

0.1 Récursivité, Python

On dispose d'une fonction P suivante (4 points) :

```
fonction  $P(X)$  renvoie un entier :  
si  $X = 1$  alors renvoyer  $X$   
si pair( $x$ ) alors renvoyer( $P(X/2)$ )  
sinon renvoyer( $P(3X+1)$ )
```

- 1- Transformer la fonction $P(X)$ en une version itérative.
- 2- Ecrire la version Python de la fonction $P(X)$ ainsi que celle de votre solution.

0.2 Algo et Python

On dispose d'un ensemble de N bâtons dont la taille de chacun est un entier positif. (5 points)

Une série d'opération de *coupure* est appliquée à ces bâtons réduisant, à chaque application, leur taille par la valeur de la taille du plus petit bâton de l'ensemble actuel et ce jusqu'à obtenir l'ensemble vide.

Exemple : soit $N = 6$ et les bâtons $\langle 5, 4, 4, 2, 2, 8 \rangle$. La première application de *coupure* trouve la plus petite taille disponible ($= 2$) et réduit les bâtons en conséquence donnant $\langle (5 - 2), (4 - 2), (4 - 2), (2 - 2), (2 - 2), (8 - 2) \rangle = \langle 3, 2, 2, 0, 0, 6 \rangle$.

On élimine les tailles nulles et on conserve $\langle 3, 2, 2, 6 \rangle$ pour l'application suivante de *coupure*.

Cette deuxième réduction (où la taille minimale est encore 2) donnera $\langle 1, 4 \rangle$, puis, le minimum étant 1, on obtient $\langle 3 \rangle$ et enfin, pour le minimum 3, on obtient $\langle \rangle$. Les opérations de coupure ont continué jusqu'à l'ensemble vide.

Ecrire le code Python de la fonction *coupure* et la partie principale qui lira N ainsi que l'ensemble de bâtons.

0.3 Complexité, Algo et Python

On veut montrer que pour l'entier N grand, la probabilité pour que deux entiers aléatoires $x, y \leq N, x \neq y$ soient premiers entre eux est proche de $\frac{6}{\pi^2} = 0.608$ (**4 points**).

Ecrire un algorithme (de la fonction) $proba_prime(N)$ qui procède comme suit :

proba_prime(N) :

compteur, x, y : des entiers initialisés à 0

Itérer N fois :

x, y = deux tirages aléatoires d'entier ($x, y \in [1..N]$)

Vérifier $x \neq y$. Sinon, recommencer cette étape d'itération

Si $pgcd(x, y) = 1$ alors incrémenter compteur

Renvoyer (*compteur* / N)

1- Ecrire la fonction Python **pgcd(a,b)** selon l'algorithme d'*Euclide* (utiliser l'opérateur *modulo*).

2- Ecrire la fonction Python qui implante **proba_prime(N)** en fournissant la partie principale qui permet de l'utiliser.

3- Supposons que la fonction $pgcd(a,b)$ est de complexité $O(\log_2(\min(a,b)))$ sachant que $\log_2(X)$ est le nombre de bits de la représentation binaire de X .

Calculer la complexité de l'algorithme **proba_prime(N)**. Préciser toutes vos hypothèses dans ce calcul.

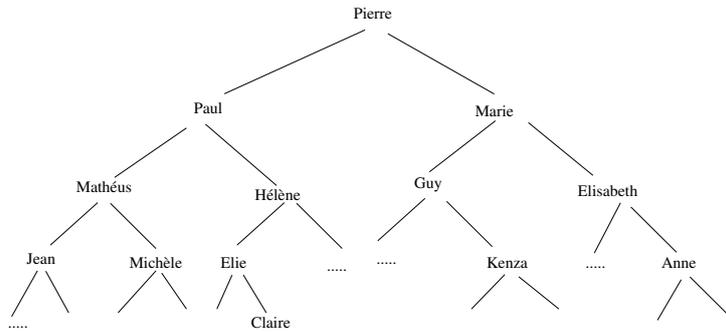
0.4 Graphe / arbre

On dispose d'un arbre généalogique binaire où chaque noeud est relié aux deux parents biologiques d'un individu. Par convention, le père est en fils-gauche et la mère en fils-droit : Paul et le père et Pierre et Marie sa mère. (**7 points**)

☞ On suppose que les enfants sont uniques et un prénom ne figure qu'une seule fois dans l'arbre.

On décide d'implanter cet arbre sous la forme suivant :

- Une liste principale **Individus** contient les prénoms. Retenons l'indice de chaque personne dans cette liste.
- Une liste de couples **Parents** qui contient les parents.



$Parents[i] = (P,M)$ où P est l'indice du père de $Individus[i]$ dans la liste $Individus$. Idem pour M .

Par exemple, si :

$Individus = ["E", "P", "M"]$ où "E" est enfant de "P" et "M",

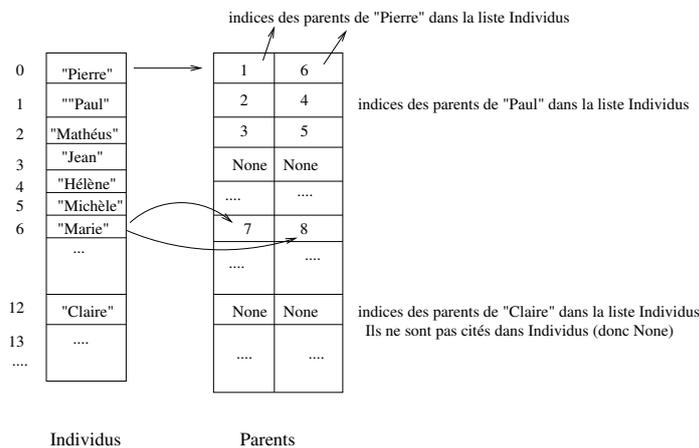
alors

$Parents = [(1, 2), (None, None), (None, None)]$ où $(1,2)$ fournit les indices de "P" et "M" dans $Individus$.

Les *None*s nous indiquent qu'à leur tours, "P" et "M" n'ont pas de parent présents dans $Individus$.

Q1- Proposer une structure en Python pour représenter cette arborescence. Elle devrait correspondre aux deux tables suivantes.

Remplissez les deux tables (listes) sous Python.



On remarque que les ajouts, modifications et suppressions sont très simples dans cette structure. Par exemple, on peut insérer des éléments en premier emplacement libre : l'indice du premier emplacement libre = la taille de la liste $Individus$ (ici = 13).

Q2- Proposer la fonction Python `index_des_Individus(Individus)` qui affiche chaque individu (prénom) ainsi que son indice dans la liste $Individus$. On devrait obtenir par exemple :

```
"Pierre" : 0
"Paul" : 1 ...
```

Q3- Proposer la fonction Python `parents(enfant, Individus, Parents)` qui permet de trouver les (noms des) deux parents (père et mère) d'un individu *enfant* (*enfant* est le nom de l'enfant).

Q4- Proposer la fonction Python `enfant(pere, mere, Individus, Parents)` qui permet de trouver le (nom de) l'enfant des noms des deux parents (père et mère $\neq None$) d'un individu.

Q5- Proposer une solution pour pouvoir représenter plusieurs enfants pour un couple.