DM0 corrigé : autour de données météorologiques

Partie I. Variations autour du minimum

Q1. On suit l'évolution des variables `p` et `i` lors de l'appel de `mini([6,2,15,2,15])`. On indique aussi la valeur de `t[i]`.

`i`	`p`	`t[i]`
–	6	–
0	6	6
1	2	2
2	2	15
3	2	2
4	2	15

Q2. (Cas $n > 0$). On définit l'invariant de boucle :

$$\mathcal{H} : \text{« } p = \min(t[0], \cdots, t[i]) \text{ »}$$

- Après le premier tour de boucle : $i = 0$ et $p = t[0]$ donc $p = \min(t[0])$. Donc $\mathcal{H}$ est vraie.
- Supposons que $\mathcal{H}$ est vraie après le k-ième tour de boucle ($1 \leq k < n$). Dans ce cas : $i = k - 1$ et $p = \min(t[0], \cdots, t[k-1])$.
  Passons le $k+1$-ième tour de boucle. Nous avons $i = k$ et :
  i. si $p \geq t[k]$ alors
  $$\forall j \in \{0, ..., k-1\}, t[j] \geq t[k].$$
  Comme $p = t[k]$ on a toujours : $p = \min(t[0], \cdots, t[k])$.
  ii. si $p < t[k]$ alors
  $$\exists j \in \{0, ..., k-1\}, t[j] < t[k].$$
  Donc $\min(t[0], \cdots, t[k-1]) = \min(t[0], \cdots, t[k])$; comme p n'est pas modifié on a toujours : $p = \min(t[0], \cdots, t[k])$.
  Après le $k+1$-ième tour de boucle $\mathcal{H}$ est vraie.
- **Terminaison.** La propriété $\mathcal{H}$ est toujours vraie après le n-ième tour de boucle. Donc $i = n - 1$ et $p = \min(t[0], \cdots, t[n-1]) = \min(t)$: c'est la valeur retournée par l'appel de `mini(t)`.

Q3. Lors de l'appel de `mini(t)` sur une liste de n éléments on réalise n comparaisons d'entiers et au plus $n+1$ affectations. La complexité temporelle de la fonctions est en $O(n)$ ou encore **linéaire** en la taille de l'entrée.

Q4. Modification pour obtenir le maximum :
Ligne 1. `def maxi(t):` *Ligne 2.* `"' Calcule le maximum d'un tableau d'entiers ou de flottants "'` *Ligne 7.* `if t[i] >= p`

Q5. On conserve la structure de la fonction `mini`. On définit une variable locale `pos` qui retient la position courante du minimum (dans la variable `p`). La variable `pos` est initialisée à 0 et sa valeur est modifiée à chaque tour de boucle où `p` est modifiée.

Q6. Définition de la fonction retournant la position d'un minimum.

```
1  def position_mini(t):
2      p=t[0]
3      pos=0
4      for i in range(len(t)):
5          if t[i] <= p:
6              p=t[i]
7              pos=i
8      return pos
```

Q7. C'est l'indice du **dernier minimum** qui est renvoyé ; pour changer ce comportement on peut remplacer la comparaison large `<=` par une comparaison stricte `<` à la ligne 5 : on obtient alors le **premier minimum**.

Q8. On donne un tableau (liste de listes) `T` en entrée. On peut calculer successivement le minimum de chaque liste interne `T[0]`,...,`T[n-1]`. On utilise une variable locale `m` qui à chaque itération retient la valeur `mini(T[i])` si la liste `T[i]` à un minimum inférieur aux précédentes. C'est le même principe que la fonction `mini` ; on obtient alors le minimum des minima des listes internes : c'est le minimum du tableau.

Q9. Définition de la fonction retournant le minimum d'un tableau (liste de listes). On utilise la fonction `mini`.

```
1  def mini2D(T):
2     m=mini(T[0])
3     for i in range(len(T)):
4        mLigne=mini(T[i])
5        if mLigne <= m:
6           m=mLigne
7     return m
```

Q10. Supposons que T est un tableau de m lignes et n colonnes. À chaque tour de boucle on réalise un appel de `mini` sur une liste de taille n et une comparaison. Il y a m tour de boucles. La complexité de la fonction est en $O(m \times n)$.

Pour un tableau carré on peut qualifier la complexité temporelle de *quadratique* en la taille de l'entrée.

Q11. Définition de la fonction `chaine_mini(l)`.

```
1  def chaine_mini(l):
2     p=l[0][1]
3     pos=0 # position du minimum courant
4     for i in range(len(T)):
5        if l[i][1] <= m:
6           m=l[i][1]
7           pos=i
8     return L[pos][0]
```

Q12. Définition de la fonction `majores_par`.

```
1  def majores_par(t,seuil):
2     cpt=0 # compteur
3     for i in range(len(T)):
4        if t[i] < seuil:
5           cpt=cpt+1
6     return cpt
```

Partie II. Recherche dans les archives de Météo-France

Q13. Une **clé primaire** d'une table c'est un attribut (ou ensemble d'attributs) tel que deux lignes distinctes ont des valeurs distinctes pour cet attribut. Si la clé est constituée de plusieurs attributs ou s'arrange pour avoir un ensemble minimal pour l'inclusion.

Table `villes`

- La colonne `nom` ne définit pas une clé primaire car deux villes distinctes peuvent avoir le même `nom` (ex : Saint Denis).
- La colonne `insee` définit une clé primaire car deux villes distinctes ont un identifiant `insee` distinct.

Table `mesures`

- La colonne `ville` ne définit pas une clé primaire car deux mesures distinctes peuvent avoir été effectuées dans la même ville (ex : 25056).
- Aucune autre colonne ne définit de clé primaire : à cet effet on peut rajouter une colonne `identifiant` ou choisir l'ensemble `ville,jour` pour avoir une clé primaire.

Q14. On peut écrire l'instruction SQL :

```
SELECT * FROM "mesures" WHERE ("Tmin"+"Tmax")/2 < 0
```

Q15. On peut écrire l'instruction SQL :

```
SELECT "insee","nom","dpt" FROM "villes" WHERE "lat">47 AND "lon">2
```

Q16. Il s'agit d'une jointure des tables `R1` et `R2` sur le critère d'égalité entre l'attribut `insee` et l'attribut `ville`. On obtient alors une nouvelle table dont les colonnes sont :

ville	jour	Tmin	Tmax	insee	nom	dpt

avec la condition que dans chaque ligne l'attribut `insee` et l'atrribut `ville` sont identiques.

On peut encore reformuler ce résultat en disant que l'on a complété chaque ligne de la table `R1` par les lignes de la tables `R2` avec comme critère de retenir l'insee, le nom et le département où a eu lieu la mesure.

On peut écrire l'instruction SQL :

```
SELECT * FROM "R1" JOIN "R2" ON "ville"="insee"
```

Partie III. Un brin d'analyse numérique

1. Manipulation de données

Q17. Nous supposons que qu'on utilise le code ASCII de sorte que chaque caractère est codé sur un octet. Chaque ligne(sauf la première) nécessite au plus 15 caractères ; il y a 365 lignes. La taille du fichier de l'ordre de 5000 octets = 5 Mo.

Q18. Les lignes 5 et 6 permettent de réaliser les instructions suivantes : si le début de la ligne du fichier n'est pas le caractère `#` (*i.e* nous ne sommes pas en première ligne) alors les variables `t`, `T1` et `T2` reçoivent respectivement le numéro du jour, la température minimale et la température maximale données dans une ligne. Le type de ces objets est : *chaîne de caractères*.

Les variables `jours`, `Tmin` et `Tmax` sont des liste (type `list`) contenant des nombres à virgule flottante (type `float`).

Q19. La méthode `append` est optimisée pour ajouter une case à une liste. La concaténation `jours = jours + [int(t)]` nécessite de récréer complétement en mémoire la nouvelle liste `jours` augentée de sa dernière case en écrasant la précédente valeur de `jours`.

Q20. Définition de la fonction `moyenne`.

```
1  def moyenne(a,b):
2      if len(a)!=len(b):
3          return None
4      moy=[(a[0]+b[0])/2]
5      for i in range(1,len(a)):
6          moy.append( (a[i]+b[i])/2 )
7      return moy
```

Q21. On peut utiliser l'instruction :

```
Tmoy=moyenne(Tmin,Tmax)
```

Q22. On peut utiliser l'instruction :

```
nb_jours_gel=majores_par(Tmoy,0)
```

2. Modélisation physique

Q23. Rappel sur la méthode d'Euler : cf cours. Les paramètres d'entrée sont :

(a) `f` : de type fonction `func` ; la fonction de deux variables y et t dans le problème de Cauchy $y' = f(y,t)$.

(b) `y0` : de type `float` ; la condition initiale $y(0)$ dans le problème de Cauchy.

(c) `dt` : de type `float` ; le pas de la discrétisation utilisé pour découper l'intervalle sur lequel on résout numériquement le problème de Cauchy.

Q24. Fonction `f`. On suppose que la fonction `cos` a été importée.

```
11  def f(theta,t):
12      return mu*(1+epsilon*cos(omega*t)) - alpha*(theta+T0)**4
```

Q25. La méthode d'Euler retourne une liste de couples donnant pour chaque valeur t_k d'une abscisse une ordonnée y_k approchant la valeur de la solution réelle $y(t_k)$. On peut considérer qu'on a les associations suivantes :

(a) Rond noir : 81 jours.

(b) Triangle bleu : 27 jours.

(c) Losange rouge : 54 jours.

Plus le pas d'intégration est petit, plus la solution numérique obtenue par la méthode d'Euler s'approche de la solution réelle du problème de Cauchy. À l'inverse, en choisissant un pas d'intégration trop important l'adéquation entre la solution numérique et la solution réelle n'est plus assurée. L'écart augmente à mesure que l'on s'éloigne du temps initial car les erreurs se cumulent à chaque pas de la discrétisation.

Q26. Les valeurs expérimentales *oscillent* autour des valeurs données par la modélisation. L'espérance de l'écart semble avoir une valeur nulle.

En prennant l'hypothèse d'un réchauffement climatique on peut penser que la courbe expérimentale va se *décaler vers le haut* avec le temps prennant une valeur moyenne sur une *pseudo-pédiode* plus élèvée. Ni le modèle, ni les mesures ne semble indiquer de réchauffement dans l'intervalle de temps considéré.