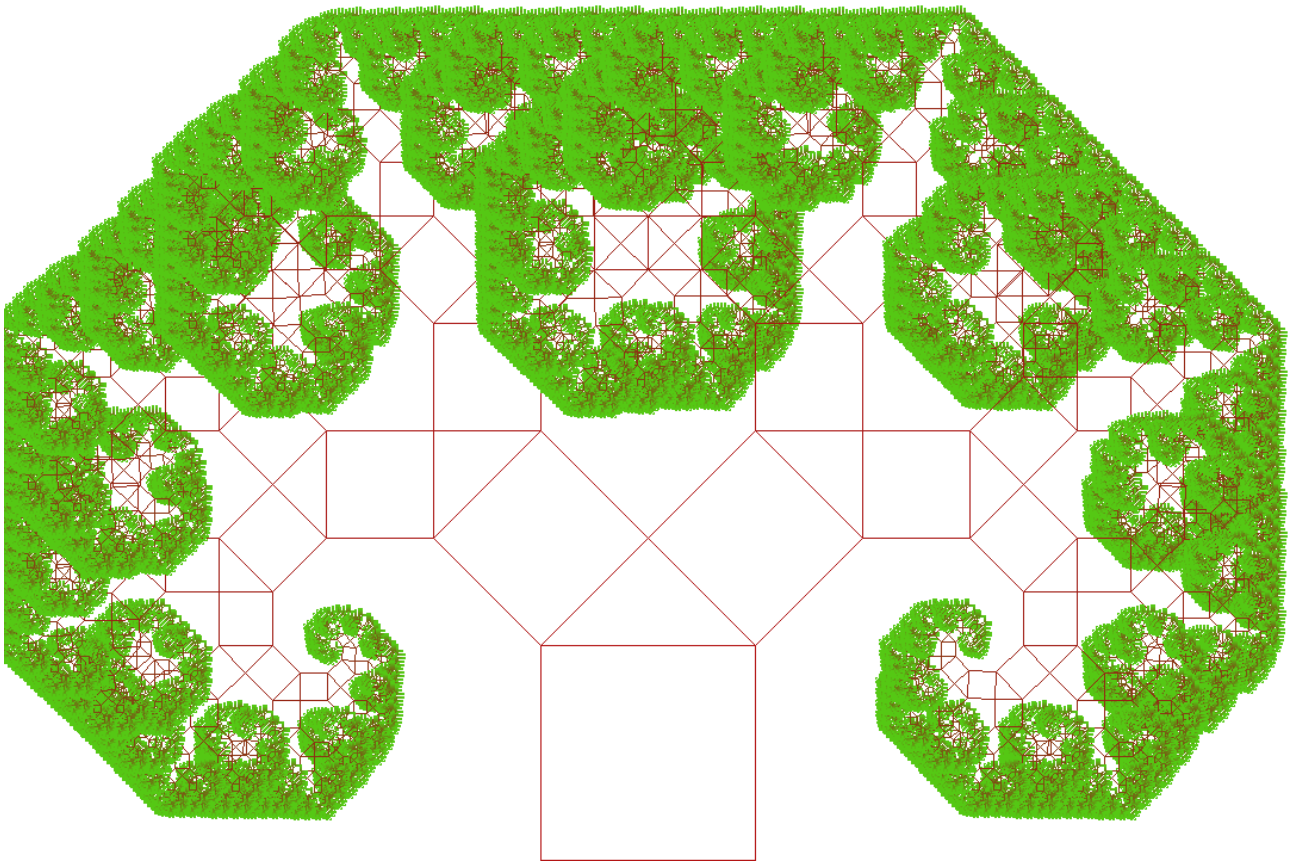
Devoir Non Surveillé 2

À rendre le 8 janvier 2024

INFORMATIQUE MP2I



Tournez la page S.V.P.

1 Étude des nombres flottants – (20 pts)

Dans cette section, on admet (sans le démontrer) que les flottants sont le sous ensemble $\mathbb{F}$ de $\mathbb{R}$ des $n2^e$ avec n et e entiers, $|n| < 2^{53}$ et $-1074 < e < 970$. La spécification des flottants garantit que l'addition sur $\mathbb{F}$, qu'on note $\oplus$, se comporte comme si le calcul avait d'abord été fait dans $\mathbb{R}$ de façon exacte puis qu'on avait arrondi le résultat. Il existe plusieurs modes d'arrondi. On considère ici l'arrondi par défaut, qui est l'arrondi au plus proche (en anglais *round to nearest*). Il s'agit de la fonction $\mathcal{N} : \mathbb{R} \to \mathbb{F}$ qui, pour un réel, x, renvoie le nombre flottant le plsu proche de x. En cas de point milieu (si x est à égale distance de deux nombres flottants), $\mathcal{N}(x)$ est celui qui a une valeur de n paire. On a donc la relation

$$\forall x \in \mathbb{F}, \forall y \in \mathbb{F}, \qquad x \oplus y = \mathcal{N}(x + y)$$

On a des relations similaires pour la soustraction $\ominus$ et la multiplication $\otimes$.

☐ 1 – L'ensemble $\mathbb{F}$ munit de la relation d'ordre usuelle $\leq$ sur les réels forme-t-il un ensemble bien fondé ?

Réponse 1. L'ensemble $\mathbb{F}$ est fini, donc on ne peut pas avoir une suite infini d'éléments strictement décroissants, car ces éléments seraient deux à deux distincts.

☐ 2 –

1. Prouver que $0, 1, 1/8, 2^{52}$ et -2^{52} sont des flottants selon cette caractérisation, mais pas $2^{52} + 1/8$.
2. Donner le plus grand nombre flottant et le plus petit nombre flottant strictement positif.
3. Donner tous les nombres flottants dans l'intervalle $[1 - 2^{-52}; 1 + 2^{-52}]$.

Réponse 2.

1. $0 = 0 \times 2^0$, $1 = 1 \times 2^0$, $1/8 = 1 \times 2^{-3}$, $2^{52} = 2^{52} \times 2^0$, $-2^{52} = -2^{52} \times 2^0$, $2^{52} + 1/8 = (2^{55} + 1)/8$ est irréductible et le numérateur est plus grand que 2^{53} donc n'est pas un flottant.
2. Donner le plus grand nombre flottant : $(2^{53} - 1) \times 2^{970}$ et le plus petit nombre flottant strictement positif : 1×2^{-1074}.
3. Donner tous les nombres flottants dans l'intervalle $[1-2^{-52}; 1+2^{-52}] = [(2^{52}-1)\times 2^{-52}; (2^{52}+1)\times 2^{-52}] = [(2^{53} - 2) \times 2^{-53}; (2^{52} + 1) \times 2^{-52}] = \{1 - 2^{-52}; 1 - 2^{-53}; 1; 1 + 2^{-52}\}$.

☐ 3 – Calculer $(-2^{52} \oplus 2^{52}) \oplus 1/8$. Calculer $-2^{52} \oplus (2^{52} \oplus 1/8)$. Conclure sur une propriété non respectée par $\oplus$.

Réponse 3. $(-2^{52} \oplus 2^{52}) \oplus 1/8 = 0 \oplus 1/8 = 1/8$
$-2^{52} \oplus (2^{52} \oplus 1/8) = -2^{52} \oplus 2^{52} = 0$
$\oplus$ n'est pas associative.

☐ 4 – Trouver un nombre flottant x non nul tel que $x \otimes x = 0$. Que dire du code suivant :

```
if (x != 0) {z = 1/(x*x);}
```

?

Réponse 4. Le plus petit nombre flottant strictement positif 1×2^{-1074} par exemple possède cette propriété. La condition `x!=0` est inutile, dans les deux cas, on peut avoir $z = \infty$ à la fin de l'exécution.

☐ 5 –

1. Trouver un nombre flottant x tel que $1 \oplus x = 1$.
2. Trouver le plus grand nombre flottant x tel que $1 \oplus x = 1$. Justifier.

Réponse 5. Le plus petit nombre flottant strictement plus grand que 1 est $1 + 2^{-52} = (2^{52} + 1)2^{-52}$. En effet, $1 = 2^e 2^{-e}$ pour tout e, et $e \geq 53$ ne donne pas un nombre flottant. Le plus grand nombre flottant x cherché est donc le plus grand nombre flottant inférieur ou égal à la moitié de 2^{-52}, soit 2^{-53}.

☐ 6 – Si l'on soustrait deux valeurs proches, mais résultats d'un calcul arrondi, on peut obtenir un résultat complètement faux. Comparer $(2^{52} \oplus 1/8) \ominus (2^{52} \ominus 1/8)$ et la valeur mathématique sans arrondi $(2^{52} + 1/8) - (2^{52} - 1/8)$.

Réponse 6. $(2^{52} \oplus 1/8) \ominus (2^{52} \ominus 1/8) = 2^{52} \ominus 2^{52} = 0$
$(2^{52} + 1/8) - (2^{52} - 1/8) = 2/8 = 1/4$.

☐ 7 – Mais ce n'est pas la dernière soustraction qui créé cette erreur, elle ne fait que mettre en lumière les erreurs précédentes. Prouver le lemme suivant : soient x et y deux nombres flottants

$$\text{si } y/2 \leq x \leq 2y, \text{ alors } x \ominus y = x - y.$$

Réponse 7. Notons que $y/2 \leq x \leq 2y \Leftrightarrow x/2 \leq y \leq 2x$, et que si un nombre f est un flottant, alors $-f$ l'est aussi. Donc sans perte de généralité, on peut supposer que $y \leq x \leq 2y$. Notons $y = n_y 2^{e_y}$, alors

$$n_y 2^{e_y} \leq x \leq n_y 2^{e_y+1}$$

donc $x = n_x 2^{e_y}$, avec $n_y < n_x < 2n_y$, et $x - y = (n_x - n_y)2^{e_y}$, avec $0 < n_x - n_y < n_y$.
$x - y$ est donc bien un flottant.

2 Permutations – C (30 pts)

On représente en C une permutation σ d'un ensemble de n éléments par un tableau de taille n, contenant les entiers de 0 à $n-1$.

☐ 8 – Écrire une fonction `int *compose(int sigma1[], int sigma2[], int n)` qui prend en entrée 2 permutations σ_1 et σ_2 et qui crée et renvoie un tableau d'entiers correspondant à la permutation $\sigma_1 \circ \sigma_2$.

Réponse 8.

C
```c
int *compose(int sigma1[], int sigma2[], int n){
  int *sigma = malloc(n*sizeof(int));
  for (int i = 0; i < n; i = i + 1){
    sigma[i] = sigma1[sigma2[i]];
  }
  return sigma;
}
```

☐ 9 – Écrire une fonction `bool est_involution(int sigma[], int n)` qui prend en entrée une permutations σ et qui renvoie `true` si σ est une involution et `false` sinon.

Réponse 9.

C
```c
bool est_involution(int sigma[], int n){
  for (int i = 0; i < n; i = i + 1){
    if (sigma[sigma[i]] != i)
      { return false; }
  }
  return true;
}
```

☐ 10 – Écrire une fonction `bool est_permutation(int sigma[], int n)` qui prend en entrée un tableau d'entiers σ et qui renvoie `true` si σ est bien une permutation et `false` sinon.

Réponse 10.

C
```c
bool est_permutation(int sigma[], int n){
  bool *vus = malloc(n*sizeof(bool));
  for (int i = 0; i < n; i = i + 1){
    int s = sigma[i]
    if (s < 0 || s >= n || vus[s])
      { free(vus); return false; }
    vus[s] = true;
  }
  free(vus); return true;
}
```

On s'intéresse maintenant aux cycles dans une permutation. Par exemple, la permutation `{ 0, 2, 1, 4, 5, 3}` possède 3 cycles : (0), $(1,2)$, et $(3,4,5)$, de longueurs 1, 2, et 3.

☐ 11 – Écrire une fonction `int taille_cycle(int i, int sigma[], int n)` qui prend en entrée un entier i et une permutation σ et qui renvoie la taille du cycle de σ contenant i.

Réponse 11.

```c
int taille_cycle(int i, int sigma[], int n){
  int cpt = 1;
  for (int j = sigma[i]; j != i; j = sigma[j])
    { cpt = cpt + 1; }
  return cpt;
}
```

☐ 12 – Écrire une fonction `int nb_cycles(int sigma[], int n)` qui prend en entrée une permutation σ et qui renvoie la nombre de cycles qu'elle contient.

Réponse 12.

```c
int nb_cycles(int sigma[], int n){
  bool vus = malloc(n*sizeof(bool));
  int cpt = 0;
  for (int i = 0; i < n; i = i + 1){
    if (!vus[i]){
      cpt = cpt + 1;
            int j = i;
      while (!vus[j]){
                  vus[j] = true;
                  j = sigma[j];
      }
    }
  }
  free(vus);
  return cpt;
}
```

On propose de parcourir toutes les permutations par ordre lexicographique pour conserver celle de poids minimal. Par exemple, la permutation qui suit $(0, 2, 4, 3, 1)$ dans l'ordre lexicographique est $(0, 3, 1, 2, 4)$.
Si $p = (p_0, p_1, \dots, p_{n-1})$ est une permutation de $[\![0, n-1]\!]$, on définit les indices suivants :

— $j = \max\{i \in [\![0, n-2]\!] \mid p_i < p_{i+1}\}$ et $j = -1$ si cet ensemble est vide.
— $k = \max\{i \in [\![j+1, n-1]\!] \mid p_j < p_i\}$ et $k = n$ si $j = -1$.

☐ 13 – Écrire une fonction `bool permut_suivante(int *p, int n)` qui renvoie `false` si p est la dernière permutation de n éléments dans l'ordre lexicographique, et qui renvoie `true` et modifie p pour qu'il prenne la valeur de la permutation suivante dans l'ordre lexicographique sinon.

Réponse 13. Premièrement, on peut remarquer qu'une permutation d'un sous ensemble de $[\![0, n-1]\!]$ est maximal pour l'ordre lexicographique si elle est triée dans le sens inverse.
L'indice j est tel que le sous-tableau $(p_i)_{j<i<n}$ est maximal. Ce sous-tableau étant trié dans le sens inverse, l'indice k est tel que p_k est le plus petit élément plus grand que p_j dans ce sous-tableau.
Pour trouver la permutation suivante, il faut changer p_j car toutes les permutations commençant par $(p_0, \dots, p_j)$ ont été testées, mais pas toutes celles commençant par $(p_0, \dots, p_{j-1})$ (par maximalité de j). Comme $(p_0, \dots, p_{j-1})$ doivent rester inchangés, on échange p_j avec p_k qui est le nombre suivant dans l'ordre. On peut remarquer que les $(p_i)_{j<i<n}$ restent triés dans le sens inverse, on a donc juste à les remettres dans le bon ordre en les inversant.

parcours des permutations dans l'ordre lexicographique C

```c
void swap(int *p, int j, int k) {
        p[j] = p[j] + p[k];
        p[k] = p[j] - p[k];
        p[j] = p[j] - p[k];
}

bool permut_suivante(int *p, int n) {
        int j = n-2;

        while (j >= 0 && p[j] >= p[j+1]) {
                j = j - 1;
        }
        if (j == -1) {
                return false;
        }
        int k = n-1;
        while (p[j] >= p[k]) {
                k = k - 1;
        }
        swap(p, j, k);
        int min = j+1,
                max = n-1;
        while (max > min) {
                swap(p, min, max);
                min = min + 1;
                max = max - 1;
        }
        return true;
}
```

☐ 14 – Écrire une fonction `void anagrammes(char mot[])` qui affiche tous les anagrammes d'un mot. On pourra utiliser l'instruction `printf("%s\n", str)` qui affiche la chaîne de caractères `str` suivie d'un retour à la ligne.

Réponse 14.

C

```c
void anagrammes(char mot[]){
  int n = 0;
  while (mot[i] != '\0') { n = n + 1; }
  int *sigma = malloc(n*sizeof(int));
  for (int i = 0; i < n; i = i + 1)
    { sigma[i] = i; }
  printf("%s\n", mot);
  while(permut_suivante(sigma, n)){
    for(int i = 0; i < n; i = i + 1)
      { printf("%c", mot[sigma[i]]); }
    printf("\n");
  }
}
```

☐ 15 – Écrire une fonction `int experience(int n)` qui prend en entrée un entier n et qui compte le nombre de permutations sur n éléments qui n'ont pas de cycle de taille supérieure à $\frac{n}{2}$.

Réponse 15.

C

```c
int experience(int n){
  int cpt = 0;
  int *sigma = malloc(n*sizeof(int));
  for (int i = 0; i < n; i = i + 1)
    { sigma[i] = i; }
  while(permut_suivante(sigma, n)){
    cpt = cpt + 1;
    for (int i = 0; 2*i < n; i = i + 1){
      if (2*taille_cycle(i, sigma, n) > n)
        { cpt = cpt - 1; }
    }
  }
  return cpt;
}
```

☐ 16 – On considère l'énigme suivante :

Cent prisonniers (sur une péniche), identifiés par un (unique) nombre entre 1 et 100, ont été condamnés à mort par le cruel Docteur No, qui leur offre cependant une dernière chance. Dans la timonerie, se trouvent 100 tiroirs (numérotés de 1 à 100), où il a aléatoirement mis les 100 numéros des prisonniers. Après concertation, ces derniers ont le droit d'entrer l'un après l'autre et de regarder le contenu de 50 tiroirs (qu'ils referment après leur passage). Ils auront, collectivement, la vie sauve si chacun d'entre eux trouve son numéro dans un tiroir. Malheureusement, ils ne peuvent pas communiquer entre eux et si l'un d'eux échoue, ils seront tous exécutés.

Un mathématicien, pessimiste, se trouve parmi les prisonniers. Selon lui, la probabilité de succès est de $2^{-100} \approx 8 \cdot 10^{-31}$.

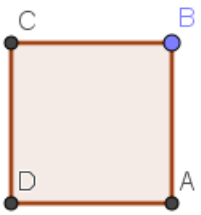
Proposer une meilleure solution que la solution naïve proposée par le mathématicien et estimer les chances de survie des prisonniers, en fonction de leur nombre n (ils ouvrent $\frac{n}{2}$ tiroirs).

Réponse 16. cf http://fabrice.orgogozo.perso.math.cnrs.fr/vulgarisation/enigmes.pdf

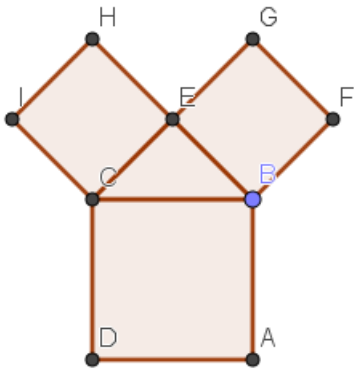
3 Arbres – OCaml (20 pts)

On souhaite dessiner des arbres en OCaml à l'aide du module Graphics. Pour cela, on s'inspire de la construction fractale de l'arbre de Pythagore :

En partant d'un carré ABCD, on lui ajoute un triangle rectangle isocèle BEC dont [BC] est l'hypoténuse, puis on ajoute deux carrés BFGE et EHIC de côtés respectivement [BC] et [EC]. On réitère ensuite cette construction avec les deux carrés obtenus.



Ordre 0



Ordre 1

☐ 17 – Écrire une fonction `carre (xa, ya : int * int) (xb, yb : int * int) : (int * int) * (int * int)` qui à partir des coordonnées des points A et B, calcule les coordonnées des points C et D, trace le carré ABCD, et renvoie le couple de couples $((x_C, y_C), (x_D, y_D))$.

Réponse 17.

```
let carre (a,b) (c,d) =
  let (e,f) = (c+b-d,d+c-a) in
  let (g,h) = (a+b-d,b+c-a) in
  moveto a b; lineto c d; lineto e f;
  lineto g h; lineto a b; ((e,f),(g,h))
```

☐ 18 – Écrire une fonction `triangle (xb, yb : int * int) (xc, yc : int * int) : (int * int)` qui à partir des coordonnées des points B et C, calcule les coordonnées du point E, trace le triangle BEC, et renvoie le couple (x_E, y_E).

Réponse 18.

```
let triangle (a,b) (c,d) =
  let (e,f) = (a+c+b-d)/2, (b+d+c-a)/2 in
  (*let (e,f) = move_slightly (e,f) in*)
  moveto a b; lineto c d;
  lineto e f; lineto a b;
  (e,f)
```

Nous sommes maintenant prêt à mettre ces fonctions bout-à-bout pour tracer l'arbre de Pythagore. Il nous faut un cas de base pour les appels récursifs, nous allons donc choisir une profondeur d (e.g., 10 au début) jusque laquelle aller, qui décroit à chaque appel récursif, et nous arrêter si $d = 0$.

☐ 19 – Compléter la fonction `arbre` ci-dessous :

```
let rec arbre (a : int * int) (b : int * int) (d : int) =
  if d <= 0 then () else
    begin
      ...
    end
```

Réponse 19.

```
let rec arbre p1 p2 n =
  set_color (rgb (80+5*n) (20+(20-n)*(20-n)/2) (20));
  if n <= 0 then () else
    begin
      let (q1,q2) = carre p1 p2 in
      let q3 = triangle q2 q1 in
      arbre q2 q3 (n-1); arbre q3 q1 (n-1)
    end
```

☐ 20 – Donner le nombre d'appels à la fonction `arbre` effectués lors du calcul de `arbre a b d`, en fonction de d.

Réponse 20. Le nombre T d'appels vérifie la récurrence : $T(0) = 1, T(n+1) = 2T(d) + 1$. Elle se résoud en $T(D) = 2^{d+1} - 1$, soit $O(2^d)$.

☐ 21 – Pour briser la symétrie de l'arbre, on décide de modifier légèrement l'emplacement des points aléatoirement, par exemple en déplaçant de quelques pixels vers la droite, vers la gauche, vers le haut, vers le bas chaque point avant de le placer définitivement. On pourra pour cela utiliser la fonction `Random.int : int -> int` qui prend en entrée un entier n et renvoie un nombre aléatoire choisi uniformément entre 0 inclus et n exclus.

Proposer une modification des fonctions précédentes afin de tracer une approximation de l'arbre de Pythagore, mais asymétrique, et envoyer vos plus jolis dessins à votre professeur.

Réponse 21.

```
let move_slightly (x,y) = (x - 10 + Random.int 21, y - 10 + Random.int 21)
```

Code complet

```
open Graphics;;
open_graph "";;
let largeur = 1080;;
let hauteur = 720;;
resize_window largeur hauteur;;

let move_slightly (x,y) = (x - 10 + Random.int 21, y - 10 + Random.int 21)

let triangle (a,b) (c,d) =
  let (e,f) = (a+c+b-d)/2, (b+d+c-a)/2 in
  (*let (e,f) = move_slightly (e,f) in*)
  moveto a b; lineto c d;
  lineto e f; lineto a b;
  (e,f);;

let carre (a,b) (c,d) =
  let (e,f) = (c+b-d,d+c-a) in
  let (g,h) = (a+b-d,b+c-a) in
  moveto a b; lineto c d; lineto e f;
  lineto g h; lineto a b; ((e,f),(g,h));;

let rec arbre p1 p2 n =
  set_color (rgb (80+5*n) (20+(20-n)*(20-n)/2) (20));
  if n <= 0 then () else
    begin
      let (q1,q2) = carre p1 p2 in
      let q3 = triangle q2 q1 in
      arbre q2 q3 (n-1); arbre q3 q1 (n-1)
    end;;

arbre (450,0) (630,0) 20;;

wait_next_event [Button_down];;
```

Fin de l'épreuve