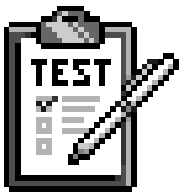


Correction du DS n° 1 d'informatique

Vendredi 16 octobre 2026

T. Lemanissier

PC lycée Carnot



1. `id_facture` est une clé primaire de la table facture et `id_client` et `id_jeu` sont des clés primaire respectivement de la table `Clients` et de la table `Jeux`, ces deux clés sont donc des clés étrangaires de la table `Factures`.

2.
```
SELECT nom, prenom, email
FROM Clients
WHERE anniversaire = "03-07" ;
```

3.
```
SELECT c.nom, c.prenom, f.date_achat
FROM clients AS c JOIN factures AS f
ON c.id_client = f.id_client
WHERE email = "myriam.sarr\@ccinp.edu" ;
```

4.
```
SELECT j.nom_jeu
FROM jeux AS j JOIN records AS r JOIN clients AS c
ON j.id_jeu = r.id_jeu AND r.id_client = c.id_client
WHERE c.nom = "Dupont" AND c.prenom = "Martin" AND date_record LIKE "%2020" ;
```

5.
```
SELECT j.nom_jeu, MAX(score)
FROM jeux AS j JOIN records AS r
ON j.id_jeu = r.id_jeu
GROUP BY r.id_jeu ;
```

6.
```
SELECT c.nom, c.emil, SUM(j.prix) AS total
FROM clients AS c JOIN factures AS f JOIN jeux AS j
ON c.id_client = f.id_client AND f.id_jeu = j.id_jeu
GROUP BY c.id_client ;
HAVING total >= 200
ORDER BY total DESC ;
```

7.
```
SELECT c.nom, c.prenom, j.nom_jeu, r1.score,  r2.score_max
FROM clients AS c
JOIN jeux AS j
JOIN records AS r1
JOIN (SELECT id_jeu, MAX(score) AS score_max
        FROM records
        GROUP BY id_jeu) AS r2
ON c.id_client = r.id_client AND r.id_jeu = j.id_jeu AND r2.id_jeu = r1.id_jeu ;
```

8. La dame à ligne 1 peut prendre la dame à la ligne 2.

9. Pour la configuration gagnante, la liste est `[1, 3, 0, 2]`, pour la configuration perdante, la liste est `[1, 3, 2, 0]`.

10.
```
def ligne(n) :
    return [0]*n
```

11.
```
def grille(n) :
    G = []
    for i in range(n) :
        G.append(ligne(n))
    return G
```

12.
```
def remplir(L) :
    n = len(L)
    G = grille(n)
    for i in range(n) :
        G[i][L[i]] = 1
    return G
```

La remplissage est bien en `O(len(L))` puisque l'on fait exactement `len(L)` itération de boucle et que chaque itération est en `O(1)` cependant la création de la liste de listes est en $O(\texttt{len(L)}^2)$.

13.
```
def affiche(G) :
    for l in G :
        print(l)
```

14. La condition (a) signifie que toute les dames sont sur des colonnes distinctes. Les conditions (b) et (c) signifient que les dames ne sont sur des diagonales respectivement montantes et descendantes distinctes.

15.
```
def tous_distincts(L) :
    presents = {}
    for x in L :
        if x not in presents :
            presents[x] = True
        else :
            return False
    return True
```

16.
```
def tous_distincts2(L) :
    n = len(L)
    for i in range(n) :
        for j in range(i) :
            if L[i] == L[j] :
                return False
    return True
```

Pour les deux fonctions, le pire des cas arrive lorsque toutes les valeurs de la liste sont distinctes.
La fonctino `tous_distincts` effectue n teste d'appartenance au dictionnaire `presents` et n ajout de clé dans ce même dictionnaire. Donc cette fonction a une complexité temporelle égale à `O(len(L))`.

Dans ce cas, la fonction `tous_distincts2` effectue $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ opération. Donc cette fonction a une complexité temporelle égale à $O(\texttt{len(L)}^2)$.

17.
```
def est_possible(L) :
    n = len(L)
    if not tous_distincts(L) :
        return False
    elif not tous_distincts([L[i]+i for i in range(n)]) :
        return False
    elif not tous_distincts([L[i]-i for i in range(n)]) :
        return False
    else :
        return True
```

18.
```
def est_solution(L, n) :
    return est_possible(L) and len(L) = n
```

19. Méthode 1 :

```
import random as rd
rd.sample(L, len(L))
```

Méthode 2 :

```
from random import sample
sample(L, len(L))
```

Méthode 1 :

```
import random
random.sample(L, len(L))
```

20.
```
def recherche_aleatoire(n, N) :
    L = [k for k in range(n)]
    for i in range(N) :
        Lper = sample(L, n)
        if est_solution(L, n) :
            return Lper
    return []
```

21. Méthode 1 :

```
def insertion(x, L, i) :
    return L[:i]+[x]+L[i:]
```

Méthode 2 :

```
def insertion(x, L, i) :
    M = []
    for j in range(len(L)) :
        if j == i :
            M.append(x)
        M.append(L[i])
    return M
```

22. Après le premier passage de la première boucle, on a :

`P = [[2, 0, 1], [0, 2, 1], [0, 1, 2]]`

Après l'exécution du code au complet, on a :

`P = [[2, 0, 1], [0, 2, 1], [0, 1, 2], [2, 1, 0], [1, 2, 0], [1, 0, 2]]`

Le code prend toute les listes de la liste `L` et stocke dans `P` toute les listes obtenues en insérant dans ces listes 2 à toute les places possibles. Au final, `P` contient toutes les permutation possibles de la liste `[0, 1, 2]`.

23.
```
def permutations(n) :
    L = [[0, 1], [0, 1]]
    for v in range(n) :
        P = []
        for l1 in L :
            for i in range(v+1) :
                p1 = insertion(v, l1, i)
                P.append(p1)
        L = [p for p in P]
    return L
```

24.
```
def configurations(n) :
    L = permutations(n)
    P = []
    for l in L :
        if est_solution(l, n) :
            P.append(l)
    return P
```

25. Pour la symétrie horizontale, on obtient :

`L1 = [2, 4, 1, 3, 0]`

Pour la rotation d'un quat de tour vers la droite, on obtient :

`L2 = [3, 1, 4, 2, 0]`

La liste `L1` est obtenue en renversant la liste.

26.
```
def symetrie(L) :
    n = len(L)
    M = []
    for i in range(n) :
        M.append(L[n-1-i])
    return M
```

27.
```
def rotation(L) :
    n = len(L)
    M = [0]*n
    for i in range(n) :
        M[L[i]] = n-1-i
    return M
```

28. La ligne 2 permet d'obtenir toute les configurations possible pour un `n` donné.
La variable `P` est une liste de listes de listes et chaque liste dans `P` contient la liste des configuration valides qui sont semblables à la première de cette liste.
La fonction `partition` génère toutes les configurations possible ensuite, elle crée la liste vide P et pour chaque configuration `c` elle teste si cette configuration à similaires à une configuration déjà rencontré auquel cas elle l'ajoute à cette liste de configurations. Si elle n'est semblable à aucune, alors c'est une nouvelle configuration et donc la fonction ajouter une nouvelle liste de configuration à `P`.

29.
```
def explore(L, n) :
    if not est_possible(L) :
        return -1
    if len(L) == n :
        return L
    for i in range(n) :
        res = explore(L + [i], n)
        if res != -1 :
            return res
    return -1
```

30.
```
def explore2(L, n) :
    if not est_possible(L) :
        return 0
    if len(L) == n :
        return 1
    nb_sol = 0
    for i in range(n) :
        nb_sol = nb_sol + explore2(L+[i], n)
    return nb_sol
```