► **Q1.** Sur 3 bits, on peut coder les entiers de 0 à 7 d'où 8 sites possibles.
Avec 7 caractères parmi 7 possible, il y a 7^7 possibilités (ou 7! si l'on comprend l'énoncé en les croyant tous distincts).
Cela fait donc 7^7 employés possible par site.
Puisqu'il n'y a que 5 sites et au plus 100 employés par site, cette technique est suffisante.

► **Q2.** Une chaîne de caractère étant un objet non mutable, ces opérations demandent des concaténations de listes (par exemple, une suppression de l'élément en position 0 demande de recopier tous les éléments à partir de celui en position 1). Ce sont donc des opérations qui ont une complexité linéaire en la longueur de la chaîne de caractères.

► **Q3.** On peut par exemple effectuer :

```
>>> s = 'Auttame'
>>> s = s[:3]+s[4:] # suppression
>>> s
'Autame'
>>> s = s[:3] + "o" + s[4:] # substitution
>>> s
'Autome'
>>> s = s[:5] + "n" + s[5:] # insertion
>>> s
'Automne'
```

Puisque l'on a effectué 3 opérations, la distance de Levenshtein est au plus 3.
Par ailleurs, il faut faire disparaître un `t` et le `a` avant le `m` et faire apparaître un `n` après donc on doit faire au moins trois opérations.
Donc la distance de Levenshtein entre ces deux chaînes est 3.

► **Q4.** Tout d'abord, on a $d(0,0)=0$ puisqu'il n'y a rien à faire pour passer d'une chaîne vide à une chaîne vide.
Pour passer d'une chaîne à $i \in [\![1,n]\!]$ caractères à une chaîne vide, il y a i suppressions de caractères à effectuer et cette valeur est clairement minimale donc :

$$\forall i \in [\![1,n]\!],\ d(i,0)=i.$$

Enfin, pour passer d'une chaîne vide à une chaîne à $j \in [\![1,p]\!]$ caractères, il y a j insertions de caractères à effectuer et cette valeur est clairement minimale donc :

$$\forall j \in [\![1,n]\!],\ d(0,j)=j.$$

► **Q5.** Soit $i \geqslant 1$ et $j \geqslant 1$. Si `a[i-1]` et `b[j-1]` sont égaux alors les modifications concernant les préfixes `a[ :i]` et `b[ :j]` sont les mêmes que celles concernant les préfixes `a[ :i-1]` et `b[ :j-1]` donc :

$$d(i,j)=d(i-1,j-1).$$

► **Q6.** Supposons maintenant `a[i-1]` et `b[j-1]` différents.
Pour transformer le mot $a_0\cdots a_{i-1}$ en $b_0\cdots b_{j-1}$, on peut

- ◇ transformer $a_0\cdots a_{i-2}$ en $b_0\cdots b_{j-1}$, puis supprimer a_{i-1}, pour un coût minimal de $d(i-1,j)+1$;
- ◇ transformer $a_0\cdots a_{i-1}$ en $b_0\cdots b_{j-2}$, puis ajouter b_{j-1}, pour un coût minimal de $d(i,j-1)+1$;
- ◇ transformer $a_0\cdots a_{i-2}$ en $b_0\cdots b_{j-2}$, puis remplacer la lettre a_{i-1} par la lettre b_{j-1}, pour un coût minimal de $d(i-1,j-1)+1$.

Comme on cherche à minimiser le coût, on a :

$$d(i,j)=1+\min\{d(i-1,j)\ ;\ d(i,j-1)\ ;\ d(i-1,j-1)\}.$$

► **Q7.** On peut par exemple écrire :

```
def mini(a, b, c):
    if a <= b and a <= c:
        return a
    if b <= a and b <= c:
        return b
    return c
```

► **Q8.**

```
def distance(a, b):
    """ entrée : a et b deux chaînes de caractères
        sortie : entier correspondant à la distance de Levenshtein entre a et b
    """
    n, p = len(a), len(b)
    d = [[0 for j in range(p+1)] for i in range(n+1)]
    for i in range(n+1):
        d[i][0] = i
    for j in range(p+1):
        d[0][j] = j
    for i in range(1, n+1):
        for j in range(1, p+1):
            if a[i-1] == b[j-1]:
                d[i][j] = d[i-1][j-1]
            else:
                d[i][j] = 1 + mini(d[i-1][j], d[i][j-1], d[i-1][j-1])
    return d[n][p]
```

► **Q9.** Les deux premières boucles ont respectivement $n+1$ et $p+1$ itérations avec à chaque fois une affectation.
La double boucle qui suit a $(n+1)\times(p+1)$ itérations et à chaque itérations, il y a un test et :

— soit une affectation ;

— soit au plus 4 comparaisons (dans `mini`) et une affectation.

La complexité est donc en $\boxed{O(np) \text{ où } n \text{ et } p \text{ sont les longueurs de deux chaînes de caractères.}}$

► **Q10.**

```
def code_bon_lvs(t):
    n = len(t)
    code_bad = [False for k in range(n)]  # si True alors il faudra mettre à 0
    for i in range(n):
        for j in range(i+1, n):
            if distance(t[i], t[j]) <= 3:
                t[i] = True
                t[j] = True
    for k in range(n):
        if code_bad[k]:
            t[k] = '000000'
```

Il suffit alors d'utiliser l'instruction :

```
code_bon_lvs(table_code)
```

► **Q11.**

```
def pourcentage_lvs(t):
    c = 0
    for x in t:
        if x == '0000000':
            c += 1
    return c/len(t)*100
```

Il suffit alors d'utiliser l'instruction :

```
pourcentage_lvs(table_code)
```

► **Q12.** On pourrait utiliser la fonction `bin` pour convertir l'entier `s` en base de 2 mais on peut s'en dispenser :

```
def liste_site(s):
    """ entrée : entier s représentant le code des sites autorisés
        sortie : liste des numéros des sites autorisés
                    ou "" si s ne correspond pas à une donnée cohérente
        On doit avoir 0 <= s <= 30 (car le site d'origine n'est pas compris)
    """
    if s == 31:
        return ""
    L = []
    for k in range(5):
        if s%2 == 1:
            L.append(k+1)
        s = s//2
    if s == 0:
        return L
    else:
        return ""
```

► **Q13.**

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

► **Q14.** L'accès exclusif aux sites 3 et 4 (en plus du site d'origine) se traduit par une valeur de `site` à 12.

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

► **Q15.** L'accès au site 2 (en plus du site d'origine) se traduit par une valeur de `site` à 2 (site 2), 3 (sites 1 et 2), 6 (sites 3 et 2), 10 (sites 4 et 2), 18 (sites 5 et 2), 7 (sites 1, 3 et 2), 11 (sites 1, 4 et 2), 19 (sites 1, 5 et 2), 14 (sites 3, 4 et 2), 22 (sites 3, 5 et 2), 26 (sites 4, 5 et 2), 15 (sites 1, 3, 4 et 2), 23 (sites 1, 3, 5 et 2), 27 (sites 1, 4, 5 et 2) ou 30 (sites 3, 4, 5 et 2).

```
SELECT COUNT(*) FROM Employes
  WHERE site = 2 OR site = 3 OR site = 6 OR site = 10 OR site = 18
                 OR site = 7 OR site = 11 OR site = 19 OR site = 14 OR site = 22 OR site = 26
                 OR site = 15 OR site = 23 OR site = 27 OR site = 30
```

► **Q16.**

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

► **Q17.**

```
SELECT ListeCategories.categorie, COUNT(*)
  FROM Employes JOIN ListeCategories ON Employes.code_categorie = ListeCategories.id
  WHERE Employes.age > 40
  GROUP BY ListeCategories.categorie
```

► **Q18.**

```
SELECT ListeCategories.categorie, COUNT(*) AS nb
  FROM Employes JOIN ListeCategories ON Employes.code_categorie = ListeCategories.id
  WHERE Employes.age > 40
  GROUP BY ListeCategories.categorie
  ORDER BY nb DESC
  LIMIT 2
```

► **Q19.**

```
SELECT ListeCategories.categorie, COUNT(*) AS nb
  FROM Employes JOIN ListeCategories ON Employes.code_categorie = ListeCategories.id
  WHERE Employes.age < 25
  GROUP BY ListeCategories.categorie
  HAVING nb > 2
```