TD07 - COMPILATEUR DE QUADTREES

On s'intéresse ici à la représentation d'images par des quadtrees, utilisés dans l'analyse, la compression et la synthèse d'images. Une image est dite *simple* si elle est de couleur uniforme. Étant donné un medium graphique (écran, imprimante, etc.) carré, qui mesure 1024×1024 pixels, on peut représenter une image par une structure d'arbre, appelé *quadtree*, dans laquelle chaque nœud a exactement 0 ou 4 fils. L'idée de base est la suivante : si une image associée à un nœud n'est pas simple, on la découpe en 4 morceaux de même taille et les fils du nœud correspondant sont les *quadtrees* associés aux 4 sous-images. Si au contraire l'image est simple, alors elle est caractérisée par une information unique, sa couleur ; dans ce cas, le nœud porte cette information et n'a pas de fils. Par exemple, si on parcourt les 4 quarts d'une image en décrivant un "Z", et si on suppose que les dix régions en lesquelles on a découpé le carré de la figure 1 sont simples, de couleurs `r`, `g` et `b`, alors cette figure peut être représentée par le *quadtree* de la figure 2. Un quadtree peut alors être représenté sous la forme d'une expression parenthésée qui décrit la structure de l'arbre correspondant. Le quadtree de la figure 2 est représenté sous la forme suivante : `(rbg((bbrg)grb))`

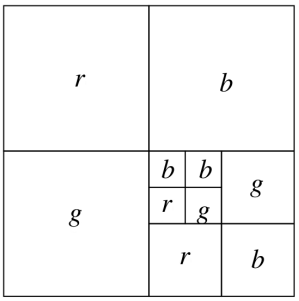

Figure 1

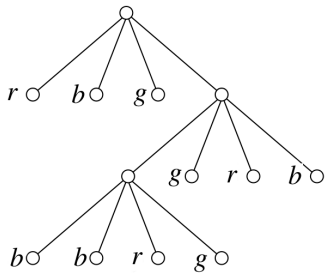

Figure 2

(1) Écrivez une grammaire G des quadtrees considérant l'alphabet des couleurs $\{\texttt{r}, \texttt{g}, \texttt{b}\}$.

Correction:

$Q \rightarrow$ ($Q\ Q\ Q\ Q$) | `r` | `g` | `b`

(2) Dessinez l'arbre de dérivation du quadtree `(((bbbr)(gbrg)rg)(bbrb)rb)`

(3) Écrivez une grammaire attribuée pour compter le nombre s de sous-carrés simples qui composent un quadtree. La veleur de cet attribut à la racine de l'arbre de dérivation pour l'exemple de la figure 1 est $s = 10$.

Correction:

règles			actions
Q	$\rightarrow$	($Q_1\ Q_2\ Q_3\ Q_4$)	$Q.s = Q_1.s + Q_2.s + Q_3.s + Q_4.s$
Q	$\rightarrow$	`r` | `g` | `b`	$Q.s = 1$

On dispose par ailleurs d'un périphérique graphique qui reconnaît l'unique instruction élémentaire

`RECT` $x_0\ y_0\ x_1\ y_1\ c$

dans laquelle x_0, y_0, x_1 et y_1 sont des nombres entiers. L'exécution de cette instruction produit le remplissage du rectangle $R = \{(x, y) \mid x_0 \leq x < x_1 \quad y_0 \leq y < y_1\}$ avec la couleur c. Un programme pour notre périphérique, appelé *programme-machine*, est une suite de telles instructions.

(4) En vous aidant de la figure 1, écrivez le programme-machine du *quadtree* `(rbg((bbrg)grb))`.

Correction:

```
RECT    0    0  512  512 r
RECT  512    0 1024  512 b
RECT    0  512  512 1024 g
RECT  512  512  640  640 b
RECT  640  512  768  640 b
RECT  512  640  640  768 r
RECT  640  640  768  768 g
RECT  768  512 1024  768 g
RECT  512  768  768 1024 r
RECT  768  768 1024 1024 b
```

(5) Écrivez une grammaire attribuée pour les attributs x_0, y_0 et t contenant les coordonnées x et y du coin supérieur gauche du carré et la taille t de ses côtés.

Correction:

règles	actions
$Q \rightarrow$ (Q_1 Q_2 Q_3 Q_4)	$Q_1.t = Q.t/2$ $Q_1.x_0 = Q.x_0$ $Q_1.y_0 = Q.y_0$
	$Q_2.t = Q.t/2$ $Q_2.x_0 = Q.x_0 + Q_2.t$ $Q_2.y_0 = Q.y_0$
	$Q_3.t = Q.t/2$ $Q_3.x_0 = Q.x_0$ $Q_3.y_0 = Q.y_0 + Q_3.t$
	$Q_4.t = Q.t/2$ $Q_4.x_0 = Q.x_0 + Q_4.t$ $Q_4.y_0 = Q.y_0 + Q_4.t$
$Q \rightarrow$ r	*print*(RECT $Q.x_0$ $Q.y_0$ $(Q.x_0 + Q.t)$ $(Q.y_0 + Q.t)$ r)
$Q \rightarrow$ g	*print*(RECT $Q.x_0$ $Q.y_0$ $(Q.x_0 + Q.t)$ $(Q.y_0 + Q.t)$ g)
$Q \rightarrow$ b	*print*(RECT $Q.x_0$ $Q.y_0$ $(Q.x_0 + Q.t)$ $(Q.y_0 + Q.t)$ b)

(6) On souhaite connaître le nombre de pixels de chacune des trois couleurs. Pour cela on crée les attributs n_R, n_G et n_B qui indiquent le nombre de pixels de chaque couleur. Écrivez une grammaire attribuée pour les attributs n_R, n_G et n_B.

Correction:

règles	actions
$Q \rightarrow$ (Q_1 Q_2 Q_3 Q_4)	$Q.n_R = Q_1.n_R + Q_2.n_R + Q_3.n_R + Q_4.n_R$
	$Q.n_G = Q_1.n_G + Q_2.n_G + Q_3.n_G + Q_4.n_G$
	$Q.n_B = Q_1.n_B + Q_2.n_B + Q_3.n_B + Q_4.n_B$
$Q \rightarrow$ r	$Q.n_R = Q.t \times Q.t$ $Q.n_G = 0$ $Q.n_B = 0$
$Q \rightarrow$ g	$Q.n_R = 0$ $Q.n_G = Q.t \times Q.t$ $Q.n_B = 0$
$Q \rightarrow$ b	$Q.n_R = 0$ $Q.n_G = 0$ $Q.n_B = Q.t \times Q.t$

(7) Écrivez en C un analyseur $LL(1)$ qui reconnaît si un texte est un *quadtree* correct.

Correction:

```
#include <stdio.h>
#include <stdlib.h>
#define erreur() printf("Erreur de syntaxe : quadtree non reconnu\n"); exit(-1)
char c;        // symbole courant, tête de lecture
FILE *input ;  // bande de lecture
void q() {
  if( c == '(' ) {
    c = getc( input );
    q();       q();       q();       q();
    if( c == ')' ) {    c = getc( input );    }
    else{     erreur();     }
  }
  else if( c == 'r' || c == 'g' || c == 'b' ) {     c = getc( input );     }
  else{     erreur();     }
}
int main( int argc, char *argv[] ) {
  input = fopen( argv[1], "r" );
  c = getc( input );  // initialiser tête de lecture
  q();                // axiome, analyse descendante
}
```

(8) Transformez l'analyseur précédent en un compilateur produisant le langage-machine du périphérique ci-dessus à partir d'une quadtree. Notez que $x_1 = x_0 + taille$ et $y_1 = y_0 + taille$.

Correction:

Cf. dossier `td07-quadtrees`, deux versions - `quadtree-min.c` sans affichage graphique et `quadtree.c` avec affichage graphique (requiert `sudo apt-get install libx11-dev`)

```
#include <stdio.h>
#include <stdlib.h>
#define erreur() printf("Erreur de syntaxe : quadtree non reconnu\n"); exit(-1)
char c;        // symbole courant, tête de lecture
FILE *input ;  // bande de lecture

void q( int t, int x0, int y0 ) {
  if( c == '(' ) {
    c = getc( input );
    q( t / 2, x0, y0 );
    q( t / 2, x0 + t / 2, y0 );
    q( t / 2, x0, y0 + t / 2 );
    q( t / 2, x0 + t / 2, y0 + t / 2 );
    if( c == ')' ) { c = getc( input ); }
    else{ erreur(); }
  }
  else if( c == 'r' || c == 'g' || c == 'b' ) {
    printf( "RECT %4d %4d %4d %4d %c\n", x0, y0, x0+t, y0+t, c );
    c = getc( input );
  }
  else { erreur(); }
}

int main( int argc, char *argv[] ) {
  input = fopen( argv[1], "r" );
input = fopen( argv[1], "r" );
  c = getc( input );  // initialiser tête de lecture
  q( 1024, 0, 0 );    // axiome, analyse descendante
  return 0;
}
```