teaching sciences for innovation

# Proposition de corrigé

Concours : Concours Commun INP

Année : 2019

Filière : TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

## A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

## Licence et Copyright



## Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

# IPT CCINP TSI 2019 | Sécurisation de l'entrée du personnel d'une entreprise

☐ **Q1.** Avec ce type de codage, l'entreprise pourrait gérer un total de $7^7$ employés par site et un nombre maximal de site de $2^3 = 8$.
Cette technique de codage est largement suffisante car chaque site de production ne peut compter que 100 ($< 7^7$ ) employés et le nombre de site est de 5 ($< 8$).

☐ **Q2.** On peut procéder de la façon suivante pour transformer `s='Auttame'` en `'Automne'` :

- suppression : `s = s[:2] + s[3:]` ainsi `s` devient `'Autame'` ;
- insertion : `s = s[:5] + 'n' + s[5:]` ainsi `s` devient `'Autamne'` ;
- substitution : `s = s[:3] + 'o' + s[4:]` ainsi `s` devient `'Automne'`.

La distance de Levenshtein est alors égale à 3.

☐ **Q3.** On obtient les matrices suivantes ligne 14 : $d = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{pmatrix}$ et ligne 26 : $d = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 2 & 1 \\ 3 & 3 & 2 \end{pmatrix}$

La distance de Levenshtein vaut donc 2.

☐ **Q4.** La complexité est en $O(n \cdot m)$. Elle provient des deux boucles imbriquées lignes 15 et 16.

☐ **Q5.** On propose l'implémentation suivante :

```
def code_bon_lvs(t) :
    n = len(t)
    for i in range(n-1) :
        for j in range(i+1, n) :
            lvs = levenshtein(t[i], t[j])
            if lvs <= 3 :
                t[j] = '0'*7
```

☐ **Q6.** On propose l'implémentation suivante :

```
def pourcentage_lvs(t):
    cpt = 0
    n = len(t)
    for i in range(n) :
        if t[i] == '0'*7 :
            cpt += 1
    val = str(int(cpt/n*100)) + '%'
    return val
```

☐ **Q7.** On propose l'implémentation suivante :

```
def liste_site(valeur):
    res = []
    if valeur > 2**5-1:
        return res
    else :
        val_b = bin(valeur)
```

```
        for k in range(len(val_b)-2) :
            if val_b[2+k] == '1':
                res += [k+1]
        return res
```

☐ **Q8.** On propose la requête SQL suivante :

```
SELECT nom, prenom, age FROM Employes WHERE age > 50;
```

☐ **Q9.** On propose la requête SQL suivante :

```
SELECT email FROM Employes WHERE site = 12;
```

☐ **Q10.** On propose la requête SQL suivante :

```
SELECT nom,categorie FROM Employes JOIN ListeCategories ON id = code_categorie
WHERE age >= 20 ORDER BY nom ASC
```

☐ **Q11.** On propose la requête SQL suivante :

```
SELECT age,COUNT(*) as nb FROM Employes GROUP BY age ORDER BY nb DESC
```

☐ **Q12.** On définit le code RGB pour

- un pixel bleu : $(0, 0, 255)_{10}$ soit $(0,0,FF)_{16}$ ;
- un pixel blanc : $(255, 255, 255)_{10}$ soit $(FF,FF,FF)_{16}$.

Le code $(10, 10, 10)_{10}$ correspond à un gris très foncé (presque noir).

☐ **Q13.** Pour chaque composante R, G ou B on a $2^8 = 256$ possibilités ; pour un code RGB on a donc $256^3$ possibilités.

☐ **Q14.** On propose les deux implémentations suivantes, l'une faisant appel à la fonction `bin()` et l'autre non :

```
def bin2(n):
    res = bin(n)
    res = res[2:]
    while len(res) < 3:
        res = '0' + res
    return res
```

```
def bin2(n):
    res = ''
    while n != 0:
        q = n // 2
        r = n % 2
        res = str(r) + res
        n = q
    while len(res) < 3 :
        res = '0' + res
    return res
```

☐ **Q15.** On propose l'implémentation suivante :

```
def num(ch):
    if len(ch) == 0:
        return -1
    else :
        n = len(ch)
        res = 0
        for k in range(1,n+1):
            res += int(ch[-k])*2**(k-1)
        return res
```

☐ **Q16.** On propose les deux implémentations suivantes :

```
def lettre(ch) :
    lettres = 'ABCDEFG'
    ind = num(ch)
    return lettres[ind-1]
```

```
def lettre(ch) :
    code = num(ch)
    return chr(code + 64)
```

☐ **Q17.** On propose l'implémentation suivante :

```
def bpf(im,posi):
    px = im.getpixel(posi)
    bpf_r = bin2(px[0])[-1]
    bpf_g = bin2(px[1])[-1]
    bpf_b = bin2(px[2])[-1]
    return bpf_r + bpf_g + bpf_b
```

☐ **Q18.** La fonction `valeur_delta()` détermine la valeur de l'intervalle delta entre chaque bloc.
La fonction additionne les 3 valeurs des composantes R, G et B du pixel traité et renvoie une valeur toujours comprise entre 40 et 128 (incluses). Dans le cas où elle serait impaire on ajoute 1 au résultat donné, d'où `inter` $\in [40, 42, ..., 126, 128]$.

☐ **Q19.** On propose l'implémentation suivante :

```
def lecture_lettres(im, posi):
    imx, imy = im.size
    delta = valeur_delta(im,posi)
    code = ''
    for k in range(1,8) :
        pos_bloc = position_bloc(posi,delta,imx,k)
        bpf_bloc = bpf(im,pos_bloc)
        lettre_bloc = lettre(bpf_bloc)
        code += lettre_bloc
    return code
```

☐ **Q20.** Cette fonction affiche le numéro du site suivi du code employé.