BAC NSI - Correction
Sujet 0 - 2021 - NSI

Objectifs

Le candidat doit choisir 3 exercices qu'il traitera sur les 5 exercices proposés.

- Exercice 1 : Notion de Pile et programmation Python.
- Exercice 2 : Programmation et récursivité.
- Exercice 3 : Arbres binaires et les arbres binaires de recherche.
- Exercice 4 : Bases de données relationnelles et le langage SQL.
- Exercice 5 : Réseaux en général et les protocoles RIP et OSPF en particulier.

Exercice 1 : Notion de Pile et programmation Python

On munit la structure de données Pile de quatre fonctions primitives définies dans le tableau ci-dessous. :

- creer_pile_vide : $\emptyset \longrightarrow$ Pile
  creer_pile_vide() : renvoie une pile vide
- est_vide : Pile $\longrightarrow$ Booléen
  est_vide(pile) : renvoie True si pile est vide, False sinon
- empiler : Pile, Élément $\longrightarrow$ Rien
  empiler(pile, element) : ajoute element au sommet de la pile
- depiler : Pile $\longrightarrow$ Élément
  depiler(pile) : renvoie l'élément au sommet de la pile en le retirant de la pile

Question 1

On suppose dans cette question que le contenu de la pile P est le suivant (les éléments étant empilés par le haut). Quel sera le contenu de la pile Q après exécution de la suite d'instructions suivante

```
1 Q = creer_pile_vide ()
2 while not est_vide(P):
3     empiler(Q, depiler(P))
```

On dépile la pile P et on empile la pile Q.

Question 2

1. **On appelle hauteur d'une pile le nombre d'éléments qu'elle contient. La fonction hauteur_pile prend en paramètre une pile P et renvoie sa hauteur. Après appel de cette fonction, la pile P doit avoir retrouvé son état d'origine. Recopier et compléter sur votre copie le programme Python suivant implémentant la fonction hauteur_pile en remplaçant les??? par les bonnes instructions.**

```
def hauteur_pile(P):
    Q = creer_pile_vide ()
    n = 0
    while  not(est_vide(P)):
        n=n+1 # c'est l'incrémentation du compteur
        x = depiler(P)
        empiler(Q,x)
    while not(est_vide(Q)):
        x=depiler(Q) # On récupère l'élément au sommet de la pile Q
        empiler(P, x)
    return n
```

2. **Créer une fonction max_pile ayant pour paramètres une pile P et un entier i. Cette fonction renvoie la position j de l'élément maximum parmi les i derniers éléments empilés de la pile P. Après appel de cette fonction, la pile P devra avoir retrouvé son état d'origine. La position du sommet de la pile est 1.**

```
def max_pile(P,i):
    '''In : P pile et i entier <= hauteur_Pile
      Out : renvoie la position j de l'élément maximum parmi les i
      derniers éléments empilés de la pile P'''
    Q=creer_pile_vide()
    n=1
    x=depiler(P)
    empiler(Q,x)
    maximum=x
    rang=1
    while  not(est_vide(P)) and n<i:
        n=n+1
        x = depiler(P)
        empiler(Q,x)
        if x>maximum:
            maximum=x
            rang=n
    while not(est_vide(Q)):
            x=depiler(Q)
            empiler(P, x)
    return rang
```

Question 3

Créer une fonction retourner ayant pour paramètres une pile P et un entier j. Cette fonction inverse l'ordre des j derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.

```
def retourner(P,j):
    '''In : P pile et i entier <= hauteur_Pile
      Out : None.
      Cette fonction inverse l'ordre des j derniers éléments empilés
      et ne renvoie rien'''
    Q = creer_pile_vide ()
    K = creer_pile_vide ()

    # on dépile les j derniers éléments de P que l'on empile dans Q
    # (ordre inversé)
    n=0
    while  n<j and not(est_vide(P)):
        empiler(Q,depiler(P)) # comme dans la question 1
        n=n+1

    # on dépile les j derniers éléments de Q que l'on empile dans K
    #  (ordre inversé) donc l'ordre redevient celui initial
    n=0
    while  n<j and not(est_vide(Q)):
        empiler(K,depiler(Q))
        n=n+1

    # on dépile les j derniers éléments de K que l'on empile dans P
    # (ordre inversé)
    n=0
    while  n<j and not(est_vide(K)):
        empiler(P,depiler(K))
        n=n+1
```

Question 4

L'objectif de cette question est de trier une pile de crêpes. On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile). On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus. Le principe est le suivant :

- *On recherche la plus grande crêpe.*
- *On retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile.*
- *On retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas.*
- *La plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.*

```
def tri_crepe(Pile):
    '''In : Pile
       out : None. Cette fonction va trier la pile'''
    n=hauteur_pile(Pile)
    while n!=0:
        k=max_pile(Pile,n)
        retourner(Pile,k)
        retourner(Pile,n)
        n=n-1
```

Exercice 2 : Programmation et récursivité

Question 1

On considère tous les chemins allant de la case (0, 0) à la case (2, 3) du tableau T donné en exemple.

Exemple avec $n = 3$ lignes et $p = 4$ colonnes.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)

Remarque historique

Compter ce nombre de chemins avec ce déplacement imposé, dans un tableau avec n lignes et p colonnes, est un problème historique. On nomme cela les chemins de Pascal, du nom de l'illustre mathématicien français Blaise Pascal (1623-1662).
En fait un on peut facilement montrer que

- Le nombre de cases pour passer de la case $(0,0)$ à la case $(n-1, p-1)$ est $n+p-1$ soit ici $3+4-1=6$.
- Le nombre de déplacements est $n+p-2$ soit $3+4-2=5$ ici,
  avec $n-1=2$ déplacements vers le bas et $p-1=3$ déplacements vers la droite.
- Par ailleurs le nombre de chemins pour passer de la case $(0,0)$ à la case $(n-1, p-1)$ est :

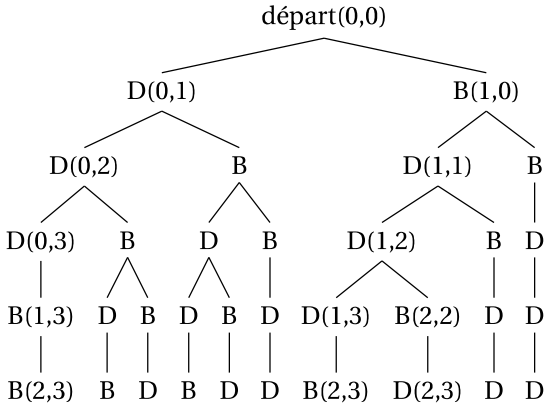
$$\binom{n+p-2}{p-1} = \binom{n+p-2}{n-1}$$

Cela revient à choisir parmi les $n+p-2=5$ déplacements possibles, les $n-1=2$ déplacements vers le bas ou les $p-1=3$ déplacements vers la droite.

$$\binom{n+p-2}{p-1} = \binom{5}{3} = \frac{5!}{3!2!} = 10 \text{ ou } \binom{n+p-2}{n-1} = \binom{5}{2} = \frac{5!}{2!3!} = 10$$

.

1. **Un tel chemin comprend nécessairement 3 déplacements vers la droite. Combien de déplacements vers le bas comprend-il?**
   Il comprend 2 chemins vers le bas.
2. **La longueur d'un chemin est égal au nombre de cases de ce chemin. Justifier que tous les chemins allant de (0, 0) à (2, 3) ont une longueur égale à 6.**
   - Pour chaque déplacement Droite ou Bas, on arrive sur une nouvelle case. On ne peut faire que 3 déplacements vers la droite et 2 vers le bas. Ce qui fait 5 cases accessibles plus la case de départ donc un chemin est de longueur 6.
   - On peut aussi faire un arbre de tous les chemins possibles (il y en a 10) et on s'aperçoit que leur longueur est 6 car chaque noeud de l'arbre correspond à une case.

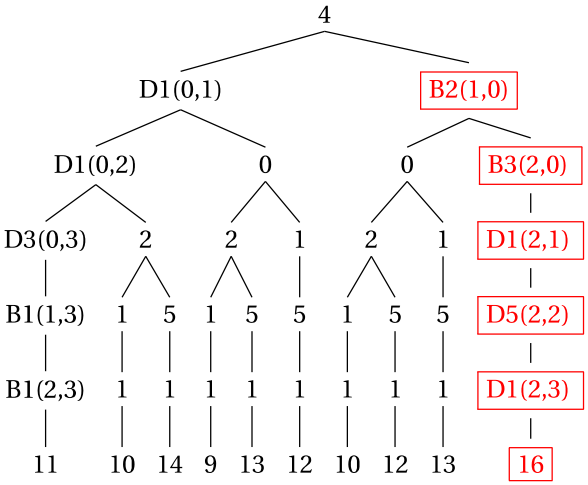


Exemple avec $n = 3$ lignes et $p = 4$ colonnes.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)

Question 2

En listant tous les chemins possibles allant de (0, 0) à (2, 3) du tableau T, déterminer un chemin qui permet d'obtenir la somme maximale et la valeur de cette somme.



(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	**(2,1)**	**(2,2)**	**(2,3)**

Le chemin qui donne la somme maximale 16 est donc le chemin :

$$\boxed{(0,0) \longrightarrow (1,0) \longrightarrow (2,0) \longrightarrow (2,1) \longrightarrow (2,2) \longrightarrow (2,3)}$$

Question 3

On veut créer le tableau T' où chaque élément T'[i][j] est la somme maximale pour tous les chemins possibles allant de (0, 0) à (i, j).

1. **Compléter et recopier sur votre copie le tableau T' donné ci-dessous associé au tableau T.**

$T =$

4	1	1	3
2	0	2	1
3	1	5	1

$T' =$

4	5	6	**9**
6	**6**	8	10
9	10	**15**	16

— Pour $T'[0][3] = 9$.
Il n'y a qu'un seul chemin possible, il est de somme $9 = 4+1+1+3$.

$$\boxed{(0,0) = 4 \longrightarrow (0,1) = 1 \longrightarrow (0,2) = 1 \longrightarrow (0,3) = 3} \quad \text{donc} \quad S = 4+1+1+3 = 9$$

— Pour $T'[1][1] = 6$.
Il n'y a que 2 chemins possibles de (0,0) à (1,1), de somme 6 et 5.

$$\boxed{(0,0) = 4 \longrightarrow (1,0) = 2 \longrightarrow (1,1) = 0} \quad \text{donc} \quad S = 4+2+0 = 6$$

$$(0,0) = 4 \longrightarrow (0,1) = 1 \longrightarrow (1,1) = 0 \quad \text{donc} \quad S = 4+1+0 = 5$$

— Pour $T'[2][2] = 15$.

Il y a 6 chemins possibles de (0,0) à (2,2). *Remarque* $\binom{4}{2} = \dfrac{4!}{2!2!} = 6$.

— $\boxed{(0,0) = 4 \longrightarrow (1,0) = 2 \longrightarrow (2,0) = 3 \longrightarrow (2,1) = 1 \longrightarrow (2,2) = 5}$ donc $S = 4+2+3+1+5 = 15$;

— $(0,0) = 4 \longrightarrow (0,1) = 1 \longrightarrow (0,2) = 1 \longrightarrow (1,2) = 2 \longrightarrow (2,2) = 5$ donc $S = 4+1+1+2+5 = 13$;

— Les 4 autres sont de sommes : $4+2+0+1+5 = 12$; $4+2+0+2+5 = 12$; $4+1+0+2+5 = 12$ et $4+1+0+1+5 = 11$.

2. **Justifier que si j est différent de 0, alors : $T'[0][j] = T[0][j] + T'[0][j-1]$.**

(0,0)	⋯	$(0, j-1)$	$(0, j)$	⋯	⋯
(1,0)	⋯	⋯	⋯	⋯	⋯
(2,0)	⋯	⋯	⋯	⋯	⋯

Le seul chemin possible pour aller à la case $(0, j)$ passe par la case $(0, j-1)$.

De ce fait $T'[0][j]$ qui est la somme maximale pour tous les chemins possibles allant de $(0,0)$ à $(0, j)$ est la somme de :

- la valeur de la case $(0, j)$ soit $T[0][j]$;

- la somme maximale pour tous les chemins possibles allant de $(0, 0)$ à $(0, j-1)$.

Soit

$$\boxed{T'[0][j] = T[0][j] + T'[0][j-1]}$$

Question 4

Justifier que si i et j sont différents de 0, alors : $T'[i][j] = T[i][j] + max(T'[i-1][j], T'[i][j-1])$.

⋯	⋯	⋯	⋯	⋯	⋯
$(i-1, 0)$	⋯	$(i-1, j-1)$	$(i-1, j)$	⋯	⋯
$(i, 0)$	⋯	$(i, j-1)$	**(i, j)**	⋯	⋯

— Le seul chemin possible pour aller à la case (i, j) passe par la case $(i, j-1)$ ou par la case $(i-1, j)$.

— De ce fait $T'[0][j]$ qui est la somme maximale pour tous les chemins possibles allant de $(0,0)$ à (i, j) est la somme de :

— la valeur de la case (i, j) soit $T[0][j]$;

— le maximum entre :
- la somme maximale pour tous les chemins possibles allant de $(0, 0)$ à $(i, j-1)$ soit $T'[i][j-1]$;
- la somme maximale pour tous les chemins possibles allant de $(0, 0)$ à $(i-1, j)$ soit $T'[i-1][j]$.

Soit

$$\boxed{T'[i][j] = T[i][j] + max(T'[i-1][j], T'[i][j-1])}$$

Question 5

On veut créer la fonction récursive somme_max ayant pour paramètres un tableau T, un entier i et un entier j. Cette fonction renvoie la somme maximale pour tous les chemins possibles allant de la case (0, 0) à la case (i, j).

1. **Quel est le cas de base, à savoir le cas qui est traité directement sans faire appel à la fonction somme_max? Que renvoie-t-on dans ce cas?**
   Le cas de base est le cas où le tableau est vide dans ce cas on renvoie zéro.
2. **À l'aide de la question précédente, écrire en Python la fonction récursive somme_max .**

```
from numpy import *

# On utilise la formule de la question 4
# T'[i][j] = T[i][j] + max(T'[ i-1 ][ j ], T'[ i ][ j-1 ])

def somme_max(Tableau):
    if size(Tableau)==0:
        return 0
    else:
        A=somme_max(Tableau[0:-1,:]) # T' [ i-1 ] [ j ]
        # on prend tout sauf la dernière ligne

        B=somme_max(Tableau[:,0:-1]) # T' [ i ] [ j-1 ]
        # on prend tout sauf la dernière colonne

        return Tableau[-1][-1] + max(A,B)
        # Tableau[-1][-1] est l'élément de la dernière ligne
        # et dernière colonne
        # C'est le T [ i ] [ j ] de notre formule
```

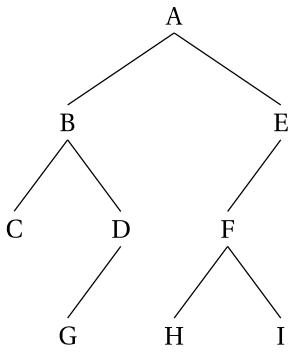
3. **Quel appel de fonction doit-on faire pour résoudre le problème initial?**
   Il suffit d'appeler : somme_max(T).

Exercice 3 : Arbres binaires et les arbres binaires de recherche

Dans cet exercice, on utilisera la convention suivante : la hauteur d'un arbre binaire ne comportant qu'un nœud est 1.

Question 1

Déterminer la taille et la hauteur de l'arbre binaire suivant :



Arbre binaire

1. On appelle **taille d'un arbre** le nombre de noeuds présents dans cet arbre.
2. Dans un **arbre binaire**, un noeud possède au plus 2 fils.
3. On appelle **profondeur d'un nœud** ou d'une feuille dans un arbre binaire le nombre de nœuds du chemin qui va de la racine à ce nœud. La racine d'un arbre est à une profondeur 1 (ici, mais cela dépend de la convention).
4. On appelle **hauteur d'un arbre** la profondeur maximale des nœuds de l'arbre.

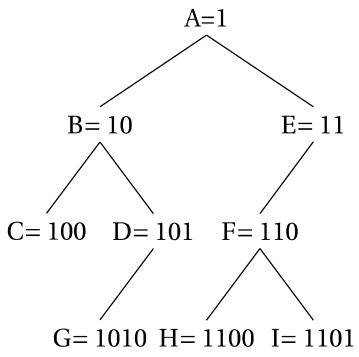
— Par définition, la taille d'un arbre est le nombre de noeud qu'il contient. Ici il y en a 9 donc la taille est 9.

— Par définition la hauteur est le nombre de noeuds du chemin le plus long dans l'arbre, ici les chemins les plus long sont ABDG, AEFH et AEFI. Comme la hauteur d'un arbre ne contenant qu'un noeud est 1, la hauteur de cet arbre est 4.

Question 2

On décide de numéroter en binaire les nœuds d'un arbre binaire de la façon suivante :
— la racine correspond à 1 ;
— la numérotation pour un fils gauche s'obtient en ajoutant le chiffre 0 à droite au numéro de son père ;
— la numérotation pour un fils droit s'obtient en ajoutant le chiffre 1 à droite au numéro de son père.

1. **Dans l'exemple précédent, quel est le numéro en binaire associé au nœud G ?**



On a donc G :1010.

2. **Quel est le nœud dont le numéro en binaire vaut 13 en décimal ?**
   $13_{10} = 1101_2$ or I :1101 donc cela correspond au noeud I.
3. **En notant h la hauteur de l'arbre, sur combien de bits seront numérotés les nœuds les plus en bas ?**
   A chaque niveau de l'arbre on rajoute 1 bit donc les numéros des noeuds les plus bas (les feuilles) contiennent h bits.
4. **Justifier que pour tout arbre de hauteur h et de taille $n \geq 2$, on a : $h \leq n \leq 2^h - 1$.**
   — Soit un arbre de hauteur h. Il contient un maximum de noeuds si il est complet. Or un arbre binaire complet de hauteur h contient $1 + 2 + 2^2 + ... + 2^{h-1}$ noeuds. C'est la somme d'une suite géométrique de raison 2 donc :

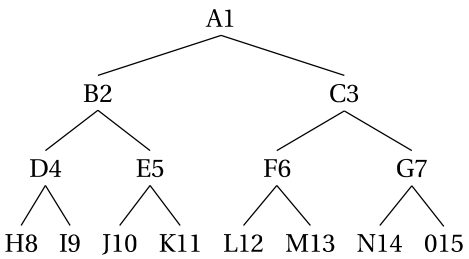
   $$1 + 2 + 2^2 + ... + 2^{h-1} = \frac{2^h - 1}{2 - 1} = 2^h - 1$$

   On a donc $n \leq 2^h - 1$.
   — Soit un arbre de hauteur h. La hauteur maximale que l'on peut obtenir avec un minimum de noeuds est le cas où chaque père n'a qu'un seul fils . Dans ce cas la hauteur est égale à n. Donc $h \leq n$.
   — Donc $h \leq n \leq 2^h - 1$

Question 3

Un arbre binaire est dit complet si tous les niveaux de l'arbre sont remplis. On décide de représenter un arbre binaire complet par un tableau de taille $n + 1$, où n est la taille de l'arbre, de la façon suivante :
— La racine a pour indice 1 ;
— Le fils gauche du nœud d'indice i a pour indice $2 \times i$;
— Le fils droit du nœud d'indice i a pour indice $2 \times i + 1$;
— On place la taille n de l'arbre dans la case d'indice 0.

1. **Déterminer le tableau qui représente l'arbre binaire complet de l'exemple précédent.**



On a donc le tableau

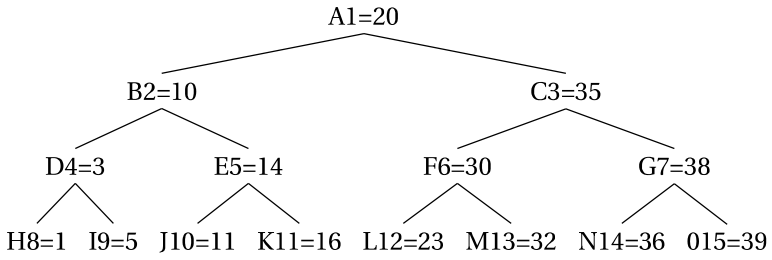
$$\boxed{[15, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O]}$$

2. **On considère le père du nœud d'indice i avec $i \geq 2$. Quel est son indice dans le tableau ?**
   Le père d'un fils d'indice i a pour indice $i/2$ si i est pair et $(i-1)/2$ sinon.

   Sous python on peut dans ce cas utiliser la fonction // .
   $a//b$ donne le quotient de la division euclidienne de a par b .

Question 4

On se place dans le cas particulier d'un arbre binaire de recherche complet où les nœuds contiennent des entiers et pour lequel la valeur de chaque noeud est supérieure à celles des noeuds de son fils gauche, et inférieure à celles des noeuds de son fils droit.
On a par exemple un arbre de ce type :



$A = [15, 20, 10, 35, 3, 14, 30, 38, 1, 5, 11, 16, 23, 32, 36, 39]$

Écrire une fonction recherche ayant pour paramètres un arbre *arbre* et un élément *element*.
Cette fonction renvoie *True* si *element* est dans l'arbre et *False* sinon. L'arbre sera représenté par un tableau comme dans la question précédente.

```
def recherche(arbre,element):
    '''In : arbre et element entier
      Out : true si element est dans la liste'''
    taille = arbre[0]
    i=1
    while i<=taille:
        if element==arbre[i]:
            return True
        elif element>arbre[i]:
            i=2*i+1
        else:
            i=2*i
    return False
```