CORRIGE DE LA COMPOSTION D'INFORMATIQUE POLYTECHNIQUE 97

Première partie : Quadtrees

1.a)Pour qu'un univers soit de profondeur $n^3 1$, il faut et il suffit qu'il soit subdivisé en quatre cellules, et que l'une au moins de ces cellules soit de profondeur exactement égale à $n - 1$, tandis que les autres sont de profondeur inférieure ou égale à $n - 1$.

Le nombre maximal de corps contenus dans un tel univers est donc atteint lorsque les quatre sous-univers sont exactement de taille $n - 1$, et comportent le nombre maximal de corps pour un tel univers. Le nombre maximal de corps pour un univers de profondeur 0 étant 1, on obtient par récurrence un nombre maximal de 4^ncorps pour un univers de profondeur n. Pour $n = 3$, cela donne 64 corps, répartis dans autant de cellules :

Le nombre minimal de corps pour qu'il y ait subdivision est minoré par 2. Or, pour tout $n^3 1$, il existe un univers de profondeur n ne comportant que 2 corps, obtenu (par exemple) en plaçant un corps dans la cellule de côté $\frac{D_u}{2^n}$ qui se trouve « en haut à droite », et un second corps dans la cellule de même taille qui se trouve juste en-dessous, comme illustré dans la figure ci-contre, pour n = 3€:

b)Soit un quadtree comportant N corps. $! $k\hat{I}N, 4^{k-1} < N \leq 4^k$.
Or chaque corps est en définitive l'unique occupant d'une cellule, donc il faut que la représentation de l'univers comporte au moins $4^{k-1} + 1$cellules. Or on a vu dans la question précédente que pour une profondeur donnée n, il ne pouvait y avoir plus de 4^ncorps. Il en résulte que la profondeur de notre quadtree vaut au moins k. Ceci nous fournit pour $N^3 1$ la minoration $profondeur\ (N)^3 k^3 E\left(log_4 N\right)$. Montrons qu'elle est atteinte :

Soit $p_0 \hat{I} N$. Le quadtree à 4^{p_0}corps dont la représentation structurée est formée d'un quadrillage homogène en autant de cellules (comme dans le premier graphique de la question a)) est de profondeur p_0, avec de plus $E\left(log_4 4^{p_0}\right) = p_0$.
On a donc en conclusion :

$$\underline{N^3 1, f\ left\ (N\ right\ ) = E\ left\ (log\ rSub\ \{\ size\ 8\{4\}\ \}\ N\ right\ )\}\ \}\ \{}$$

c)Nous avons établi en a) qu'il existait pour tout $n^3 1$un quadtree de profondeur n et contenant 2 corps exactement. Soit $N^3 2$. Soit $n^3 1$. Le quadtree de profondeur n dont la représentation structurée est figurée par le deuxième graphique de la question a) peut être modifié en remplaçant le corps de la cellule supérieure par un sous-univers comportant $N - 1$des N corps de l'univers initial. Ceci représente un état intermédiaire de la subdivision en sous-cellules de l'univers initial, dont l'état final fournira un quadtree de profondeur au moins égale à n. Donc on peut pour tout $N^3 2$ construire un univers à N corps représenté par un quadtree de profondeur arbitrairement grande. D'où :

Il n'existe pas de majoration fonction de N de la profondeur d'un quadtree à N corps

d)On reconnaît en $max\left(\left|p_c.x - p_{c'}.x\right|, \left|p_c.y - p_{c'}.y\right|\right)$ la norme sup de la différence des vecteurs position des corps c et c'. d représente donc le minimum des distances (au sens de cette norme) mutuelles des corps de l'univers. Le fait de supposer $d > 0$est une hypothèse bizarre voulant probablement dire qu'on suppose que deux corps distincts ne peuvent pas se trouver à la même position, ce qui semble raisonnable ! (quoique... avec la physique moderne...)

Soit $n^3 1$. Supposons la grosse cellule initiale maillée en 4^ncellules de même coté $\frac{D_U}{2^n}$. Si deux corps sont placés dans la même cellule, cela impose que leur distance au sens de la norme sup soit inférieure ou égale à $\frac{D_U}{2^n}$. Donc, si $d > \frac{D_U}{2^n}$, chaque maille ne pourra contenir qu'au plus un corps. Donc la profondeur d'un quadtree pour lequel $d > \frac{D_U}{2^n}$ est inférieure ou égale à n.

Mais pour d fixé, il existe un unique entier $n^3 1$tel que $\frac{D_U}{2^n} < d \leq \frac{D_U}{2^{n-1}}$, et par conséquent la profondeur du quadtree correspondant sera inférieure ou égale à n, avec $n = 1 + E\left(log_2 \frac{D_U}{d}\right)$. On en déduit que :

$$\underline{la\ profondeur\ d©un\ quadtree\ est\ au\ plus\ gale\ \hat{}\ 1 + E\left(log_2 \frac{D_U}{d}\right)}$$

e)L'hypothèse sur la représentation en machine des grandeurs homogènes à une distance se traduit par le fait qu'une telle grandeur g vérifie $1'2^e \leq g \leq \left(2^m - 1\right)2^e$. Donc $d^3 2^e$ et $D_U \leq \left(2^m - 1\right)2^e$.

Il en résulte que $\frac{D_U}{d} \leq 2^m - 1$, ce qui entraîne $1 + E\left(log_2 \frac{D_U}{d}\right) \leq m$.

<u>La profondeur des quadtrees manipulés par notre programme ne peut excéder m</u>

Deuxième partie - Caml : Construction du quadtree

2.a)
```
let add u v = {x = u.x +. v.x ; y = u.y +. v.y}
and sub u v = {x = u.x -. v.x ; y = u.y -. v.y};;
```

b)
```
let scal m u = {x = m *. u.x ; y = m *. u.y};;
```

c)
```
let carre u = u.x *. u.x +. u.y *. u.y;;
```

3. La numérotation des sous-cellules sera réalisée comme dans l'exemple du début de l'énoncé, et commencera à 0, pour s'adapter à l'indexage habituel des tableaux en Caml. Le schéma ci-contre symbolise mon choix :3 0 2 1 .

a)
```
let indice_fille p_c p =
let v = sub p p_c
in if v.x >= 0. then if v.y >= 0. then 0 else 1 else if v.y <= 0. then 2 else 3;;
```

(on remarquera un choix arbitraire de la cellule fille lorsque la position du corps est sur une ligne du quadrillage, ce qui dans la pratique n'a aucune importance, ce cas ayant une probabilité très faible (mais non nulle, en raison de la finitude des grandeurs représentables en machine) de se produire)

b)
```
let position_fille p_c taille i =
  let u = {x = 1. ; y = 1.} and v = {x = 1. ; y = -1.} and m = taille /. 4.
  in
  match i with
     0 -> add p_c (scal m u)
    |1 -> add p_c (scal m v)
    |2 -> sub p_c (scal m u)
    |_ -> sub p_c (scal m v);;
```

4.a) La fonction `insere_corps` est manifestement récursive. Soient `c` le corps à insérer, `q` le quadtree adaptatif dans lequel on l'insère, `position` la position du centre de la cellule que `q` représente, et `côté` son côté.

- Si `q` est vide, on retourne la feuille constituée du seul corps `c`.
- Si `q` est un noeud, on calcule l'indice de la fille contenant `c`, et on appelle récursivement la fonction `insere_corps` en lui donnant pour arguments `c`, la fille en question, la position de la fille et la moitié de `côté`.

- Si `q` est une feuille, constituée d'un corps `c'`, on construit d'abord un noeud dont la cellule est constituée d'une masse arbitrairement fixée à 0, d'une position arbitrairement fixée à `vecteur_nul` (puisque l'on ne demande pas de mettre ces deux champs à jour), ainsi que d'un tableau formé de 4 arbres vides.
  Puis on insère successivement `c'` et `c` dans ce noeud.

```
let rec insere_corps c q position côté =
  match q with
     Vide -> Feuille c
    |Noeud cellule ->
      let i = indice_fille position c.pos
      in insere_corps c cellule.filles.(i) (position_fille position côté i) (côté /.
2.)
    |Feuille c' ->
      let nouveau =
        Noeud {cm_mass = 0. ; cm_pos = vecteur_nul ; filles = make_vect 4 Vide}
    in insere_corps c (insere_corps c' nouveau position côté) position côté;;
```

b)Nous allons démontrer par récurrence que si$d > 0$et `côté est` assez grand, et sous certaines hypothèses sur `c`, `q` et `côté`, alors `insère_corps c q position côté` termine bien. Plus précisément, on se donne$d > 0$, et on va prouver la proposition suivante:
$H(n)_{n^3 0}$: pour tout quadruplet (`c,q,position,côté`) vérifiant les conditions suivantes :

I. la plus petite distance mutuelle de deux corps de `q` ou d'un corps de `q` avec `c` est supérieure ou égale à δ

II. la cellule de côté `côté` et de centre `position` contient les positions de `c` et de tous les corps de `q`

III. $2^{n-1} \leq \frac{co\hat{}t}{d} < 2^n$

la fonction `insere_corps c q position côté` termine bien.

$\underline{H(0)}$: la condition III. impose que $co\hat{}t < d$, donc les conditions I. et II. entraînent que `q` est vide.
Or dans ce cas, `insère_corps` se contente de retourner une feuille contenant `c`, donc termine bien.

$\underline{H(n) \Rightarrow H(n+1)}$: Soit un quadruplet (`c,q,position,côté`) vérifiant I.,II., et III. . On a donc $2^n \leq \frac{co\hat{}t}{d} < 2^{n+1}$.

* si `q` est vide, alors `insère_corps c q position côté` termine bien.

* si `q` est un noeud, alors la fonction `indice_fille` retourne l'indice de la fille définissant le quadrant de sommet `position` dans lequel se trouve `c`, mais sans vérification de l'appartenance effective de `c` à la cellule définie par la fille en question. Cette appartenance est en fait assurée par la condition II., qui implique l'appartenance de `c` à la cellule mère, donc à l'une de ses quatre cellules filles.
La fonction `position_fille` calcule alors le centre `position'` de la cellule fille en question.
Le côté de cette cellule-fille est `côté/.2.`. Or cette cellule peut contenir `c` et tous les corps du quadtree `q'` qu'elle représente. Il en résulte que le quadruplet (`c,q',position',côté/.2.`) vérifie la condition II. Ce quadruplet vérifie aussi la condition I., car tous les corps de `q'` sont dans `q`. Enfin, il vérifie III. à l'ordre n, car $2^n \leq \frac{co\hat{}t}{d} < 2^{n+1} \Rightarrow 2^{n-1} \leq \frac{co\hat{}t/.2.}{d} < 2^n$.
D'après l'hypothèse de récurrence, `insère_corps c q' position' côté/.2.` termine bien, et il en est donc de même de `insère_corps c q position côté`.

* si `q` est une feuille, contenant un corps `c'`, alors il y a création d'un quadtree `nouveau`, provisoirement mal formé, puisqu'un quadtree ne peut être un noeud que s'il contient au moins deux corps.
Ensuite on insère `c'` dans `nouveau`, et on obtient encore un noeud mal formé, contenant uniquement le corps `c'`. Enfin, on insère `c` dans ce noeud `nouveau'`. Or les deux appels de `insère_corps` correspondants se font à l'aide des quadruplets (`c',nouveau,position,côté`) et (`c',nouveau,position,côté`), vérifiant de façon évidente les trois conditions à l'ordre n + 1. On se ramène donc au cas précédent, et on conclut que ces deux appels terminent bien, car le raisonnement fait ci-dessus reste valable quelque soit le nombre de corps effectivement présents dans le noeud, puisque ces corps n'interviennent pas dans l'argumentation.

On a donc bien montré $H(n)$ par récurrence pour tout n. Par conséquent, si l'on se donne `c`, `q` et `position`, et si l'on suppose que $d > 0$, il suffit de choisir `côté` assez grand pour vérifier II., et d'appliquer $H(n)$ pour l'unique entier n vérifiant III. (il s'agit de $n = 1 + E\left(ln_2 \frac{co\hat{}t}{d}\right)$), et on est assuré que `insère_corps c q position côté` termine.

`insère_corps c q position côté` termine, pourvu que le côté de la cellule racine soit assez grand et que δ > 0

c)D'après la question 1.b), la profondeur des quadtrees ne peut pas être bornée stricto sensu indépendamment de N. Il faut en fait comprendre que l'on impose, pour tout N, une profondeur maximale m des quadtrees de taille inférieure ou égale à N, et que l'on ne s'intéresse à la complexité asymptotique de `construire_arbre` que pour cette classe de quadtrees. Cette contrainte est légitimée par la remarque de la question 1.e), mais peut aussi être justifiée par des raisons relevant de la physique, en imposant un minimum pour δ (par exemple le minimum observé), ce qui conduit au même résultat d'après la question 1.d).

- Il est clair tout d'abord que la fonction `construire_arbre` devant insérer N corps dans un quadtree initialement vide, elle est de complexité $T(N)$ au moins linéaire. Donc $T(N) = \Omega(N)$.
- D'autre part, les opérations suivantes sont de façon évidente de complexité constante :
  - le calcul de l'indice de la fille destinée à recevoir un corps lors de l'insertion dans un noeud : complexitéα_1
  - l'insertion d'un corps dans un quadtree vide : complexitéα_2
  - la création d'un nouveau noeud `nouveau` à quatre filles vides : complexitéα_3
  - l'insertion d'un corps dans un noeud, lorsque la fille destinée à le recevoir est vide : complexitéα_4

  Appelons a un majorant de la somme (car il peut y avoir cumul) de ces complexités, et montrons par récurrence sur m que si la profondeur des quadtrees est bornée par m indépendamment de N, alors la complexité $t(m)$ de l'insertion d'un corps dans un quadtree est majorée par $a(m + 1)$.

  a)c'est vrai pour $m = 0$€: en effet, un quadtree de profondeur au plus nulle est soit vide, soit une feuille. Mais l'insertion dans une feuille fait passer la profondeur du quadtree à au moins 1, ce qui est exclu. Donc l'insertion ne peut se produire que dans un quadtree vide, et est donc de complexité$\alpha_2 \leq a$. Donc $t(0) \leq a(0 + 1)$.

  b) supposons que $t(m) \leq a(m + 1)$, c'est-à-dire que la complexité de l'insertion dans un quadtree dont la profondeur est inférieure ou égale à m est inférieure ou égale à $a(m + 1)$. Evaluons alors la complexité de l'insertion dans un quadtree dont la profondeur est inférieure ou égale à $m + 1$:

  - si le quadtree est vide, la complexité est$\alpha_2 \leq a \leq a(m + 2)$.
  - si le quadtree est un noeud, on est ramené, après détermination de la fille convenable, à l'insertion du corps dans cette fille. Or cette fille est de profondeur majorée par m. On a donc dans ce cas une complexité inférieure ou égale à $\alpha_1 + t(m) \leq a + a(m + 1) = a(m + 2)$.
  - si le quadtree est une feuille, il y a création du quadtree `nouveau`, insertion du corps contenu par la feuille dans `nouveau`, calcul de la fille destinée à recevoir le corps de départ, et insertion de ce corps dans cette fille, ce qui donne une complexité majorée par $\alpha_3 + \alpha_4 + \alpha_1 + t(m) \leq a + a(m + 1) = a(m + 2)$.

  On trouve donc dans tous les cas une complexité majorée par $a(m + 2)$, d'où $t(m + 1) \leq a(m + 2)$.

  Conclusion intermédiaire : la complexité de `insère_corps c q position côté` est un $O(m)$.

  La fonction `construire_arbre` étant essentiellement constituée d'une boucle de N tours réalisant chacun l'insertion d'un corps dans un quadtree, elle est donc de complexité $NO(m) = O(N)$par rapport à N, puisqu'on a supposé m indépendant de N. Donc $T(N) = O(N)$.
- En conclusion, on a donc $T(N) = Q(N)$.

Sous les conditions de la question 1.e), l'algorithme `construire_arbre` est de complexité linéaire

Troisième partie - Caml : Calcul des forces

5. Si l'arbre-univers est vide ou est une feuille, il n'y a rien à faire, et d'ailleurs les champs à positionner ne sont pas définis. Si c'est un noeud, il suffit d'examiner tour à tour les quatre filles de l'arbre à l'aide d'une boucle, en mettant à jour les champs `cm_mass` et `cm_pos` du noeud (qui ont été laissés à 0 par `construire_arbre`). Si une fille est

vide, on ne fait rien. Si c'est une feuille, on ajoute la masse `c.mass` du corps qu'elle contient à `cm_mass` et le produit de `c.pos` par `c.mass` à `cm_pos`. Si c'est un noeud, on lui applique d'abord `barycentres` pour positionner sa masse totale et son vecteur-position, et on procède ensuite comme dans le cas d'un simple corps. Enfin, il ne reste plus qu'à diviser la somme pondérée des vecteurs positions contenue dans le champ `cm_pos` à l'issue de cette boucle par `cm_mass` pour obtenir la valeur définitive de `cm_pos`.

```
let rec barycentres = function
   Noeud cell -> for i = 0 to 3
                 do
                   let f = cell.filles.(i)
                   in
                   match f with
                     Noeud cl-> barycentres f;
                                cell.cm_mass<- cell.cm_mass +. cl.cm_mass;
                                cell.cm_pos <- add cell.cm_pos (scal cl.cm_mass
cl.cm_pos)
                    |Feuille c -> cell.cm_mass <- cell.cm_mass +. c.mass;
                                  cell.cm_pos  <- add cell.cm_pos (scal c.mass c.pos)
                    |Vide ->()
                 done;
                 cell.cm_pos <- scal (1. /. cell.cm_mass) cell.cm_pos
  |_           -> ();;
```

6.a) Nous allons réaliser un parcours en largeur de `arbre`, qui se prête mieux à l'évaluation de complexité demandée à la question suivante :

```
let grav_approx pos arbre taille =
  let acc_approx = ref vecteur_nul and forêt = ref [arbre] and côté = ref taille
  in
  let rec traite f t =
    match f with
       [] -> []
      |Vide::q -> traite q t
      |(Feuille c)::q ->
              let r = sub c.pos pos
              in
              if r <> vecteur_nul then
                let d = sqrt (carre r)
                in acc_approx := add !acc_approx (scal (c.mass /. (d *. d *. d)) r)
              else ();
              traite q t
      |(Noeud cl)::q ->
              let r = sub cl.cm_pos pos
              in
              let d = sqrt (carre r)
              in
              if t < d *. theta (* évitons les divisions par zéro! *) then
                (acc_approx := add !acc_approx (scal (cl.cm_mass /. (d *. d *. d)) r);
                traite q t)
              else
                cl.filles.(0)::cl.filles.(1)::cl.filles.(2)::cl.filles.(3)::(traite q
t)
  in
  while !forêt <> [] do forêt := traite !forêt !côté; côté := !côté /. 2. done;
  !acc_approx;;
```

Remarque
On aurait aussi pu écrire un algorithme récursif classique :

```
let rec grav_approx pos arbre taille =
```

```
  match arbre with
     Vide -> vecteur_nul
    |Feuille c ->
      let r = sub c.pos pos
      in
      if r = vecteur_nul then vecteur_nul
      else let d = sqrt (carre r) in scal (c.mass /. (d *. d *. d)) r
    |Noeud cell ->
      if taille < (sqrt (carre (sub cell.cm_pos pos))) *. theta then
        let c' = {mass=cell.cm_mass ; pos=cell.cm_pos ; vel=vecteur_nul ;
acc=vecteur_nul}
        in grav_approx pos (Feuille c') taille
      else
        begin
          let acc_approx = ref vecteur_nul
          in
          for k = 0 to 3
          do
            acc_approx := add !acc_approx (grav_approx pos cell.filles.(k) (taille /.
2.))
          done;
          !acc_approx
        end;;
```

b) Soit c un corps et C une cellule de taille donnée D . Le centre de masse de cette cellule est un point ω de cette cellule. Pour que celle-ci soit subdivisée pendant le calcul de l'accélération de c, il faut que $d(c,w) < \frac{D}{q}$. Mais la cellule étant un carré de côté D, on a $sup_{M\hat{I}C} d(w,M) \leq D\sqrt{2}$.

Donc, en vertu de l'inégalité triangulaire,
$M\hat{I}C, d\ left\ (c, M\ right\ ) <\ \{\ \{D\}\ over\ \{q\}\ \}\ +\ D\ sqrt\{2\}\ =\ D\ left\ (\{\ \{1\}\ over\ \{q\}\ \}\ +\ \ sqrt\{2\}\ right\ )\}\{$, donc C est inclus dans le disque de centre c et de rayon $D\left(\frac{1}{q}+\sqrt{2}\right)$. Les cellules de même taille D étant disjointes deux à deux, la somme des aires des cellules de taille D incluses dans le disque précédent ne peut excéder l'aire du disque. Si l'on appelle n le nombre des cellules de taille D qui sont subdivisées pendant le calcul de l'accélération de c, on a donc la relation $nD^2 \leq pD^2\left(\frac{1}{q}+\sqrt{2}\right)^2$, soit après simplification, une majoration de n par une fonction K(θ) de θ seul :

$$K(q) = p\left(\frac{1}{q}+\sqrt{2}\right)^2$$

c)Soit $T(N)$ la complexité de `ajuste_approx`.

- La fonction `grav_approx` est essentiellement constituée d'une boucle.
  Chaque tour de boucle fait appel à la fonction `traite`, qui est une fonction récursive classique de parcours d'une liste d'arbres : traitement de la tête, puis appel récursif sur la queue. Le traitement de la tête consiste à tester s'il s'agit d'un arbre vide (on ne fait rien), d'une feuille (on ajoute à `!acc_approx` la contribution du corps qu'elle contient), d'une cellule vérifiant le critère d'approximation (on fait comme pour une feuille, en identifiant la cellule à un corps) ou d'une cellule ne vérifiant pas ce critère. Dans les trois premiers cas, il y a exécution de quelques instructions de complexité constante, puis appel récursif sur la queue. Dans le dernier cas, on fait d'abord un appel récursif sur la queue, puis une quadruple concaténation. Dans tous les cas, la relation de récurrence sur la complexité est du type $u_{n+1} = u_n + Q(1)$, ce qui implique que `traite` est de complexité linéaire par rapport à la longueur de la liste.
- De plus, `traite` retourne la liste de toutes les filles des cellules subdivisées lors du calcul de l'accélération provoquée par l'ensemble des arbres de la liste `f` sur le corps. Par conséquent, il est aisé de montrer par récurrence que si tous les arbres de `f` ont une profondeur bornée par p, alors tous les arbres de la forêt retournée par `traite` ont une profondeur bornée par $p-1$. Comme cette fonction est initialement appelée sur la forêt `[arbre]`, et que par l'hypothèse 1.e, `arbre` est de profondeur inférieure ou égale à m, il arrivera

nécessairement un moment où tous les arbres contenus dans `!forêt` seront l'arbre vide ou une feuille, et alors le retour de `traite` sera la forêt vide, ce qui arrêtera la boucle et assure la terminaison de la fonction `grav_approx`.

- Mais revenons au calcul de complexité. D'après ce qui précède, il y aura donc au plus m + 1 tours de boucles dans la boucle. De plus, à une profondeur donnée, toutes les sous-arbres d'un quadtree représentent une cellule de même côté. D'après la question 6.b), il y en a donc au plus $K(\theta)$ qui sont subdivisées, ce qui assure que la forêt retournée par `traite` comporte au plus $4K(\theta)$ arbres. La complexité de `traite` est donc majorée par une constante indépendante de N, et il en est de même de l'instruction `forêt := traite !forêt !côté`, qui effectue une recopie de la liste retournée dans l'emplacement-mémoire référencé par `forêt`.
  - En conclusion, la fonction `grav_approx` effectue, outre un nombre borné d'instructions d'initialisation, un nombre borné (par m + 1) de tours de boucles effectuant chacune un nombre borné (par $2 \times 4K(\theta) + 1$) d'opérations. Donc :

    `grav_approx` est une fonction de complexité constante en N

  - Il est clair ensuite que la fonction `ajuste_approx` devant traiter N corps, elle est de complexité au moins linéaire. Donc $T(N) = \Omega(N)$.
- De plus, `ajuste_approx` est essentiellement constituée d'une boucle de N tours, chacun réalisant un appel à `grav_approx`, qui est de complexité constante. D'où le résultat :

  Sous les conditions de la question 1.e), la fonction `ajuste_approx` est de complexité linéaire en N