Informatique Tronc Commun
Concours Blanc (Mines 2023, 2h)
Corrigé

Toute erreur signalée sur le corrigé est susceptible de rapporter des points bonus.

1. $\overline{100}^{16} = 1 \times 16^2 = 256$, donc le montant versé est 2 dollars et 56 cents.
2. Il s'agit du caractère j.
3. ```
   SELECT Count(*)
   FROM Glyphe
   WHERE groman = True
   ```
4. ```
   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
   ```
5. ```
   SELECT fnom, Count(*)
   FROM Famille
   JOIN Police ON Famille.fid = Police.fid
   GROUP BY fnom
   ORDER BY fnom
   ```
6. ```
   def points(v : [[[float]]]) -> [[float]]:
       Lp = []
       for ligne in v:
           for point in ligne:
               Lp.append(point)
       return Lp
   ```

   (le sujet comporte une erreur sur le type de retour de la fonction, on renvoie bien une liste de points, donc une liste de liste de nombres. Ces annotations de type sont purement indicatives).
7. ```
   def dim(l : [[float]], n : int) -> [float]:
       R = []
       for point in l:
           R.append(point[n])
       return R
   ```
8. ```
   def largeur(v : [[[float]]]) -> float:
       Lx = dim(points(v), 0)
       return max(Lx) - min(Lx)
   ```
9. ```
   def obtention_largeur(police : str) -> [float]:
       R = []
       for lettre in "abcdefghijklmnopqrstuvwxyz":
           R.append(largeur(glyphe(lettre, police, True)))
           R.append(largeur(glyphe(lettre, police, False)))
       return R
   ```
10. ```
    def transforme(f : callable, v : [[[float]]]) -> [[[float]]]:
        v2 = []
        for ligne in v:
            v2.append([f(point) for point in ligne])
        return v2
    ```

11. Toutes les abscisses des points sont divisées par 2. Cela a pour effet de contracter les glyphes en divisant leur largeur par 2.

12.
```
def penche( : [[[float]]]) -> [[[float]]]:
    def f(p : [float]) -> [float]:
        return [p[0 + 0.5 * p[1], p[1] ]
    return transforme(f, v)
```

13. Les points encrés sont (0,0), (1,0), (2,1), (3,1), (4,1), (5,2), (6,2)

14. Puisque `dx` est négatif, il n'y a aucune itération dans la boucle `for`, et seul le point (9,8) est encré. On peut prévenir ce comportement en ajoutant en début de fonction `assert p0[0] <= p1[0]`

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

    Le segment tracé n'est donc pas formé de points contigus, et contient des trous. Cela est dû au fait que `dx` $<$ `dy`.

16. On échange les rôles joués par `dx` et `dy` :

```
def trace_quadrant_sud(im:img, p0:(int), p1:(int)):
    assert p0[1] <= p1[1]
    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)
```

17.
```
def trace_segment(im:Image, p0:(int), p1:(int)):
    x0, y0 = p0
    x1, y1 = p1
    dx, dy = x1 - x0, y1 - y0
    if x0 <= x1 and dy <= dx: trace_quadrant_est(im, p0, p1)
    elif x1 <= x0 and -dy <= -dx: trace_quadrant_est(im, p1, p0)
    elif y0 <= y1 and dx <= dy: trace_quadrant_sud(im, p0, p1)
    else: trace_quadrant_sud(im, p1, p0)
```

18.
```
def position(p:(float), pz:(int), taille:int) -> (int):
    x,y = p
    xz,yz = pz
    return (floor(x*taille) + xz, -floor(y*taille) + yz)
```

19.
```
def affiche_car(page:img, c:str, police:str, roman:bool, pz:(int),
taille:int) -> int:
    v = glyphe(c, police, roman)
    for ligne in v:
        x, y = position(ligne[0], pz, taille)
        trace_segment(page, (x,y), (x,y))
        n = len(ligne)
        for k in range(n-1):
            debut = position(ligne[k], pz, taille)
            fin = position(ligne[k+1], pz, taille)
            trace_segment(debut, fin)
    return taille*largeur(v)
```

20.
```
def affiche_mot(page:Image, mot:str, ic:int, police:str, roman:bool,
pz:(int), taille:int) -> (int):
    debut = pz
    for car in mot:
```

```
        largeur = affiche_car(page, car, police, roman, debut, taille)
        debut = debut[0] + largeur + ic, debut[1]
    return (debut{0] - ic, debut[1])
```

21. L'algorithme ajoute les mots à la ligne courante tant que c'est possible, et passe à une nouvelle ligne dès que c'est nécessaire. Il est glouton car cjaque choix est fait localement, et n'est jamais modifié par la suite.

22. (a) La première ligne va des indices $i = 0$ à $j = 2$, on calcule un coût de 0
    La deuxième ligne va des indices $i = 3$ à $j = 3$, on calcule un coût de 16
    La troisème ligne va des indices $i = 4$ à $j = 4$, on calcule un coût de 16

    (b) La première ligne va des indices $i = 0$ à $j = 1$, on calcule un coût de 9
    La deuxième ligne va des indices $i = 2$ à $j = 3$, on calcule un coût de 1
    La troisème ligne va des indices $i = 4$ à $j = 4$, on calcule un coût de 16

    On obtient un total de 32 pour a) et 26 pour b), l'algorithme par programmation dynamique donne donc une solution plus harmonieuse sur cet exemple.

23. (L'énoncé est un peu confus, `m` n'est pas défini et doit être remplacé par `lmots`. Les variables globales `lmots` et `memo` n'ont pas besoin d'être prises en argument par la fonction)

```
memo = {len(lmots) : 0}

def progd_memo(i:int,lmots:[int],L:int,memo:{int:int}) -> int:
    if i in memo: return memo[i]
    else:
        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
```

24. Pour l'algorithme récursif naïf, on a la relation de récurrence

$$C(n) = \sum_{i=1}^{n} (C(n-i) + O(i^2))$$

    En négligeant le terme en $O(i^2)$, on obtient déjà une complexité exponentielle.

    Pour la fonction `prog_bashaut`, on a une complexité en $O(n^3)$, du fait des deux boucles imbriquées et des appels à `cout` en $O(n)$.

    Le calcul de bas en haut est beaucoup plus rapide que le calcul récursif naïf.

25.
```
def lignes(mots:[str],t:[int],L:int) -> [[str]]:
    i = 0
    R = []
    while i<len(t):
        R.append(mots[i:t[i])
        i = t[i]
    return R
```

26.
```
def formatage(lignesdemots:[[str]],L:int) -> str:
    texte = ""
    for ligne in lignesdemots:
        nbLettres = sum([len(mot) for mot in ligne])
        nbEspaces = L -nbLettres
        if len(ligne) == 1:
            texte = texte + ligne[0] + " "*nbEspaces + "\n"
```

```
        else:
            nbCoupures = len(ligne) - 1
            q = nbEspaces // nbCoupures
            r = nbEspaces % nbCoupures
            Espaces = [q+1]*r + [q]*(nbCoupures -r)
            lignef = ""
            for i in range(nbCoupures):
                lignef = lignef + ligne[i] + " "*Espaces[i}
            lignef = lignef + ligne[nbCoupures] + "\n"
            texte = texte + lignef
    return texte
```