j =0;j<(tailleTab);j++)
                  printf("%d",tabMots[i1][j]);
            printf("\n");
      }

      int distMin = distanceMinimale(tabMots,nbMots,tailleTab);
      printf("\nDistance minimale : %d\n\n",distMin);
      //affichage de la validité du mot tab
```

```
	if (verification(tab,tailleTab,tabMots,nbMots)==-1)
		printf("tab: Mot errone!\n");
	else
		printf("tab: Mot correct!\n");

	//affichage du tableau tab2
	printf("affichage de tab2:\n");
	for (i =0;i<tailleTab;i++)
		printf("%d,",tab2[i]);
	printf("\n");

	//affichage de la validité du mot tab2
	if (verification(tab2,tailleTab,tabMots,nbMots)==-1)
		printf("tab2: Mot errone!\n");
	else
		printf("tab2: Mot correct!\n");

	//affichage des mots les plus proches de tab2
	correction(tab2,tailleTab,tabMots,nbMots);
	//affichage du tableau

	/*printf("tab2 corrigé\n");
	for (i =0;i<tailleTab;i++)
		printf("%d,",tab2[i]);
	printf("\n");*/
return 0;
}

/*****************************************************************************/

/*****************************************************************************/
int codage(int tab[TAILLEMAX],int *tailleTab)
{
	//initialisation des tableaux de règles
	int tabr1[2]={1,4}; // trad: r1=(x1+x3+x4)modulo2
	int tabr2[2]={1,3};
	int tabr3[3]={1,3,4};
	int tabr4[3]={1,2,4};
	//initialisation et calcul des parités rm1 et rm2
	int r1=0;
	int r2=0;
	int r3=0;
	int r4=0;
	int rm1,rm2,rm3,rm4,rm5;
	//somme sans modulo r1
	for (int i=0;i<=1;i++)
	{
		r1 = r1 + tab[tabr1[i]-1];

	}
	//rm1 = r1 modulo 2
	rm1 = r1%2;
	//pareil que r1 et rm1
	for (int j=0;j<=1;j++)
	{
		r2 = r2 + tab[tabr2[j]-1];
	}
	rm2 = r2 % 2;
	//rm3
	for (j=0;j<=2;j++)
	{
		r3 = r3 + tab[tabr3[j]-1];
	}
	rm3 = r3 % 2;
	//rm4
	for (j=0;j<=2;j++)
	{
		r4 = r4 + tab[tabr4[j]-1];
```

```
	}
	rm4 = r4 % 2;
	 //augmentation de la taille du tableau et ajout des bits de parité
	(*tailleTab)++;
	tab[*tailleTab-1]= rm1;
	(*tailleTab)++;
	tab[*tailleTab-1]= rm2;
	(*tailleTab)++;
	tab[*tailleTab-1]= rm3;
	(*tailleTab)++;
	tab[*tailleTab-1]= rm4;
	return 0;
}

/*****************************************************************************/

/*****************************************************************************/
int enumeration(int tailleTab,int nbBitParite,int tabMots[TAILLEMAX][TAILLEMAX],
int *nbMots)
{
	//calcul du nombre de mots possibles
	*nbMots = pow(2,tailleTab-nbBitParite);
	//définition du nombre d'alternance entre les séries de 0 et de 1
	int nbAlternance=2;
	//pour chaque colonne
	for (int numCols=0;numCols<(tailleTab-nbBitParite);numCols++)
	{
		//on clacule la longueur de 0 qu'il faudra mettre
		int lg=((*nbMots)/nbAlternance);
		int debut = 0;
		//pour chaque alternance
		for (int x=0;x<(nbAlternance/2);x++)
		{
			//on met le nombre de zero qu'il faut
			for (int x1=debut;x1<debut+lg;x1++)
				tabMots[x1][numCols]=0;
			//on met le nombre de 1 qu'il faut
			for (int x2=debut+lg;x2<(debut+lg+lg);x2++)
				tabMots[x2][numCols]=1;
			//on décale le début
			debut = debut+(lg*2);
		}
		//a la fin, on met a jour le nombre d'alternances
		nbAlternance = nbAlternance*2;
	}
	//initialisation de la taille du mot
	int taille=tailleTab-nbBitParite;
	//pour chaque mot, on calcule ses bits de parité
	for (int i=0;i<*nbMots;i++)
	{
		codage(tabMots[i],&taille);
		taille=taille-nbBitParite;
	}
	return 0;
}

/*****************************************************************************/

/*****************************************************************************/
int verification(int tab[TAILLEMAX],int tailleTab, int tabMots[TAILLEMAX]
[TAILLEMAX],int nbMots)
{
	//pour chaque mot de l'énumération
	for (int i=0;i<nbMots;i++)
	{
		//si la distance de hamming est nulle, on renvoie que le mot est
correct
		if (distanceHamming(tabMots[i],tab,tailleTab)==0)
```

```
                  return 0;
      }
      //sinon on retourne qu'il n'est pas correct
      return -1;
}

/*****************************************************************************/

/*****************************************************************************/
int correction(int tab[TAILLEMAX],int tailleTab, int tabMots[TAILLEMAX]
[TAILLEMAX],int nbMots)
{

      //initialisation de la distance à sa valeur maxim possible(c.a.d. la
taille du mot)
      int distance=tailleTab;
      //distance temporaire
      int distTemp;
      //indice de l'élément trouvé
      int ind=0;
      //recherche de la distance minimale
      //pour chaque mot de l'énumération
      for (int i=0;i<nbMots;i++)
      {
            //si la distance courante est inférieure à la distance en mémoire
            if ((distTemp = distanceHamming(tabMots[i],tab,tailleTab))<distance)
            {
                  //elle devient la nouvelle distance
                  distance = distTemp;
                  ind = i;
            }

      }
      //on affiche les mots les plus proches
      printf("mot(s) le(s) plus proche(s):\n");
      for (i=0;i<nbMots;i++)
      {
            //pour chaque mot ou la distance est minimum
            if (distanceHamming(tabMots[i],tab,tailleTab)==distance)
            {
                  //on l'affiche
                  for (int j =0;j<tailleTab;j++)
                  printf("%d,",tabMots[i][j]);
                  printf("\n");
            }

      }
      /*for (int k=0;k<tailleTab;k++)
      {
            tab[k]=tabMots[ind][k];
      }*/
      return 0;
}
/*****************************************************************************/

/*****************************************************************************/
int distanceHamming(int tab1[TAILLEMAX],int tab2[TAILLEMAX],int tailleTab)
{
      //initialisation du nombre de différences
      int diff=0;
      //pour chaque bit des mots
      for (int j=0;j<tailleTab;j++)
            //si les bits sont différents
            if (tab1[j]!=tab2[j])
                  //on augmente la distance
                  diff++;
      return diff;
}
```

```
/*****************************************************************************/
/*****************************************************************************/
int distanceMinimale(int tabMots[TAILLEMAX][TAILLEMAX],int nbMots,int tailleTab)
{
	//initialisation de la distance à la taille maximum
	int distanceMinimale=tailleTab;
	int distTemp;
	//pour chaque mot
	for (int i=0;i<nbMots;i++)
	{
		//on le compare avec tous les autres mots
		for (int j=i+1;j<nbMots;j++)
		//si la distance courante est inférieure à la distance en mémoire
		if ((distTemp = distanceHamming(tabMots[i],tabMots[j],tailleTab))
<distanceMinimale)
		{
			//elle devient la nouvelle distance
			distanceMinimale = distTemp;
		}

	}
	//on retourne la distance minimale
	return distanceMinimale;

}
```

1.2 Trace d'exécution du programme :

```
Affichage de tab:
0,0,0,0,0,0,0,0,

affichage de l'enumeration:
00000000
00011011
00100110
00111101
01000001
01011010
01100111
01111100
10001111
10010100
10101001
10110010
11001110
11010101
11101000
11110011

Distance minimale : 2

tab: Mot correct!

affichage de tab2:
0,0,0,0,0,1,1,0,

tab2: Mot errone!

mot(s) le(s) plus proche(s):
0,0,1,0,0,1,1,0,

Press any key to continue
```

2 TRIS

2.1 Code source

```
//*************************************************************************//
// Nom du programme                | AlgoTri.cpp                           //
//*************************************************************************//
// But                             | Implémente différents tris            //
//                                 |                                     //
//*************************************************************************//
// Auteurs                         | David Montet / ALexandre Villemaine   //
//*************************************************************************//
// Date de Création / modification | 21/09/04 / 27/09/04                   //
//*************************************************************************//

//*************************************************************************//
// Inclusion de fichiers
//
//*************************************************************************//
#include <math.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

//*************************************************************************//
// Définition de constantes
//
//*************************************************************************//
#define TAILLEMAX 64000

//*************************************************************************//
// En-têtes des fonctions
//
//*************************************************************************//
void triParExtraction(int t[TAILLEMAX],int n,int * nbIter,int debut,int pas);
void creerTableau(bool tri, int taille, int tab[TAILLEMAX]);
void afficherTableau(int t[TAILLEMAX],int n);
void triInsertion(int t[TAILLEMAX],int n,int debut,int pas, int *nbIter );
void pourvoir(int t[TAILLEMAX], int i, int taille, int *nbIter);
int remplir_tab ( int t[TAILLEMAX] , int N);
bool verifTri(int t[TAILLEMAX],int n);
int demander_taille();
int InsertionParDichotomie(int val,int t[TAILLEMAX],int *n, int * nbIter);

//*************************************************************************//
// Programme principal
//
//*************************************************************************//
int main(int argc, char* argv[])
{
      //initialisations
      int tab[TAILLEMAX],tab2[TAILLEMAX],tab3[TAILLEMAX],tab4[TAILLEMAX];
      int n=0;
      int pas=0;
      int nbIter=0;
      int debut=0;
      bool go=true;
      int prof;
      time_t t1,t2;
      double t;

      //saisie du nombre d'elements
      n=demander_taille();
      printf("Nombre d'elements dans la liste : %d\n\n",n);
```

```
	//creation des tableaux remplis non triés
	creerTableau(false,n,tab);
	creerTableau(false,n,tab2);
	creerTableau(false,n,tab3);

	//tri par promotion
	time(&t1);
	nbIter=0;
	prof = remplir_tab(tab4,n);
	for (int i=pow(2,prof-1)-2; i>=0 ;i--)
		pourvoir (tab4, i, prof, &nbIter);
	for ( i=0; i<n;i++)
		pourvoir (tab4, 0, prof ,&nbIter);
	time(&t2);
	t = difftime(t2,t1);
	printf("Tri par promotion en %d iterations et %f secondes\n\n",nbIter,t);

	//tri par extraction
	nbIter=0;
	time(&t1);
	triParExtraction(tab,n,&nbIter,0,1);
	time(&t2);
	t = difftime(t2,t1);
	printf("Tri par extraction en %d iterations et %f secondes\n\n",nbIter,t);

	//tri par insertion
	time(&t1);
	nbIter=0;
	triInsertion(tab2,n,0,1,&nbIter);
	time(&t2);
	t = difftime(t2,t1);
	printf("Tri par insertion en %d iterations et %f secondes\n\n",nbIter,t);

	//tri SHELL
	nbIter=0;
	pas=n/2;
	time(&t1);
	while (go)
	{
		for(debut=0;debut<pas;debut++)
		{
			triInsertion(tab3,n,debut,pas,&nbIter);
		}
		if (pas==1) go=false;
		else pas=pas/2;
	}
		time(&t2);
	t = difftime(t2,t1);
	printf("Tri SHELL en %d iterations et %f secondes\n\n",nbIter,t);

	//test si les tableaux sont bien triés à la fin
	if (verifTri(tab,n)) printf("tableau tab trie\n");
	else printf("tableau tab non trie\n");
	if (verifTri(tab2,n)) printf("tableau tab2 trie\n");
	else printf("tableau tab2 non trie\n");
	if (verifTri(tab3,n)) printf("tableau tab3 trie\n");
	else printf("tableau tab3 non trie\n");
	if (verifTri(tab4,n)) printf("tableau tab4 trie\n");
	else printf("tableau tab4 non trie\n");

	return 0;
}
/*********************************************************************/

/*********************************************************************/
void triParExtraction(int t[TAILLEMAX],int n,int * nbIter,int debut,int pas)
{
```

```
	//chaque t[i] represente un element du tableau
	for (int i=debut;i+pas<n;i=i+pas)
	{
		//Pour chacun des elements t[j]qui se trouvent à droite du t[i]
		for (int j=i+pas; j<n;j=j+pas)
		{
			//si l'element t[i] est supérieur à t[j], on échange les
éléments
			if (t[i]>t[j])
			{
				int tmp = t[i];
				t[i]=t[j];
				t[j]=tmp;
			}
			(*nbIter)++;
		}

	}
	//printf("%d\n",*nbIter);
}
/*****************************************************************************/

/*****************************************************************************/
void creerTableau(bool tri, int taille, int tab[TAILLEMAX])
{
	//initialisation nécessaire du timer
	srand ( time(NULL) );

	//pour chaque element du tableau jusqu'à la taille taille
	for (int i=0;i<taille;i++)
	{
	  //si le tableau est trié
	  if (tri)
	      tab[i]=i;
	  //sinon on génère au hasard dans [0,4*la taille du tableau]
	  else
	      tab[i]=rand()%taille*4;
	}

}
/*****************************************************************************/

/*****************************************************************************/
void afficherTableau(int t[TAILLEMAX],int n)
{
	//affichage du tableau
	for(int i=0;i<(n-1);i++)
	{
		printf( "%d,",t[i]);
	}
	printf("%d\n\n",t[n-1]);
}
/*****************************************************************************/

/*****************************************************************************/
void triInsertion(int t[TAILLEMAX],int n,int debut,int pas, int *nbIter )
{
	int j, tmp;
    //on tri les elements du tableau en partant de debut et avec un pas de pas
	for(int i = debut+pas; i+pas <= n; i=i+pas)
	{
	  tmp = t[i];
	  j = i;
	      //on fait remonter le plus grand
	      while((j-pas >= 0) && (t[j-pas] > tmp))
	      {
	      t[j] = t[j-pas];
			j = j-pas;
```

```
                    (*nbIter)++;
            }
            t[j] = tmp;
      }
}
/*****************************************************************************/

/*****************************************************************************/
void pourvoir(int t[TAILLEMAX], int i, int prof, int *nbIter)
{
      //fonction récursive qui fait remonter le maximum au sommet de l'arbre
      int T = pow (2, prof);

      if (i <= pow (2, prof-1) - 2 )
      {
            (*nbIter)++;
            if (t[2*i+1] >= t[2*i+2] )
            {
                  t[i] = t[2*i+1];
                  pourvoir (t, 2*i+1, prof,nbIter);
            }
            else
            {
                  t[i] = t[2*i+2];
                  pourvoir (t, 2*i+2, prof,nbIter);
            }
      }
      else
            if ( i <= T)
                  t[i] = 0;

}
/*****************************************************************************/

/*****************************************************************************/
int remplir_tab ( int t[TAILLEMAX] ,int N)
{

      //rempli le tableau avec des nombres aléatoires compris entre 0 et taille
du tableau *4
      //(pour eviter d'avoir trop de nombres identiques
      srand ( time(NULL) );
      int prof=0;
      bool go=true;
      int elem;

      //recuperation de la profondeur de l'arbre
      while (go)
      {
            prof++;
            if (pow(2,prof)>N)
                  go = false;
      }
      prof++;

      //remplissage du tableau
      for(int i=pow(2,prof-1)-1; i<pow(2,prof-1)+N-1 ; i++)
      {
            t[i]=rand()%(N*4);
      }
      for (i=pow(2,prof-1)+N-1 ;i<=pow(2,prof)-2;i++)
            t[i]=0;
      return prof ;
}
/*****************************************************************************/

/*****************************************************************************/
int demander_taille()
```

```
{
	//saisie du nombre d'éléments
	int taille;
	printf("Saisir le nombre d'elements (16384 max): \n");
	scanf("%d",&taille);

	return taille;
}

bool verifTri(int t[TAILLEMAX],int n)
{
	//verifie si le tableau est trié
	bool good=true;
	for (int i=0; i<n-1 && good!=false; i++)
	{
		if (t[i] >t[i+1])
			good=false;
	}
	return good;
}
/*****************************************************************************/

/*****************************************************************************/
int InsertionParDichotomie(int val,int t[TAILLEMAX], int *n, int * nbIter)
{
	*nbIter=0;
	int debut = 0;
	int fin   = *n;
	int milieu;
	int ind=0;
	bool go=true;
	while(go)
	{
		(*nbIter)++;
		milieu = (fin+debut) / 2;
		if (debut==milieu || milieu==fin)
		{
			go=false;
			ind = milieu;
		}
		if (val< t[milieu])
			fin = milieu;

		else
		{
			if (val == t[milieu])
			{
				ind = milieu;
				go = false;
			}
			else
				debut=milieu;

		}
	}
	if (ind!=0 )
	{
		ind++;
	}
	(*n)++;
	for (int i=*n;i>=(ind);i--)
	{
		t[i]=t[i-1];
	}
	t[ind]=val;

	return 0;
}
```

/***/

2.2 Trace d'exécution

```
Saisir le nombre d'elements (16384 max):
16384

Nombre d'elements dans la liste : 16384

Tri par promotion en 311279 iterations et 0.000000 secondes

Tri par extraction en 134209536 iterations et 3.000000 secondes

Tri par insertion en 66735504 iterations et 1.000000 secondes

Tri SHELL en 1155703 iterations et 0.000000 secondes

tableau tab trie
tableau tab2 trie
tableau tab3 trie
tableau tab4 trie

Press any key to continue
```