Informatique Mines 2023 : La typographie informatisée. Corrigé des parties I, II et VI

Partie I

Q1) 100 en base 16 correspond à $16^2 = 2^8 = 256$.

L'hexa dollar vaut donc 2.56 dollars ...

Q2) On obtient la lettre j.

Partie II

Q6) En fait, il y a une petite erreur dans l'énoncé, car la valeur renvoyée par la fonction `points` est une liste de points, donc de type `[[float]]`.

```
def points(v) :
    L = []
    for M in v :
        for point in M :
            L.append(point)
    return L
```

Q7)

```
def dim(l,n) :
    return [p[n] for p in l]
```

Q8) La largeur est l'écart entre l'abscisse minimale et l'alscisse maximale des points de la représentation vectorielle.

Il convient ici d'utiliser les deux fonctions précédentes.

```
def largeur(v) :
    L = dim(points(v),0)
    return max(L) - min(L)
```

Q9) Il faut utiliser la fonction glyphe décrite dans l'énoncé juste avant la partie II

```
def obtention_largeur(police) :
    L = []
    for c in range(26) :
        L.append(largeur(glyphe(c,police,True)))
        L.append(largeur(glyphe(c,police,False)))
    return L
```

Q10)

```
def transforme(f,v) :
```

```
    L = []
    for M in v :
        L.append([ f(point) for point in M]
    return L
```

Q11) On diminue ainsi la largeur du glyphe tout en conservant sa forme.

Q12)

```
def penche(v) :
    def f(p) :  return [p[0]+0.5*p[1],p[1]]
    return transforme(f,v)
```

Partie VI

Q21) On parcourt la liste des mots en remplissant chaque ligne au maximum et en passant à la ligne lorsque le mot en cours de lecture ne tient pas sur la ligne courante. Il est dit glouton dans la mesure où on ne tient compte que du mot en cours de lecture pour décuder du passage à la ligne.

Q22) Pour l'exemple a), le couple (i, j) vaut respectivement $(0, 2)$, $(3, 3), (4, 4)$ sur chaque ligne.
La somme des coûts associés est donc $(10 - 2 - 2 - 4 - 2)^2 + (10 - 6)^2 + (10 - 6)^2 = 32$.

Pour l'exemple b), le couple (i, j) vaut respectivement $(0, 1)$, $(2, 3), (4, 4)$ sur chaque ligne.
La somme des coûts associés est donc $(10 - 2 - 2 - 4)^2 + (10 - 2 - 6)^2 + (10 - 6)^2 = 24$.

La solution obtenue par programmation dynamique est donc plus équilibrée.

Q23) L'algorithme consiste à envisager toutes les possibilités de placements sur la première ligne des mots pris dans la liste `lmots` à partir du mot d'indice i et de calculer récursivement la solution optimale. Le problème de l'algorithme récursif naïf est d'appeler récursivement la fonction `algo_recursif` avec les mêmes paramètres un grand nombre de fois.Le dictionnaire `memo` permet de mémoriser les valeurs déjà calculées :

```
def progd_memo(i,lmots,L,memo) :
    if (i in memo) :  return memo[i]
    mini = float("inf")
    for j in range(i+1,len(lmots)+1) :
        d = progd_memo(j,lmots,L)+cout(i,j-1,lmots,L)
        if d < mini :  mini = f
    memo[i] = mini
    return mini
```

Lorsque la valeur a déjà été calculée, on la renvoie directement. Sinon, on la calcule, puis on la renvoie.

Remarque : On pourrait ici tout aussi bien utiliser un tableau plutôt qu'un dictionnaire, puisque la clé est un indice i variant de 0 à $n-1$, où n est le nombre de mots.

Q24) On note $c(i,j)$ le coût de la fonction `cout(i,j,lmots,L)`.

Ainsi, $c(i,j-1)$ demande $O(j-i)$ additions et soustractions.

Dans le cas de l'algorithme récursif `algo_recursif` : on note $f(n)$ le coût de `algo_recursif(0,lmots,L)`, où n le nombre de mots.

Le coût de `algo_recursif(i,lmots,L)` est $f(n-i)$. D'où (avec les hypothèses de l'énoncé) :

$$f(0) = 0 \text{ et } f(n) = \sum_{i=1}^{n} (f(n-i) + K) = \sum_{i=0}^{n-1} f(i) + Kn$$

D'où $f(n) = f(n-1) + Kn + \sum_{i=0}^{n-2} f(i) = 2f(n-1) + K$, avec

On écrit la relation sous la forme $(f(n) + K) = 2(f(n-1) + K)$, d'où on obtient $f(n) = O(2^n)$.

Dans le cas de l'algorithme `progd_bashaut`, le nombre d'additions vérifie

$$g(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} (j-i), \text{ qui est donc en } O(n^3)$$

L'interêt de la programmation dynamique est de gagner de façon considérable dans la complexité en temps en stockant des informations afin de ne pas faire des calculs redondants.

Q25) Il convient de bien comprendre le sens de $t[i]$.

Pour tout $0 \leq i < n$, $t[i]$ donne l'indice du dernier mot de la première ligne lors du placement optimisé des mots d'indices $\geq i$. Autrement dit, si on commençait au i-ième mot, la première contiendrait tous les mots d'indices compris entre i et $t[i]$ exclu (au singulier). On a toujours $i \leq t[i] < n$.

Pour en déduire le placement optimisé pour tous les mots, on part de $i = 0$, on place les mots jusqu'à l'indice $t[0] - 1$ sur la première ligne, puis on repart de $i = t[0]$, et on place les mots jusqu'à l'indice $t[t[0]] - 1$, puis on repart de $i = t[t[0]]$, etc ... D'où l'algorithme :

```
def lignes(mots,t,L) :
    L = []
    i = 0 ; n = len(mots)
    while t[i]<n :
        L.append([mots(i,t[i])])    # L est une liste de sous-listes
        i = t[i]
    return L
```

Q26) Il faut comprendre que dans l'exemple donné, la chaîne de caractères attendue est :

```
"ut    enim" + "\n" + "ad  minima" + "\n" + "veniam"
```

On utilise notamment les fonctions précédemment définies :

```
def formatage(lignesdemots,L) :
    ### on récupère d'abord la liste des mots :
    mots = []
    for L in lignesdemots :
        for m in L : mots.append(m)
    ### on calcule ensuite le tableau t :
    t = [0]*len(mots)
    progd_bashaut(lmots,L,t)    # ici, la fonction modifie t
    ### on détermine enfin le nouveau placement des mots par lignes :
    justifiees = lignes(mots,t,L)
    ### on conclut en appliquant aux lignes (sauf la dernière) une fonction auxilaire format
    s = ""    # on initialise à la chaîne de caractères vide
    r = len(justifiees)
    for k in range(r-1) :  s = s + format(justifiees[k],L) + "\n"
    s = s + justifiees[r-1]
    return s
```

Et la fonction auxiliaire `format` est définie de la façon suivante :

```
def format(ligne:[str],L:int) -> str :
    n = len(ligne) ; q = L//(n-1) ; r = L%(n-1)
    ### on a par division euclidienne L = (n-1)q + r = r*(q+1) + (n-1-r)*q
    ### on va prendre les r premiers écarts égaux à q+1 espaces, et q pour les suivants
    s = ""
    for k in range(r) :
        s = s + ligne[k] + (q+1) * " "
    for k in range(r,n-1) :
        s = s + ligne[k] + q * " "
    s = s + ligne[n-1]
    return s
```