MPSI 2/2014 — GÉNÉALOGIE — 1-phô

♡1 Je vous donne l'arbre généalogique de la famille *(codé comme l'arborescence des dossiers/répertoires d'un disque dur)*. Complétez *(éventuellement sur ce document)* ce qui manque : •1 pt.•

nom	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
indice montant	8	6	5	21	19	25	0	1	18	14	2	3	27	29	26		10
indice descendant	9	7	12	22	20	32		24	23		13	4	28	30	31	16	11

♡2 Qui est l'ancètre commun à tout le monde ? •1 pt.•
Combien a-t-il de descendants directs et indirects ? •1 pt.• Combien a-t-il d'enfants ? •1 pt.•

♡3 Qui sont les individus sans descendants ? •1 pt.•

♡4 Combien `10` a-t-il d'enfants, et combien de descendants ? •1 pt.•

♡5 Donnez le père, le grand-père, et les aïeux successifs de `3`. •1 pt.•

♡6 Si `8` a un nouvel enfant, pouvez vous indiquer le script qui va modifier les deux listes "indices montants" et "indices descendants". •1 pt.•

MPSI 2/2014 — TUYAUX — 1-phô

Un lemme mathématique pour commencer : on donne quatre réels r, R, b et B vérifiant $r \leqslant R$ et $b \leqslant B$. Montrez : $|r-b|+|R-B| \leqslant |r-B|+|R-b|$. •2 pt.•

```
-----+----+-----------+--------+---------------- ou -----+----+-+------------------+----------------
.....r....b...........R........B................ ou .....r....R.b..................B...............
```

Vous disposez d'un stock de `N` tuyaux rouges de tailles diverses stockées dans une liste nommée `R`, et de `N` tuyaux bleus de longueurs diverses aussi stockées dans une liste nommée `B`. Exemple : `R=[5, 12, 8, 16]` et `B=[6, 8, 13, 20]`.
Vous devez les vendre par lots de deux : un rouge un bleu. Mais dans chaque lot, les deux tuyaux doivent être de même longueur *(il peut s'agir de tuyaux eau chaude/eau froide)*. Il va donc faloir couper soit le bleu *(exemple : `R[1]` couplé avec `B[3]`)*, soit le rouge *(exemple : `R[3]` couplé avec `B[2]`)*, soit aucun *(exemple : `R[2]` couplé avec `B[1]`)*.
Votre objectif : avoir le moins possible de pertes, c'est à dire trouver σ pour minimiser $\sum_{k=0}^{N-1} |R[K)-B[\sigma(k)]|$.

♠1 Ecrivez la procédure `pertes` qui pour deux listes `R` et `B` et une permutation `sigma` *(donnée aussi sous forme de liste)* calcule la somme indiquée ci dessus. •2 pt.•
Dans la mesure du possible, votre procédure devra retourner le booléen `False` si les trois listes n'ont pas la même longueur.

♠2 S'il faut tester toutes les permutations possibles, si le calcul de $\sum_{k=0}^{N-1} |R[K]-B[\sigma(k)]|$ prend 10^{-10} seconde, mais que N vaut 20, combien de temps vous faudrait il pour étudier tous les cas ? •2 pt.•

2^{20}	$20!$	20^{20}	20^{10}	
10^{6}	10^{18}	10^{26}	10^{13}	

♠3 Prouvez que la perte minimale pour notre exemple est de 6. •3 pt.•

♠4 Prouvez que pour minimiser la perte, il faut placer apparier le tuyau rouge le plus court avec le tuyau bleu le plus court. •3 pt.•

♠5 Déduisez que la perte est minimale si les deux listes sont triées dans le même ordre. •2 pt.•

♠6 Ecrivez une procédure qui donne la liste des couples `[R[k], B[sigma(k)]]` pour `k` de `0` à

N-1 en supposant que vous disposez de la méthode `sort` qui trie une liste *(syntaxe : `tamairen.sort()` si la liste s'appelle `tamairen`)*[1]. •3 pt.•

♠7 Ecrivez une procédure *(même peu efficace)* qui donne cette liste des couples si vous n'avez pas la métode `sort`. •3 pt.•

MPSI 2/2014

SCRIPT

1-phô

♣1 Que va faire ce script : •5 pt.•

```
from random import randrange
def leurdeschamps(L) :
....if len(L) == 1 :
........return(L[0])
....else :
........demi = len(L)//2
........a = leurdeschamps(L[0 :demi])
........b = leurdeschamps(L[demi, len(L)])
........if a > b :
............return(a)
........else :
............return(b)
LL = [randrange(1000) for k in range(10000)]
print(leurdeschamps(LL))
```

Votre note sur cette question dépendra de votre rédaction, explication, justification.

MPSI 2/2014

BASES DE BONNETS

1-phô

On envisage une base de données pour la gestion des Prépas du lycée faite de deux tables : `eleves` et `professeurs`. Les champs de la table `eleves` sont les suivants :

champ	nom	prenom	date_naissance	classe	LV1	LV2	option	regime
type	char(20)	char(20)	date	char(5)	char(8)	char(8)	char(8)	char(12)
exemple	HARTY	Kenty	data error	MPSI2	anglais	francais	Info	externe

Ceux de la table `professeurs` sont les suivants :

champ	nom	prenom	MPSI1	MPSI2	PCSI	MP	MP*	PC	PC*	matieres	date_naissance
type	Char(20)	char(20)	int(1)	int(1)	int(1)	int(1)	int(1)	int(1)	int(1)	char(30)	date
exemple	Papanico	Bob	1	1	1	1	1	0	0	SII	

On vous demande de saisir diverses requêtes, à vous de les formuler en langage SQL :

nombre d'élèves de sup faisant anglais langue vivante 1
nom des élèves de PCSI externes
nom, prénom et classe des élèves de Spé ayant moins de 18 ans
liste des élèves prénommés Alexandre ou même Alex, Alexis..., triés par classe
liste des professeurs de la MP*
liste des professeurs de la filière MP (quatre classes) triés par âge
liste nominale des professeurs de mathématiques
liste des professeurs de l'élève Lebanc
vérifier qu'aucun élève de PCSI n'a pris option informatique
envoyer tout de suite en MP* les élèves prénommé(e)s Lucille

Rappel des mots du langage SQL : `SELECT, ORDER BY, WHERE, FROM, COUNT, MAX, MIN, AVG, UPDATE, DELETE.`

[1] oui, "ta mère en short"

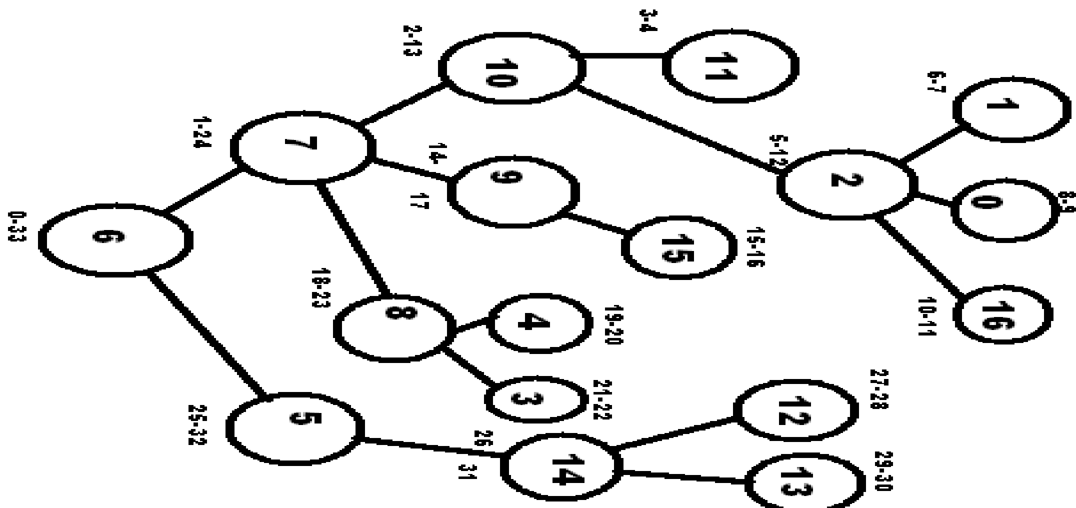
2014-15

Un arbre généalogique, ou l'arborescence d'un disque dur. 1-phô

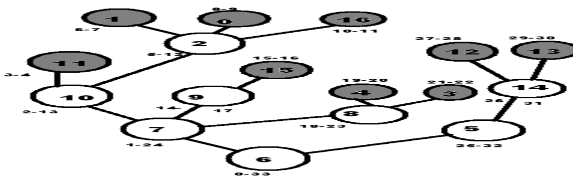
L'ancêtre est évidemment celui qui a l'indice ascendant égal à 0. C'est 6.
Son indice descendant est le double du nombre de personnes dans la famille puisque chaque peronne a deux indices au total. On complète donc : 33 comme indice descendant *(et on ne descend pas plus bas)*.

Il manque encore dans le tableau deux indices : 15 et 17
On a deux cases où les mettre.
Comme l'indice ascendant est plus petit que l'indice descendant, la seule possibilité est

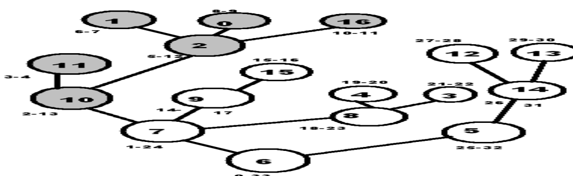
nom	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
indice montant	8	6	5	21	19	25	0	1	18	14	2	3	27	29	26	15	10
indice descendant	9	7	12	22	20	32	33	24	23	17	13	4	28	30	31	16	11



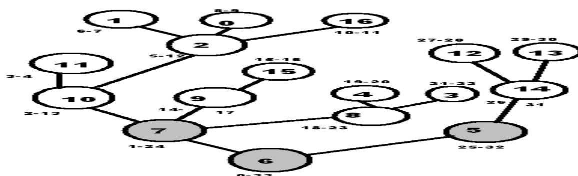
Les individus n'ayant pas de descendants sont ceux pour lesquels la différence d'indices est de 1. Il s'agit donc de 0, 1, 3, 4, 11, 12, 13, 15, 16.



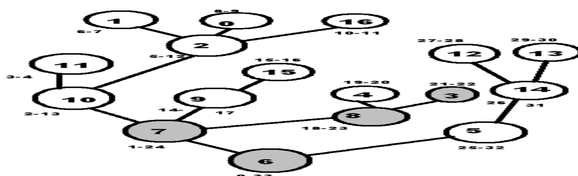
L'individu 10 a des enfants puisque la différence d'indices est plus grande que 1. On la calcule : $13-2=11$. Il a cinq descendants.
On les détecte à leurs indices compris entre 2 et 13 : 0, 1, 2, 11 et 16.
Parmi eux, seul 2 n'est pas en fin de branche, et a des enfants : 0, 1 et 16.
Par élimination, 11 est descendant direct de 10. On résume : 10 a deux enfants : 2 et 11. Et il a trois petits enfants : 0, 1 et 16 tous descendants de 2.



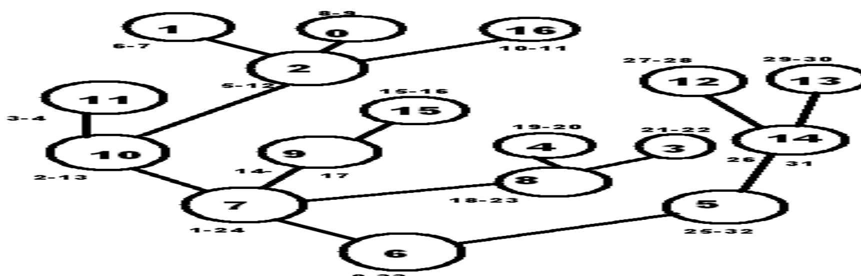
On cherche les descendants directs de la racine 6.
Déjà, il y a 7 *(indice montant 1)*. Son indice descendant est 24. Ceux d'indice entre 2 et 23 sont dans la brache de 7.
On cherche qui a pour indice montant 25. C'est le deuxième enfant de 6. Et c'est 5.
Son indice descendant est 32. On est donc en fin de liste. Il n'y a plus personne au delà.
Les deux enfants du grand aïeul 6 sont 7 et 5.



On retrouve la généalogie de 3. Son indice descendant est 22. Or, un individu a pour indice descendant 23. C'est donc son père *(et il était le cadet de ce père nommé 8)*.
On cherche si quelqu'un a pour indice descendant 24. C'est 7, qui est donc son grand père.
On continue : 25 n'est indice descendant de personne, mais au contraire indice montant de 5. C'est donc que 5 est le frère de 7.
On est donc au même rang. Tous ceux dont l'indice est entre 25 et 32 *(indice descendant de 5)* ne sont donc que des cousins lointaints de 3.



On pouvait aussi reconstituer l'arbre par ses indices et tout lire visuellement.



L'élément 8 a un nouvel enfant. Celui ci aura donc pour indice montant 23 et pour indice descendant 24.
8 garde son indice montant, et son indice descendant devient 25.
Il en va de même de tous les indices qui suivaient, ils sont tous augmentés de 2.
Par exemple, 14 aura pour nouveaux indices 28 et 33.
Par exemple, 7 garde son indice montant 1 et adopte pour indice descendant 26.
Si l'on considère que les indices montants et descendants sont codés dans deux listes de noms naturels :

```
for individu in arbre :
....if montant[individu] > 22 :
........montant[individu] += 2
....if descendant[individu] > 22 :
........descendant[individu] += 2
```

Et on n'oublie pas `montant[17] , descendant[17] = 23, 24` pour le petit nouveau.

Des tuyaux. — 2014-15 — 1-phô

On commence par le lemme mathématique qui servira sans doutes plus loin.

On a donc quatre réels $r \leqslant R$ et $b \leqslant B$. On mesure des distances.
On peut utiliser l'inégalité triangulaire, mais il y a quand même deux termes dans chaque membre, alors que l'inégalité triangulaire crée des termes en plus dans un membre.
On va étudier les différents cas de figure sur l'ordre de ces quatre nombres. Fort heureusement il n'y a pas 24 cas puisque certains ordres sont imposés.

	$d = \|r-b\| + \|R-B\|$	$\delta = \|r-B\| + \|R-b\|$	différence $\delta - d$
$r \leqslant R \leqslant b \leqslant B$	$b-r+B-R$	$B-r+b-R$	0
$r \leqslant b \leqslant R \leqslant B$	$b-r+B-R$	$B-r+R-b$	$2.(R-b) \geqslant 0$
$r \leqslant b \leqslant B \leqslant R$	$b-r+R-B$	$B-r+R-b$	$2.(R-b) \geqslant 0$
$b \leqslant B \leqslant r \leqslant R$	$r-b+R-B$	$r-B+R-b$	0
$b \leqslant r \leqslant B \leqslant R$	$r-b+R-B$	$B-r+R-b$	0
$b \leqslant r \leqslant R \leqslant B$	$r-b+B-R$	$B-r+R-b$	$2.(R-r) \geqslant 0$

On constate que dans tous les cas, la différence est positive.
On a bien $|r-b| + |R-B| \leqslant |r-B| + |R-b|$.
Si il s'agit de mesurer des distances, mieux vaut regrouper les deux petits entre eux et les deux grands entre eux.

Si les listes des longueurs et de la permutation ont la même longueur, on a juste à cumuler une somme :

```
def pertes(R, B, sigma) :
....S = 0
....for k in range(len(L)) :
........S = S+abs(R[k]-B[sigma[k]])
....return(S)
```

La fonction `abs` est d'accès direct. Si vous aviez des doutes, il fallait l'importer du module `math`.
On sollicite un indice $\sigma(k)$ en allant chercher un terme d'indice k dans une liste `sigma`, d'où `sigma[k]` avec des crochets et non `sigma(k)` qui correspondrait à l'appel d'une fonction ou procédure.
Enfin, on rappelle que le résultat d'une procédure *(introduite par def)* est retourné par `return`. On en fait ensuite ce qu'on veut. Ce n'est pas du tout un print qui affiche un résultat mais qu'on ne peut exploiter, sauf si on a des yeux et un crayon, ce qui n'est pas le cas de l'ordinateur.

Si l'on veut tester que les listes ont la même longueur :

```
def pertes(R, B, sigma) :
....if not(len(R) == len(B) and len(L) == len(sigma)) :
........return(False)
....else :
........S = 0
........for k in range(len(R)) :
............S = S+abs(R[k]-B[sigma[k]])
........return(S)
```

Un programmeur zélé vérifiera que tous les indices `sigma[k]` sont aussi compris entre `0` et `N`.

Un petit calcul d'ordre de grandeurs avant de se ruer dans la programmation.
Si on a 20 tuyaux de chaque sorte, les listes sont de longueur 20. On a 20! permutations possibles.
Chaque calcul prend 10^{-10} seconde(s). On a donc besoin de $20!/10^{10}$ secondes.
On trouve un nombre de l'ordre de 10^8 secondes. On divise par 60 pour avoir des minutes, et encore par 60 pour avoir des heures, puis par 24 et 365. On trouve un résultat de l'ordre de $\boxed{7 \text{ ans}}$
Ce serait un peu idiot de lancer un ordinateur sur un tel calcul pour vendre des bouts de tuyau...

On regarde notre exemple :

rouges	=====	============	========	================
bleues	======	========	=============	====================

On prend la permutation identité et on mesure les chutes :

rouges	=====	============	========	================
bleues	======	========	=============	====================
14	=	====	=====	====

On trie les deux listes et on apparie :

rouges	=====	========	============	================
bleues	======	========	=============	====================
6	=		=	====

On constate que la valeur 6 est une valeur possible. Pourquoi ne peut on avoir moins ?

Une solution consiste à tester les vingt quatre permutations et à calculer pour chacune la perte totale. C'est évidemment une méthode un peu lourde.

On comprend qu'il vaut mieux apparier le tuyau rouge 5 avec le tuyau bleu 6 pour optimiser les pertes. On se doute bien que mettre le 5 avec le 20 créerait une chute de 15 fort peu judicieuse *(même si finalement on pourrait peut être tenter d'apparier cette chute avec le morceau de* 13*, mais on ne part pas dans cette voie de redécoupages).*
Il faut quand même le prouver.
Supposons que le 5 rouge soit avec un autre tuyau bleu α *(*$\alpha \in \{8,\ 13,\ 20\}$*)*. Le 6 bleu est alors avec un rouge a *(*$a \in \{8,\ 12,\ 16\}$*)*. On va montrer que la perte est alors plus grande que dans le cas 5 avec 6 *(et a avec α)* : $\begin{array}{|c|c|c|}\hline 5 & a & \text{autres} \\ \hline \alpha & 6 & \text{autres} \\ \hline \end{array}$ contre $\begin{array}{|c|c|c|}\hline 5 & a & \text{autres} \\ \hline 6 & \alpha & \text{autres} \\ \hline \end{array}$

Les deux pertes sont $|5-\alpha|+|a-6|+\sum_{autres}|rouge-bleu|$ et $|5-6|+|a-\alpha|+\sum_{autres}|rouge-bleu|$.
Comme on a deux fois le même terme $\sum_{autres}|rouge-bleu|$ il suffit de comparer $|5-\alpha|+|a-6|$ et $|5-6|+|a-\alpha|$.

Notre lemme permet de conclure : la configuration $\begin{array}{|c|c|c|}\hline 5 & a & \text{autres} \\ \hline 6 & \alpha & \text{autres} \\ \hline \end{array}$ minimise les pertes.

On recommence en comparant la liste $\begin{array}{|c|c|c|}\hline 5 & 8 & 12,\ 16 \\ \hline 6 & 8 & 13,\ 20 \\ \hline \end{array}$ avec $\begin{array}{|c|c|c|}\hline 5 & 8 & 12,\ 16 \\ \hline 6 & 13 & 8,\ 20 \\ \hline \end{array}$ ou $\begin{array}{|c|c|c|}\hline 5 & 8 & 12,\ 16 \\ \hline 6 & 20 & 13,\ 8 \\ \hline \end{array}$.
C'est le même lemme qui montre qu'il vaut mieux choisir le premier modèle.
On termine avec $\begin{array}{|c|c|c|c|}\hline 5 & 8 & 12 & 16 \\ \hline 6 & 8 & 13 & 20 \\ \hline \end{array}$ meilleure que $\begin{array}{|c|c|c|c|}\hline 5 & 8 & 12 & 16 \\ \hline 6 & 8 & 20 & 13 \\ \hline \end{array}$.
Bref, c'est bien la configuration $\begin{array}{|c|c|c|c|}\hline 5 & 8 & 12 & 16 \\ \hline 6 & 8 & 13 & 20 \\ \hline \end{array}$ qui optimise, avec perte $1+0+1+4$.

On reprend le raisonnement ci dessus pour montrer qu'on a intérêt à mettre ensemble le plus petit rouge et le plus petit bleu.

On note r le plus petit rouge et b le plus petit bleu. Si on ne les met pas ensemble $\begin{array}{|c|c|c|}\hline r & a & \text{autres} \\ \hline \alpha & b & \text{autres} \\ \hline \end{array}$.

Or, avec $\begin{array}{|c|c|c|}\hline r & a & \text{autres} \\ \hline b & \alpha & \text{autres} \\ \hline \end{array}$, le lemme donne $|r-b|+|a-\alpha|+autres \leqslant |r-\alpha|+|a-b|+autres$.
On comprend qu'il vaut mieux coupler les deux plus petits (Julie et Iqbal ?).
On mes met de côté, et on recommence avec ce qu'il reste.
Etape par étape, par récurrence, on montre qu'il faut apparier le i^{eme} tuyau rouge r_i (par ordre croissant) avec le i^{eme} tuyau bleu.

On passe à l'algorithme dans le cas où l'on dispose de la méthode `sort`.

```
R.sort()
B.sort()
S = 0
for k in range(len(R)) :
....S += abs(R[k]-B[k])
print(S)
```

Dans le cas où on ne dispose pas de la procédure `sort`, on va la reconstruire. Et tant pis si la manière n'est pas efficace, puisque on ne va pas travailler sur des listes de grande longueur.
On créer une petite procédure qui prend une liste, cherche le plus petit élément et l'efface de la liste :

```
def extrait_min(L) :
....LL = [] #on initialise une liste vide qu'on va remplir peu à peu
....mini = L[0]
....for k in range(1, len(L)) : #on va parcourir la liste L (sans son premier terme déjà regardé)
........elt = L[k] #on prend donc un élément de L d'indice non nul
........if elt < mini : #si on détecte une descente...
.............LL.append(mini) #on oublie le minimum précédent, donc on le met dans la liste LL
............mini = elt #on mémorise le nouveau minimum
........else : #si on n'a pas détecté une descente
.............LL.append(elt) #on transfère dans LL le nouveau nombre sans intérêt
....return(mini, LL) #le minimum est mémorisé, et la liste est un peu plus courte
```

On note qu'avec cette procédure, la liste `LL` contient bien tous les éléments de `L` sauf son minimum, mais l'ordre a été modifié.

On peut alors mettre en boucle cette procédure pour `R` et `B` :

```
n = len(R) #on mémorise la longueur des listes car on va les modifier
pertes_min = 0 #on initialise une somme nulle qu'on va augmenter au fur et à mesure
for k in range(n) : #on va solliciter extrait_min n fois
....a, R = extrait_min(R) #on sort le minimum des listes déjà réduites R et B
....b, B = extrait_min(B) #et on racourcit ces deux listes
....perte_min = perte_min + abs(b-a) #on calcule un terme de plus de la somme
print(perte_min) #on affiche la perte totale du cas optimisé
```

Un script sur des listes aléatoires.	2014-15 1-phô

On analyse diverses parties du script qui est formé d'une importation, de la définition d'une procédure visiblement récursive *(elle se rappelle elle même)*, de la création d'une liste et de la sollicitation de la procédure.

```
from random import randrange
```

On importe la fonction qui engendre des nombres "aléatoires".

```
LL = [randrange(1000) for k in range(10000)]
```

On crée justement une liste aléatoire de 10.000 termes entre 0 inclus et 1000 exclu.

```
print(leurdeschamps(LL))
```

On sollicite la procédure sur la liste qu'on vient de créer. On aurait pû compacter en

```
print(leurdeschamps([randrange(1000) for k in range(10000)]))
```

On définit donc une procédure qui prendra en variable un certain objet `L` qui sera une liste si l'on en croit la suite du programme mais aussi le test `len(L) == 1`.

```
def leurdeschamps(L) :
....if len(L) == 1 :
........return(L[0])
```

Si la liste est de longueur 1, on renvoit son élément d'indice 0 c'est à dire son unique élément. Sinon, on se lance dans une suite d'instructions.

```
........demi = len(L)//2
........a = leurdeschamps(L[0 :demi])
........b = leurdeschamps(L[demi, len(L)])
```

On sollicite la procédure sur les deux demi-listes `L[0 :demi]` et `L[demi, len(L)]`, et on stocke les résultats dans deux variables `a` et `b`.

```
........if a > b :
...........return(a)
```

```
........else :
............return(b)
```
On retourne le maximum de `a` et de `b` par simple test.
On comprend qu'on commence donc à parler de maximum.
Si la liste a deux éléments `[x, y]`, on coupe en deux et on sollicite la procédure pour les deux listes `[x]` et `[y]`. Elle renvoit les deux nombres `x` et `y`, qu'elle compare et elle renvoit alors le plus grand des deux.
Si la liste a quatre éléments `[x, y, z, t]`, elle coupe en deux et calcule `leurdeschamps([x, y])` ainsi que `leurdeschamps([z, t])`. Elle retourne donc $Max(x, y)$ et $Max(z, t)$ en vertu de l'étude précédente. Elle les compare et retourne finalement $Max(Max(x, y), Max(z, t))$, c'est à dire $Max(x, y, z, t)$.
On comprend qu'elle détermine donc le maximum des éléments d'une liste.
Si il convient de vraiment démontrer ce résultat, on le fait par récurrence forte sur les nombre d'élements de la liste.
C'est initialisé. Supposons ensuite que la propriété soit vraie pour toute liste de longueur inférieure ou égale à `n`. On donne une liste de longueur `n+1`. Elle se coupe en deux listes `L[0 :demi]` et `L[demi,n+1]`. Comme chacune est de longueur inférieure ou égale à `n`, les deux nombres `a=leurdeschamps(L[0 :demi])` et `b=leurdeschamps(L[demi,n+1])` sont les deux maxima `Max(L[0],...L[demi-1])` et `Max(L[demi],...L[n])`. Le test if `a>b` va donc retourner au final le plus grand des deux, c'est à dire `Max(L[0],...L[n])`.
Ceci achève la récurrence.
C'est ce type de démonstration qu'on vous demandera en option informatique.
Et le principe de cette démarche qui coupe le tas en deux et recommence s'appelle "*diviser pour régner*". Bref, on tire une liste aléatoire et on en cherche le maximum.

2014-15

Bases de données. 1-phô

nombre d'élèves de sup faisant anglais langue vivante 1
`SELECT COUNT(*) FROM eleves WHERE LV1="Anglais" OR LV1="ANGLAIS"`
nom des élèves de PCSI externes
`SELECT nom FROM eleves WHERE classe="PCSI" AND regime="externe"`
nom, prénom et classe des élèves de Spé ayant moins de 18 ans
`SELECT nom, prenom, classe FROM eleves WHERE date_naissance<(calcul) AND classe classe = "MP" OR classe = "MP*" OR classe = "PC"OR classe = "PC*"`
liste des élèves prénommés Alexandre ou même Alex, triés par classe
`SELECT nom, prenom FROM eleves WHERE prenom LIKE "%Alex%" SORT BY classe`
liste des professeurs de la MP*
`SELECT nom FROM professeurs WHERE MP*=1`
liste des professeurs de la filière MP (quatre classes) triés par âge
`SELECT nom, prenom FROM professeurs WHERE classe IN {MPSI1, MPSI2, MP, MP*} SORT BY date_naissance`
liste nominale des professeurs de mathématiques
`SELECT nom FROM professeurs WHERE matiere LIKE "%math%"`
liste des professeurs de l'élève Lebanc
`SELECT nom, prenom, classe FROM eleves WHERE nom="Leblanc" or nom="LEBLANC"` `SELECT nom FROM professeurs WHERE (...)=True`
vérifier qu'aucun élève de PCSI n'a pris option informatique
`SELECT COUNT(*) FROM eleves WHERE classe="PCSI" AND option="Info"`
envoyer tout de suite les élèves prénommées Lucille en MP*
`UPDATE eleves SET classe="MP*" WHERE prenom="Lucille"`

M.P.S.I.2 2014 33 points 2015 Charlemagne p 1-phô q