|  |  |
| --- | --- |
|  | **COMPLEXITÉ** |
|  | **Master 1 IL**  **Groupe 2**  **2018** |

|  |
| --- |
| **[ Algorithmique avancÉe et complexitÉ ]** |
| E-mail : geronimotoutcourt@gmail.com |

Rapport de TP N°6 COMPLEXITÉ : Le problème SAT

**BOUDOUR Mehdi / 201500008386/ TP: Le problème SAT**



*Ce document présent les solutions en 5 étapes : (1) les algorithmes écris en pseudo-code. (2) le calcul de la complexité au pire des cas. (3) Implémentation de l’algorithme en langage C. (4) capture de l’exécution de l’algorithme. (5) représentation graphique de l’évolution du temps d’exécution en fonction de N. Le programme C complet contenant les détails (affichage, calcul du temps d’exécution,…) d’implémentation est présenté à la fin du document.*

# I. Algorithme *Résolution du* problème SAT:

# - Une proposition atomique est une variable booléenne, c’est-à-dire prenant ses valeurs dans l’ensemble BOOL= {VRAI, FAUX}.

# - Un littéral est une proposition atomique ou la négation d’une proposition atomique.

# - Une proposition atomique est aussi appelée littéral positif ; et la négation d’une proposition atomique est appelé littéral négatif.

# - Une clause est une disjonction (somme logique ou (or)) de littéraux.

# Etant données m propositions atomiques p1, …, pm, une instanciation du m-uplet (p1, …, pm) est un élément de {VRAI, FAUX}. Une instanciation (e1, …, em) de (p1,

# …, pm) satisfait une clause c (notée (e1, …, em) c) si et seulement si l’une des conditions suivantes est satisfaite :

# 1- il existe i Є {1, …, m} tel que (ei = VRAI) et (pi occurre ou existe dans c) ; 2- il existe i Є {1, …, m} tel que (ei = FAUX) et (┐pi occurre ou existe dans c).

# Une instanciation satisfait une conjonction (produit logique et (and)) de clauses si et seulement si elle satisfait chacune de ses clauses. Une conjonction de clauses est satisfiable si et seulement s’il existe une instanciation la satisfaisant. Une instanciation satisfaisant une conjonction est dite solution ou modèle de la conjonction.

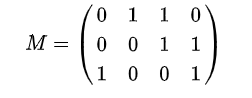
# Le problème SAT est maintenant défini comme suit :

# - Entrée : une conjonction C de n clauses construites à l’aide de m propositions atomiques p1, …, pm ;

# - Sortie : la conjonction C est-elle satisfiable ?

# *STRUCTURE PROPOSÉ : Matrice Binaire représentant une Conjonction.*

# 



# ⤇ M = (┐a ⋁ b ⋁ c ⋁ ┐d) ⋀ (┐a ⋁ ┐b ⋁ c ⋁ d) ⋀ (a ⋁ ┐b ⋁ ┐c ⋁ d)

# Déclaration: structure Conjonction

**TYPE ENREGISTREMENT CONJONCTION ;**

**MATRICE:TABLEAU [1..N][1..M] D'ENTIER;**

**NBCLAUSE,NBLITERRAUX:ENTIER;**

**FINENREG;**

# Implémentation : En langage C

**//Type Matrice**

**typedef struct Matrice Matrice ;**

**struct Matrice**

**{**

**long \*\*Pointer;**

**long n,m;**

**};**

# Déclaration: structure Instanciation (liste linéaire chainée) #1ère\_Optimisation

**TYPE ENREGISTREMENT INSTANCIATION**

**VALEUR:ENTIER;**

**SUIV: ^INSTANCIATION;**

**FINENREG;**

# Implémentation : En langage C

**//Type Instanciation Liste Linéaire Chainée**

**typedef struct Instanciation Instanciation ;**

**struct Instanciation**

**{**

**long valeur;**

**Instanciation \*suiv;**

**};**

# Algorithme : *Primitive de Manipulation des Instanciations*

**FONCTION AJOUTER(E/ I: ^INSTANCIATION, E/ VALEUR:ENTIER): ^INSTANCIATION**

**P: ^INSTANCIATION;**

**DEBUT**

**ALLOUER(P);**

***O(4)***

**P->VALEUR=VALEUR; P->SUIV=I;**

**RETOURNER P;**

**FIN;**

# Implémentation : En langage C

**//Primitives de Manipulation des Instanciations**

**Instanciation \*Ajouter(Instanciation \*I,long valeur)**

**{**

**Instanciation \*P=(Instanciation \*)malloc(sizeof(Instanciation ));**

**P->valeur=valeur; P->suiv=I;**

**return P;**

**}**

# Algorithme : *VerifierClause : Vérifie si une Clause est vrai pour une instance Donnée*

**FONCTION VERIFIERCLAUSE(E/ CLAUSE :TABLEAU[1..NBLITERRAUX] D' ENTIER , E/ NBLITERRAUXINSTANCE:ENTIER):BOOLEEN**

**J,BIT:ENTIER;**

**DEBUT**

**BIT = INSTANCE;**

**POUR J=NBLITERRAUX-1 JUSQU'A 0 PAS -1 FAIRE**

**SI (CLAUSE[J] = BIT MOD 2) ALORS**

***O(1+nbLiterraux)***

**RETOURNER VRAI;**

**FIN SI**

**BIT=BIT/2;**

**FIN POUR**

**RETOURNER FAUX;**

**FIN;**

# Implémentation : En langage C

**//Verifier : V�rifie si une Clause est vrai pour une instance Donn�e**

**long VerifierClause(long \*Clause,long nbLiterraux,long instance)**

**{**

**long j,bit = instance;**

**for(j=nbLiterraux-1;j>=0;j--)**

**{**

**if(Clause[j] == bit%2)**

**{**

**return 1;**

**}**

**bit=bit/2;**

**}**

**return 0;**

**}**

# Algorithme : *Solutions : retournes les instanciations pour lesquelles une Clause est vrai parmi les instanciations données en paramètre*

**FONCTION SOLUTIONS(E/ CLAUSE : TABLEAU D' ENTIER,E/ NBLITERRAUX:ENTIER ,E/INSTANCES : ^INSTANCIATION):^INSTANCIATION**

***C(Solution) = O(m\*2^m)***

**I,BIT,J,TAILLETV:ENTIER;**

**SOLUTIONS,P,Q: ^INSTANCIATION;**

**DEBUT**

**SOLUTION = NULL;**

**SI (INSTANCES=NULL)ALORS //PAS D'INSTANCES EN ENTREE**

**//TESTER TOUTE LA TABLE DE VERITE 2^M**

**TAILLETV = PUISSANCE(2,NBLITERRAUX);**

**POUR I=0 JUSQU'A TAILLET FAIRE**

**SI (VERIFIERCLAUSE(CLAUSE,NBLITERRAUX,I)=VRAI)**

***O(1+nbLiterraux)***

**ALORS**

***O(2^nbLitterraux \* (5+nbLitterraux))***

**SOLUTIONS = AJOUTER(SOLUTIONS,I);**

***O(4)***

**FIN SI;**

**FIN POUR;**

**SINON // INSTANCES != NULL IL YA DES VALEURS A TESTER**

**//PARCOURS DE LA LISTE DES INSTANCIATIONS**

**P=INSTANCE**

**TANT QUE ( P <> NULL;) FAIRE**

**SI(VERIFIERCLAUSE(CLAUSE,NBLITERRAUX,P->VALEUR) = VRAI)**

***O(1+nbLiterraux)***

**ALORS**

***O(2^nbLitterraux \* (5+nbLitterraux))***

**SOLUTIONS = AJOUTER(SOLUTIONS,P->VALEUR);**

***O(4)***

**FIN SI;**

**Q=P ; P=P->SUIV ; LIBERER(Q);**

**FIN TANT QUE;**

**FIN SI;**

**//RETOURNER LA LISTE DES INSTANCES POUR LESQUELLE CLAUSE EST VRAI**

**RETOURNER SOLUTIONS;**

**FIN;**

# Implémentation : En langage C

**//Solutions : retournes les instanciations pour lesquelles une Clause est vrai**

**//Parmi les instanciations donn�es en param�tre**

**Instanciation \*Solutions(long \*Clause,long nbLiterraux ,Instanciation \*Instances)**

**{**

**long i,bit, j;**

**Instanciation \*solutions =NULL ,\*P,\*Q;**

**if(Instances==NULL) //Pas d'instances en Entr�e**

**{//Tester pour toute la Table de V�rit� 2^m**

**long tailleTV = (long)pow(2,nbLiterraux);**

**for(i=0 ; i<tailleTV ; i++)**

**{**

**if(VerifierClause(Clause,nbLiterraux,i))**

**{solutions = Ajouter(solutions,i); }**

**Affichage(solutions);**

**}**

**}**

**else // Instances != NULL il ya des valeurs � Tester**

**{   //Parcours de la Liste des instanciations**

**for(P=Instances; P != NULL;)**

**{**

**if(VerifierClause(Clause,nbLiterraux,P->valeur))**

**{   solutions = Ajouter(solutions,P->valeur); }**

**Q=P ; P=P->suiv ; free(Q); //Lib�rer la cellue lu de Instances**

**}**

**}**

**//retourner la Liste des instances pour lesquelle Clause est VRAI**

**return solutions;**

**}**

# Algorithme : *Validation Version 1*

**FONCTION VALIDATION1(E/ CONJ :CONJONCTION,INT VALEUR) :BOOLEEN**

**I:ENTIER;**

**P,INSTANCES:^INSTANCIATION;**

**DEBUT**

***O(4)***

**P=NULL;**

**INSTANCES = AJOUTER(P,VALEUR);**

**SI (INSTANCES=NULL) ALORS**

**RETOURNER 0;**

**SINON**

**POUR I=1 JUSQU'A CONJ.NBCLAUSES FAIRE**

***O(nbClauses\*nbLitterraux)***

***O(nbLiterraux)***

**INSTANCES =**

**SOLUTIONS(CONJ.MATRICE[I],CONJ.NBLITERRAUX,INSTANCES);**

**SI (INSTANCES=NULL) ALORS**

**RETOURNER FAUX;**

**FIN SI;**

**FIN POUR;**

**FIN SI;**

**RETOURNER VRAI;**

**FIN;**

# Complexité :

# Au pire des cas : (INSTANCES <> NULL)

C(Version1)=

= ( -1 + 1)\*

= nbClauses\*nbLitterraux ∼ ***O(n\*m)*** ***Polynomiale***

# Au meilleur des cas : (INSTANCES = NULL)

C(Version1)= ***(1)***

# Implémentation : En langage C

**long Validation1(long \*\*Conjonction ,long nbClauses, long nbLiterraux,long valeur)**

**{**

**long i;**

**Instanciation \*P; P=NULL;**

**Instanciation \*instances = Ajouter(P,valeur);**

**if(instances==NULL) return 0;**

**else**

**for(i=1;i<nbClauses;i++)**

**{**

**instances = Solutions(Conjonction[i],nbLiterraux,instances);**

**if(instances==NULL) return 0;**

**}**

**return 1;**

**}**

# Exécution :

# *Cet Algorithme est si rapide qu’il a été difficile de relever des valeurs de temps significatives.*

# Algorithme : *Validation Version 2*

**FONCTION VALIDATION2(1(E/ CONJ :CONJONCTION) : ENTIER**

**I:ENTIER;**

**INSTANCES:^INSTANCIATION**

***O(2^nbLitterraux)***

**DEBUT**

**INSTANCES = SOLUTIONS(CONJ.MATRICE[0],CONJ.NBLITERRAUX,NULL);**

**SI(INSTANCES=NULL) ALORS**

**RETOURNER 0;**

**SINON**

**POUR I=1 JUSQU'A CONJ.NBCLAUSES FAIRE**

***O(nbLitterraux\*2^nbLitterraux)***

**INSTANCES =**

***O(nbLitterraux\*2^nbLitterraux)***

**SOLUTIONS(CONJ.MATIRE[I],CONJ.NBLITERRAUX,INSTANCES);**

**SI (INSTANCES=NULL) ALORS**

**RETOURNER FAUX;**

**FIN SI;**

**FIN POUR;**

**FIN SI;**

**RETOURNER VRAI;**

**FIN;**

# Complexité :

# Au pire des cas : (INSTANCES <> NULL)

C(Version2)=

= ( -1 + 1)\*

= nbClauses\*

∼ ***O( Exponetielle***

# Au meilleur des cas : (INSTANCES = NULL)

C(Version2)= ***()***

# Implémentation : En langage C

**long Validation2(long \*\*Conjonction ,long nbClauses, long nbLiterraux)**

**{**

**long i;**

**Instanciation \*instances = Solutions(Conjonction[0],nbLiterraux,NULL);**

**if(instances==NULL) return 0;**

**else**

**for(i=1;i<nbClauses;i++)**

**{**

**instances = Solutions(Conjonction[i],nbLiterraux,instances);**

**if(instances==NULL) return 0;**

**}**

**return 1;**

**}**

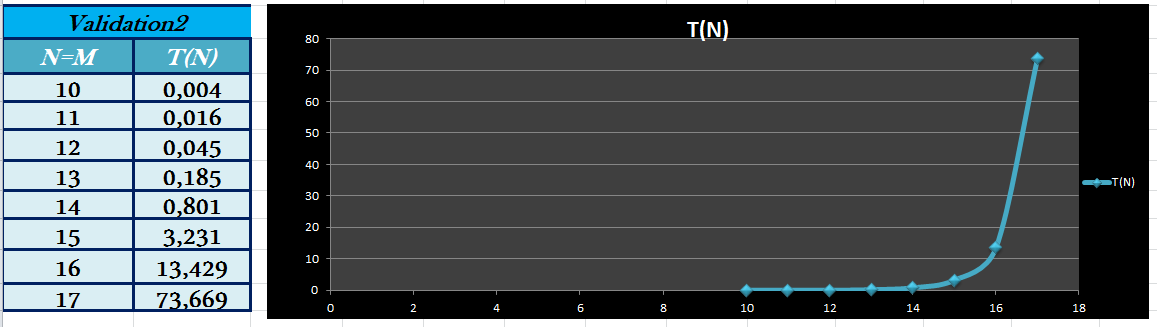
# Exécution :

# *Affichage du temps d’exécution de l’algorithme pour chaque valeur de N (T = le temps d’exécution calculé pour chaque exécution de la fonction Validation2).*

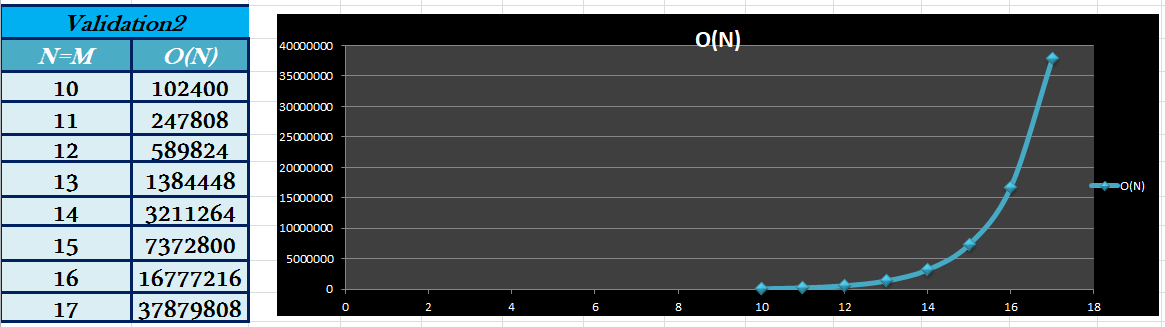
# C:\Users\Moflawer\Desktop\Dol_Gul_Dur\WorkShop_Tree\C\Almost_Done\TP_COMPLEXITÉ\SAT\exe.PNG

# Représentation Graphique :

# *Graphe du temps d’exécution en fonction de N.*



# *Graphe de la complexité théorique en fonction de N.*



# Algorithme : *Validation Version 3*

**FONCTION VALIDATION3(E/ CONJ:CONJONCTION) : ^INSTANCIATION**

**I:ENTIER;**

**INSTANCES:^INSTANCIATION**

***O(2^nbLitterraux)***

**DEBUT**

**INSTANCES = SOLUTIONS(CONJ.MATRICE[0],CONJ.NBLITERRAUX,NULL);**

**SI (INSTANCES=NULL) ALORS**

**RETOURNER NULL;**

**SINON**

**POUR I=1 JUSQU'A CONJ.NBCLAUSES FAIRE**

***O(nbLitterraux\*2^nbLitterraux)***

***O(nbLitterraux\*2^nbLitterraux)***

**INSTANCES =**

**SOLUTIONS(CONJ.MATRICE[I],CONJ.NBLITERRAUX,INSTANCES);**

**SI (INSTANCES=NULL) ALORS**

**RETOURNER NULL;**

**FIN SI;**

**FIN POUR ;**

**FIN SI;**

**RETOURNER INSTANCES;**

**FIN;**

# Complexité : De même que la version 2

# Au pire des cas : (INSTANCES <> NULL)

C(Version3)= ∼ ***O( Exponetielle***

# Au meilleur des cas : (INSTANCES = NULL)

C(Version3)= ***()***

# Implémentation : En langage C

**Instanciation \*Validation3(long \*\*Conjonction ,long nbClauses, long nbLiterraux)**

**{**

**long i;**

**Instanciation \*instances = Solutions(Conjonction[0],nbLiterraux,NULL);**

**if(instances==NULL) return NULL;**

**else**

**for(i=1;i<nbClauses;i++)**

**{**

**instances = Solutions(Conjonction[i],nbLiterraux,instances);**

**if(instances==NULL) return NULL;**

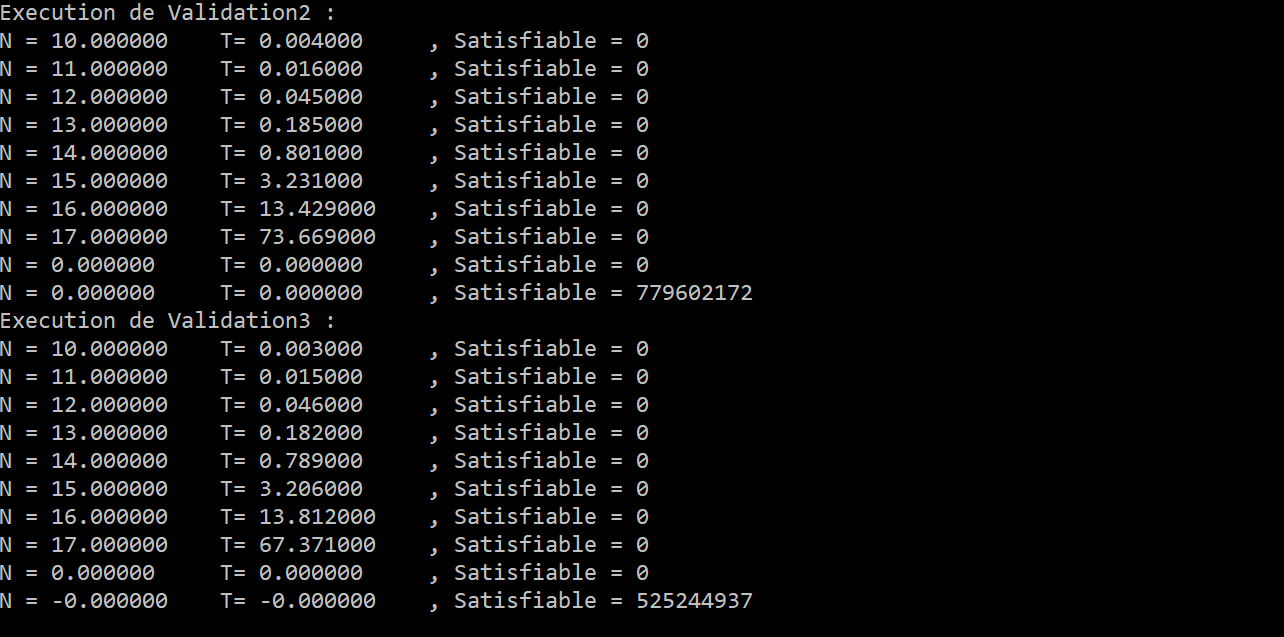
**}**

**return instances;**

**}**

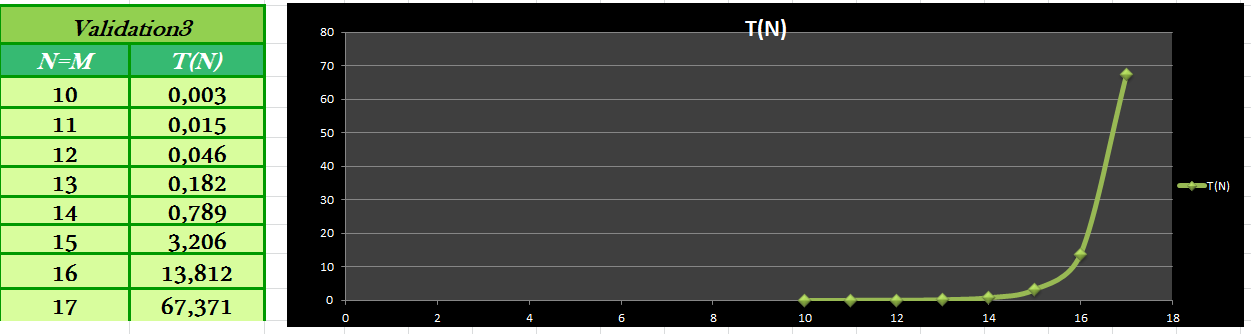
# Exécution :

# *Affichage du temps d’exécution de l’algorithme pour chaque valeur de N (T = le temps d’exécution calculé pour chaque exécution de la fonction Validation3).*

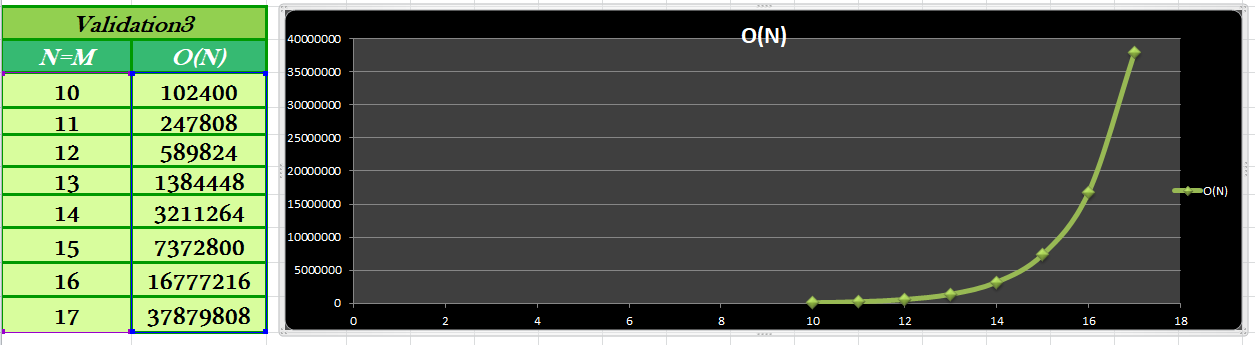
****

# Représentation Graphique :

# *Graphe du temps d’exécution en fonction de N.*



# *Graphe de la complexité théorique en fonction de N.*



(\*)Code Source du Programme complet :

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

**#include <math.h>**

**//Type Matrice**

**typedef struct Matrice Matrice ;**

**struct Matrice**

**{**

**long \*\*Pointer;**

**long n,m;**

**};**

**//Type Instanciation Liste Lin�aire Chain�e**

**typedef struct Instanciation Instanciation ;**

**struct Instanciation**

**{**

**long valeur;**

**Instanciation \*suiv;**

**};**

**void Affichage(Instanciation \*tete)**

**{**

**Instanciation \*P;**

**long i;**

**if(tete==NULL)**

**{**

**//printf("LA LISTE EST VIDE !");**

**}**

**else**

**{**

**P=tete;**

**//printf("[");**

**while(P!=NULL)**

**{**

**//printf("%d,",P->valeur);**

**P=P->suiv;**

**}**

**//printf("end]\n");**

**}**

**}**

**void Affiche(long T[],long N)**

**{**

**long i=0;**

**//printf("[");**

**for(i=0;i<N-1;i++)**

**//printf("%d,",T[i]);**

**//printf("%d]\n",T[N-1]);**

**;**

**}**

**//Primitives de Manipulation des Instanciations**

**Instanciation \*Ajouter(Instanciation \*I,long valeur) //O(4)**

**{**

**Instanciation \*P=(Instanciation \*)malloc(sizeof(Instanciation ));**

**P->valeur=valeur; P->suiv=I;**

**return P;**

**}**

**//Verifier : V�rifie si une Clause est vrai pour une instance Donn�e**

**long VerifierClause(long \*Clause,long nbLiterraux,long instance) // O(1+nbLiterraux)**

**{**

**//printf("\n\t\tV�rification : Clause = ");Affiche(Clause,nbLiterraux);**

**//printf("\n\t\tpour instanciation = %d.",instance);**

**long j,bit = instance;**

**for(j=nbLiterraux-1;j>=0;j--)**

**{**

**if(Clause[j] == bit%2)**

**{**

**//printf("\n\t\t\tVRAI pour instanciation = %d.",instance);**

**return 1;**

**}**

**bit=bit/2;**

**}**

**//printf("\n\t\t\tFAUSSE pour instanciation = %d.",instance);**

**return 0;**

**}**

**//Solutions : retournes les instanciations pour lesquelles une Clause est vrai**

**//Parmi les instanciations donn�es en param�tre**

**Instanciation \*Solutions(long \*Clause,long nbLiterraux ,Instanciation \*Instances) // O(m\*2^m)**

**{**

**//printf("\n\tRecherche de Solutions: pour Clause = ");Affiche(Clause,nbLiterraux);**

**//printf("\n\tparmi les instanciations = "); Affichage(Instances);**

**long i,bit, j;**

**Instanciation \*solutions =NULL ,\*P,\*Q;**

**if(Instances==NULL) //Pas d'instances en Entr�e**

**{//Tester pour toute la Table de V�rit� 2^m**

**long tailleTV = (long)pow(2,nbLiterraux);**

**for(i=0 ; i<tailleTV ; i++) //O(2^nbLitterraux \* (5+nbLitterraux))**

**{**

**if(VerifierClause(Clause,nbLiterraux,i))//O(1+nbLiterraux)**

**{solutions = Ajouter(solutions,i); }// O(4)**

**Affichage(solutions);**

**}**

**}**

**else // Instances != NULL il ya des valeurs � Tester**

**{   //Parcours de la Liste des instanciations**

**for(P=Instances; P != NULL;) //au pire cas O(2^nbLitterraux \* (5+nbLitterraux))**

**{**

**if(VerifierClause(Clause,nbLiterraux,P->valeur)) //O(1+nbLiterraux)**

**{   solutions = Ajouter(solutions,P->valeur); }// O(4)**

**Q=P ; P=P->suiv ; free(Q); //Lib�rer la cellue lu de Instances**

**}**

**}**

**//retourner la Liste des instances pour lesquelle Clause est VRAI**

**//printf("\n\tLes Solutions touvees = ");Affichage(solutions);**

**return solutions;**

**}**

**long Validation1(long \*\*Conjonction ,long nbClauses, long nbLiterraux,long valeur) //O(n\*m) - Omega(1)**

**{**

**long i;**

**Instanciation \*P; P=NULL;**

**Instanciation \*instances = Ajouter(P,valeur);**

**if(instances==NULL) return 0;**

**else**

**for(i=1;i<nbClauses;i++) //O(nbClauses\*nbLitterraux\*2^nbLitterraux)**

**{**

**instances = Solutions(Conjonction[i],nbLiterraux,instances); //O(nbLitterraux)**

**if(instances==NULL) return 0;**

**}**

**return 1;**

**}**

**long Validation2(long \*\*Conjonction ,long nbClauses, long nbLiterraux) //O(n\*m\*2^m) - Omega(2^m)**

**{**

**long i;**

**Instanciation \*instances = Solutions(Conjonction[0],nbLiterraux,NULL);**

**if(instances==NULL) return 0;**

**else**

**for(i=1;i<nbClauses;i++) //O(nbClauses\*nbLitterraux\*2^nbLitterraux)**

**{**

**instances = Solutions(Conjonction[i],nbLiterraux,instances); //O(nbLitterraux\*2^nbLitterraux)**

**if(instances==NULL) return 0;**

**}**

**return 1;**

**}**

**Instanciation \*Validation3(long \*\*Conjonction ,long nbClauses, long nbLiterraux) //O(n\*m\*2^m) - Omega(2^m)**

**{**

**long i;**

**Instanciation \*instances = Solutions(Conjonction[0],nbLiterraux,NULL);**

**if(instances==NULL) return NULL;**

**else**

**for(i=1;i<nbClauses;i++) //O(nbClauses\*nbLitterraux\*2^nbLitterraux)**

**{**

**instances = Solutions(Conjonction[i],nbLiterraux,instances); //O(nbLitterraux\*2^nbLitterraux)**

**if(instances==NULL) return NULL;**

**}**

**//printf("\nLa Solution Finale = ");Affichage(instances);**

**return instances;**

**}**

**//Matrice Triï¿½ Ordre dï¿½croissant**

**long \*\*Conjonction(long n)**

**{**

**long i,j,\*\*T=(long \*\*)malloc(n\*sizeof(long \*));**

**for(i=0;i<n;i++)**

**{**

**T[i] = (long \*)malloc(n\*sizeof(long ));**

**for(j=0;j<n;j++)**

**{T[i][j]= rand() % 2;}**

**}**

**return T;**

**}**

**double \*\*Calcul\_des\_Temps(double \*\*tab , long algorithme)**

**{**

**Instanciation \*P=NULL;**

**long j,verdict,\*\*M;**

**for(j=0 ; j<8 ; j++)**

**{**

**long \*\*Conj = Conjonction((long)tab[0][j]);**

**clock\_t begin = clock();**

**switch(algorithme)**

**{**

**case 1: verdict = Validation1(Conj,(long)tab[0][j],(long)tab[0][j],15); break;**

**case 2: verdict = Validation2(Conj,(long)tab[0][j],(long)tab[0][j]); break;**

**case 3: P = Validation3(Conj,(long)tab[0][j],(long)tab[0][j]); break;**

**}**

**clock\_t end = clock();**

**tab[1][j] = (double)(end - begin) / CLOCKS\_PER\_SEC;**

**if(P!=NULL)**

**tab[2][j] = 1;**

**else tab[2][j]=verdict;**

**}**

**return tab;**

**}**

**double \*\*Tableau\_de\_Valeurs23(void)**

**{**

**long i ;**

**double \*\*tab;**

**tab = (double \*\*)malloc(4\*sizeof(double \*));**

**for(i=0 ; i<3 ; i++) tab[i] = (double \*)malloc(8\*sizeof(double));**

**tab[0][0]=10;**

**tab[0][1]=11;**

**tab[0][2]=12;**

**tab[0][3]=13;**

**tab[0][4]=14;**

**tab[0][5]=15;**

**tab[0][6]=16;**

**tab[0][7]=17;**

**for(i=0 ; i<8 ; i++)tab[1][i] = 0 ;**

**return tab;**

**}**

**void Afficher\_Tableau\_de\_Valeurs(double \*\*tab)**

**{**

**long j,verdict;**

**for(j=0 ; j<10 ; j++)**

**{**

**verdict = (int)tab[2][j];**

**printf("N = %f \t T= %f \t , Satisfiable = %d \n",tab[0][j],tab[1][j],tab[2][j]);**

**}**

**}**

**void AfficherMatrice(long \*\*M,long n,long m)**

**{**

**long i,j;**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<m;j++)**

**printf("%d\t",M[i][j]);**

**printf("\n");**

**}**

**}**

**int \*\*MatrixToPointer(int \*M,int rows ,int cols)**

**{**

**int i,j,\*\*R=(int \*\*)malloc(rows\*sizeof(int \*));**

**for (i = 0; i < rows; i++) {**

**\*(R+i)=(int \*)malloc(cols\*sizeof(int ));**

**for (j = 0; j < cols; j++) {**

**R[i][j]= \*(M + i \* cols + j);**

**}**

**}**

**return R;**

**}**

**/\***

**Matrice LectureConjonction(void)**

**{**

**int n,m,i,j;**

**printf("Lecture de la Conjonction:\n");**

**printf("Entrez le nombre de clauses : ");**

**scanf("%d",&n); getchar();**

**printf("Entrez le nombre de litteraux des clauses : ");**

**scanf("%d",&m); getchar();**

**int \*\*Conjonction=(int \*\*)malloc(n\*sizeof(int \*));**

**for(i=0;i<n;i++) Conjonction[i]=(int \*)malloc(m\*sizeof(int));**

**printf("Remplir la matrice :\n");**

**for(i=0;i<n;i++)**

**{**

**printf("Saisie de la clause %d :\n",(i+1));**

**for(j=0;j<m;j++)**

**{**

**printf("\t%c = ",(j+97));**

**scanf("%d",&(Conjonction[i][j]));**

**getchar();**

**}**

**}**

**Matrice Mat;**

**Mat.m=m;**

**Mat.n=n;**

**Mat.Pointer=Conjonction;**

**return Mat;**

**}**

**\*/**

**int main(int argc, char \*argv[])**

**{**

**printf("Execution de Validation2 :\n");**

**Afficher\_Tableau\_de\_Valeurs(Calcul\_des\_Temps(Tableau\_de\_Valeurs23(),2));**

**printf("Execution de Validation3 :\n");**

**Afficher\_Tableau\_de\_Valeurs(Calcul\_des\_Temps(Tableau\_de\_Valeurs23(),3));**

**/\*  int C[4][4] ={**

**{1 ,0 ,1 ,0 },**

**{0 ,0 ,0 ,1 },**

**{1 ,1 ,0 ,0 },**

**{0 ,0 ,1 ,1 }**

**};**

**//Affichage(Validation(MatrixToPointer(&C[0][0],4,4),4,4));**

**//Version 1**

**printf("Version 1 :\n");**

**Matrice Conj = LectureConjonction();**

**printf("\nLa conjontion = \n");**

**AfficherMatrice(Conj.Pointer,Conj.n,Conj.m);**

**printf("\nEntrez l'instanciation : ");**

**int instance ; scanf("%d",&instance); getchar();**

**if(Validation1(Conj.Pointer,Conj.n,Conj.m,instance)) printf("\nLinstance %d est Solution.",instance);**

**else printf("\nLinstance %d n'est pas Solution.",instance);**

**//Version 1**

**printf("\nVersion 2 :\n");**

**Conj = LectureConjonction();**

**printf("\nLa conjontion = \n");**

**AfficherMatrice(Conj.Pointer,Conj.n,Conj.m);**

**if(Validation2(Conj.Pointer,Conj.n,Conj.m)) printf("\nLa Conjonction est Satisfiable.");**

**else printf("\nLa Conjonction n'est pas Satisfiable.");**

**//Version3**

**printf("\nVersion 2 :\n");**

**Conj = LectureConjonction();**

**printf("\nLa conjontion = \n");**

**AfficherMatrice(Conj.Pointer,Conj.n,Conj.m);**

**Affichage(Validation3(Conj.Pointer,Conj.n,Conj.m));**

**\*/**

**getchar();**

**return 0;**

**}**