Algorithmique PEIP2

Correction TD1 : Boucles et conditionnelles

1 Analyse d'algorithme

On considère l'algorithme suivant.

Algorithme 1 : EstPremier

Entrées : $n \geq 1$: entier
Sorties : r: booléen
$i \leftarrow 2$
$r \leftarrow$ `Vrai`
tant que $i < n$ *et* r **faire**
  si $n \bmod i = 0$ **alors**
    $r \leftarrow$ `Faux`
  $i \leftarrow i + 1$
retourner r

1. *Que fait cet algorithme ?*

   Cet algorithme renvoie `Vrai` si n est un nombre premier, et `Faux` sinon. En effet, si l'on se trouve dans le corps de la boucle `tant que` avec une valeur de `i` donnée au cours d'une exécution de l'algorithme, alors $n \bmod j \neq 0$ pour tout $j < i$, ce qui se montre facilement par récurrence. Si à la sortie de la boucle on a $i < n$, alors `r` est faux, ce qui signifie qu'il existe i tel que $n \bmod i = 0$ et donc n n'est pas premier. Si au contraire $i \geq n$ à la sortie de la boucle, alors on s'est trouvé dans le corps de la boucle avec $i = n - 1$, ce qui signifie que $n \bmod j \neq 0$ pour tout $j \leq n - 2$, et donc que n est premier.

2. *Toutes les itérations de boucle sont-elles nécessaires ? Comment l'améliorer ?*

   Si n n'est pas premier, et i est son plus petit facteur premier, alors $i^2 \leq n$. On peut donc remplacer $i < n$ par $i * i \leq n$ dans la condition de sortie de boucle. De plus, l'instruction `Si` peut être synthétisée en la remplaçant par $r \leftarrow (n \bmod i \neq 0)$. Enfin, on peut déduire la valeur finale de `r` en considérant la valeur finale de `i`, ce qui permet de se passer totalement de la variable `r`. On obtient alors :

Algorithme 2 : EstPremier

Entrées : $n \geq 1$: entier
Sorties : r: booléen
$i \leftarrow 2$
tant que $i * i \leq n$ *et* $n \bmod i \neq 0$ **faire**
  $i \leftarrow i + 1$
retourner $i * i > n$

   Il est également possible de traiter le cas $i = 2$ à part, puis de ne tester que les valeurs impaires de $i \geq 3$, ce qui revient à incrémenter i de 2 en 2 après l'avoir initialisé à 3.

2 Conditionnelles

1. *Écrire un programme* `bissextile` *qui prend en paramètre un entier* `a` *représentant une année et qui renvoie* `Vrai` *si cela correspond à une année bissextile, et* `Faux` *sinon.*

Il est plus facile d'exprimer les conditions qui rendent une année bissextile que leur négation. On commence donc par énumérer ces conditions en renvoyant `Vrai` pour chacune d'entre elles. Si aucune de ces conditions n'est remplie, alors on peut renvoyer `Faux`.

Algorithme 3 : Bissextile

Entrées : a: entier
si $a \bmod 400 = 0$ **alors**
  retourner *Vrai*
si $a \bmod 4 = 0$ *et* $a \bmod 100 \neq 0$ **alors**
  retourner *Vrai*
retourner *Faux*

2. *Écrire un algoritme* `jours` *qui prend en paramètre un entier* `m` *représentant un mois (1 pour janvier, 12 pour décembre), et un entier* `a` *représentant une année. Cet algorithme devra calculer le nombre de jours de ce mois.*

   Le mois de février est le seul de l'année dont le nombre de jours n'est pas 30 ou 31; on le traite donc en premier pour s'en débarrasser. Les autres mois alternent entre 30 et 31 jours jusqu'au mois de juillet, puis entre 31 et 30 jours ; on peut donc exploiter cette régularité pour simplifier la disjonction de cas.

Algorithme 4 : jours

Entrées : m, a: entiers
si $m = 2$ **alors**
  si *Bissextile(a)* **alors**
    retourner 29
  sinon
    retourner 28
sinon si $m \leq 7$ **alors**
  retourner $30 + (a \bmod 2)$
sinon
  retourner $r \leftarrow 31 - (a \bmod 2)$
retourner r

3 Enumérations

1. *Écrire un algorithme prenant en paramètre un entier* `n`*, et qui affiche tous les entiers pairs inférieurs à* `n`*.*

   On peut soit énumérer les multiples de 2 sous la forme $\{2i : i \in [\lfloor n/2 \rfloor]\}$, soit utiliser le fait que l'écart entre deux nombres pairs consécutifs est toujours 2 (cette deuxième solution est un peu moins coûteuse, car elle ne nécessite aucune multiplication).

Algorithme 5 : pairs

Entrées : n: entier
pour i *de* 0 *à* $\lfloor n/2 \rfloor$ **faire**
  `Afficher(2*i)`

Ou bien :

Algorithme 6 : pairs

Entrées : n: entier
pour i *de* 0 *à* n *par pas de* 2 **faire**
  `Afficher(i)`

2. *Écrire un algorithme prenant en paramètre un entier* `n`*, et qui affiche tous les carrés parfaits inférieurs à* `n`*.*

   Encore une fois il y a deux approches possibles : soit on énumère par les racines, soit on utilise le fait qu'on peut passer de i^2 à $(i+1)^2$ en additionnant $j := 2i + 1$.

Algorithme 7 : pairs

Entrées : n: entier
$i \leftarrow 0$
tant que $i * i \leq n$ **faire**
    `Afficher(i*i)`
    $i \leftarrow i + 1$

Ou bien :

Algorithme 8 : pairs

Entrées : n: entier
$i \leftarrow 0$
$j \leftarrow 1$
tant que $i \leq n$ **faire**
    `Afficher(i)`
    $i \leftarrow i + j$
    $j \leftarrow j + 2$

4 Boucles

1. *Écrire un algorithme* `log` *prenant en paramètre deux nombres* `b` *et* `x`*, et qui renvoie la partie entière de* $\log_b(x)$.

   On calcule les puissances successives de b dans une variable y, et on s'arrête quand on dépasse x.

Algorithme 9 : log

Entrées : $b, x > 1$: nombres
$k \leftarrow 0$
$y \leftarrow 1$
tant que $y \leq x$ **faire**
    $y \leftarrow b * y$
    $k \leftarrow k + 1$
retourner $k - 1$

   Au début et à la fin de chaque itération de la boucle, on a $y = b^k$. Quand on sort de la boucle, on a $b^k = y > x$, tandis que $b^{k-1} \leq x$, ce qui signifie que $k - 1$ est bien la solution recherchée.

2. *On suppose que l'on dispose de la fonction* `EstPremier(n)` *qui détermine si l'entier* `n` *est premier ou non. Écrire un algorithme* `NonPremiersConsécutifs` *qui prend en paramètre un entier* `k`*, et qui renvoie le plus petit entier* `n` *tel que tous les entiers compris dans l'intervalle* $[n + 1; n + k]$ *ne sont pas premiers.*

   On propose deux solutions.

   Dans la première, on fait prendre à une variable n toutes les valeurs premières, successivement, et on cherche à l'aide d'une variable m que l'on incrémente à partir de $n + 1$ la valeur première suivante. On s'arrête si on a trouvé un écart $> k$ entre n et m.

   Dans le seconde, on teste la primalité de tous les nombres à partir de $n = 3$. À chaque fois qu'on tombe sur un nombre non premier, on incrémente un compteur, que l'on fait retomber à 0 dès qu'on rencontre un nombre premier. Dès qu'on a atteint k avec ce compteur, on a rencontré k nombres non-premiers consécutifs, et n vaut le k-ème tel nombre, donc la solution est $n - k$.

   À noter qu'il existe toujours une solution à ce problème, ce qui garantit que les algorithmes terminent en temps fini : aucun nombre dans l'intervalle $[(k+1)! + 2, (k+1)! + (k+1)]$ n'est premier, et donc la solution est au plus $(k + 1)! + 2$.

Algorithme 10 : NonPremiersConsecutifs

Entrées : $k \geq 1$: entier
$m \leftarrow 3$
répéter
  $n \leftarrow m$
  répéter
    $m \leftarrow m + 1$
  jusqu'à `EstPremier(m)`
jusqu'à $m - n > k$
retourner n

Lorsque l'on sort de la boucle interne, n et m sont deux nombres premiers consécutifs. Lorsque l'on sort de la boucle principale, on a $m - n > k$, et donc n est le plus petit nombre premier tel que le suivant est à distance supérieure à k. Donc n est la solution recherchée.

Algorithme 11 : NonPremiersConsecutifs

Entrées : $k \geq 1$: entier
$n \leftarrow 3$
$i \leftarrow 0$
tant que $i < k$ **faire**
  $n \leftarrow n + 1$
  si `EstPremier(n)` **alors**
    $i \leftarrow 0$
  sinon
    $i \leftarrow i + 1$
retourner $n - k$

On a comme invariant de boucle que $n-i$ est le dernier nombre premier rencontré. Lorsque l'on sort de la boucle, on a $i = k$, et donc tous les nombres compris dans l'intervalle $[n - k + 1, n]$ ne sont pas premiers. Cela garantie que $n - k$ est la solution recherchée.

3. *Écrire un algorithme* `EntierMystère` *qui prend en paramètre un entier mystère* `x` *à faire deviner à un utilisateur. Cet algorithme demande de façon répétée à l'utilisateur de rentrer une valeur jusqu'à ce qu'il ait deviné la valeur* `x` *auquel cas il le félicite et arrête le jeu, et en précisant si la valeur à deviner est supérieure ou inférieure à celle proposée en cas d'échec.*

   On répète une boucle principale jusqu'à ce que l'entrée utilisateur corresponde à l'entier mystère, auquel cas on affiche le texte de réussite.

Algorithme 12 : EntierMystère

Entrées : $x \geq 1$: entier
`Afficher("J'ai choisi un nombre mystère, essayez de le deviner !")`
répéter
  `Afficher("Quel est le nombre mystère selon vous ?")`
  $y \leftarrow$ `SaisieUtilisateur()`
  si $x < y$ **alors**
    `Afficher("Le nombre mystère est plus petit.")`
  si $x > y$ **alors**
    `Afficher("Le nombre mystère est plus grand.")`
jusqu'à $y = x$
`Afficher("Bravo, c'est la bonne réponse !")`

5 Tirage au sort

1. *Écrire un algorithme* `pièce` *qui prend en paramètre un nombre* $p \in [0, 1]$*, et qui renvoie* `Pile` *avec probabilité* p*, et* `Face` *avec probabilité* $1 - p$*.*

   Si x est un nombre aléatoire uniforme dans l'intervalle $[0, 1[$, alors $\Pr[x \leq p] = p$. Il suffit donc d'associer `Pile` à l'événement $x \leq p$, et `Face` à l'événement contraire.

Algorithme 13 : pièce

Entrées : p: nombre
$x \leftarrow$ `Random()`
si $x \leq p$ **alors**
  retourner *Pile*
sinon
  retourner *Face*

2. *Écrire un algorithme* `dé6()` *qui simule un lancer de dé à* 6 *faces, et renvoie donc un entier aléatoire uniforme compris entre* 1 *et* 6*.*

   Si x est un nombre aléatoire uniforme dans $[0, 1[$, on pourrait découper l'intervalle $[0, 1[$ en 6 parts égales de la forme $[i/6, (i+1)/6[$, et renvoyer l'indice i correspondant à l'intervalle qui contient x. Pour simplifier, on peut considérer $6x$ qui est aléatoire uniforme sur l'intervalle $[0, 6[$. Or l'intervalle $[0, 6[$ coupé en 6 parts égales donne des intervalles de la forme $[i, i+1[$.

Algorithme 14 : de6

Entrées : p: nombre
$x \leftarrow 6 *$ `Random()`
$i \leftarrow 0$
tant que $x > i$ **faire**
  $i \leftarrow i + 1$
retourner i

On pourrait aussi retourner $1 + E(x)$, si on a accès à la fonction E qui renvoie la partie entière d'un nombre.

3. *On cherche à simuler l'expérience suivante. On dispose d'un dé à 6 face, et d'un dé à 8 faces. On fait une séquence de lancers, en utilisant d'abord le dé à 6 faces, et en changeant de dé après chaque résultat pair. Écrire un algorithme* `lancers` *qui prend en paramètre un entier* `n`*, et qui simule* n *lancers de dés selon l'expérience décrite ci-dessus.*

   On définit `de8` en remplaçant 6 par 8 dans le corps de l'algorithme `de6`.

   On propose deux approches. Dans la première, les appels consécutifs à `de6()` sont fait dans une même boucle; de même pour les appels consécutifs à `de8()`.

   Dans le seconde, on utilise une variable booléenne `switch` qui sert à déterminer si on doit faire un appel à `de6()` ou à `de8()`. Il suffit d'inverser la valeur de `switch` à chaque fois qu'on obtient une valeur paire.

Algorithme 15 : lancers

Entrées : n: entier
$i \leftarrow 1$
tant que $i \leq n$ **faire**
  répéter
    $x \leftarrow$ `de6()`
    `Afficher(x)`
    $i \leftarrow i + 1$
  jusqu'à $i > n$ *ou* $x \bmod 2 = 0$
  répéter
    $x \leftarrow$ `de8()`
    `Afficher(x)`
    $i \leftarrow i + 1$
  jusqu'à $i > n$ *ou* $x \bmod 2 = 0$

Algorithme 16 : lancers

Entrées : n: entier
`switch` $\leftarrow$ `Vrai`
pour i *de* 1 *à* n **faire**
  si *switch* **alors**
    $x \leftarrow$ `de6()`
  sinon
    $x \leftarrow$ `de8()`
  si $x \bmod 2 = 0$ **alors**
    `switch` $\leftarrow \neg$`switch`

6 Impôt sur le revenu

On cherche à écrire un algorithme `impôt` qui calcule l'impôt sur le revenu d'un ménage, en prenant en paramètre la valeur `r` du revenu imposable, et le nombre `p` de parts fiscales du foyer.

- Le revenu net imposable est obtenu en diminuant de 10% le revenu imposable.
- Le quotient familial `qf` est obtenu en divisant le revenu net imposable par le nombre de parts.
- Ensuite, appliquer le barême 2021 pour calculer l'impôt brut. Seule la partie du quotient familial appartenant à la tranche d'impôt est imposée selon le taux correspondant. Par exemple, avec $\texttt{qf} = 10184 = 10084 + 100$, l'impôt sera $10084 \times 0\% + 100 \times 11\% = 11$.

tranche de l'impôt	taux d'imposition
$[0; 10084]$	0%
$[10084; 25710]$	11%
$[25710; 73516]$	30%
$[73516; 158122]$	41%
$[158122; +\infty[$	45%

On calcule les tranches les unes après les autres, en commençant par la plus élevée.

Algorithme 17 : impôt

Entrées : r, p: nombres
`net` $\leftarrow 0.9 * r$
`qf` $\leftarrow$ `net`$/p$
`impot` $\leftarrow 0$
si *qf* > 158122 **alors**
  `tranche` $\leftarrow$ `qf` $- 158122$
  `impot` $\leftarrow$ `impot` $+ 0.45 *$ `tranche`
  `qf` $\leftarrow$ `qf` $-$ `tranche`
si *qf* > 73516 **alors**
  `tranche` $\leftarrow$ `qf` $- 73516$
  `impot` $\leftarrow$ `impot` $+ 0.41 *$ `tranche`
  `qf` $\leftarrow$ `qf` $-$ `tranche`
si *qf* > 25710 **alors**
  `tranche` $\leftarrow$ `qf` $- 25710$
  `impot` $\leftarrow$ `impot` $+ 0.3 *$ `tranche`
  `qf` $\leftarrow$ `qf` $-$ `tranche`
si *qf* > 10084 **alors**
  `tranche` $\leftarrow$ `qf` $- 10084$
  `impot` $\leftarrow$ `impot` $+ 0.11 *$ `tranche`
  `qf` $\leftarrow$ `qf` $-$ `tranche`
retourner *impot*