G. Hunault
L3 Info

Angers, juin 2019

Décomposition, Conception et Réalisation d'Applications

1. Trajets

On dispose de fichiers d'informations concernant des trajets de personnes de ville à ville par jour, mois et an nommés TRAJ*.DATA. Voici quelques exemples de lignes à traiter, sachant que les véritables fichiers comportent chacun plusieurs centaines de lignes et qu'on a plusieurs dizaines de tels fichiers. Les personnes sont repérées par un identifiant de personne, noté ici P suivant d'un nombre entier formaté avec des zéros de tête non significatifs.

Chaque fichier a toujours en ligne numéro 1 le rappel des colonnes, comme ci-dessous. Il peut y avoir des lignes vides.

```
Personne   Départ      Arrivée Jour Mois An

P00008     Angers      Nantes   17   12  2008
P00538     Paris       Nice     14   01  2009
P00532     La_Rochelle Limoges  25   10  2008
```

Question 1.1

Donner la ou les commandes UNIX qui permettent d'afficher la liste triée par ordre alphabétique de toutes les villes d'arrivée sur l'ensemble des fichiers TRAJ*.DATA.

Vous ne devez utiliser que les commandes cat, grep, head, cut, awk, wc, uniq, sort, tr mais vous n'êtes pas obligé(e) de toutes les utiliser.

Si vous utilisez awk, il faut que ce soit en mode *one-liner*.

Vous devez justifier votre choix de commande(s).

Question 1.2

Donner la ou les commandes UNIX qui permettent de savoir quelle ville est la plus fréquente comme ville d'arrivée sur l'ensemble des fichiers TRAJ*.DATA. En cas d'ex-aequo, on affichera la ville dont le nom est en premier par ordre alphabétique.

Vous ne devez utiliser que les commandes cat, grep, head, cut, awk, wc, uniq, sort, tr mais vous n'êtes pas obligé(e) de toutes les utiliser.

Si vous utilisez awk, vous avez le droit à un programme que vous fournirez, bien sûr.

Là encore, vous devez justifier votre choix de commande(s).

Question 1.3

En admettant que les informations précédentes sur les trajets sont stockées dans une table nommée `trj` (TRajets Journaliers) d'une base de données `TE` (Trajets Entreprise), quelle commande MySQL permettrait d'afficher les villes et leur fréquence cumulée sur l'ensemble des années pour les villes les plus fréquentes comme villes d'arrivée ?

On se limitera aux 5 premières lignes d'affichage trié par fréquence cumulée décroissante puis éventuellement par nom croissant de ville.

2. Distances

Les fichiers précédents donnaient juste les trajets. On dispose d'un fichier complémentaire nommé distances.txt qui contient les distances entre les villes de ces trajets, au format triangulaire inférieur.

Voici un extrait d'un tel fichier de distances :

	Angers	La_Rochelle	Limoges	Nantes	Nice	Paris
Angers	0					
La_Rochelle	187	0				
Limoges	270	220	0			
Nantes	90	165	320	0		
Nice	920	900	700	980	0	
Paris	300	470	390	390	930	0

Pour ce qui suit, on admettra qu'il y a au plus 100 villes en tout impliquées dans les trajets.

Afin de simplifier le problème, on considérera que dans le fichier des distances les villes sont triées par ordre alphabétique et que la ligne 1 contient la liste de toutes les villes, comme dans l'exemple précédent.

Question 2.1

On voudrait construire en Python la variable symétrique complète nommée matDist liée à ces données, c'est-à-dire la matrice

	Angers	La_Rochelle	Limoges	Nantes	Nice	Paris
Angers	0	187	270	90	920	300
La_Rochelle	187	0	220	165	900	470
Limoges	270	220	0	320	700	980
Nantes	90	165	320	0	980	390
Nice	920	900	700	980	0	930
Paris	300	470	390	390	930	0

Expliquer quelle méthode vous utiliseriez pour réaliser la lecture du fichier distances.txt et la construction de la matrice en Python.

Question 2.2

Donner le code Python correspondant à cette méthode. Vous pouvez utiliser toutes les fonctions de Python existantes, même celles qui n'ont pas été présentées en cours, TD ou TP, y compris issues de modules comme *NumPy*.

Question 2.3

On voudrait remplacer la matrice symétrique de toutes les distances par un tableau de distances des villes deux à deux, avec la distance `Ville_1` à `Ville_2` et de `Ville_2` à `Ville_1` utilisée une seule fois, rangée dans l'ordre `Ville_1` avant `Ville_2` par ordre alphabétique.

Pour l'exemple proposé, voici le début et la fin de ce tableau :

```
Ville_1      Ville_2        Distance
Angers       La_Rochelle    187
Angers       Limoges        270
Angers       Nantes          90
[...]
Nantes       Paris          390
Nice         Paris          930
```

Combien a-t-on de distances à conserver avec 6 villes à partir des 36 distances initiales, sachant qu'on ne garde pas la distance d'une ville à elle-même ?

Et dans le cas général, combien a-t-on de distances à conserver avec n villes, toujours sans la distance d'une ville à elle-même ?

Question 2.4

Expliquer quelle méthode vous utiliseriez pour réaliser la transformation de la matrice de toutes les distances en tableau des distances deux à deux en R.

On veut obtenir au bout du compte un tableau (*data.frame*) de toutes les distances présentes triées par ordre décroissant de distance. On ne veut garder la distance de `Ville_1` à `Ville_2` qu'une seule fois, avec `Ville_1` avant `Ville_2` par ordre alphabétique.

Pour l'exemple proposé, voici le début et la fin de ce *data.frame* :

```
Ville_1      Ville_2        Distance

Nantes       Nice           980
Nice         Paris          930
Angers       Nice           920
[...]
La_Rochelle  Nantes         165
Angers       Nantes          90
```

Bien sûr, on ne veut pas non plus ici des distances d'une ville avec elle-même.

Question 2.5

Donner le code R d'une fonction nommée mat2dist() qui prend en entrée une matrice comme matDist et qui renvoie le *data.frame* demandé, composé de toutes les distances possibles triées par ordre décroissant de distance, les villes étant triées comme indiqué.

Vous pouvez utiliser toutes les fonctions de R existantes, même celles qui n'ont pas été présentées en cours, TD ou TP, qu'elles soient chargées de base ou accessibles via un *package*.

3. Trajets et distances

On veut maintenant relier les trajets et les distances afin de savoir qui parcourt le plus de kilomètres à l'année.

Question 3.1

Si vous aviez le choix entre Awk, Perl, Php, Python, Javascript, R et Sql, quel langage choisiriez-vous et surtout, pourquoi choisiriez-vous ce langage, pour effectuer cette recherche de personne(s) parcourant le plus de kilomètres à l'année :

1 - si toutes les données sont dans des fichiers séparés ?

2 - si toutes les données ont été fusionnées dans un seul fichier ?

3 - si toutes les données sont accessibles sur Internet via une API (de type REST par exemple, mais pas uniquement) ?

Question 3.2

On suppose maintenant que les données pour les trajets sont disponibles par année, accessibles par une URL comme `http://trajets/2017.txt`, `http://trajets/2018.txt`... où chaque fichier est formaté comme à la question 1. On veut produire en PHP conceptuel un tableau XHTML comme celui fourni en annexe, à l'aide d'une fonction nommée `trajets()`. Cette fonction admet deux paramètres, dont le deuxième est facultatif : une année de début et une année de fin. Si l'année de fin n'est pas précisée, on prend l'année courante. Ainsi `trajets(2010,2015)` affiche les comptages pour les années 2010 à 2015 alors que `trajets(2016)` affiche les comptages pour les années 2016 à 2019 vu que nous sommes en 2019.

Donner les grandes lignes de la méthode que vous comptez utiliser et le pseudo-code PHP conceptuel associé. Vous pouvez inventer des sous-programmes à condition de préciser leur role et leurs paramètres.

Question 3.3

Donner le code PHP conceptuel détaillé qui implémente la fonction `trajets()`.

Question 3.4

Que faudrait-il modifier dans ce code PHP pour que le tableau soit triable au niveau des colonnes An, Personne et NbTrajets ?

ANNEXE : extrait du tableau des comptages à produire en XHTML

Le tableau complet est issu de l'instruction `trajets(2017) ;`

```
<table border="1" cellpadding="10"
       class='collapse' summary="comptages trajets" >

<tr>
  <th align='center'> An         </th>
  <th align='center'> Personne   </th>
  <th align='center'> NbTrajets </th>
</tr>

<tr>
  <td align='right'> 2017   </td>
  <td align='left' > P00532 </td>
<  td align='right'> 5      </td>
</tr>

<tr>
  <td align='right'> 2017   </td>
  <td align='left' > P00007 </td>
  <td align='right'> 25     </td>
</tr>

[...]

<tr>
  <td align='right'> 2019   </td>
  <td align='left' > P01275 </td>
<  td align='right'> 20     </td>
</tr>

</table>
```

ESQUISSE DE CORRIGÉ

Question 1.1

Il faut récupérer ici la valeur de chaque ville d'arrivée, soit la colonne numéro 3, sauf pour les lignes vides et pour la lligne 1 de chaque fichier. Une instruction awk est sans doute le plus simple ici.

Pour le tri avec unicité de chaque ville affichée, sort est suffisant.

Voici donc une réponse concise :

```
awk -e ' (FNR>1) && (NF>0) { print $3 } ' TRAJ*.DATA | sort -u
```

Question 1.2

Il est sans doute possible d'extraire le champ numéro 3 avec cut et de compter avec le *duo* de commandes sort/uniq -c, mais là encore awk est sans doute plus simple et plus court à écrire pour cumuler les fréquences. On utilise ensuite sort et head pour n'avoir que la première ville.

On pourrait donc écrire

```
awk -f vpf.awk TRAJ*.DATA | sort -rk 2 | sort -k 1 | head -n 1
```

On notera ici l'emploi volontaire de deux commandes sort successives.

Le code-source de vpf.awk est le suivant :

```
# cumul des villes d'arrivée (colonne 3)

(FNR>1) && (NF>0) {
  cntV[ $3 ]++
} # fin de récupération

# affichage des villes et de leur fréquence

END {
   for (ville in cntV) {
      print ville " " cntV[ ville ]
   } # fin pour
} # fin de END
```

Question 1.3

Si toutes les informations sont dans la base de données, le code MySQL doit ressembler à

```
SELECT arrive AS ville , COUNT( arrive ) AS frequence
       FROM TE.trj
       GROUP BY arrive
       ORDER BY frequence DESC , arrive ASC LIMIT 5 ;
```

Question 2.1

Après lecture de la ligne 1 du fichier, on connait le nombre n de villes. On créee alors une matrice M de dimensions (n, n) initialisée à 0 partout. On lit ensuite ligne par ligne les informations. A la ligne i, on lit les valeurs $M_{i,j}$ pour j de 1 à i. On recopie chaque $M_{i,j}$ en $M_{j,i}$.

Après réflexion, on lit toutes les lignes du fichier qu'on met dans un tableau de lignes nommé `tdl` à l'aide de la méthode Python nommé `readlines`.

Question 2.2

```
# coding: iso8859-1

# lecture de la matrice triangulaire inférieure
# et création de la matrice symétrique complète des distances

import re
import numpy as np

nfdd = "distances.txt"
fdd  = open(nfdd,"r")  # ouverture du fichier
tdl  = fdd.readlines() # transfert dans un tableau de chaines

for nbl,lig in enumerate(tdl) :  # parcours du tableau

  lig = re.sub("\s+"," ",lig)
  ldv = lig.split(" ")

  # la ligne 1 (indice 0) permet de connaitre le nombre
  # nbv de villes
```

```
  if nbl==0 :

     # on initialise la matrice avec des zéros

     nbv = len(ldv) - 2
     md  = np.zeros(shape=(nbv,nbv))

  else :

     i = 0
     for elt in ldv :
         i = i + 1
         if i>1 and elt!='' :
            md[ nbl-1 , i-2 ] = float(elt)
         # fin si
     # fin pour

  # fin si

#  fin pour lig


print("Matrice des distances")
print(md)
```

Question 2.3

Avec n villes, il y a n^2 distances dont n distances égales à 0 sur la diagonale de la matrice. Chacune des $n(n-1)$ distances restantes est comptée deux fois. Il ne faut donc conserver que $n(n-1)/2$ distances. Pour $n = 6$, cela fait donc 15 distances à conserver.

Question 2.4

On parcourt la matrice triangulaire inférieure des distances sauf la diagonale. Pour chaque couple (i, j) avec $j > i$ on remplit une nouvelle ligne du data.frame résultat avec les noms des villes et la distance. Une fois le data.frame rempli, on utilise la fonction `order()` pour trier selon le critère demandé.

Question 2.5

```
##############################################################################
#
# fonction qui crée un data frame de distance de ville à ville
# à partir d'une matrice complète symétrique de distances
#
##############################################################################

mat2dist <- function(mdd) {

##############################################################################

 # préparation du data.frame résultat

 nbv  <- nrow(mdd)
 noms <- c("Ville_1","Ville_2","Distance")
 dfd  <- data.frame(matrix(NA,nrow=nbv*(nbv-1)/2,ncol=length(noms)))
 names(dfd) <- noms

 # parcours de la matrice triangulaire inférieure
 # et remplissage du data.frame

 nbl <- 0
 for (i in (1:(nbv-1))) {
     for (j in ((i+1):nbv)) {
       nbl <- nbl + 1
       dfd[nbl,1] <- names(mdd)[i]
       dfd[nbl,2] <- names(mdd)[j]
       if (dfd[nbl,1] > dfd[nbl,2] ) { dfd[nbl,1:2] <- dfd[nbl,2:1] }
       dfd[nbl,3] <- mdd[i,j]
     } # fin pour j
 } # fin pour i

 # tri du data.frame par distance décroissante

 idx <- order(dfd$Distance,decreasing=TRUE)

 return(dfd[idx,])

} # fin de fonction mat2dist

#################### programme principal
```

```
# lecture de la matrice symétrique

matd <- read.table("distances2.txt",header=TRUE,row.names=1)
cat("\nMatrice symétrique complète des distances\n")
print(matd)

# construction et affichage du data frame associé

cat("\nData.frame des distances ville à ville (ordre décroissant)\n")
ddd <- mat2dist(matd)
print(ddd,right=FALSE)
```

Question 3.1

Le langage n'est sans doute pas un problème ici, sauf si les données sont sur Internet, auquel cas il faudrait, pour utiliser Awk, commencer par rapatrier les données, par exemple à l'aide de `wget` ou `curl`.

La raison principale de ce non choix est qu'après la lecture de la matrice des distances ville à ville, le cumul des trajets pour un individu se fait à l'aide de tableaux associatifs, présents dans tous les langages cités. Puisque chaque personne est en colonne 1, les villes en colonnes 2 et 3, l'année en colonne 6, le cumul pour chaque ligne lue est donc, à peu de choses près, défini par, avec une syntaxe à la Awk :

```
cumulpers[ $1 , $6 ] += matDist[ $2 , $3 ]
```

si `matDist` est la matrice des distances ville à ville. Il ne reste plus ensuite qu'à trier.

Question 3.2

On commence par initialiser la date de fin à l'année courante si la date de fin n'est pas fournie.

Une fois les données lues, disons dans le tableau `$tabComptages` via la fonction `lectureDonnees($anDeb,$anFin) ;`, on utilise les fonctions conceptuelles `table()` et `entetesTableau()` pour afficher le début du tableau.

A l'aide d'une boucle sur chacun des éléments du tableau `$tabComptages`, on génère des éléments `<tr>` et `<td>` via les fonctions conceptuelles `tr()` et `td()`. En fin de parcours, on utilise `finTable()`.

Question 3.3

```
<?php

include("std7.php") ;

##########################################################

function trajets($anDeb,$anFin=0) {

##########################################################

if ($anFin==0) { $anFin = date("Y") ; } ;

h1("Trajets de $anDeb à $anFin") ;

blockquote() ;

# on simule le rapatriement des données, soit
#  $tabComptages = lectureFichiers($anDeb,$anFin) ;
# par un petit tableau, pour tester la fonction

$tabComptages = array() ; $idl= 0 ;

$idl++ ;
$tabComptages[$idl] = array() ;
$tabComptages[$idl]["an"]   = 2017  ;
$tabComptages[$idl]["pers"] = "P00532" ;
$tabComptages[$idl]["cnt"]  = 5 ;

$idl++ ;
$tabComptages[$idl] = array() ;
$tabComptages[$idl]["an"]   = 2017  ;
$tabComptages[$idl]["pers"] = "P00007" ;
$tabComptages[$idl]["cnt"]  = 25 ;

$idl++ ;
$tabComptages[$idl] = array() ;
$tabComptages[$idl]["an"]   = 2019  ;
$tabComptages[$idl]["pers"] = "P01275" ;
$tabComptages[$idl]["cnt"]  = 20 ;

table(1,10,"collapse","comptages trajets") ;
```

```
sdl() ;

entetesTableau("An Personne NbTrajets") ;
sdl() ;

$nbl = count($tabComptages) ;
for ($idl=1;$idl<=$nbl;$idl++) {
  tr() ;
     td("R") ; echo $tabComptages[$idl]["an"]   ; fintd() ;
     td()    ; echo $tabComptages[$idl]["pers"] ; fintd() ;
     td("R") ; echo $tabComptages[$idl]["cnt"]  ; fintd() ;
  fintr() ;
} # fin pour idl
finTable()   ;

finblockquote() ;

} ; # fin de fonction trajets

##  exemple d'utilisation

debutPageMinimal("trajets","std.css") ;
blockquote() ;
trajets(2017) ;
finblockquote() ;
finPageMinimal() ;

?>
```

Question 3.4

Comme vu en cours et en TD, un tri interactif dans une page Web peut se faire avec le code Javascript nommé sorttable.

Il faut inclure le fichier `sorttable.js` dans la partie `<head>` de la page, via l'élément `<script>`, attribut `src` puis définir ou ajouter la valeur `sortable` à l'attribut `class` de chaque élément `<table>` des tableaux à trier.