Aix*Marseille université

PORTAIL RENÉ DESCARTES

**Science informatique : Examen**
**2h**

Année 2022-23

**Documents et calculatrice interdits**

*Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre.*

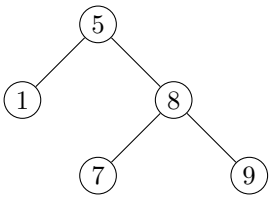
Dans les deux premiers exercices, pour manipuler les arbres, on utilise les fonctions de l'interface vue en cours, à savoir

- le constructeur `arbre_vide()` renvoyant un arbre vide ;
- le constructeur `nouveau_noeud(valeur, enfant_gauche, enfant_droit)` renvoyant un arbre dont on donne la valeur de la racine, l'enfant gauche et l'enfant droit ;
- le prédicat `est_vide(arbre)` testant si l'arbre est vide ;
- les fonctions d'accès `valeur(arbre)`, `enfant_gauche(arbre)` et `enfant_droit(arbre)`, renvoyant respectivement la valeur, l'enfant gauche et l'enfant droit de l'arbre supposé non vide donné en argument.

**Exercice 1** On considère la fonction `mystere` ci-dessous, qui prend en argument un arbre binaire de recherche *non vide*.

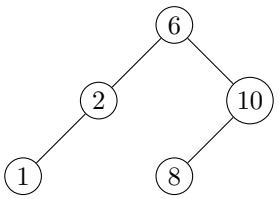
```
def mystere(arbre):
  enfant = enfant_droit(arbre)
  if est_vide(enfant):
    return valeur(arbre)
  else:
    return mystere(enfant)
```

1. Que retourne la fonction quand on lui passe en argument l'arbre suivant ?

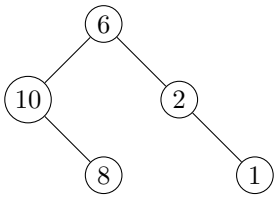



2. Décrivez en une phrase ce que la fonction `mystere` calcule dans le cas général.
3. Quel est l'ordre de grandeur de la complexité dans le pire des cas de la fonction `mystere` en fonction de la hauteur $h$ de l'arbre passé en argument ?
4. Quel est l'ordre de grandeur de la complexité dans le pire des cas de la fonction `mystere` lorsque l'argument est un arbre binaire de recherche *équilibré* de taille $n$ ? On exprimera la complexité en fonction de $n$. (Pour rappel, les arbres binaires de taille $n$ *équilibrés* sont les arbres binaires de hauteur minimale parmi les arbres binaires de taille $n$.)

**Exercice 2** Donnez le code python d'une fonction récursive `miroir` prenant en argument un arbre binaire de recherche `arbre` et renvoyant un arbre binaire `reflet`, tel qu'un parcours infixe de `reflet` lise les entiers contenus dans `arbre` dans l'ordre décroissant. Par exemple, sur l'entrée :

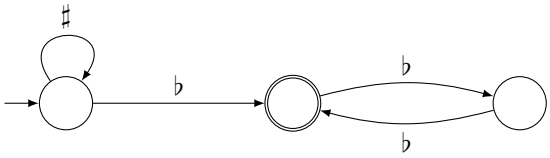



la fonction devrait retourner :



**Exercice 3**

1. Donnez l'alphabet ainsi que l'ensemble de toutes les séquences de lettres acceptées par l'automate ci-dessous.

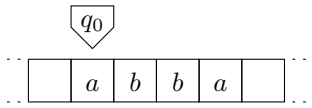



2. Donnez un automate fini qui reconnaît les séquences sur l'alphabet $\{a, b\}$ qui terminent par la séquence $aba$. Par exemple, il doit accepter les séquences $aba, aaba, abbababa, \ldots$ et refuser les séquences $abab, abbba, bbba, \ldots$

**Exercice 4** Considérons une machine de Turing qui travaille sur l'alphabet $a, b, \not{a}, \not{b}$ et qui accepte (c'est-à-dire, termine son exécution dans l'état $OK$) exactement toutes les séquences de caractères en entrée qui ont autant de $a$ que de $b$ ; les autres séquences sont rejetées (c'est-à-dire la machine s'arrête dans un état différent de $OK$). Voici sa table de transition, où une case blanche est dénotée par le symbole $\text{␣}$ et l'état initial est $q_0$ :

| | état | symbole lu | symbole écrit | sens | état suivant |
|---|---|---|---|---|---|
| 1 | $q_0$ | $a$ | $\text{␣}$ | $\rightarrow$ | $q_a$ |
| 2 | $q_0$ | $b$ | $\text{␣}$ | $\rightarrow$ | $q_b$ |
| 3 | $q_0$ | $\not{a}$ | $\text{␣}$ | $\rightarrow$ | $q_0$ |
| 4 | $q_0$ | $\not{b}$ | $\text{␣}$ | $\rightarrow$ | $q_0$ |
| 5 | $q_0$ | $\text{␣}$ | $\text{␣}$ | $\leftarrow$ | $OK$ |
| 6 | $q_a$ | $a$ | $a$ | $\rightarrow$ | $q_a$ |
| 7 | $q_a$ | $b$ | $\not{b}$ | $\leftarrow$ | $q_L$ |
| 8 | $q_a$ | $\not{a}$ | $\not{a}$ | $\rightarrow$ | $q_a$ |
| 9 | $q_a$ | $\not{b}$ | $\not{b}$ | $\rightarrow$ | $q_a$ |
| 10 | $q_b$ | $a$ | $\not{a}$ | $\leftarrow$ | $q_L$ |
| 11 | $q_b$ | $b$ | $b$ | $\rightarrow$ | $q_b$ |
| 12 | $q_b$ | $\not{a}$ | $\not{a}$ | $\rightarrow$ | $q_b$ |
| 13 | $q_b$ | $\not{b}$ | $\not{b}$ | $\rightarrow$ | $q_b$ |
| 14 | $q_L$ | $a$ | $a$ | $\leftarrow$ | $q_L$ |
| 15 | $q_L$ | $b$ | $b$ | $\leftarrow$ | $q_L$ |
| 16 | $q_L$ | $\not{a}$ | $\not{a}$ | $\leftarrow$ | $q_L$ |
| 17 | $q_L$ | $\not{b}$ | $\not{b}$ | $\leftarrow$ | $q_L$ |
| 18 | $q_L$ | $\text{␣}$ | $\text{␣}$ | $\rightarrow$ | $q_0$ |

1. Voici la configuration initiale de la machine de Turing sur l'entrée $abba$, avec la tête de lecture et écriture placée sur le premier symbole de l'entrée :




   Simulez l'exécution de la machine en montrant pour chaque étape de calcul le contenu du ruban, la position de la tête de lecture et écriture et l'état dans lequel la machine se trouve. Est-ce que l'entrée est acceptée ?

2. Simulez également la machine de Turing sur l'entrée $aab$. Est-ce que cette entrée est acceptée ?
3. Modifiez la table de transition de la machine de Turing (en ajoutant, supprimant ou modifiant des transitions) pour construire une nouvelle machine qui accepte *les séquences de caractères où le nombre de a et de b est différent* ; la machine devra donc accepter, par exemple, la séquence $aaaaab$ et rejeter la séquence $bababa$. Il n'est pas nécessaire de recopier les transitions qui restent identiques entre les deux machines.

Aix*Marseille université

PORTAIL RENÉ DESCARTES

**Science informatique : Examen**
**2h**

Année 2022-23

**Documents et calculatrice interdits**

*Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre.*

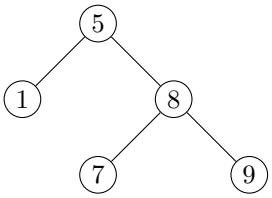
Dans les deux premiers exercices, pour manipuler les arbres, on utilise les fonctions de l'interface vue en cours, à savoir

- le constructeur `arbre_vide()` renvoyant un arbre vide ;
- le constructeur `nouveau_noeud(valeur, enfant_gauche, enfant_droit)` renvoyant un arbre dont on donne la valeur de la racine, l'enfant gauche et l'enfant droit ;
- le prédicat `est_vide(arbre)` testant si l'arbre est vide ;
- les fonctions d'accès `valeur(arbre)`, `enfant_gauche(arbre)` et `enfant_droit(arbre)`, renvoyant respectivement la valeur, l'enfant gauche et l'enfant droit de l'arbre supposé non vide donné en argument.

**Exercice 1** On considère la fonction `mystere` ci-dessous, qui prend en argument un arbre binaire de recherche *non vide*.

```
def mystere(arbre):
  enfant = enfant_droit(arbre)
  if est_vide(enfant):
    return valeur(arbre)
  else:
    return mystere(enfant)
```

1. Que retourne la fonction quand on lui passe en argument l'arbre suivant ?




**Solution :** La fonction retourne 9.

2. Décrivez en une phrase ce que la fonction `mystere` calcule dans le cas général.

**Solution :** La fonction `mystere` renvoie la valeur maximale contenue dans l'arbre.

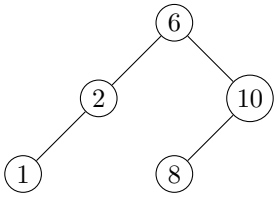
3. Quel est l'ordre de grandeur de la complexité dans le pire des cas de la fonction `mystere` en fonction de la hauteur $h$ de l'arbre passé en argument ?

**Solution :** Le pire des cas correspond à un arbre possédant une unique feuille et tel que seul l'enfant droit de chaque noeud interne est non vide, conduisant à une complexité dans le pire des cas en $O(h)$.

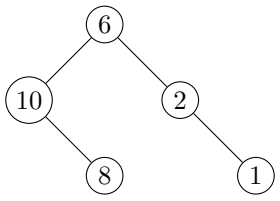
4. Quel est l'ordre de grandeur de la complexité dans le pire des cas de la fonction `mystere` lorsque l'argument est un arbre binaire de recherche *équilibré* de taille $n$ ? On exprimera la complexité en fonction de $n$. (Pour rappel, les arbres binaires de taille $n$ *équilibrés* sont les arbres binaires de hauteur minimale parmi les arbres binaires de taille $n$.)

**Solution :** Le nombre d'appels récursifs est borné par la hauteur de l'arbre, qui, pour un arbre équilibré, est en $O(\log(n))$. Cette borne est atteinte pour tout arbre binaire de recherche équilibré (en d'autres termes, tous les cas sont des « pire cas » quand on se restreint aux arbres binaires de recherche équilibrés). On obtient donc une complexité dans le pire des cas en $O(\log(n))$.

**Exercice 2** Donnez le code python d'une fonction récursive `miroir` prenant en argument un arbre binaire de recherche `arbre` et renvoyant un arbre binaire `reflet`, tel qu'un parcours infixe de `reflet` lise les entiers contenus dans `arbre` dans l'ordre décroissant. Par exemple, sur l'entrée :



la fonction devrait retourner :

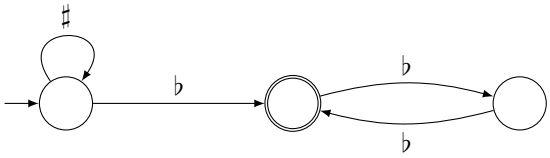



**Solution :**

```
def miroir(arbre):
  if est_vide(arbre):
    return arbre_vide()
  else:
    return nouveau_noeud(valeur(arbre),
                         miroir(enfant_droit(arbre)),
                         miroir(enfant_gauche(arbre)))
```

**Exercice 3**

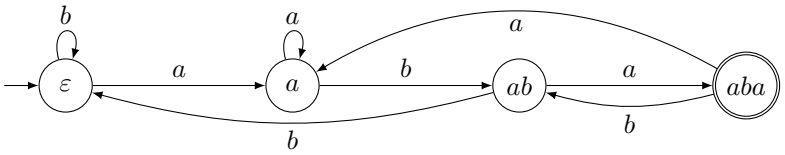
1. Donnez l'alphabet ainsi que l'ensemble de toutes les séquences de lettres acceptées par l'automate ci-dessous.




**Solution :** L'automate a pour alphabet $\{\sharp, \flat\}$. Il reconnait l'ensemble des séquences contenant d'abord un nombre quelconque de $\sharp$, suivi d'un nombre impair de $\flat$.

2. Donnez un automate fini qui reconnaît les séquences sur l'alphabet $\{a, b\}$ qui terminent par la séquence $aba$. Par exemple, il doit accepter les séquences $aba, aaba, abbababa, \ldots$ et refuser les séquences $abab, abbba, bbba, \ldots$

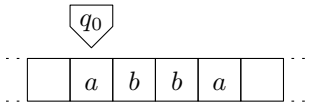
**Solution :**




**Exercice 4** Considérons une machine de Turing qui travaille sur l'alphabet $a, b, \not{a}, \not{b}$ et qui accepte (c'est-à-dire, termine son exécution dans l'état $OK$) exactement toutes les séquences de caractères en entrée qui ont autant de $a$ que de $b$ ; les autres séquences sont rejetées (c'est-à-dire la machine s'arrête dans un état différent de $OK$). Voici sa table de transition, où une case blanche est dénotée par le symbole ␣ et l'état initial est $q_0$ :

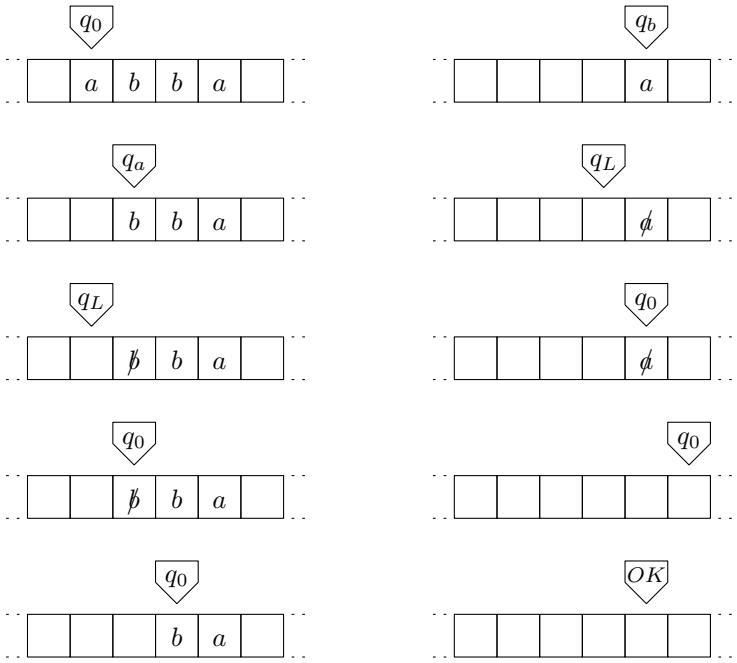
| | état | symbole lu | symbole écrit | sens | état suivant |
|---|---|---|---|---|---|
| 1 | $q_0$ | $a$ | $\sqcup$ | $\rightarrow$ | $q_a$ |
| 2 | $q_0$ | $b$ | $\sqcup$ | $\rightarrow$ | $q_b$ |
| 3 | $q_0$ | $\not\triangleleft$ | $\sqcup$ | $\rightarrow$ | $q_0$ |
| 4 | $q_0$ | $\not\triangleright$ | $\sqcup$ | $\rightarrow$ | $q_0$ |
| 5 | $q_0$ | $\sqcup$ | $\sqcup$ | $\leftarrow$ | $OK$ |
| 6 | $q_a$ | $a$ | $a$ | $\rightarrow$ | $q_a$ |
| 7 | $q_a$ | $b$ | $\not\triangleright$ | $\leftarrow$ | $q_L$ |
| 8 | $q_a$ | $\not\triangleleft$ | $\not\triangleleft$ | $\rightarrow$ | $q_a$ |
| 9 | $q_a$ | $\not\triangleright$ | $\not\triangleright$ | $\rightarrow$ | $q_a$ |
| 10 | $q_b$ | $a$ | $\not\triangleleft$ | $\leftarrow$ | $q_L$ |
| 11 | $q_b$ | $b$ | $b$ | $\rightarrow$ | $q_b$ |
| 12 | $q_b$ | $\not\triangleleft$ | $\not\triangleleft$ | $\rightarrow$ | $q_b$ |
| 13 | $q_b$ | $\not\triangleright$ | $\not\triangleright$ | $\rightarrow$ | $q_b$ |
| 14 | $q_L$ | $a$ | $a$ | $\leftarrow$ | $q_L$ |
| 15 | $q_L$ | $b$ | $b$ | $\leftarrow$ | $q_L$ |
| 16 | $q_L$ | $\not\triangleleft$ | $\not\triangleleft$ | $\leftarrow$ | $q_L$ |
| 17 | $q_L$ | $\not\triangleright$ | $\not\triangleright$ | $\leftarrow$ | $q_L$ |
| 18 | $q_L$ | $\sqcup$ | $\sqcup$ | $\rightarrow$ | $q_0$ |

1. Voici la configuration initiale de la machine de Turing sur l'entrée *abba*, avec la tête de lecture et écriture placée sur le premier symbole de l'entrée :




Simulez l'exécution de la machine en montrant pour chaque étape de calcul le contenu du ruban, la position de la tête de lecture et écriture et l'état dans lequel la machine se trouve. Est-ce que l'entrée est acceptée ?

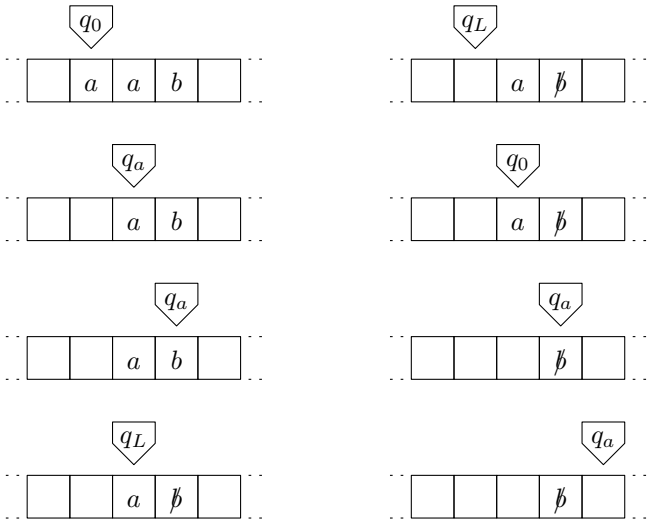
**Solution :**




Comme la machine s'arrête dans l'état $OK$, l'entrée *abba* est acceptée.

2. Simulez également la machine de Turing sur l'entrée *aab*. Est-ce que cette entrée est acceptée ?

**Solution :**




La machine s'arrête dans l'état $q_a$, qui n'est pas l'état d'acceptation, donc elle rejette l'entrée *aab*.

3. Modifiez la table de transition de la machine de Turing (en ajoutant, supprimant ou modifiant des transitions) pour construire une nouvelle machine qui accepte *les séquences de caractères où le nombre de a et de b est différent* ; la machine devra donc accepter, par exemple, la séquence *aaaaab* et rejeter la séquence *bababa*. Il n'est pas nécessaire de recopier les transitions qui restent identiques entre les deux machines.

**Solution :** La nouvelle machine doit rejeter quand la machine originelle accepte (c'est-à-dire quand on rencontre une case blanche dans l'état $q_0$) et accepter quand on trouve une case blanche dans les états $q_a$ ou $q_b$ ; en effet, dans ces états on a déjà trouvé un $a$ ou un $b$ et on cherche respectivement un $b$ ou un $a$ pour les équilibrer.
Il est donc suffisant de supprimer la transition numéro 5 et d'ajouter les deux transitions suivantes :

| | état | symbole lu | symbole écrit | sens | état suivant |
|---|---|---|---|---|---|
| 19 | $q_a$ | $\sqcup$ | $\sqcup$ | $\leftarrow$ | *OK* |
| 20 | $q_b$ | $\sqcup$ | $\sqcup$ | $\leftarrow$ | *OK* |