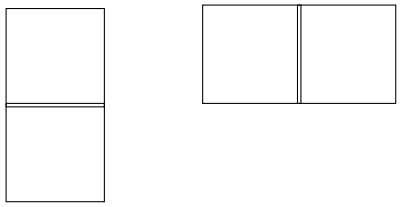
1

Géométrie et Images

« *Des représentations. Des figures. Des intersections. Des pavages.* »

1.1 Pavage d'un rectangle par des dominos

On appelle domino une pièce formée de deux carrés unité adjacents.



Un pavage d'un rectangle de largeur n et de hauteur k est un recouvrement des cases du rectangle par des dominos, chaque domino recouvrant deux cases et chaque case étant couverte par exactement un domino.

Question 1

Combien y a-t-il de pavages du rectangle $n \times 1$?

Question 2

Soit u_n le nombre de pavages du rectangle $n \times 2$. Écrire une fonction de calcul de u_n.

Question 3

Soit u_n le nombre de pavages du rectangle $n \times 3$. Proposer un algorithme de calcul de u_n. Écrire la fonction correspondante.

Question 4

Que pensez-vous du nombre de pavages du carré $n \times n$?

CORRIGÉ

Question 1

Le rectangle $n \times 1$ n'admet pas de pavage si n est impair, sa surface étant impaire. Si n est pair, il y a un pavage unique.

Question 2

Soit u_n le nombre de pavages du rectangle $n \times 2$. Considérons les deux cases les plus à droite du rectangle. Soit elles sont recouvertes par un même domino vertical, auquel cas le reste de la figure forme un rectangle $(n-1) \times 2$ qui doit également être pavé, soit elles sont recouvertes par deux dominos horizontaux l'un au dessus de l'autre, auquel cas le reste de la figure forme un rectangle $(n-1) \times 2$. On a donc la récurrence :

$$u_n = u_{n-1} + u_{n-2},$$

avec les conditions initiales $u_1 = 1$ et $u_2 = 2$. On reconnaît d'ailleurs là la suite de Fibonacci. Programmer la récurrence donne la fonction suivante :

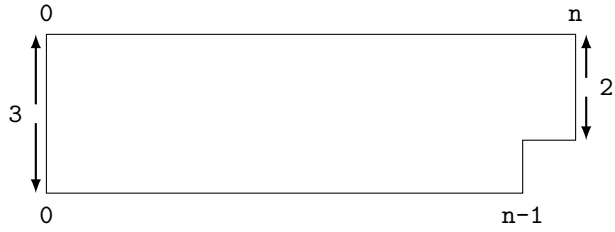
```
let largeur2 (n:int) : int =
  if n=1 then 1 else
  if n=2 then 2 else

  let a = ref 1 and b = ref 2 in
  for i=3 to n do
    (* Calcul de u_i *)
    let tmp = !a + !b in
    a := !b;
    b := tmp
  done;
  !b
```

Cette solution, très simple, permet de calculer u_n en $O(n)$ opérations. Il est cependant possible de faire mieux : en effet, il est bien connu qu'on peut calculer u_n explicitement, et on obtient $u_n = (\alpha^{n+1} - \beta^{n+1})/\sqrt{5}$, où α et β sont les deux racines de l'équation $x^2 = x + 1$. Ainsi, on a simplement besoin d'une fonction calculant la n-ième puissance d'un nombre réel, ce qui peut se faire en $O(\log_2 n)$ multiplications en utilisant la décomposition binaire de n. Voir ci-dessous une fonction dite « d'exponentiation rapide » pour calculer a^n.

```
let rec puiss (x:int) (n:int) : int = match n with
  | 0 -> 1
  | n when (n mod 2) = 0 -> puiss (x * x) (n / 2)
  | n -> (puiss (x * x) ((n-1)/2)) * x
```

Question 3

On a maintenant un rectangle de largeur 3. Soit w_n le nombre de pavages de la figure suivante, obtenue à partir du rectangle en enlevant le coin en bas à droite :



Pour calculer u_n, on regarde les dominos recouvrant les trois cases les plus à droite du rectangle. Soit ce sont trois dominos horizontaux, auquel cas il reste un pavage du rectangle de taille $n-2$, soit ce sont un domino horizontal et un domino vertical, disposés de deux façons possibles, auquel cas il reste une region du type ci-dessus, c'est à dire un rectangle de taille $n-1$ auquel il manque un coin (par symétrie, que le coin soit en haut à gauche ou en haut à droite, le nombre de pavages est le même). On a donc la récurrence :

$$u_n = 2w_{n-1} + u_{n-2},$$

De même, on observe facilement que :

$$w_n = u_{n-1} + w_{n-2}.$$

De plus, $u_n = 0$ si n est impair et $w_n = 0$ si n est pair (car la surface doit être paire). Les conditions initiales sont : $u_2 = 3$ et $w_1 = 1$. D'où la fonction suivante, qui à chaque étape calcule w_{2i-1} et u_{2i} (les autres valeurs sont 0 par parité de la surface) :

```
let largeur3 (n:int) : int =
  if n mod 2 = 1 then 0 else
  let wn = ref 1 in (* valeur de w1 *)
  let un = ref 2 in (* valeur de u2 *)
  for i=2 to (n/2) do
    (* Calcul de w2i-1 et de u2i *)
    wn := !wn + !un;
    un := 2 * !wn + !un
  done;
  !un
```

Ici encore, on peut résoudre explicitement le système et se ramener à une formule qui peut être évaluée en $O(\log_2 n)$ multiplication de réels.

Question 4

Tout d'abord, par parité, le nombre de pavages est 0 si n est impair. Si n est pair, soit u_n le nombre de pavages. On sait que chaque carré 2×2 peut être pavé de deux manières par des dominos. Donc en partitionnant un carré $n \times n$ en $n/2 \times n/2$ petits carrés 2×2, on obtient la borne inférieure :

$$u_n \geq 2^{n^2/4}.$$

Par ailleurs, tout pavage peut être décrit par un mot de $n^2/2$ bits comme suit : si le carré en haut à gauche est recouvert par un domino horizontal, le premier bit est 0, sinon il est 1. Supposons que les k premiers bits décrivent la position de k dominos. Si la case la plus en haut à gauche de la région restante est recouverte par un domino horizontal, le $(k+1)$-ième bit est 0, sinon il est 1. Ceci décrit une application injective de l'ensemble des pavages vers l'ensemble des mots de longueur $n^2/2$, et donc :

$$u_n \leq 2^{n^2/2}.$$

En fait, ce problème est bien connu en physique statistique, où le nombre de pavages correspond au nombre de configurations de molécules diatomiques sur une surface. En 1961, Kasteleyn propose une formule compliquée donnant la valeur exacte de u_n et calcule sa valeur asymptotique :

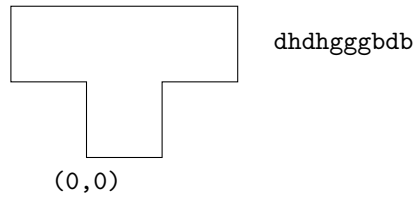
$$\frac{\log_2(u_n)}{n} \xrightarrow[n\to\infty]{} \sum_{r\geq 0} \frac{(-1)^r}{\pi(2r+1)^2} \approx 0.2916...$$

Cependant, le nombre de pavages du cube $n \times n \times n$ par des dominos est toujours un problème ouvert.

⋆⋆⋆⋆⋆⋆⋆⋆

1.2 Surface d'un polygone dessiné sur un réseau

Soit P un polygone dont toutes les arêtes sont horizontales ou verticales et tous les sommets à coordonnées entières. On le suppose donné par un mot décrivant son contour, de la façon suivante :
- on part de $(0, 0)$
- on lit les lettres du mot une à une : « d » signifie qu'on fait un pas d'une unité vers la droite, « g » vers la gauche, « h » vers le haut et « b » vers le bas. Par exemple :



Question 1

Écrire une fonction qui test si le polygone est fermé, c'est à dire si le dernier point du contour coïncide avec le premier.

Question 2

Écrire une fonction qui détermine les abscisses minimale et maximale $xmin$ et $xmax$ des points du polygone.

Question 3

Soient $ymin$ et $ymax$ définis de façon similaire. On se donne une matrice d'entiers `t` de taille $n \times m$ initialement remplie par des 0. On veut « dessiner » le polygone dans `t`, c'est à dire mettre des 0 et des 1 dans les entrées `t.(i).(j)` de façon que l'ensemble des entrées égales à 1 décrive le contour du polygone. Autrement dit, la matrice doit être comme une « fenêtre » par laquelle on voit une portion de $\mathbb{Z}^2$. Écrire une fonction à cet effet.

Question 4

Écrire une fonction pour calculer la surface du polygone.

Question 5

Proposer une solution si le polygone est dessiné sur le réseau triangulaire (pavage du plan par des triangles équilatéraux) au lieu du réseau carré.

CORRIGÉ

Question 1

Il suffit de faire le tour du polygone à partir de $(0, 0)$ en calculant à chaque pas la variation de l'abscisse et de l'ordonnée. On suppose le polygone donné par une chaîne de caractères.

```
let est_ferme (poly:string) : bool =
  let n = String.length poly in
  let x = ref 0 and y = ref 0 in
  for i=0 to n-1 do
    match poly.[i] with
      | 'h' -> incr y
      | 'b' -> decr y
      | 'd' -> incr x
      | _ -> decr x
  done;
  (!x, !y) = (0,0)
```

Question 2

Là encore, il suffit de faire un parcours du contour en regardant à chaque pas la variation de l'abscisse et en mettant à jour `xmin` et `xmax` lorsque c'est nécessaire.

```
let abscisses (poly:string) : (int * int) =
  let n = String.length poly in
  let x = ref 0  in
  let xmax = ref 0 and xmin = ref 0 in
  for i=0 to n-1 do
    match poly.[i] with
      | 'd' -> incr x; if !x > !xmax then xmax := !x
      | 'g' -> decr x; if !x < !xmin then xmin := !x
      | _ -> ()
  done;
  (!xmin, !xmax)
```

On supposera dans la suite disposer dans la suite d'une fonction `ordonnees` analogue.

Question 3

Le polygone va pouvoir se tenir dans le tableau si et seulement si $xmax - xmin + 1$ et $ymax - ymin + 1$ sont tous deux inférieurs ou égaux à n. Attention à bien dessiner le contour tel qu'il apparaît dans $\mathbb{Z}^2$, et non une image de ce contour par rotation ou symétrie. En particulier, faire varier l'indice des lignes de la matrice de 0 à $n-1$ revient à faire varier l'ordonnée du point de $\mathbb{Z}^2$ de $ymax$ à $ymax - n + 1$, et faire varier l'indice des colonnes de 0 à $n-1$ revient à faire varier l'abscisse du point de $\mathbb{Z}^2$ de $xmin$ à $xmin + n - 1$. L'application qui à un point de $\mathbb{Z}^2$ associe une entrée de la matrice doit donc être :

$$\sigma : (x, y) \mapsto \texttt{t.(ymax-y).(x-xmin)}.$$

Ainsi, $(0, 0)$ a pour image `t.(ymax).(-xmin)`, et à chaque lettre lue on a :
« d » : `j` augmente de 1
« g » : `j` diminue de 1
« h » : `i` diminue de 1
« b » : `i` augmente de 1.
D'où la fonction :

```
let dessin (poly:string) : int array array =
  let xmin, xmax = abscisses poly in
  let ymin, ymax = ordonnees poly in
  let n = ymax - ymin + 1 and m = xmax - xmin + 1 in
  let t = Array.make_matrix n m 0 in

  let l = String.length poly in
  let i = ref ymax and j = ref (-xmin) in
  t.(!i).(!j) <- 1;
  for k=0 to l-1 do
    (match poly.[k] with
      | 'd' -> incr j
      | 'g' -> decr j
      | 'h' -> decr i
      | _ -> incr i);
    t.(!i).(!j) <- 1
  done; t
```

Question 4

Le calcul de la surface A peut sembler difficile, mais en fait il suffit d'utiliser la formule de Green-Riemann (cas particulier de la formule de Stokes) pour que la programmation devienne très simple.