Informatique — MPSI/PCSI, 2023–2024

Corrigé du concours blanc

Exercice 1 – Parcours routier

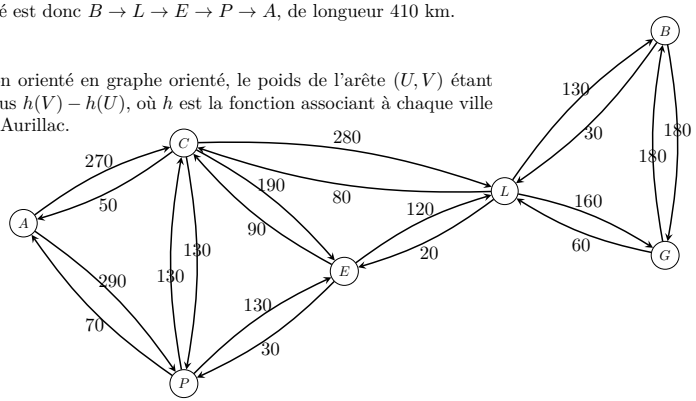
Question 1.1. On indique entre parenthèses le dernier sommet ayant modifié la valeur, afin de faciliter la lecture.

	A	B	C	E	G	L	P
Initialisation	$+\infty$	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
Étude de : B	$+\infty$	0	$+\infty$	$+\infty$	180(B)	80(B)	$+\infty$
Étude de : L	$+\infty$	0	260(L)	150(L)	180	80	$+\infty$
Étude de : E	$+\infty$	0	260	150	180	80	230(E)
Étude de : G	$+\infty$	0	260	150	180	80	230
Étude de : P	410(P)	0	260	150	180	80	230
Étude de : C	410	0	260	150	180	80	230
Étude de : A	410	0	260	150	180	80	230

Le plus court chemin recherché est donc $B \to L \to E \to P \to A$, de longueur 410 km.

Question 1.2.

1. On transforme le graphe non orienté en graphe orienté, le poids de l'arête (U, V) étant le poids de l'arête initiale plus $h(V) - h(U)$, où h est la fonction associant à chaque ville sa distance à vol d'oiseau d'Aurillac.



2.

	A	B	C	E	G	L	P
Initialisation	$+\infty$	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
Étude de : B	$+\infty$	0	$+\infty$	$+\infty$	180(B)	30(B)	$+\infty$
Étude de : L	$+\infty$	0	110(L)	50(L)	180	30	$+\infty$
Étude de : E	$+\infty$	0	110	50	180	30	80(E)
Étude de : P	150(P)	0	110	50	180	30	80
Étude de : C	150	0	110	50	180	30	80
Étude de : A	150	0	110	50	180	30	80

Le plus court chemin recherché est donc toujours $B \to L \to E \to P \to A$, de longueur $150 + 260 - 0 = 410$ km. Mais cette fois-ci, on l'a obtenu en une étape de moins.

Exercice 2 – *Il y a de la tension dans l'air !*

Question 2.1. Pour calculer la moyenne des éléments d'une séquence, il suffit d'en faire la somme puis de diviser par le nombre d'éléments. Initialisant la somme avec son éléments neutre, un simple parcours par éléments suffit. Le code suivant convient.

```
def moy(U):
    s=0
    for e in U:
        s+=e
    return(s/len(U))
```


Question 2.2. Pour détecter une valeur qui dépasserait un seuil, il s'agit de parcourir successivement les éléments tant qu'ils ne dépassent pas ce seuil. Ainsi, si une valeur dépasse le seuil, alors la boucle s'arrête à un indice `i` strictement plus petit que la longueur de la séquence. Le code suivant convient.

```
def surtension(U, seuil=220):
    i=0
    while i<len(U) and U[i]<seuil:
        i+=1
    return(i<len(U))
```

Question 2.3. Pour trouver le maximum d'un séquence, il faut utiliser la méthode du candidat qui consiste à initialiser le premier candidat avec le premier élément de la séquence, puis à parcourir tous les autres éléments en les comparant au candidat, que l'on met à jour si besoin. Le code suivant convient.

```
def max(U):
    c=U[0]
    for e in U[1:]:
        if e>c:
            c=e
    return(c)
```

Question 2.4. De façon similaire, en inversant simplement la relation d'ordre, il vient :

```
def min(U):
    return(-max([-e for e in U]))
```

Question 2.5. Pour définir l'amplitude crête-à-crête, il suffit de faire la différence entre le maximum et le minimum de la séquence ; ce que fait le code suivant.

```
def vpp(U):
    return(max(U)-min(U))
```

Question 2.6. De façon naïve, il est possible de réaliser un parcours par indices tel qu'il vienne par compréhension :

```
def soustraction(U, V):
    return([U[k]-V[k] for k in range(len(U))])
```

et que l'on peut même abréger sous la forme :

```
def soustraction(U, V):
    return([u-v for u,v in zip(U,V)])
```

où la fonction `zip` permet de parcourir simultanément les éléments des séquences de mêmes longueurs données en argument.

Question 2.7. Un front est montant si :

$$(U_i < m) \wedge (U_{i+1} > m)$$

où le « $\wedge$ » est le « ET » logique. En faisant attention au fait que les indices sont pris par paire, et donc $i \in [\![0, n-2]\!]$, le code suivant convient pour détecter les fronts.

```
def fronts(U):
    f=[]
    m = moy(U)
    for i in range(len(U)-1):
        if U[i]<m and U[i+1]>m:
            f.append(i)
    return(f)
```

Question 2.8. La période correspond à la durée entre les instants de détection des fronts montants que l'on trouve par simple image dans `T` avec les indices trouvés avec la fonction `fronts`, soit :

```
I = [T[i] for i in fronts(U)]
```

Pour calculer les « périodes » associées, il faut faire les différences :

$$\{T_k = I_{k+1} - I_k,\ k \in [\![0, \text{len}(I) - 2]\!]\}$$

ce que l'on peut faire avec la fonction `soustraction` sous la forme :

```
P = soustraction(I[1:],I[:-1])
```

ou les coupes `[1:]` et `[:-1]` signifient respectivement tous les éléments sauf le premier et sauf le dernier. On réalise enfin la moyenne des éléments de cette séquence de longueur $\text{len}(I) - 2$ avec la fonction `moy`. C'est ce que fait le code suivant.

```
def periodes(T,U):
    I = [T[i] for i in fronts(U)]
    P = soustraction(I[1:],I[:-1])
    return(len(P), moy(P))
```

Question 2.9. Dans le cas qui nous occupe, la subdivision des instants n'est pas forcément régulière et on a donc :

$$x_{k+1} - x_k \neq \left(\frac{b-a}{n}\right) \implies \mathcal{A}(n) = \frac{1}{2}\sum_{k=0}^{n-1} (f(x_k) + f(x_{k+1})) \times (x_{k+1} - x_k)$$

Il vient l'implémentation :

```
def trapezes(T, V):
    s=0
    for k in range(len(V)-1):
        s+=(V[i]+V[i+1])*(T[i+1]-T[i])
    return(s/2)
```

où l'on ne fait qu'une seule fois la division par 2.

Question 2.10. Calculer la valeur efficace avec le plus grand nombre de périodes $n \geqslant 1$ possible, consiste à calculer :

$$U_{\text{eff}} = \sqrt{\frac{1}{nT}\int_0^{nT} u^2(t)\,\mathrm{d}t}$$

où nT est la durée entre les instants du premier et du dernier fronts montants détectés avec

```
I = [i for i in fronts(U)]
a,b = I[0], I[-1]
```

et se traduit par `T[b]-T[a]`. Ainsi, l'intervalle « $[a, b]$ » de calcul de l'intégrale correspond a `T[a:b+1]`, le `+1` après `b` venant de ce que la deuxième valeur de la coupe est le successeur du dernier élément. Dès lors, pour calculer la valeur efficace, on forme une séquence des u^2 de même longueur que `T[a:b+1]` directement par compréhension

```
V = [u**2 for u in U[a:b+1]]
```

puis on calcule l'intégrale par simple appel de la fonction `trapezes`. On veille enfin à diviser par nT avant de prendre la racine carrée. Le code suivant convient.

```
def efficace(T,U):
    I = [i for i in fronts(U)]
    a,b = I[0], I[-1]
    return((Trapezes(T[a:b+1], [u**2 for u in U[a:b+1]])/(T[b]-T[a]))**.5)
```

Exercice 3 – Tri de chaînes de caractères

On notera que la fonction `rang` peut s'écrire simplement :

```
def rang(c):
    return(ord(c)-97)
```

Question 3.1. On utilise la méthode du candidat. Le code suivant convient.


```
def lettres(L):
    n = len(L[0])
    for e in L[1:]:
        if len(e)>n:
            n=len(e)
    return(n)
```

Question 3.2. La seule difficulté est de trouver la liste de `T` où mettre chaque liste de `L`, notée `e`. Comme on trie, sur le caractère `i`, alors l'indice correspond à `rang(e[i])`.

```
def comptage(L,i):
    T=[[] for k in range(26)]
    for e in L:
        T[rang(e[i])].append(e)
    return(T)
```

Question 3.3. Comme l'argument de la fonction est une liste de listes, il faut deux boucles imbriquées pour accéder à toutes les chaînes de caractères qu'elles contiennent et ajouter chacune à une liste finale. Le code suivant convient.

```
def fusion(T):
    M=[]
    for L in T:
        for e in L:
            M.append(e)
    return(M)
```

Question 3.4. Le code permet de calculer les listes suivantes :

```
A=['bcd', 'dab', 'ccc', 'ecf', 'add', 'ada', 'daa']
B=['ada', 'daa', 'dab', 'ccc', 'bcd', 'add', 'ecf']
C=['daa', 'dab', 'ccc', 'bcd', 'ecf', 'ada', 'add']
D=['ada', 'add', 'bcd', 'ccc', 'daa', 'dab', 'ecf']
```

On peut remarquer que la liste `D` est triée en ordre alphabétique.

Question 3.5. D'après ce qu'on vient de voir, il suffit de trier puis fusionner la liste `L` en fonction du caractère d'indice $p-1$, puis de recommencer avec le caractère d'indice $p-2$, etc, jusqu'au premier caractère. Le code suivant convient.

```
def trip(L):
    p = len(L[0])
    for i in range(p):
        L=fusion(comptage(L,p-1-i))
    return(L)
```

Question 3.6. On commence par adapter la fonction `comptage` dans le cas où le mot serait de longueur moins grande et donc n'aurait pas de caractère. On lui associe le rang 1 et donc `a` prend le rang 2, et ainsi de suite. On branche alors en fonction du nombre de lettres. On définit la fonction :

```
def recomptage(L,i):
    T=[[] for k in range(27)]
    for e in L:
        if i<len(e):
            T[1+rang(e[i])].append(e)
        else:
            T[0].append(e)
    return(T)
```

On procède ensuite de la même façon, en faisant simplement attention à initialiser `p` avec le nombre maximal de lettres. Le code suivant convient.

```
def dico(L):
    p = lettres(L)
    for i in range(p):
        L=fusion(recomptage(L,p-1-i))
    return(L)
```

— Fin du corrigé —