Exercice 1. Dichotomie en récursif

Recherche dichotomique. On rappelle qu'il est possible de tester l'appartenance d'un élément `x` à une liste `L` en temps logarithmique grâce à une recherche par dichotomie. La fonction itérative correspondante est donnée ci-dessous.

```
def appartient_dicho_it(L, e):
    i = 0
    j = len(L)-1
    while i <= j:
        m = (i+j)//2
        if L[m] == e:
            return True
        elif L[m] < e:
            i = m+1
        else:
            j = m-1
    return False
```

1. Dans le cours sur les fonctions récursives, on a vu comment convertir une fonction itérative en fonction récursive. Convertir la fonction `appartient_dicho_it` en une fonction récursive `appartient_dicho_rec`.

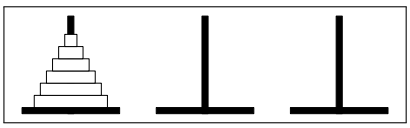
Exponentiation rapide. Pour tout réel $x \in \mathbb{R}_+^*$ et tout entier $n \in \mathbb{N}$:

$$\begin{cases} x^n = 1, & \text{si } n = 0. \\ x^n = (x \times x)^{\frac{n}{2}}, & \text{si } n > 0 \text{ est pair.} \\ x^n = x \times (x \times x)^{\frac{n-1}{2}}, & \text{si } n > 0 \text{ est impair.} \end{cases}$$

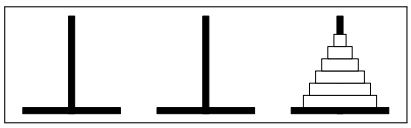
2. En utilisant cette formule, écrire une fonction récursive `expo_rapide` qui prend en entrée x, n, et renvoie x^n. Si n est strictement négatif, votre fonction déclenchera une erreur. Vous devez écrire la fonction `expo_rapide` directement sans passer par une fonction intermédiaire et sans utiliser l'opérateur `**`.

Exercice 2. Tours de Hanoï

Le problème des ***tours de Hanoï*** est un jeu de réflexion dans lequel des disques sont disposés sur trois plots. Chaque disque a un diamètre différent des autres et ne doit jamais se trouver au dessus d'un disque de diamètre inférieur. Initialement, tous les disques sont positionnés sur le plot numéro 1 et le but est de les déplacer sur le plot numéro 3.

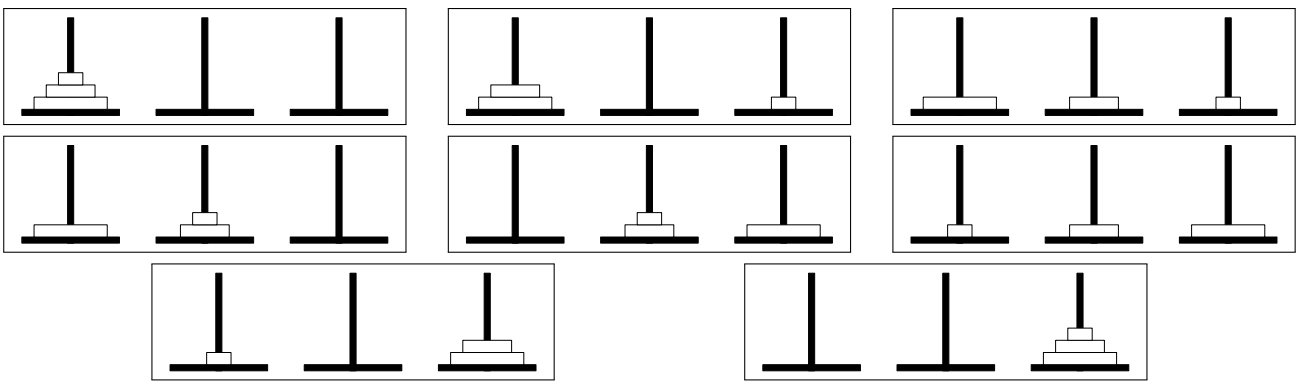


Situation initiale



Situation finale

Pour cela, à chaque étape, il est possible de déplacer un disque se trouvant au sommet d'un plot vers le sommet d'un autre plot. Il faut également penser à respecter la contrainte stipulant qu'un disque ne peut jamais se trouver au dessus d'un autre disque de diamètre inférieur. Par exemple, avec trois disques, la stratégie optimale pour résoudre le problème des tours de Hanoï est :



En Python, un déplacement est représenté par un couple d'entiers. Par exemple, le couple `(1,2)` signifie que le disque se trouvant au sommet du plot numéro `1` a été déplacé vers le plot numéro `2`. Les déplacements avec 3 disques schématisés ci-dessus sont représentés par :

```
[(1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3)]
```

1. À l'aide d'une interface en ligne, résoudre le problème des tours de Hanoï avec 4 disques. On pourra par exemple utiliser :

`https://www.toupty.com/jeutourshanoi.html`

Pour résoudre le problème des tours de Hanoï, on adopte une stratégie récursive. Soit `n` $\in \mathbb{N}$ le nombre de disques, soit `d` $\in$ `{1, 2, 3}` le numéro du plot sur lequel se trouvent les disques au début et `f` $\in$ `{1, 2, 3}` $\setminus$ `{d}` le numéro du plot sur lequel ils doivent se trouver à la fin. Dans l'exemple précédent, on avait `d = 1` et `f = 3`. On a alors deux cas en fonction de la valeur de `n` :

- Lorsque `n = 0`, il suffit de ne rien faire pour résoudre le problème des tours de Hanoï.
- Lorsque `n > 0`, on note `m` l'unique élément de `{1, 2, 3}` $\setminus$ `{d, f}`, puis :
  - À l'aide d'un appel récursif, on déplace les `(n - 1)` premiers disques du plot numéro `d` vers le plot numéro `m`.
  - On déplace le disque se trouvant sur le plot numéro `d` vers le plot numéro `f`.
  - À l'aide d'un appel récursif, on déplace les `(n - 1)` disques qui se trouvent sur le plot numéro `m` vers le plot numéro `f`.

2. Écrire une fonction récursive `hanoi` qui prend en entrée `n`, `d`, `f`, et renvoie la liste des déplacements à effectuer pour résoudre le problème des tours de Hanoï. Vous devez écrire la fonction `hanoi` directement sans fonction intermédiaire.
3. Déterminer la taille de la liste renvoyée par la fonction `hanoi`. Justifiez votre réponse.

Exercice 3. Suite de Fibonacci

La suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est définie récursivement : $\begin{cases} F_0 = 0, F_1 = 1 \\ F_{n+2} = F_{n+1} + F_n \quad \text{pour tout } n \geqslant 0. \end{cases}$

```
def fibo(n):
    """fibo(n: int) -> int"""
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```

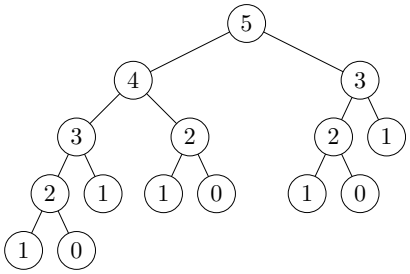


FIGURE 1

1. Trouver une valeur de n pour laquelle la fonction `fibo` met environ 1 seconde pour calculer F_n.

La valeur de n trouvée à la question 1 est relativement petite. Cela est dû au fait que Python effectue plusieurs fois les mêmes appels récursifs. Par exemple, l'arbre des appels récursifs pour calculer F_5 est présenté figure 1 ; il y a donc 2 appels à `fibo(3)`, 3 appels à `fibo(2)`, 5 appels à `fibo(1)` et 3 appels à `fibo(0)`.

2. Pour tout $n \in \mathbb{N}$, on note A_n le nombre d'appels à la fonction `fibo` lors de l'exécution de `fibo(n)`. Par exemple, $A_5 = 15$. Établir une formule de récurrence pour A_n.

Grâce à la formule trouvée question 2, il est possible de montrer que pour tout $n \in \mathbb{N}$:

$$A_n = -1 + c_1\phi_1^n + c_2\phi_2^n \qquad \text{avec} \qquad \begin{cases} c_1 = \dfrac{5+\sqrt{5}}{5} \\ c_2 = \dfrac{5-\sqrt{5}}{5} \end{cases} \qquad \text{et} \qquad \begin{cases} \phi_1 = \dfrac{1+\sqrt{5}}{2} \\ \phi_2 = \dfrac{1-\sqrt{5}}{2} \end{cases}$$

Comme $|\phi_2| < 1 < \phi_1$, on en déduit que lorsque n tend vers l'infini : $A_n = \mathcal{O}(\phi_1^n)$. On parle de ***complexité exponentielle*** en n.

3. Écrire une fonction récursive `fibo_bis` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie F_n. Votre fonction devra être de complexité linéaire en n. Si vous n'y arrivez pas, vous pouvez lire l'indication en bas de page [1]. Vérifier que la fonction `fibo_bis` permet de calculer F_{100} instantanément.

Remarque. Essayer de calculer F_{1000} avec la fonction `fibo_bis` déclenche une erreur car en Python, le nombre d'appels récursifs est limité. On parle de ***dépassement de la taille de la pile*** (stack overflow). Pour augmenter le nombre maximal d'appels récursifs, il faut utiliser la commande :

```
import sys
sys.setrecursionlimit(10000)
```

1. On pourra d'abord écrire une fonction itérative, puis la convertir en fonction récursive

Exercice 4. Sous-ensembles de $[\![0; n-1]\!]$

Il s'agit de l'exercice facultatif du TP 3 (à refaire à l'aide d'une fonction récursive).

Écrire une fonction récursive « `sous_liste(n: int, k: int) -> list[list[int]]` » qui prend en entrée deux entiers `(n, k)` $\in \mathbb{N}^2$, et renvoie une liste contenant toutes les sous-listes triées de `[0, 1, ..., n-1]` de taille `k`. Par exemple avec `n = 4`, on obtient (l'ordre des sous-listes n'a pas d'importance) :

Pour `k = 0` : `[[]]`,
Pour `k = 1` : `[[0], [1], [2], [3]]`,
Pour `k = 2` : `[[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]`,
Pour `k = 3` : `[[0,1,2], [0,1,3], [0,2,3], [1,2,3]]`,
Pour `k = 4` : `[[0,1,2,3]]`,
Pour `k = 5` : `[]`.

Vous devez écrire la fonction `sous_liste` directement sans utiliser de fonction intermédiaire.

Indications (essayez de faire l'exercice sans lire ce qui suit). Soit L_n la liste `[0, 1, ..., n-1]`. On a plusieurs cas en fonction des valeurs de `n` et `k` :

- Si `k = 0`, alors la seule sous-liste de L_n de taille `k` est la liste vide.
- Sinon, si `k > n`, alors il n'existe pas de sous-liste de L_n de taille `k`.
- Sinon, une sous-liste de L_n peut contenir l'élément `n-1` ou bien ne pas contenir l'élément `n-1`. On procède donc de la manière suivante :
  - On génère toutes les sous-listes triées de taille `k-1` de L_{n-1} à l'aide d'un appel récursif. Soit `S` l'une de ces sous-listes alors `S + [n-1]` est une sous-liste triée de L_n de taille `k`.
  - On génère toutes les sous-listes triées de taille `k` de L_{n-1} à l'aide d'un appel récursif. Soit `S` l'une de ces sous-listes alors `S` est une sous-liste triée de L_n de taille `k`.

  On pourra utiliser des boucles `for` dans les deux étapes ci-dessus.

Exercice 5. Tours de Hanoï - Interface graphique

À l'aide du module `Tkinter`, créer une interface graphique permettant de jouer au jeu des tours de Hanoï. On pourra utiliser l'aide en ligne de `Tkinter`, les fonctions décrites dans l'exercice 5 du TP 1 et s'inspirer du fichier annexe disponible sur la page du cours.

Exercices à rendre au plus tard le 14/01/2024 à 20h

Exercice 6.

Les trois fonctions demandées dans cet exercice peuvent être écrites directement sans définir de fonction intermédiaire. Si vous n'y parvenez pas, vous pouvez écrire une fonction itérative, puis la convertir en fonction récursive. La procédure de conversion a été illustrée dans le cours sur les fonction récursives.

1. Écrire une fonction récursive `somme_chiffres` qui calcule la somme des chiffres d'un nombre. Par exemple, la somme des chiffres de 45448 est $4+5+4+4+8=25$. Dans cette question, on s'interdit de convertir l'entier en une chaîne de caractères.
2. Écrire une fonction récursive « `est_croissante(L: list[int]) -> bool` » qui renvoie `True` si `L` est croissante et `False` sinon.
3. Écrire une fonction récursive « `est_palindrome(s: str) -> bool` » qui renvoie `True` si `s` est un palindrome et `False` sinon. On rappelle que `s` est un palindrome si et seulement si `s == s[::-1]`. On attend ici que vous parcouriez la chaîne de caractères ; tester directement l'égalité précédente ne sera pas accepté.

Exercice 7. Permutations des éléments d'une liste

Écrire une fonction récursive `permutation` qui prend en entrée une liste d'entiers `L`, et renvoie une liste de listes d'entiers contenant toutes les permutations de `L`. Par exemple, avec `L = [1,2,3]`, on obtient (l'ordre de sous-listes n'a pas d'importance) :

`[[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]`.

Vous devez écrire la fonction `permutation` directement sans utiliser de fonction intermédiaire.

Indications (essayez de résoudre l'exercice sans lire ce qui suit). Voici une stratégie possible :

- On parcourt tous les indices valides pour `L` à l'aide d'une boucle `for` (on pourrait aussi utiliser une fonction récursive, mais le code serait moins lisible). Pour chaque indice `i` :
  - Soit `L'` la liste `L` privée de `L[i]`.
  - Grâce à un appel récursif, on calcule `M'` la liste contenant toutes les permutations de `L'`.
  - Pour chaque liste `P'` appartenant à `M'`, on calcule `P` la liste obtenue en ajoutant `L[i]` au début de `P'`.
- On renvoie la liste de toutes les listes `P` obtenues dans les différents tours de boucle.

Exercice 8. Baguenaudier (exercice facultatif)

Cet exercice est issu du site France-ioi :

`http://www.france-ioi.org/algo/task.php?idChapter=671&idTask=1504`

Vous avez découvert un nouveau jeu très amusant. Celui-ci se joue sur un plateau de N cases alignées numérotées de 1 à N sur lesquelles on peut poser des jetons.

- À tout moment, chaque case contient au plus un jeton.
- Une case contenant un jeton est dite remplie, une case n'en contenant aucun est dite vide.
- Au début de la partie, toutes les cases sont remplies.

Les règles du jeu sont les suivantes :

- À tout moment on peut vider ou remplir la case 1.
- Si la case 1 est remplie, alors on peut remplir ou vider la case 2. (Règle valable pour $N \geqslant 2$)
- Pour tout $3 \leqslant K \leqslant N$, on peut remplir ou vider la case K lorsque les $K-2$ premières cases sont vides et que la case $K-1$ est remplie. (Règle valable pour $N \geqslant 3$)

Le but du jeu est de vider toutes les cases. Vous avez décidé d'écrire une fonction de signature « `remplir(N: int) -> list[int]` » permettant de résoudre le jeu.

Entrée. L'entrée est composée d'un unique entier : $N \geqslant 1$ le nombre de cases du plateau.

Sortie. La fonction `remplir` doit renvoyer une liste d'entiers compris entre 1 et N décrivant les différents coups. Chaque entier représente l'indice d'une case à remplir ou à vider.

Exemple. Pour $N = 4$, on obtient la liste `[2, 1, 4, 1, 2, 1, 3, 1, 2, 1]`. Voici une représentation correspondant à cette liste :