TP : Piles et Récursivité

I) Piles

Exercice 1

1) Implémenter des piles en Python à l'aide de listes : écrire trois fonctions
- `creer`, pour créer une pile
- `empiler(c,x)` pour rajouter un élément à une pile.
- `depiler(c)` pour enlever un élément d'une pile (attention, il y a un test à faire [1]).

2) (Si vous avez fait le 1) en moins de 10 minutes) Même question avec un tableau NumPy : on pourra stocker la longueur de la liste dans la première case. Si la pile est pleine, on pourra doubler la taille du tableau.

II) Suite de Fibonacci

On définit la suite (u_n) par

$$u_0 = 0 \qquad u_1 = 1 \qquad \forall n \in \mathbb{N} - \{0,1\} \quad u_n = u_{n-1} + u_{n-2}$$

C'est la « suite de Fibonacci », une suite d'entiers qui a été utilisée au XIIIe siècle pour décrire la croissance d'une population de lapins (on compte les couples à chaque génération, sans décès).
Vous remarquerez que c'est une suite récurrente linéaire : vous connaissez des techniques pour calculer directement le n-ième terme. Mais ces techniques, appliquées à l'informatique, font utiliser des `float` et donc potentiellement perdre en précision (en fait, avec le théorème des accroissements finis on réussit à maîtriser l'erreur... mais ce n'est pas le sujet ici).
Tous les algorithmes que vous écrirez doivent être testé !!

A) Version récursive naïve

Exercice 2

1) Écrire une fonction récursive qui calcule le n-ième terme de la suite de Fibonacci.

2) Tester la rapidité (un n raisonnable pourra être obtenu par dichotomie). Que constate-t-on ?

3) Expliquer le phénomène à l'aide d'un arbre décrivant le fonctionnement de votre programme. Quelle semble être la complexité de cet algorithme ?

B) En partant du bas

Exercice 3

Écrire un algorithme itératif (c'est-à-dire une boucle) qui calcule u_n « du bas vers le haut », donc en partant de u_0 et u_1. Donner la complexité de cet algorithme.

C) Version avec mémoïsation

On revient à des algorithmes récursifs, en essayant d'améliorer la complexité.

Exercice 4

Améliorer l'algorithme récursif naïf en stockant les u_k déjà calculés (et donc en vérifiant, à chaque étape, si on a ou non déjà calculé le u_n que l'on cherche). Donner la complexité du nouvel algorithme.

D) Encore mieux

On admet les formules : $\forall n \in \mathbb{N}^*$ $\left\{ \begin{array}{rcl} u_{2n+1} & = & u_{n+1}^2 + u_n^2 \\ u_{2n} & = & 2u_n u_{n-1} + u_n^2 \end{array} \right.$

Exercice 5

Écrire un algorithme récursif avec mémoïsation qui utilise ces formules. Quelle est sa complexité (sans preuve : on pourra se contenter de faire un calcul dans le cas où n est d'une forme sympathique).

1. `if`, ou `assert` pour ceux qui veulent.

III) Dessin

Exercice 6
Écrire une fonction récursive qui permet de tracer l'arbre ci-dessous (ou des motifs semblables).