Master 1 d'informatique fondamentale
École Normale Supérieure de Lyon

Vincent DELAITRE

19 Avril 2009

DM de Algorithmique Arithmétique

1 Description des fichiers et du fonctionnement

- **main.c** : Contient la fonction 'main'. Parse les options de la ligne de commande puis lance l'algorithme pour générer la multiplication. Les options possibles sont **-M nombre** où 'nombre' est la constante pour laquelle on souhaite générer un algorithme de multiplication. Par défaut, cet algorithme est généré dans un fichier MultX.c où 'X' est le nom de l'algorithme utilisé. Pour changer le fichier de sortie, utiliser l'option **-f fichier**.
- **makefile** : Faire 'make' pour plus d'informations. 'make clean' fait le ménage.
- **test.c** : Programme de test des fichiers générés. Après compilation du programme de test de multiplication par M (cf 'make'), taper './MultX N' où 'X' est le nom de l'algorithme utilisé et 'N' un nombre pour afficher le résultat de $M * N$.
- **generate.c**, **generate.h** : Fonctions associées à la génération des fichiers.
- **naif.c**, **booth.c**, **bernstein.c**, **lefevre1.c**, **lefevre2.c** : Algorithmes pour la multiplication par une constante.
- **common_lefevre.c**, **common_lefevre.h** : Fonctions communes à 'lefevre1.c' et 'lefevre2.c'.

Question 1

L'implémentation se trouve dans le fichier "bernstein.c".

J'ai choisi de représenter l'ensemble des possibilités par un arbre dont chaque noeud interne a quatres fils. L'utilisation de Branch & bound permet de stopper l'exploration de l'arbre dès que la complexité de la solution en cours dépasse celle de la meilleure déjà obtenue. Une fois l'arbre construit, on le parcourt en sélectionnant les meilleures branches pour générer la multiplication. La vitesse d'exécution est bonne pour une complexité exponentielle et la génération du programme est instantanée.un vairable

Question 2

Soit $\mathbf{M}$ un entier. On note $\mathbf{T}(\mathbf{M})$ l'arbre binaire correspondant à la trace de l'exécution de l'algorithme de Lefèvre sur M : un noeud interne est un appel récursif sur M' et R (où $M = \pm M' \pm 2^k \cdot M' \pm R$) et une feuille est un appel à l'algorithme de Booth. On note $\#\mathbf{T}(\mathbf{M})$ le nombre de feuilles de $T(M)$. Enfin, on note $\mathbf{s}(\mathbf{x})$ le nombre de bit non nuls dans l'encodage de Booth modifié de x.

Montrons que $\#T(M) \leq \frac{s(M)}{2} + 1$ par induction structurelle sur $T(M)$. Si $T(M)$ est une feuille, la propriété est vérifiée. Sinon, on a appelé l'algorithme de Lefèvre récursivement sur M' et R, avec $s(M') \geq 2$. Ainsi, $T(M)$ se décompose en deux sous-arbres $T(M')$ et $T(R)$. Par hypothèse d'induction :

$$\#T(M) = \#T(M') + \#T(R) \leq \frac{s(M')+s(R)}{2} + 2 \leq \frac{2 \cdot s(M')+s(R)}{2} + 1.$$

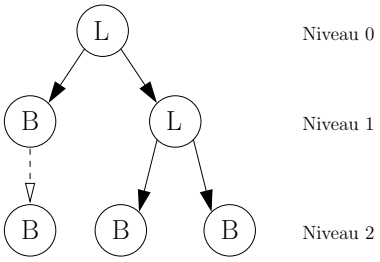
Or, d'après l'algorithme de Lefèvre, on a choisi M' de tel facon que M' et $2^k \cdot M'$ ne se chevauchent pas (si le i$^{\text{ème}}$ bit de l'encodage de Booth de l'un est non nul, le i$^{\text{ème}}$ bit de l'autre est nul) et le reste est l'ensemble des bits non nuls n'apparaissant ni dans M' ni dans $2^k \cdot M'$. On a donc $s(M) = 2 \cdot s(M') + s(R)$ ce qui montre la propriété.

D'après cette propriété, le nombre de noeuds internes de $T(M)$ est donc inférieur à $s(M)/2$. Il y a donc O(s(M)) appels récursifs à l'algorithme de Lefèvre et O(s(M)) à l'algorithme de Booth. Notons n la taille de M. Comme l'encodage de Booth modifié fait

la même taille que M, on a $s(M) = O(n)$. L'algorithme de Booth est linéaire en la taille de l'entrée, il s'exécute en $O(n)$. L'algorithme de Lefèvre effectue une recherche de motif en $O(n^2)$ (on cherche des motifs pour toute taille d). On obtient une complexité totale en $O(s(M) \cdot n^2 + s(M) \cdot n) = O(n^3)$. L'algorithme de Lefèvre est polynomial en la taille de M.

Question 3

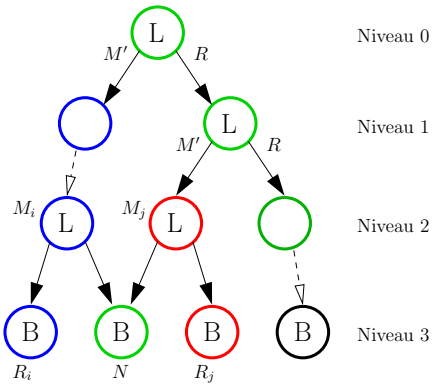
J'ai implémenté directement les fonctions qui allaient me servir pour l'algorithme de Lefèvre amélioré et je les ai appliquées au cas particulier de l'algorithme de Lefèvre. J'ai représenté un M_i par la structure *node_struct* de *common_lefevre.h*. L'ensemble des M_i est représenté par la structure *level_struct*. Le déroulement du calcul se fait par étape, en faisant grandir l'ensemble des M_i. Pour cela, on cherche simplement des motifs dans le premier de la liste. Si on n'en trouve pas on passe au second, jusqu'à échouer sur tout les M_i.



Les noeuds générés par Lefèvre sont marqués (L), ceux générés par Booth sont marqués (B). Cette représentation n'est pas optimale car le graphe est en fait un arbre binaire, mais elle convient tout à fait pour l'algorithme amélioré.

Question 4

Cette fois pour passer au niveau suivant, il faut comparer tous les (M_i, M_j) du niveau pour trouver le meilleur motif. Si on ne trouve aucun motif, on calcule tous les noeuds du niveau en utilisant Booth comme dans le dessin ci-dessous :



Il reste ensuite à colorer le graphe pour assigner une variable à chaque noeud (fonction *assign_vars* dans *common_lefevre.c*). Pour cela, on part du bas, et on colorie R_i, R_j et N. Ensuite, récursivement, on colorie d'abord M_i et M_j de la même couleur respectivement que R_i et R_j du niveau du dessous. On colorie ensuite R_i, R_j et N du niveau actuel de telle manière qu'ils aient une couleur différente de celle de chaque noeud du niveau. On obtient ainsi le coloriage précédent.

Question 5

Nombre d'additions ou de soustractions nécessaires pour chaque algorithme :

	AlgoNaif	AlgoBooth	AlgoBernstein	AlgoLefevre1	AlgoLefevre2
45	3	3	2	2	2
113	3	2	2	2	2
341	4	4	3	3	3
861	6	5	4	4	4
1717	6	5	4	4	4
20061	8	6	5	5	5
543413	8	9	8	5	5
47804853381	13	15	10	9	11
799144290325165979	36	22	31	16	15
905502432259640355	25	25	21	17	16
391745696951556693	29	24	25	15	13

Question 6

Il serait intéressant de regarder ce que donnerait l'algorithme de Bernstein appliqué au code de Booth et non au développement binaire de M. Mais peut-être que cela ne donnerait aucun résultat car les 1 sont trop dispersés. Concernant la recherche de motif, je n'ai pas vraiment de suggestions : l'algorithme de Lefèvre est efficace, je ne vois pas comment l'améliorer.