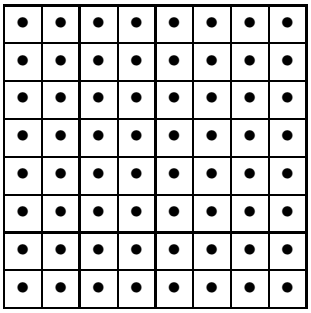
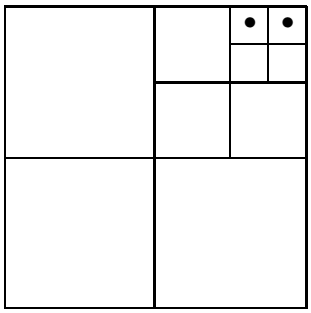
Corrigé feuille d'exercices d'informatique n°11

Question 1a.





Question 1b.

Un arbre quaternaire de profondeur p contenant au plus 4^p feuilles, on a $4^p \leqslant N$ soit $p \leqslant \frac{\ln N}{\ln 4}$ d'où $f(N) = \left\lceil \frac{\ln N}{\ln 4} \right\rceil$. Cette minoration est optimale,

– dans le sens de l'énoncé : pour $p_0 \in \mathbb{N}$ il existe $N_0 = 4^{p_0}$ tel que $f(N_0) = p_0$ et il existe un quadtree à N_0 corps de profondeur p_0 obtenu comme au **1a** en subdivisant l'univers en 4^{p_0} carrés et en plaçant un corps dans chaque carré ;

– dans un sens plus général : pour $N \in \mathbb{N}$ il existe un quadtree à N corps de profondeur $p = f(N)$ obtenu en divisant l'univers en 4^p carrés et en plaçant les N corps d'une manière quelconque à raison d'un seul corps par carré (il y a alors au moins un groupe de 4 carrés contenant deux corps car $4^{p-1} < N$).

Question 1c.

En adaptant l'exemple du **1a**, on voit qu'il existe des quadtrees de profondeur arbitrairement grande contenant seulement deux corps, et de même, il existe des quadtrees à $N \geq 2$ corps de profondeur arbitrairement grande.

Question 1d.

On pense que l'auteur demande un *majorant* de la profondeur et non une *borne supérieure*. Dans un quadtree de profondeur p il y a au moins un carré de côté $\frac{D_{\mathcal{U}}}{2^{p-1}}$ contenant deux corps, donc $\delta \leqslant \frac{D_{\mathcal{U}}}{2^{p-1}}$ soit $p \leqslant 1 + \frac{\ln(D_{\mathcal{U}}/\delta)}{\ln 2}$.

Question 1e.

Le plus petit nombre strictement positif représentable est 2^e donc $\delta \geq 2^e$. Le plus grand nombre représentable est $2^e(2^m - 1)$ donc $D_{\mathcal{U}} \leqslant 2^e(2^m - 1)$. On en déduit $\frac{D_{\mathcal{U}}}{\delta} \leqslant 2^m - 1$ et donc $p \leqslant 1 + \frac{\ln(2^m - 1)}{\ln 2}$ ce qui implique $p \leqslant m$ car p est entier.

Corrigé en Caml

Question 2abc.

```
let add v1 v2 = {x=v1.x+.v2.x; y=v1.y+.v2.y};;
let sub v1 v2 = {x=v1.x-.v2.x; y=v1.y-.v2.y};;
let scal m u  = {x=m*.u.x; y=m*.u.y};;
let carre u   = u.x*.u.x +. u.y*.u.y;;
```

Question 3.

On numérote ainsi :

2	0
3	1

Question 3ab.

```
let indice_fille p_c p =
  (if p_c.y > p.y then 1 else 0) + 2*(if p_c.x > p.x then 1 else 0);;

let position_fille p_c taille i =
  let t = taille/.4.0 in
  match i with
  | 0 -> add p_c {x =   t; y =   t}
  | 1 -> add p_c {x =   t; y = -.t}
  | 2 -> add p_c {x = -.t; y =   t}
  | _ -> add p_c {x = -.t; y = -.t}
;;
```

Question 4a.

```
(* insère un corps dans un quadtree *)
let rec insere_corps corps arbre p_c taille = match arbre with
| Vide -> Feuille(corps)
| Noeud(cell) ->
   insere_dans_filles corps cell.filles p_c taille;
   arbre
| Feuille(c) ->
   let f = make_vect 4 Vide in
   insere_dans_filles c     f p_c taille;
   insere_dans_filles corps f p_c taille;
   Noeud{cm_mass=0.0; cm_pos=vecteur_nul; filles=f}

(* insère un corps dans un tableau de filles *)
and insere_dans_filles corps filles p_c taille =
  let i = indice_fille   p_c corps.pos in
  let p = position_fille p_c taille i  in
  filles.(i) <- insere_corps corps filles.(i) p (taille/.2.0)
;;
```

Question 4b.
On dit qu'un appel `insere_corps corps arbre p_c taille` est licite si :

☐ – `corps` représente un corps dont les coordonnées vérifient

$$\texttt{p\_c.x} - \texttt{taille}/2 \leqslant \texttt{corps.pos.x} < \texttt{p\_c.x} + \texttt{taille}/2$$
$$\texttt{p\_c.y} - \texttt{taille}/2 \leqslant \texttt{corps.pos.y} < \texttt{p\_c.y} + \texttt{taille}/2 \text{ ;}$$

– `arbre` représente un quadtree adaptatif cohérent centré en `p_c` de taille `taille`, c'est à dire que chaque corps vérifie les inégalités précédentes à chaque niveau de division, et qu'aucune cellule n'est inutilement divisée ;

– la position de `corps` est distincte de celles de tous les corps contenus dans `arbre`.

On démontre alors à la fois la correction et la terminaison d'un appel licite par récurrence sur `taille`. Soit δ la plus courte distance entre deux corps de `arbre` ou entre un corps de `arbre` et `corps`, telle que définie en **1.c**, avec la convention $\delta = +\infty$ si l'arbre est vide.

☐
– Si $\texttt{taille} < \delta$: par définition de δ, il n'y a aucun corps dans `arbre` situé à une distance inférieure à `taille` de `corps`, donc `arbre` = `Vide`. Dans ce cas, l'appel termine et retourne le résultat correct,?Feuille(corps)?.

– Si la correction et la terminaison sont prouvées pour tout appel licite tel que $\texttt{taille} < t$, considérons un appel licite avec $\texttt{taille} < 2t$.

☐ – Si `arbre` = `Vide` il y a terminaison et correction du résultat.

– Si `arbre` est un nœud, `insere_dans_filles` sélectionne la bonne cellule où poursuivre l'insertion et génère un appel licite à `insere_corps` avec un côté $\texttt{taille}/2 < t$. Cet appel termine et renvoie un résultat correct par hypothèse de récurrence, et l'arbre retourné est placé dans la bonne fille, il y a donc encore terminaison et correction du résultat.

– Si `arbre` est une feuille, le premier appel à `insere_dans_files` divise la cellule en 4 et place le corps déjà présent dans le bon quadrant de manière évidente. Le deuxième appel à `insere_dans_files` produit alors un appel récursif à `insere_corps` licite, que les corps soient dans le même quadrant ou non. Comme précédement, on conclut à la terminaison et à la correction du résultat.

Question 4c.
D'après le raisonnement précédent, `insere_corps` effectue un nombre borné d'opérations avant de s'appeler éventuellement à travers `insere_dans_files` pour le même corps et le paramètre $\texttt{taille}/2$ car le premier appel à `insere_dans_files` dans le cas `Feuille(c)` a une complexité constante. La complexité de l'insertion d'un corps dans un arbre est donc dominée par la profondeur de l'arbre, elle même majorée par m. La complexité de l'insertion de N corps dans un arbre vide est dominée par mN, donc est un $O(N)$ puisque m est constant. Ce majorant est optimal car chaque corps est examiné donc donne lieu à au moins une opération.

Question 5.

```
$(* ajuste les champs$ cm_mass $et$ cm_pos $et retourne *)$
$(* le couple (masse,somme pond{\'e}r{\'e}e des positions)  *)$
let rec mass_pos(arbre) = match arbre with
| Vide        -> (0.0, vecteur_nul)
| Feuille(c)  -> (c.mass, scal c.mass c.pos)
| Noeud(cell) ->
    let m = ref(0.0) and p = ref(vecteur_nul) in
    for i=0 to 3 do
      let (mi,pi) = mass_pos(cell.filles.(i)) in
      m := !m +. mi;
      p := add !p pi
    done;
    cell.cm_mass <- !m;
    cell.cm_pos  <- scal (1.0/. !m) !p;
    (!m,!p)
;;

let barycentres(arbre) = let _ = mass_pos(arbre) in ();;
```

Question 6a.

```
$(* fonction auxilliaire : distance num{\'e}rique et vectorielle *)$
let distance a b = let d = sub b a in (sqrt(carre(d)),d);;

$(* fonction auxilliaire : acc{\'e}l{\'e}ration de la masse$ m $plac{\'e}e en$ b $sur$ a $*)$
let accel a b m = let (r,d) = distance a b in scal (m/.r/.r/.r) d;;

$(* acc{\'e}l{\'e}ration de l'univers sur un corps *)$
let rec grav_approx pos arbre taille = match arbre with
| Vide        -> vecteur_nul
| Feuille(c)  -> if c.pos = pos then vecteur_nul else accel pos c.pos c.mass
| Noeud(cell) ->
    let (r,_) = distance pos cell.cm_pos in if taille < r*.theta
    then accel pos cell.cm_pos cell.cm_mass
    else begin
      let acc = ref(vecteur_nul) and t = taille/.2.0 in
      for i=0 to 3 do acc := add !acc (grav_approx pos cell.filles.(i) t) done;
      !acc
    end
;;
```

Question 6b.

Pour qu'une cellule de taille D soit subdivisée, il faut que son centre de masse soit situé à une distance inférieure à D/θ du corps c considéré, donc cette cellule doit être entièrement incluse dans le disque fermé de centre c et de rayon $D/\theta + D\sqrt{2}$ car le centre de masse est géographiquement inclus dans la cellule si tous les corps le sont. Comme toutes les cellules de même taille sont deux à deux disjointes, la somme des aires des cellules divisables est majorée par l'aire du disque précédent, donc le nombre de ces cellules est majoré par $K(\theta) = \lfloor \pi(1/\theta + \sqrt{2})^2 \rfloor$.

Question 6c.

Un appel `grav_approx c arbre t` génère au plus $K(\theta)$ appels `grav_approx c arbre' (t/2^k)` pour tout entier k d'après ce qui précède et la profondeur de l'arbre étant majorée par m, le nombre total d'appels à `grav_approx` générés est majoré par $mK(\theta)$ qui est une constante. Chacun de ces appels effectue, hors appels récursifs, un nombre borné d'opérations, donc la complexité de `grav_approx` est bornée et celle de `ajuste_approx` est un $O(N)$.

Programmes en Pascal

Les prototypes des fonctions demandées ne sont pas tous conformes à la syntaxe du Pascal en vigueur dans les classes préparatoires : une fonction ne peut retourner un résultat non scalaire et les descriptions de type dans une en-tête de procédure ou de fonction (`function cree_feuille(c:1..N):arbre`) ne sont pas autorisées, il faut donner un nom au type ?1..N?. Ces restrictions ne sont pas supportées par tous les compilateurs Pascal, en particulier le traducteur ?p2c? de la Free Software Foundation a permi de compiler avec succès toutes les procédures et fonctions qui suivent.

Question 2abc.

```
procedure add(u,v : vecteur; var r: vecteur);
begin
   r.x := u.x + v.x;
   r.y := u.y + v.y
end;

procedure sub(u,v: vecteur; var r: vecteur);
begin
   r.x := u.x - v.x;
   r.y := u.y - v.y
end;

procedure scal(m: real; u: vecteur; var r: vecteur);
begin
   r.x := m*u.x;
   r.y := m*u.y
end;

function carre(u : vecteur): real;
begin
   carre := u.x*u.x + u.y*u.y
end;
```

Question 3ab.

```
function indice_fille(p_c,p: vecteur): integer;
var res: integer;
begin
   if p_c.y > p.y then res := 1 else res := 0;
   if p_c.x > p.x then res := res + 2;
   indice_fille := res
end;

function position_fille(p: vecteur; taille: real; i: integer): vecteur;
var t: real; v: vecteur;
begin
```

```
   t := taille/4.0;
   if i >= 2 then v.x := -t else v.x := t;
   if odd(i) then v.y := -t else v.y := t;
   add(p,v,v);
   position_fille := v
end;
```

Question 4a.

```
procedure insere_dans_filles(c: 1..N; var a: arbre; p: vecteur; taille: real);
forward;

procedure insere_corps(c: 1..N; var a: arbre; p: vecteur; taille: real);
var b: arbre;
begin
   if a = nil then a := cree_feuille(c)
   else if a^.nature = Noeud then insere_dans_filles(c,a,p,taille)
   else begin
      b := cree_noeud(0.0,vecteur_nul,nil,nil,nil,nil);
      insere_dans_filles(a^.corps,b,p,taille);
      insere_dans_filles(c,b,p,taille);
      dispose(a);
      a := b
   end
end;

procedure insere_dans_filles;
var i: integer; q: vecteur;
begin
   i := indice_fille(p,univers[c].pos);
   q := position_fille(p,taille,i);
   insere_corps(c,a^.filles[i],q,taille/2)
end;
```

Question 5.

```
procedure mass_pos(a: arbre; var m: real; var pos: vecteur);
var i: integer;
   mi: real;
 posi: vecteur;
begin
   if a=nil then begin m := 0.0; pos := vecteur_nul end
   else if a^.nature = Feuille then begin
      m := univers[a^.corps].mass;
      scal(m,univers[a^.corps].pos,pos)
   end
   else begin
      m := 0.0;
      pos := vecteur_nul;
      for i:=0 to 3 do begin
         mass_pos(a^.filles[i],mi,posi);
         m := m + mi;
         add(pos,posi,pos)
      end;
      a^.cm_mass := m;
```

```
      scal(1.0/m,pos,a^.cm_pos)
   end
end;

procedure barycentres(a: arbre);
var m : real;
   pos: vecteur;
begin
   mass_pos(a,m,pos)
end;
```

Question 6a.

```
procedure distance(a,b: vecteur; var r: real; var d: vecteur);
begin
   sub(a,b,d);
   r := sqrt(carre(d))
end;

function accel(a,b: vecteur; m: real): vecteur;
var r : real;
   d  : vecteur;
begin
   distance(b,a,r,d);
   scal(m/r/r/r,d,d);
   accel := d
end;

function grav_approx(pos: vecteur; a: arbre; taille: real): vecteur;
var r,t: real;
      p: vecteur;
      i: integer;
begin
   if a=nil then grav_approx := vecteur_nul
   else if a^.nature = Feuille then begin
      if pos = univers[a^.corps].pos then grav_approx := vecteur_nul
      else grav_approx := accel(pos,univers[a^.corps].pos,univers[a^.corps].mass)
   end
   else begin
      distance(pos,a^.cm_pos,r,p);
      if taille < r*theta
      then grav_approx := accel(pos,a^.cm_pos,a^.cm_mass)
      else begin
         t := taille/2.0;
         p := vecteur_nul;
         for i:= 0 to 3 do add(p,grav_approx(pos,a^.filles[i],t),p);
         grav_approx := p
      end
   end
end;
```

Programmation

Voir le fichier `exo_info_11(corrige).ml` pour une implémentation complète du mouvement à N corps, avec résolution approchée du système différentiel par une méthode d'Euler d'ordre 2. L'exécution de ce programme fournit le code postscript de la figure suivante où l'on a représenté la division de l'univers en quadtree adaptatif un dixième de seconde avant la collision entre les corps rouge et bleu ...