TD d'Éléments d'Algorithmique n° 3

Listes, piles et arbres binaires

I) Modélisation

Exercice 1. *Modélisation d'un jeu de cartes.*

On considère un jeu de cartes classique où chaque carte est définie par sa valeur (dans l'ordre : 1,2,3,4,5,6,7,8,9,10,Valet,Dame,Roi) et son enseigne (pique, cœur, carreau et trèfle). Une main à n cartes est un ensemble de n cartes distinctes.

1. Combien y a-t-il de cartes différentes possibles ?
2. Proposez une représentation d'une carte.
3. Proposez un ordre *total* $\preceq$ sur les cartes respectant l'ordre des valeurs, c'est-à-dire que si c_1 et c_2 sont deux cartes et que la valeur de c_1 est strictement plus petite que la valeur de c_2 (par exemple, c_1 a pour valeur "8" et c_2 a pour valeur "Dame"), alors $c_1 \preceq c_2$.
4. Écrivez un fonction successeur qui prend en paramètre une carte et qui renvoie son successeur pour l'ordre $\preceq$. Utilisez cette fonction pour énumérer toutes les cartes dans l'ordre $\preceq$.
5. Combien y a-t-il de mains de n cartes ?
6. On représente une main de n cartes comme un tableau de n cartes trié dans l'ordre $\preceq$. Écrivez un algorithme qui énumère toutes les mains de n dans l'ordre lexicographique (induit par $\preceq$).

II) Listes

Exercice 2. *Longueur d'une liste.*

- Écrire un algorithme itératif qui calcule la longueur d'une liste.
- Écrire une version récursive du même algorithme.

Exercice 3. *Ajouter à la fin de la liste.*

- Écrire un algorithme itératif qui ajoute un élément à la fin d'une liste.
- Écrire une version récursive du même algorithme.

Exercice 4. *Chainer deux listes.*

- Écrire un algorithme itératif qui chaîne deux listes.
- Écrire une version récursive du même algorithme.

Exercice 5. *Supprimer un élément d'une liste.*

On cherche à concevoir un algorithme prenant en argument une liste L d'entiers et un entier n, et qui enlève de L toute occurence de n et retourne le résultat. Si n n'apparaît pas dans L, l'algorithme retourne donc L.
Par exemple, sur la liste 1, 2, 1, 3, 4, 1, 5 et l'entier 1, l'algorithme renverra la liste 2, 3, 4, 5.

- Écrire une version itérative de cet algorithme.
- Écrire une version récursive du même algorithme.

Exercice 6. *Insertion triée.*

- On suppose donnée une liste d'entiers, triée par ordre croissant. Écrire un algorithme récursif qui va insérer un nouvel élément dans la liste à sa position correcte, de manière à ce que la nouvelle liste soit encore triée.
- Écrire ensuite, à l'aide de l'algorithme précédent, un algorithme qui fait une copie triée d'une liste.

Exercice 7. *Inverser une liste en place.*

- Écrire un algorithme qui inverse l'ordre des éléments dans une liste sans utiliser de nouvelles listes mais simplement en réorganisant les éléments de la liste.

III) Piles

Exercice 8. *Retournement de pile.*

Donner un algorithme prenant en entrée une pile et renvoyant la pile contenant les mêmes éléments, mais dans l'ordre inverse (le sommet de la pile originale sera son dernier élement, etc).

Exercice 9. *Implémenter des piles à l'aide d'un tableau.*

Pour un entier $n \in \mathbb{N}$, expliquer comment implémenter deux piles dans un seul tableau de taille n, en supposant que la somme des hauteurs de ces deux piles ne dépasse jamais n. Les opérations d'empilement et de dépilement devront être données en $O(1)$.

Dans un deuxième temps, on expliquera comment éliminer la restriction sur les hauteurs des piles.

Exercice 10. *Trier des piles d'entiers.*

1. Ecrire un algorithme prenant en entrée une pile de nombres entiers, et retournant une pile contenant les mêmes éléments mais dans laquelle les nombres pairs sont en-dessous des nombres impairs. L'algorithme pourra utiliser une pile auxiliaire.
2. Ecrire un algorithme prenant en entrée une pile de nombres entiers, et retournant une pile contenant uniquement ses éléments pairs. On demande de plus qu'à la fin de l'algorithme la pile fournie en argument retrouve son état initial. L'algorithme pourra utiliser une pile auxiliaire.

Exercice 11. *Piles et bon parenthésage.*

On suppose qu'une pile est donnée sur un alphabet de deux symboles (et). En dépilant successivement ses éléments (en partant donc du sommet de la pile), on obtient une suite de parenthèses ; on dit que c'est le mot représenté par la pile. On s'intéresse aux mots *bien parenthésés*, c'est-à-dire aux mots tels que toute parenthèse fermante) ferme effectivement une unique parenthèse ouvrante (qui la précède, et que toute parenthèse ouvrante soit fermée par une unique parenthèse fermante. Par exemple, (()(())) et (((())(((()))()))(()()) sont bien parenthésés, tandis que ((())) et)(()())(ne le sont pas.

1. Ecrire un algorithme prenant en entrée une pile représentant un mot et retournant vrai si la pile représente un mot bien parenthésé et faux sinon.

2. Ecrire un algorithme prenant en entrée une pile représentant un mot et retournant -1 si la pile représente un mot bien parenthésé et la position de la première parenthèse problématique sinon (la position du symbole en haut de la pile est 0, en le dépilant on accède à la position 1, etc).

Exercice 12. *Piles et palindromes.*

On appelle *mot* une suite de caractères. On appelle *miroir* d'un mot le mot obtenu en lisant celui-ci à l'envers. Par exemple, *pile* a pour miroir *elip*. On dit qu'un mot est un *palindrome* s'il est égal à son miroir.
Ecrire un algorithme prenant en entrée un mot et renvoyant vrai si celui-ci est un palindrome et faux sinon. On pourra utiliser une pile auxiliaire.