

# Devoir à la maison

## corrigé

### 1 Arbres AVL

**Exercice 1** On prend la convention qu'un arbre vide est de hauteur nulle et qu'un arbre à un seul noeud est de hauteur 1 comme vu en cours pour les ABRs.

L'arbre complet a le maximum de noeuds :  $2^h - 1$ .

Les arbres ayant le moins de noeuds sont ceux qui sont déséquilibrés au maximum, pour chaque noeud de l'arbre, la hauteur du sous-arbre gauche et du sous-arbre droit diffèrent d'exactement 1.

**Exercice 2** On a facilement  $\log_2(n+1) \leq h$ .

Les AVL ayant un nombre minimal de noeuds (en fonction de  $h$ ) vérifient la récurrence suivante :  $n(h) = 1 + n(h-1) + n(h-2)$ , avec  $n(1) = 1$  et  $n(2) = 2$ . Soit  $F_h$  le  $h$ -ième terme de la suite de Fibonacci et  $\phi = \frac{1+\sqrt{5}}{2}$ , on montre par récurrence que  $n(h) = F_{h+2} - 1$ . Donc  $n(h) = \frac{1}{\sqrt{5}}(\phi^{h+2} - \phi^{-(h+2)}) - 2 \leq$

$\frac{1}{\sqrt{5}}\phi^{h+2} - 2$  d'où on déduit  $h \leq \frac{\log_2(\sqrt{5}(n+1))}{\log_2\phi}$  soit pour avoir une formule similaire à la borne inférieure  $h \leq \frac{\log_2((n+1))}{\log_2\phi}$  avec  $\frac{1}{\log_2\phi} = 1.4402\dots$   
 $h \leq 1.441 \log_2(n+1)$

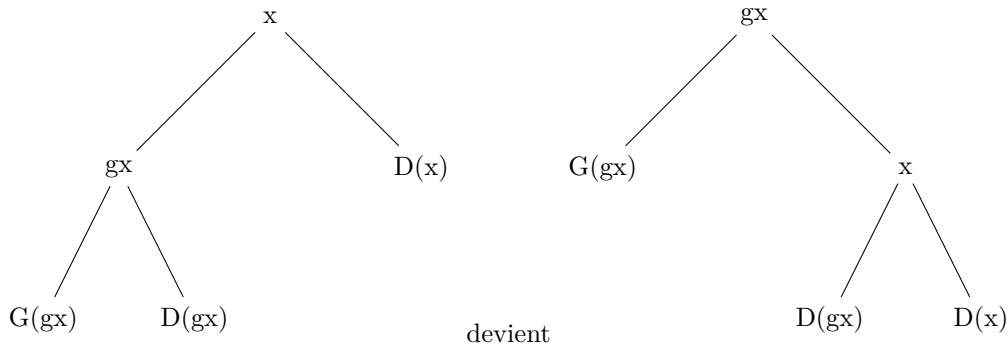
**Exercice 3** On conserve la propriété des ABR lors de la première étape d'insertion, dont si une propriété n'est plus vérifiée c'est que "pour tout sommets  $x$  de l'arbre les hauteurs des sous-arbres gauche et droite de  $x$  ne diffèrent que de 1 au plus".

$|h(G(x')) - h(D(x'))| = 2$  où  $x'$  est le plus petit sous arbre non équilibré, c'est à dire que  $G(x')$  et  $D(x')$  sont eux équilibrés.

**Exercice 4** Erreur dans l'énoncé. Si  $D(x)$  et  $G(x)$  étaient de même hauteur, après insertion leur hauteur différent d'au plus 1, par contre le sous arbre où est ajouté  $y$  peut devenir déséquilibré,  $A(x)$  n'est donc pas nécessairement équilibré. Par contre si  $D(x)$  et  $G(x)$  sont encore équilibrés après l'insertion de  $y$ ,  $A(x)$  est lui aussi équilibré.

**Exercice 5**

Cas 1 :

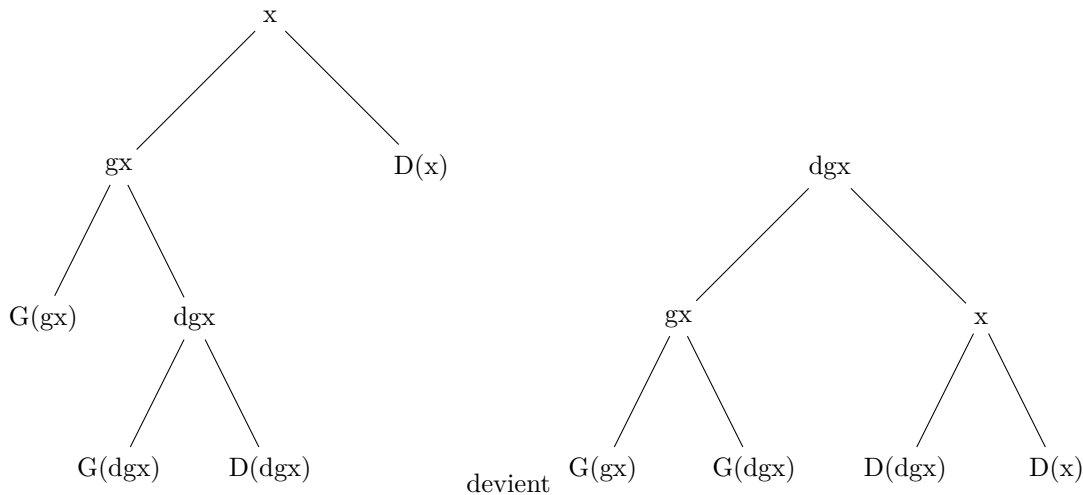


Les noeuds à gauche (respectivement à droite) de  $x$  ou de  $gx$  sont toujours du même coté, nous avons donc toujours un ABR.

Par hypothèse, les sous-arbres de  $x$  sont équilibrés. Soit  $h = h(x)$  avant insertion, après insertion  $h(G(G(x))) = h - 1$  (au lieu de  $h - 2$ ),  $h(D(x)) = h - 3$  (pour qu'il y ait déséquilibre) et  $h(D(G(x))) = h - 2$  (avec  $h - 1$  il y aurait eu un déséquilibre en  $x$  avant l'insertion, avec  $h - 3$  le déséquilibre après insertion serait aussi en  $gx$  contrairement à l'hypothèse).

Après rotation, en notant les  $h'$  la nouvelle hauteur et en gardant les autres notations. Les arbres qui n'ont pas été modifiés sont toujours équilibrés et de même hauteur. Comme avant la rotation  $D(gx)$  et  $D(x)$  diffèrent d'exactly 1 et  $h'(x) = h - 1$ .  $h'(G(gx)) = h$ . L'arbre est donc à nouveau équilibré et de même hauteur  $h$  qu'avant l'insertion.

Cas 2 :



On n'a à nouveau pas changé les positions relatives gauche/droite entre les noeuds, donc on a toujours un ABR.

Comme précédemment, soit  $h = h(x)$  avant insertion, après insertion  $h(dgx) = h - 1$  (au lieu de  $h - 2$ ),  $h(D(x)) = h - 3$  (pour qu'il y ait déséquilibre) et  $h(G(G(x))) = h - 2$  (avec  $h - 1$  il y aurait eu un déséquilibre en  $x$  avant l'insertion, avec  $h - 3$  le déséquilibre après insertion serait aussi en  $gx$  contrairement à l'hypothèse) et entre  $G(dgx)$  et  $D(dgx)$  l'un est de hauteur  $h - 2$  (celui qui a reçu  $y$ ) et l'autre de hauteur  $h - 3$  (même arguments que pour  $G(G(x))$  précédemment).

Les hauteurs de  $G(Gx)$  et  $G(dgx)$  diffèrent donc d'au plus un, de même pour  $D(dgx)$  et  $D(x)$ , de plus  $h'(gx) = h - 1$  et  $h'(x) = h - 1$  ou  $h - 2$  et le nouvelle arbre est équilibré et de même hauteur  $h$  qu'avant l'insertion de  $y$ .

**Exercice 6** On retranscrit simplement les dessins précédent.

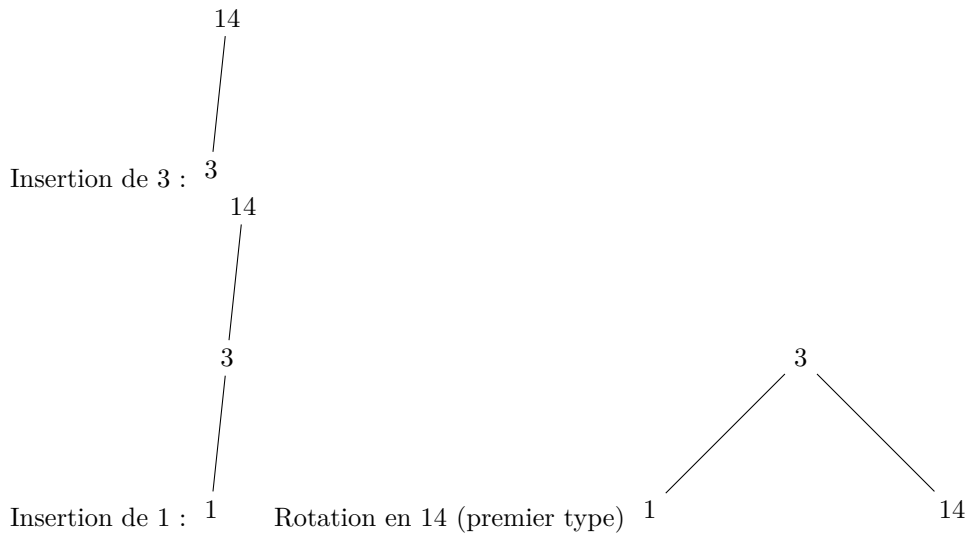
```
recalculer_h(x){
  h(x) <- max(h(G(x)), h(D(x))) + 1
}
```

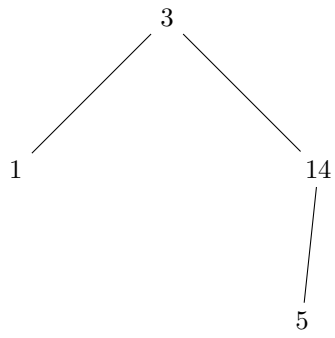
```

Equilibre(x){
  si ( h(G(G(x)))=2+h(D(G(x))) ou h(G(G(x)))=1+h(D(G(x))) ) alors {
    gx <- G(x);
    p(gx) <- p(x); si x=G(p(x)) alors G(p(x)) <- gx; sinon D(p(x)) <- gx
    G(x) <- D(gx); p(D(gx)) <- x;
    p(x) <- gx; G(px) <- x
    recalculer h(x) et h(gx)
  }
  sinon si ( h(D(D(x)))=2+h(G(D(x))) ou h(D(D(x)))=1+h(G(D(x))) ) alors {
    dx <- D(x);
    p(dx) <- p(x); si x=D(p(x)) alors D(p(x)) <- dx; sinon G(p(x)) <- dx
    D(x) <- G(dx); p(G(dx)) <- x;
    p(x) <- dx; D(px) <- x;
    recalculer_h(x); recalculer_h(dx);
  }
  sinon si ( h(D(G(x)))=2+h(G(G(x))) ou h(D(G(x)))=1+h(D(x)) ) alors{
    gx <- G(x);
    dgx <- D(G(x));
    p(dgx) <- p(x) ; si x=G(p(x)) alors G(p(x)) <- dgx; sinon D(p(x)) <- dgx;
    p(D(dgx)) <- x; G(x) <- D(dgx);
    p(gx) <- dgx; G(dgx) <- gx;
    D(gx) <- G(dgx); p(dgx) <- gx;
    recalculer_h(x); recalculer_h(gx); recalculer_h(dgx);
  }
  sinon sinon si ( h(G(D(x)))=2+h(D(D(x))) ou h(G(D(x)))=1+h(G(x)) ) alors{
    dx <- D(x);
    gdx <- G(D(x));
    p(gdx) <- p(x) ; si x=D(p(x)) alors D(p(x)) <- gdx; sinon G(p(x)) <- gdx;
    p(G(gdx)) <- x; D(x) <- G(gdx);
    p(dx) <- gdx; D(gdx) <- dx;
    G(dx) <- D(gdx); p(gdx) <- dx ;
    recalculer_h(x); recalculer_h(dx); recalculer_h(gdx)
  }
}
}

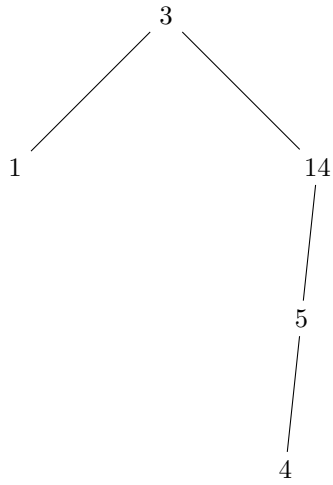
```

**Exercice 7** Insertion de 14 : 14



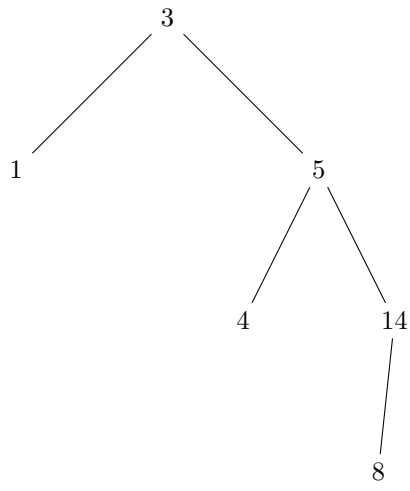
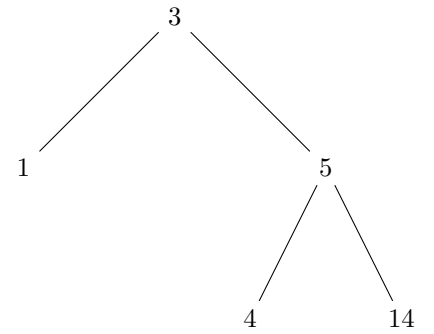


Insertion de 5



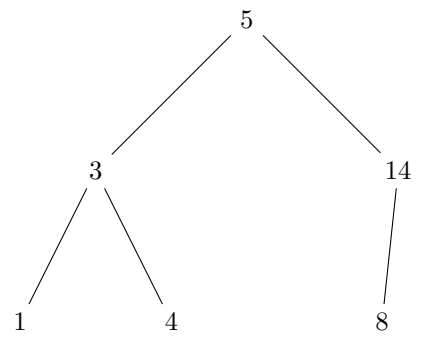
Insertion de 4

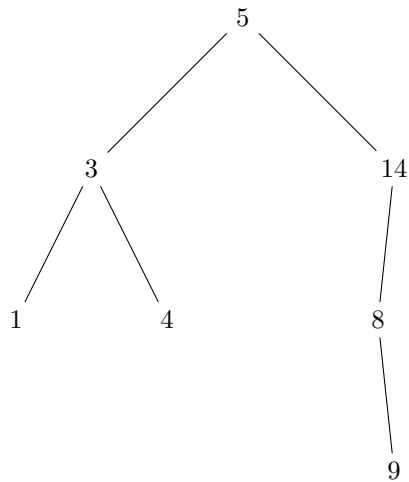
rotation en 14 (premier type) :



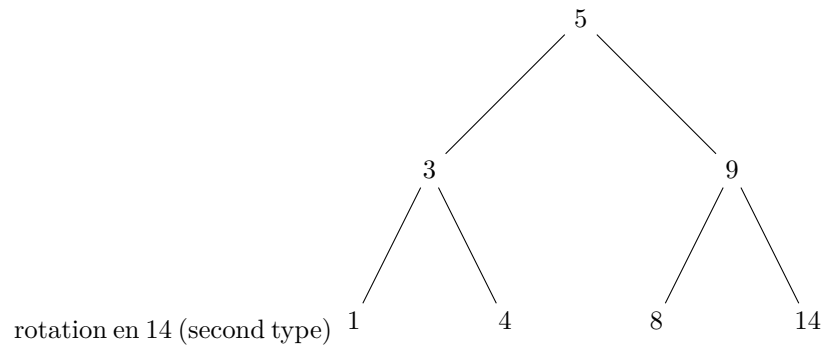
Insertion de 8 :

rotation en 3 (premier type) :



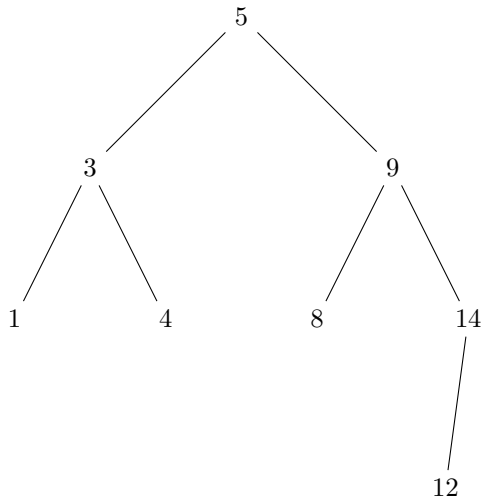


Insertion de 9



rotation en 14 (second type)

Insertion de 9 :



**Exercice 8** Une seule rotation (i.e. appel à équilibrer) est suffisante car on a vu précédemment qu'après une insertion induisant un déséquilibre, la rotation rééquilibre l'arbre et le nouveau arbre a la même hauteur qu'avant l'insertion, les règles d'équilibre sont donc toujours respectées.

### Exercice 9

```

Insérer(x){
  Insérer(x,racine);
}
  
```

```

Insérer(x,c){
  //Insérer x comme dans un ABR.
  si v(x)<v(c) alors{
    si G(c) est vide alors{ G(c)<-x;p(x)<-c;maj(x); }
    sinon Insérer(x,G(c));
  }
}
  
```

```

}
sinon {
  si D(c) est vide alors{ D(c)<-x;p(x)<-c;maj(x); }
  sinon Insérer(x,D(c));
}
}

maj(x){
  tant que x différent de NULL { //Actualiser la hauteur des ancêtres jusqu'à ce :
  h <- h(x);
  recalculer_h(h);
  si h=h(x) alors x <- NULL; //soit h(x) ne change pas
  sinon si (|h(G(x))-h(D(x))|=2) alors {Equilibre(x); x <- NULL;} //soit on atteint un conflit
  sinon x <- p(x);
}
}

```

**Exercice 10** En comptant le nombre d'accès aux noeuds, la complexité est  $\Theta(h)$  c'est à dire logarithmique, précisément car on parcourt au plus un noeud par niveau lors de l'insertion et donc on accède à un nombre constant de noeuds pour chaque niveau (le noeud courant et ses deux fils), de même lors de la mise à jour des hauteurs et la rotation se fait en temps constant.

## 2 La bâtîère

**Exercice 11** return B[1,1] suffit. Temps  $O(1)$ .

**Exercice 12** On déplace le problème par adja-échanges. Attention si  $B[i, j] > B[i + 1, j]$  et  $B[i, j] > B[i, j + 1]$  (problème sur ligne et colonne) on ne fait qu'un adja-échange mais le bon, de sorte de ne pas créer deux problèmes. Exemple avec problème du 4 en (1, 1) :

4	1
2	3

On ne peut échanger 4 qu'avec 1. Si échange avec 2 on obtient deux problèmes :

2	1
4	3

Récurivement cela donne :

**Exercice 13** Soit  $x$  l'élément en  $(i, j)$  et  $g, d, h, b$  ses voisins : 

	h	
g	x	d
	b	

 Dans une bâtîère correcte on

a  $\max(h, g) \leq x \leq \min(b, d)$ . Si  $x$  est incorrect seulement deux cas sont possibles :

- $\max(h, g) \geq x$ . Alors on échange  $x$  avec  $h$  ou  $g$  et on maintient que  $x \leq \min(b, d)$  après chaque échange. Donc on ne va que vers le haut et vers la gauche.
- sinon,  $x \geq \min(b, d)$ . Et là on n'ira que vers le bas et la droite avant rétablissement de l'équilibre car  $\max(h, g) \leq x$  est maintenu.

La complexité de **Rétablir** est donc  $O(n + m)$ .

Pour montrer que l'on rétablit bien la correction de la bâtîère (non demandée ici mais servira exercice 17), il faut prouver qu'on maintient l'invariant "une seule case est incorrecte, en  $(i, j)$ ". Donc on finit par échouer sur un bord (car  $i + j$  soit décroît de 1 à chaque appel, soit augmente de 1 à chaque appel, et est compris entre 2 et  $n + m$ ).

Après échange on peut créer des problème dans la nouvelle case  $B[i \pm 1, j \pm 1]$  mais ils seront réglés récursivement. Montrons que  $B[i, j]$  est maintenant correcte. On a  $h < b, g < d, h < d$  et  $g < b$  par propriétés de la bâtîère. Il faut regarder quatre cas :

**Rétablir**( B : bâtière  $n \times m$ ,  $i, j$  : position d'une case incorrecte);

**début**

```

si ( $i < n$  et  $B[i, j] > B[i + 1, j]$ ) ou ( $j < m$  et  $B[i, j] > B[i, j + 1]$ ) alors
  // NB : cas "élément trop grand" donc adja-échange avec min(bas,droite)
  si ( $i < n$  et  $j < m$  et  $B[i, j + 1] > B[i + 1, j]$ ) ou ( $j = m$ ) alors
     $tmp = B[i, j]; B[i, j] = B[i + 1, j]; B[i + 1, j] = tmp;$ 
    Rétablir( $B, i + 1, j$ );
  sinon
     $tmp = B[i, j]; B[i, j] = B[i, j + 1]; B[i, j + 1] = tmp;$ 
    Rétablir( $B, i, j + 1$ );

sinon si ( $i > 1$  et  $B[i, j] < B[i - 1, j]$ ) ou ( $j > 1$  et  $B[i, j] < B[i, j - 1]$ ) alors
  // NB : cas "élément trop petit" donc adja-échange avec max(haut,gauche)
  si ( $i > 1$  et  $j > 1$  et  $B[i, j - 1] < B[i - 1, j]$ ) ou ( $j = 1$ ) alors
     $tmp = B[i, j]; B[i, j] = B[i - 1, j]; B[i - 1, j] = tmp;$ 
    Rétablir( $B, i - 1, j$ );
  sinon
     $tmp = B[i, j]; B[i, j] = B[i, j - 1]; B[i, j - 1] = tmp;$ 
    Rétablir( $B, i, j - 1$ );

sinon Rien : l'équilibre est rétabli

```

**fin**

-  $x > \min(b, d)$  et  $\min(b, d) = b$ . Alors après échange on a 

	h	
g	b	d
	x	

 : aucun problème pour  $b$

- si  $x > \min(b, d)$  et  $\min(b, d) = d$  alors 

	h	
g	d	x
	b	

 est correct

- si  $x < \max(h, g)$  alors selon le cas on arrive en 

	x	
g	h	d
	b	

 ou 

	h	
x	g	d
	b	

**Exercice 14** On remarque que seul le cas "élément trop grand" de l'exercice 12 sert...

**ExtraireMinimum**(B : bâtière);

**début**

```

 $x = B[1, 1];$ 
 $B[1, 1] = +\infty;$ 
Rétablir( $B, 1, 1$ );
retourner  $x$ ;

```

**fin**

**Exercice 15** Là, seul le cas "élément trop petit" sert...

**Exercice 16** D'après l'exercice 13 **Rétablir** est en  $O(n+m)$ , le reste en temps constant, donc  $O(n+m)$ .

```

Inserer(B : bâtière, x : réel);
début
  | B[n, m] = x;
  | Rétablir(B, n, m);
fin

```

**Exercice 17** On construit une bâtière correcte des  $n^2$  éléments (première boucle) puis on en extrait  $n^2$  fois le minimum (deuxième boucle) ce qui produit donc tous les éléments en ordre trié. Notez que l'on n'extrait jamais les  $+\infty$  rajoutés par **ExtraireMin** qui sont plus grands que les éléments du tableau.

On a besoin de prouver que **Insérer** et **ExtraireMin** sont corrects. Clairement ils font bien les opérations demandées sur le tableau, mais peuvent modifier la propriété de bâtière sur un élément : celui en  $(n, n)$  ou en  $(1, 1)$  respectivement. On a besoin de la correction de **Rétablir**. Celle-ci a été prouvée par anticipation exercice 13.

**Exercice 18** Les **Insérer** et **Rétablir** sont en  $O(n)$  on est donc en  $O(n^3)$  car chacun est appelé  $O(n^2)$  fois. Attention, piège ! Le nombre de case est  $n^2$ . Pour trier  $a$  éléments il faut donc un temps  $O(a\sqrt{a})$ . C'est moins bien que QuickSort et Tri-par-tas en  $O(a \log a)$  mais mieux que Tri-Bulles en  $O(a^2)$ .

**Exercice 19** Chaque ligne étant un tableau trié, on peut faire  $n$  recherches dichotomiques de  $x$ , une par ligne. On est alors en  $O(n \log m)$ .

On peut calculer plus efficacement. Appellons *frontiere*( $i$ ) l'indice où l'on passe de valeurs  $< x$  à des valeurs  $> x$  dans la ligne  $i$  (si  $x$  n'appartient pas à la ligne). Or, si  $i < i'$  alors  $frontiere(i) \geq frontiere(i')$ . L'idée est de rester en permanence sur la frontière, en partant du coin supérieur droit : on incrémentera  $i$   $n$  fois et on décrémentera  $j$   $m$  fois. C'est donc clairement  $O(n + m)$ . L'invariant prouvant la correction est que  $j = frontiere(i)$  avant chaque incrément de  $i$  (par convention la frontière vaut 1 si la ligne n'a que des éléments  $> x$  et  $m$  si que des  $< x$ .)

```

recherche B : bâtière  $n \times m$ , x : valeur cible;
début
  | i = 1;
  | j = m;
  tant que i ≤ n faire
    | tant que j ≥ 2 et B[i, j] > x faire
      | j = j - 1;
      si B[i, j] = x alors retourner (i, j);
      i = i + 1;
  retourner (0, 0) // si ici : échec
fin

```

**Exercice 20** L'anti-diagonale (éléments  $(i, j)$  avec  $i + j = 1 + \min(n, m)$ ) est constituée d'éléments sans aucun ordre : on ne peut rechercher dedans plus vite que la recherche dans un tableau non trié. Donc la recherche dans une bâtière est en  $\Omega(\min(n, m))$ .

### 3 Arbres binomiaux

**Exercice 21** On peut observer facilement que, pour tout  $n \geq 0$ , l'hauteur de l'arbre  $B_n$  est exactement  $n$ . Pour tout  $n \geq 0$  et  $k \leq n$ , on denote par  $S_n^k$  le nombre de sommets de profondeur  $k$  dans l'arbre  $B_n$ .

On prouve par induction sur  $n$  que

$$\text{pour tout } n \text{ et pour tout } k \leq n, S_n^k = \binom{n}{k}. \quad (1)$$

Pour  $n = 0$ , on obtient qu'il y a un seul sommet de profondeur 0 qui est vrai.

Supposons que la propriété (1) est vraie pour  $n$  et prouvons qu'elle est vraie aussi pour  $n + 1$ .

Pour  $k \in \{0, n + 1\}$  c'est facile à prouver que  $S_{n+1}^k = 1$ . L'arbre  $B_{n+1}$  a un seul noeud de profondeur 0, la racine, et un seul noeud de profondeur  $n + 1$ .

Maintenant, soit  $0 < k < n + 1$ . Par définition,  $B_{n+1}$  est formé de la réunion des deux copies  $B_n^{(g)}$  et  $B_n^{(d)}$ . On peut observer que l'ensemble de sommets de profondeur  $k$  de  $B_{n+1}$  est l'union de :

- l'ensemble de sommets de profondeur  $k - 1$  de  $B_n^{(g)}$  et
- l'ensemble de sommets de profondeur  $k$  de  $B_n^{(d)}$ .

Donc,  $S_{n+1}^k = S_n^{k-1} + S_n^k$ . Par l'hypothèse d'induction,  $S_n^{k-1} = \binom{n}{k-1}$  et  $S_n^k = \binom{n}{k}$  qui implique :

$$S_{n+1}^k = \binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k}.$$

**Exercice 22** On prouve la propriété suivante par induction sur  $n$  :

la racine de  $B_n$  a  $n$  fils et  $2^{n-k-1}$  sommets ont  $k$  fils, pour  $0 \leq k < n$

Elle est évidemment vraie pour  $n = 0$ . Maintenant, supposons qu'elle est vraie pour  $n$  et prouvons qu'elle est vraie pour  $n + 1$ .

Par définition, les fils de la racine de  $B_{n+1}$  sont les fils de la racine dans la copie  $B_n^{(d)}$  plus la racine de  $B_n^{(g)}$ . Par l'hypothèse d'induction, la racine de  $B_n^{(d)}$  a  $n$  fils et donc la racine de  $B_{n+1}$  a  $n + 1$  fils.

Soit  $k$  un entier tel que  $0 \leq k < n + 1$ . Pour  $k = n$ , on doit prouver qu'il y a  $2^{n+1-k-1} = 2^{n+1-n-1} = 2^0 = 1$  noeud avec  $n$  fils. En fait, ce noeud est la racine de  $B_n^{(g)}$  (qui a  $n$  fils par l'hypothèse d'induction). Si  $k < n$  alors, par l'hypothèse inductive,  $B_n^{(g)}$  et  $B_n^{(d)}$  contient chacun  $2^{n-k-1}$  noeuds avec  $k$  fils. Donc,  $B_{n+1}$  qui est formé de la réunion de  $B_n^{(g)}$  et  $B_n^{(d)}$  contient  $2 \cdot 2^{n-k-1} = 2^{n+1-k-1}$  noeuds avec  $k$  fils.

**Exercice 23** On prouve par induction sur  $n$  que

pour tout  $k$ , un noeud est de profondeur  $n - k$  ssi son numéro a  $k$  chiffres de "1" en écriture binaire

Pour  $n = 0$  c'est évidemment vraie.

Supposons que la propriété est vraie pour  $n$  et prouvons qu'elle est vraie pour  $n + 1$  aussi. Un sommet de  $B_{n+1}$  est un sommet de  $B_n^{(g)}$  ou de  $B_n^{(d)}$ . S'il est un sommet de  $B_n^{(g)}$  alors il va apparaître sur la même position dans les parcours postfixe de  $B_n^{(g)}$  et  $B_{n+1}$  (donc, on va lui associer le même nombre dans les deux arbres). Par l'hypothèse inductive, un sommet de  $B_n^{(g)}$  est de profondeur  $n - k$  dans  $B_n^{(g)}$  ssi son numéro a  $k$  chiffres de "1" en écriture binaire. Un sommet de  $B_n^{(g)}$  de profondeur  $n - k$  dans  $B_n^{(g)}$  est de profondeur  $n - k + 1$  dans  $B_{n+1}$ . Donc, pour tout sommet  $v$  qui appartient à  $B_n^{(g)}$ ,  $v$  est de profondeur  $n + 1 - k$  dans  $B_{n+1}$  ssi son numéro a  $k$  chiffres de "1" en écriture binaire.

Il nous reste à prouver la même propriété pour les sommets de  $B_n^{(d)}$ . On peut observer que un sommet de  $B_n^{(d)}$  qui apparait sur la position  $i$  dans le parcours postfixe apparait sur la position  $2^n + i$  dans le parcours postfixe de  $B_{n+1}$  (parce qu'on doit parcourir avant tous les  $2^n$  noeuds de  $B_n^{(g)}$ ). Par l'hypothèse inductive, un sommet de  $B_n^{(d)}$  est de profondeur  $n - k$  dans  $B_n^{(d)}$  ssi son numéro a  $k$  chiffres de "1" en écriture binaire. Un sommet de  $B_n^{(d)}$  a la même profondeur  $B_n^{(d)}$  et  $B_{n+1}$ . Donc, pour tout sommet  $v$  qui appartient à  $B_n^{(d)}$ ,  $v$  est de profondeur  $n - k$  dans  $B_{n+1}$  ssi son numéro a  $k + 1$  chiffres de "1" en écriture binaire. Comme  $n - k = (n + 1) - (k + 1)$  on peut conclure.

**Exercice 24** On prouve par induction sur  $n$  que

le nombre de fils d'un sommet est égal au nombre de "1" qui suivent le dernier "0" dans l'écriture binaire de son numéro

Pour  $n = 0$  c'est évidemment vraie.

Supposons que la propriété est vraie pour  $n$  et prouvons qu'elle est vraie pour  $n + 1$  aussi. La racine de  $B_{n+1}$  a  $n$  fils et elle est numérotée par  $2^n - 1$ . Dans l'écriture binaire de  $2^n - 1$  il n'y a pas de "0" et donc, la propriété est vraie pour la racine.

On a vu dans l'exercice précédent que le numéro associé à un sommet de  $B_n^{(g)}$  est le même dans les arbres  $B_n^{(g)}$  et  $B_{n+1}$ . Aussi, un sommet de  $B_n^{(g)}$  a le même nombre de fils dans les deux arbres. Donc, la propriété est vraie pour n'importe quelle noeud de  $B_n^{(g)}$ .

Le numéro associé à un sommet  $v$  de  $B_n^{(d)}$  dans  $B_{n+1}$  est  $2^n + i$ , où  $i$  est le numéro associé à  $v$  dans  $B_n^{(d)}$ . Le nombre de "1" qui suivent le dernier "0" dans l'écriture binaire de  $2^n + i$  est égal au nombre de "1" qui suivent le dernier "0" dans l'écriture binaire de  $i$ . Comme  $v$  a le même nombre de fils dans  $B_n^{(d)}$  et  $B_{n+1}$ , on obtient que la propriété est vraie aussi pour n'importe quelle noeud de  $B_n^{(d)}$ .

**Exercice 25** On prouve par induction sur  $i$  que

$$B_i \text{ a } 2^i - 1 \text{ arêtes.} \quad (2)$$

$B_0$  a  $2^0 - 1 = 0$  arêtes.

Supposons que la propriété est vraie pour  $i$  et prouvons qu'elle est vraie pour  $i + 1$  aussi. On peut observer que le nombre d'arêtes de  $B_{i+1}$  est  $2 \cdot N + 1$ , où  $N$  est le nombre d'arêtes de  $B_i$ . En utilisant l'hypothèse inductive, on obtient que le nombre d'arêtes de  $B_{i+1}$  est  $2 \cdot (2^i - 1) + 1 = 2^{i+1} - 1$ .

Par la définition de  $F_n$  et la propriété (2), le nombre d'arêtes de  $F_n$  est :

$$\sum_{i \in I_n} (2^i - 1) = \left( \sum_{i \in I_n} 2^i \right) - \left( \sum_{i \in I_n} 1 \right) = n - \nu(n).$$

**Exercice 26** La forêt  $F_n$  est l'union de  $B_i$  pour tout  $i \in I_n$ . Dans chaque  $B_i$ , la propriété "si  $y$  est fils de  $x$  alors  $c(y) > c(x)$ " implique que la clé associée à la racine de  $B_i$  est inférieure à toutes les clés associées aux sommets de  $B_i$ . Donc, pour trouver la clé minimale dans la file binomiale  $F_n$  il suffit de comparer seulement les racines de tous les arbres  $B_i$  contenus dans  $F_n$ . Comme on a  $\nu(n)$  arbres, il suffit de faire seulement  $\nu(n) - 1$  comparaisons.

## 4 Tournois

**Exercice 27** Définissons un **puits** comme un sommet d'où ne part aucun arc. Dans un tournoi de  $n$  éléments il arrive donc  $n - 1$  arcs dans un puits. Il y a au plus un puits par tournoi.

Maintenant considérons une **marche**, comme dans l'exercice des graphes eulériens. On part d'un sommet et on suit les arcs

- soit jusqu'à être coincé dans un puits
- soit jusqu'à repasser pour la deuxième fois par un sommet

Clairement on termine toujours sur l'une des deux conditions car il y a un nombre fini d'éléments. Cette marche ne détecte pas toujours un cycle. En revanche, dans un tournoi sans puits, elle terminera toujours sur la deuxième condition donc elle détecte bien un cycle (qui contient ce sommet répété).

D'autre part, on remarque que dans un ordre total il y a un puits (le dernier élément  $x_n$ ) et que quand on l'**enlève** on obtient l'ordre total  $x_1 \dots x_{n-1}$  où  $x_{n-1}$  est à son tour un puits.

D'où l'algorithme :

1. Tant que le tournoi a un puits, l'enlever
2. Si on a enlevé tous les puits alors on a un ordre total
3. Sinon commencer une marche à n'importe quel sommet.
4. La première répétition dans cette marche donne un cycle (les éléments entre la première et la deuxième occurrence de l'élément répété)

C'est la preuve constructive que l'on voulait

**Exercice 28** On implémente plus précisément cet algorithme.  $T$  est une matrice  $n \times n$  avec  $T[i, j] = 1$  si  $ij$  est un arc,  $-1$  si  $ji$  est un arc,  $0$  sinon (alors  $i = j$ ).

**Données :** un tournoi  $T$  de  $n$  éléments

Ordre : liste vide;

**tant que**  $T$  a un puits  $p$  **faire**

  AjouterEnQueue(Ordre,  $p$ );

**si**  $|\text{Ordre}| = n$  **alors**

  Afficher("Ce tournoi est l'ordre total"); AfficherListe(Ordre);

**sinon**

  Marche : liste vide;

$x =$  un sommet de  $T$ ;

**tant que**  $x \notin$  Marche **faire**

    AjouterEnTete(Marche,  $x$ );

    Soit  $xy$  un arc (donc soit  $y$  tel que  $xRy$ );

$x = y$

  Cycle = SousListe( Marche, PremiereOccurrence( Marche,  $x$ ), FinListe(Marche));

  Afficher("Ce tournoi contient le cycle"); AfficherListe(Cycle);

Complexité : rechercher un puits se fait en  $O(n^2)$  (il faut chercher une ligne ne contenant pas de 1). Extraire le sommet se fait aussi en  $O(n^2)$ . Les opérations sur les listes sont en  $O(n)$  au pire. On est donc en  $O(n^3)$ .

On peut faire mieux pour les puits en triant les sommets par degré entrant croissants. Un graphe est alors un ordre total si les degrés entrants sont  $0, 1, 2, \dots, n - 1$ . On construit les degrés en  $O(n^2)$  et on trie en  $O(n \log n)$  donc tout devient en  $O(n^2)$ , linéaire en la taille de l'entrée.