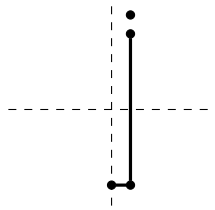
Mines-Ponts informatique 2023 : Corrigé

Partie I

Q1

$100_{16} = 1 \times 16^2 + 0 \times 16 + 0 = 256$ cents soit 2,56 dollar.

Q2



On obtient le caractère `j` minuscule.

Partie II

Q3

```
SELECT COUNT(*) FROM Glyphe WHERE groman = 'True';
```

Q4

```
SELECT gdesc FROM Glyphe JOIN Caractere ON Glyphe.code = Caractere.code
JOIN Police ON Police.pid = Glyphe.pid
WHERE car = 'A' AND pnom = 'Helvetica' AND groman = 'False';
```

Q5

```
SELECT fnom, COUNT(Famille.fid) FROM Famille JOIN Police
ON Famille.fid = Police.fid GROUP BY Famille.fid ORDER BY fnom;
```

Partie III

Q6

```
def points(v):
    liste = []
    for segment in v:
        for p in segment:
            liste.append(p)
    return liste

def points(v):
    return [p for segment in v for p in segment]
```

Q7

```
def dim(lst, n):
    liste = []
    for p in lst:
        liste.append(p[n])
    return liste

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

Q8

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

Q9

```
def obtention_largeur(police):
    lst_largeur = []
    for c in 'abcdefghijklmnopqrstuvwxyz':
        lst_largeur.append(largeur(glyphe(c, police, True)))
        lst_largeur.append(largeur(glyphe(c, police, False)))
    return lst_largeur
```

Q10

```
def transforme(f, v):
    liste = []
    for segment in v:
        lst = []
        for p in segment:
            lst.append(f(p))
        liste.append(lst)
    return liste
```

Q11

Réduction en largeur (abscisses divisées par 2). Le glyphe est deux fois moins large.

Q12

```
def penche(v):
    liste = []
    for segment in v:
        lst = []
        for x, y in segment:
            lst.append([x+0.5*y, y])
        liste.append(lst)
    return liste
```

Partie IV

Q13

(0, 0), (1, 0), (2, 1), (3, 1), (4, 1), (5, 2), (6, 2)

Q14

(9, 8) et (1, 9). La boucle n'est pas exécutée car `dx` est négatif.

`assert x1 > x0` ou `assert dx > 0`

Q15

(3, 0), (4, 4), (5, 8). Les points sont isolés.

Q16

```
def trace_quadrant_sud(im, p0, p1):
    x0, y0 = p0
    x1, y1 = p1
    dx, dy = x1 - x0, y1 - y0
    im.putpixel(p0, 0)
    for i in range(1, dy):
        p = (x0 + floor(0.5 + dx * i / dy), y0 + i)
        im.putpixel(p, 0)
    im.putpixel(p1, 0)
```

Q17

```
def trace_segment(im, p0, p1):
    if p0 == p1:
        im.putpixel(p0, 0)
    else:
        x0, y0 = p0
        x1, y1 = p1
        dx, dy = x1 - x0, y1 - y0
        if abs(dy) <= abs(dx): # horizontal
            if x1 < x0:
                p0, p1 = p1, p0
            trace_quadrant_est(im, p0, p1)
        else: # vertical
            if y1 < y0:
                p0, p1 = p1, p0
            trace_quadrant_sud(im, p0, p1)
```

Partie V

Q18

```
def position(p, pz, taille):
    x = pz[0] + floor((taille-1) * p[0])
    y = pz[1] - floor((taille-1) * p[1])
    return [x, y]
```

Q19

```
def affiche_car(page, c, police, roman, pz, taille):
    desc = glyphe(c, police, roman)
    for segment in desc:
        p0 = position(segment[0], pz, taille)
        if len(segment) == 1:
            im.putpixel(p0, 0)
        else:
            for i in range(1, len(segment)):
                p1 = position(segment[i], pz, taille)
                trace_segment(page, p0, p1)
                p0 = p1
    return floor(largeur(desc) * (taille-1)) + 1
```

Q20

```
def affiche_mot(page, mot, ic, police, roman, pz, taille):
    for c in mot:
        pz[0] = pz[0] + affiche_car(page, c, police, roman, pz, taille)
        pz[0] = pz[0] + ic
    return pz
```

Q21

Pour chaque mot, deux choix : l'ajouter à la ligne en cours si c'est possible ou l'ajouter à une nouvelle ligne. Le premier est considéré comme le « meilleur » choix. L'algorithme proposé est glouton car pour chaque mot, on applique le « meilleur » choix.

Q22

1er cas : indices (0, 2) puis (3, 3) puis (4, 4)
ligne 1 $(10 - [(2 + 4 + 2) + 2])^2 = 0$, ligne 2 $(10 - 6)^2 = 16$, ligne 3 $(10 - 6)^2 = 16$

2e cas : indices (0, 1) puis (2, 3) puis (4, 4)
ligne 1 $(10 - [(2 + 4) + 1])^2 = 9$, ligne 2 $(10 - [(2 + 6) + 1])^2 = 1$, ligne 3 $(10 - 6)^2 = 16$
La deuxième méthode est plus intéressante (coût 26 contre 32).

Q23

```
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, memo) + cout(i, j-1, lmots, L)
        if d < mini:
            mini = d
    memo[i] = mini
    return mini
```

Q24

Complexité fonction `cout` : $j - i$

Complexité fonction `algo_recursif` : $C(0)$ avec $C(i) = \sum_{j=i+1}^{n}(C(j) + j - i)$. On en déduit $C(i) \geq 2C(i + 1)$ d'où $C(0) \geq 2^n$. La complexité est exponentielle.

Complexité fonction `progd_bashaut` : pour calculer `M[i]` avec la fonction `coût`, on a au moins $\sum_{k=1}^{n-i} k$. Donc pour `M[0]`, $\sum_{i=1}^{n} \frac{i(i+1)}{2}$ qui est de l'ordre de $\sum_{i=1}^{n} i^2$. Donc complexité en n^3.

Q25

```
def lignes(mots, t, L):
    lst = []
    i = 0
    while i < len(t):
        lst.append(mots[i:t[i]])
        i = t[i]
    return lst
```

Q26

```
def formatage(lignesdemots, L):
    ch = ''
    for liste in lignesdemots:
        if len(liste) == 1:
            ch = ch + liste[0] + ' ' * (L - len(liste[0])) + '\n'
        else:
            ch_ligne = liste[0]
            nb_espaces = L - sum([len(mot) for mot in liste])
            espace = (nb_espaces) // (len(liste)-1)
            reste = (nb_espaces) % (len(liste)-1)
            for mot in liste[1:]:
                if reste == 0:
                    ch_ligne = ch_ligne + ' ' * espace + mot
                else:  # pour justifier à droite ?
                    ch_ligne = ch_ligne + ' ' * (espace+1) + mot
                    reste = reste - 1
            ch = ch + ch_ligne + '\n'
            # retour à la ligne à la fin ?
    return ch
```

Remarques sur quelques « problèmes » du corrigé officiel.

Q3 : pas vraiment de type booléen dans les sgbd. Donc on peut accepter `where groman = True` et `where groman = 'True'`, ce dernier étant le seul correct à mon avis.

Q4 : `USING` n'est pas au programme

Q5 : `ORDER BY fnom` et pas `ORDER BY famille.fid`

Q20 : `return curseur` plutôt que `return curseur[0]`

Q21 : La justification de glouton est un peu légère. Il faut préciser quels sont les choix possibles, quel est le choix considéré comme le « meilleur » et dire qu'on applique ce choix à chaque étape.

Q22 : il manque la précision des indices i et j pour chaque ligne.

Q23 : le nom de la fonction et des paramètres ne correspond pas à ce qui est demandé, l'appel récursif n'est pas correct (les fonctions m et c n'existent pas).

Q24 : pour le deuxième cas, la complexité me semble cubique et pas quadratique. (Pour chaque i, on appelle la fonction `cout` avec les paramètres (i, i), (i, i+1), (i, i+2), ... (i, n), donc le calcul de chaque M[i] est quadratique.)

Q26 : fonction incorrecte, les lignes à un seul mot ne sont pas traitées séparément donc division par 0 avec (n-1), la justification à droite n'est pas traitée (cas où le nombre d'espaces à placer n'est pas un multiple de (n-1)).