Université Denis-Diderot
Licence I
Informatique: IF2

Examen Informatique IF2

Première session, durée 3 heures.
Tous les documents sont interdits.
(5 pages)

14

Cet examen comporte un questionnaire à choix multiples que vous devez rendre avec votre copie.
Pour vous identifier sur le questionnaire tout en conservant l'anonymat vous devez choisir un identifiant quelconque que vous mettrez en tête de votre copie et que vous reporterez sur le questionnaire et sur toutes les copies supplémentaires que vous allez rendre.

Les questions sont indépendantes. Le barème est donné à titre indicatif.

1. On dispose d'une urne contenant des boules rouges et des boules noires. On suppose qu'il y a au moins 2 boules dans l'urne et qu'on dispose d'une réserve infinie de boules noires. On tire deux boules dans l'urne, si les deux boules sont rouges, on les jette et on remet dans l'urne une boule noire, si une boule est noire et l'autre rouge, on remet dans l'urne la boule rouge, si les deux boules sont noires on remet dans l'urne une boule noire. Le problème est de déterminer en fonction du nombre de boules noires et de boules rouges quelle est la couleur de la dernière boule dans l'urne.

   (a) S'il y a n boules au départ dans l'urne au bout de combien de tirages n'y aura-t-il plus qu'une seule boule dans l'urne? (0,5 pt)

   (b) S'il y a deux boules rouges et une boule noire, quelle sera la couleur de la dernière boule? (0,5 pt)

   (c) S'il y a deux boules noires et une boule rouge, quelle sera la couleur de la dernière boule? (0,5 pt)

   (d) Quelle propriété concernant la parité du nombre de boules rouges est conservée à chaque tirage? (1 pt)

   (e) Peut-on prédire en fonction du nombre initial de boules rouges et de boules noires quelle sera la couleur de la dernière boule dans l'urne? Si oui donner une telle condition permettant de savoir quelle sera cette couleur. Justifier votre réponse grâce à la question précédente (1,5 pt)

2. Pierre et Jacques jouent à un jeu: Pierre pense à un nombre x compris entre $0 \leq x < n$ et Jacques doit déterminer ce nombre en posant des questions à Jacques. Jacques connaît n et Pierre est obligé de répondre honnêtement aux questions.

   (a) Jacques a le droit uniquement de proposer un nombre m et de demander à Pierre si le nombre proposé est égal ou non au nombre x. Définir une stratégie pour Jacques. Dans le meilleur cas, combien Jacques posera de questions? Dans le pire cas, combien de questions posera-t-il? (1 pt)

   (b) Jacques a le droit de proposer un nombre m et de demander si ce nombre est plus grand, égal ou plus petit que x. On suppose que $n = 2^k$ pour un certain k. Définir une stratégie pour Jacques qui vous paraît la meilleure possible. Dans le meilleur cas, combien Jacques posera de questions? Dans le pire cas, pour la stratégie choisie, combien de questions posera-t-il? Que peut-on dire si n n'est pas une puissance de 2? (1,5 pt)

   (c) On suppose que $n = 2^k$ pour un certain k. Jacques a le droit de proposer un nombre i $(0 \leq i < k)$ et de demander dans la représentation de m en binaire si le i-ème bit de x est 0 ou 1. Jacques pose cette question pour chaque i $(0 \leq i < k)$. Combien de questions en fonction de n, sont nécessaires pour déterminer x? (1,5 pt)

3. On suppose dans cette partie que l'on dispose d'une classe `MaPileEntier` qui implémente l'interface `PileEntier` suivante :

```
interface PileEntier{
    public boolean estVide();
    public void empiler(int i);
    public int depiler();
    public int sommet();
}
```

On suppose que cette implémentation assure les propriétés usuelles des piles: `estVide()` retourne `true` si la pile est vide, `empiler(i)` insère l'entier `i` au sommet de la pile, `depiler()` retourne et supprime la valeur entière au sommet de la pile et `sommet()` retourne (sans supprimer) la valeur entière au sommet de la pile.

(a) On suppose que l'on dispose de la classe suivante:

```
class NoeudEntier {
    int item;
    NoeudEntier suivant;
    public NoeudEntier(int i) {
        item=i;
        suivant=null;
    }
    public NoeudEntier(int i, NoeudEntier n){
        item=i;
        suivant=n;
    }
}
```

En définissant les méthodes déclarées dans `PileEntier` définir une classe `MaPileEntier` qui réalise une implémentation de `PileEntier` en utilisant `NoeudEntier` et qui, bien entendu, assure les propriétés usuelles des piles. (2 pt)

(b) Une pile p d'entiers est triée si ses éléments sont dans le bon ordre, plus précisément, p est triée si:

- soit p est vide,
- soit si v est le sommet de la pile et si q est la pile obtenue après $p.depiler()$ alors soit q est vide, soit $v < q.sommet()$ et q est triée.

L'insertion ordonnée de x dans une pile triée p consiste à insérer x dans la pile de façon à ce que la nouvelle pile obtenue soit triée.

Remarquons que pour une insertion ordonnée de x dans une pile triée, on peut procéder ainsi:

- si la pile est vide alors il suffit d'empiler x,
- sinon si x est plus petit que le sommet de la pile alors il suffit d'empiler x,
- sinon si y est le sommet de la pile il suffit de dépiler, de faire une insertion ordonnée de x, puis d'empiler y.

i. Écrire une méthode récursive qui permet d'insérer un entier dans une pile `MaPileEntier` triée. (2 pt)

ii. Pour insérer x combien, en fonction du nombre d'éléments dans la pile, peut-on être amené à faire d'opérations empiler et dépiler? (0,5pt)

iii. Écrire une méthode qui à partir d'un tableau d'entiers affiche les éléments de ce tableau triés en utilisant une pile et la méthode insérer définie précédemment. (1 pt)

iv. Quelle est dans le pire cas le nombre d'opérations empiler et dépiler pour trier le tableau. (0,5 pt)