Concours Commun INP (CCINP), France

Session 2026, filière TPC

Informatique

Durée : 3 heures

Ce que traite le sujet

- Partie I.1 à I.3 : codage, affichage et validité
- Partie I.4 et I.5 : recherche aléatoire et permutations
- Partie I.6 et I.7 : configurations similaires et graphe des coups
- Partie II : base de données d'un serveur de jeu en ligne

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/tpc/2026/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

ÉPREUVE SPÉCIFIQUE - FILIÈRE TPC

INFORMATIQUE

Durée : 3 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- *Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, bleu clair ou turquoise, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.*
- *Ne pas utiliser de correcteur.*
- *Écrire le mot FIN à la fin de votre composition.*

Les calculatrices sont interdites.

Le sujet est composé de deux parties indépendantes et de deux annexes.

Important : pour chaque question, vous pouvez utiliser les fonctions des questions précédentes même si vous ne les avez pas toutes implémentées.

Seul le Document Réponse (DR) doit être rendu dans son intégralité (le QR Code doit être collé sur la première page du Document Réponse).

A

Autour de jeux

Partie I - Des algorithmes pour résoudre le problème des `n` reines

Organisation de la partie I : les différentes sous-parties sont très largement indépendantes entre elles. On signale toutefois que :

- les **sous-parties I.4** et **I.5** nécessitent d'admettre la question **Q11** si elle n'a pas été traitée ;
- la **sous-partie I.6** nécessite d'admettre la question **Q17** si elle n'a pas été traitée ;
- la **sous-partie I.7** nécessite d'admettre la question **Q10** si elle n'a pas été traitée.

Quelques rappels de syntaxe Python figurent dans l'**annexe 2**.

Les jeux de société (échecs, dames, reversi, awale, etc.) sont une grande source d'inspiration pour tester l'efficacité de différents algorithmes.
Nous nous proposons ici d'en étudier plusieurs autour d'un défi issu du jeu d'échecs.

Considérons une grille de taille $n \times n$. Une reine est une pièce du jeu qui peut se déplacer du nombre de cases que l'on souhaite le long d'une ligne, d'une colonne ou d'une diagonale montante ou descendante (**figure 1**).
On cherche à placer `n` reines dans cette grille de telle façon qu'il n'y ait pas plus d'une reine sur chaque ligne, sur chaque colonne et sur chaque diagonale montante ou descendante [1] (**figure 2**).

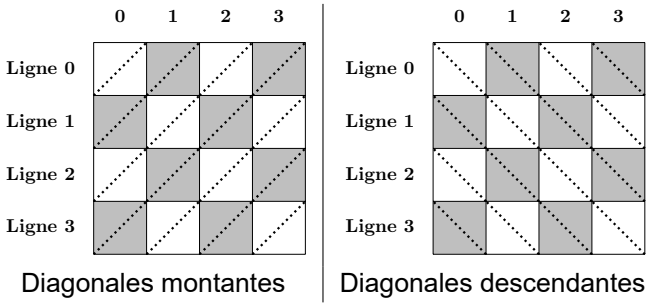


Figure 1 - Diagonales dans le cas `n = 4`

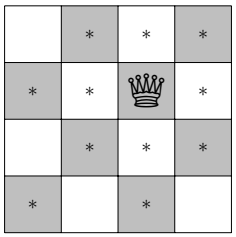


Figure 2 - Une reine a été placée : les cases qu'elle contrôle et qui ne peuvent pas être occupées par d'autres reines sont figurées par le symbole *

I.1 - Introduction

Pour `n = 3` et donc une grille 3×3, le problème n'a pas de solution. Deux reines suffisent à contrôler toutes les cases. Il est donc impossible de placer une troisième reine tout en respectant la consigne. La **figure 3** illustre cette situation dans un cas, les autres s'en déduisant par symétrie ou rotation.

1. Ce problème a été posé en 1848 par Max Bezzel pour un échiquier ordinaire, partiellement résolu par Carl Friedrich Gauss, puis complètement résolu par Franz Nauck en 1850.

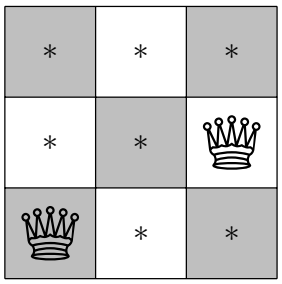


Figure 3 - Cas `n = 3` : deux reines contrôlent toutes les cases

Pour une grille 4×4 (**figure 4**) ainsi que pour des grilles de taille supérieure, il y a toujours une solution au problème étudié.

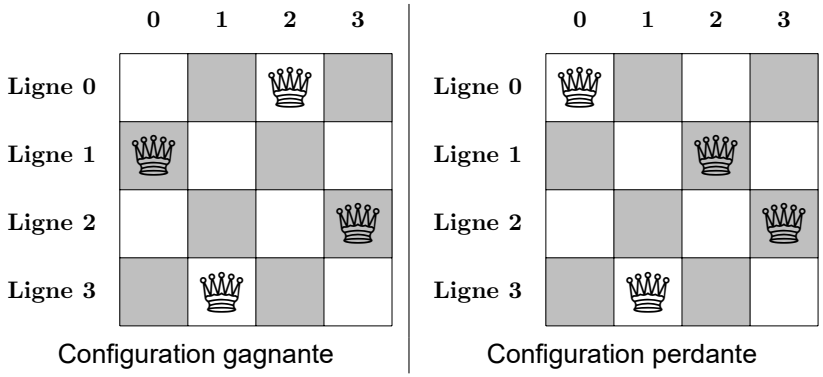


Figure 4 - Deux exemples de configurations pour `n = 4`

Q1. Expliquer pourquoi la configuration perdante de la **figure 4** ne résout pas le problème.

Q2. On décide de coder une configuration par une liste `L` telle que `L[i]` soit égale au numéro de colonne où se trouve la reine de la ligne `i`. Par exemple, la configuration de la **figure 5** sera codée par la liste `L = [2, 4, 1, 3, 0]`.

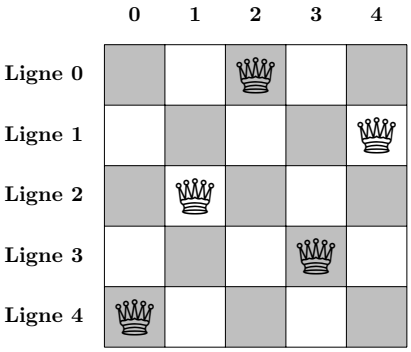


Figure 5 - Un exemple de configuration gagnante pour `n = 5`

Sur le **DR**, donner les listes décrivant les configurations de la **figure 4**.

I.2 - Mise en place d'un affichage

Les solutions que nous obtiendrons dans les sous-parties suivantes seront des listes de `n` positions pour les reines (comme décrit à la **sous-partie I.1**). Pour aider l'utilisateur, nous souhaitons afficher la grille solution trouvée et pour cela nous implémentons quelques fonctions basiques sur les listes Python.

Q3. Écrire une fonction `ligne` prenant pour paramètre un entier naturel `n` et qui renvoie une liste de taille `n` constituée uniquement de `0`.

Par exemple, pour `n = 5`, l'appel `ligne(5)` renverra `[0, 0, 0, 0, 0]`.

Remarque : l'utilisation de structures de données du module `numpy` comme `array` n'est pas autorisée.

Q4. Écrire une fonction `grille` prenant pour paramètre un entier `n` et qui renvoie une liste de `n` listes distinctes, chacune contenant `n` fois l'entier `0`.

Par exemple, pour `n = 3`, l'appel `grille(3)` renverra `[[0, 0, 0], [0, 0, 0], [0, 0, 0]]`.

On considère ainsi que pour une grille `G`, représentée par une liste de listes, `G[i][j]` correspond à la case de la i^{e} ligne et j^{e} colonne.

Q5. Compléter la fonction `remplir` prenant pour paramètre une liste `L` de taille `n` représentant une configuration. Cette fonction crée une grille de taille `n` $\times$ `n` ne contenant que des zéros. Puis, elle remplace `G[i][j]` par la valeur `1` lorsque `L[i] = j`. Enfin, elle renvoie la grille ainsi obtenue.

Remarque : une fonction ayant une complexité temporelle en `O(len(L))` (c'est-à-dire linéaire en la taille de la liste) sera valorisée.

Par exemple, pour `n = 4` :

```
>>> G = remplir([1, 3, 0, 2])
>>> G
[[0, 1, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 0]]
```

Q6. La représentation des grilles produite par l'interpréteur n'est pas visuellement satisfaisante (pour l'utilisateur). Écrire une fonction `affiche` qui prend pour paramètre une grille `G`, qui ne renvoie rien mais affiche la grille `G` ligne par ligne.

Par exemple :

```
>>> G = remplir([1, 3, 0, 2])
>>> G
[[0, 1, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 0]]
>>> affiche(G)
[0, 1, 0, 0]
[0, 0, 0, 1]
[1, 0, 0, 0]
[0, 0, 1, 0]
```

I.3 - Tester si une configuration est valide

Considérons une liste `L` ne contenant que des entiers entre `0` et `n-1` inclus. On admet que cette liste de positions pour les reines résout le problème étudié pour une grille de taille `n` $\times$ `n` si et seulement si :

- (a) tous les `L[i]` sont distincts ;
- (b) tous les `L[i]+i` sont distincts ;
- (c) tous les `L[i]-i` sont distincts ;
- (d) cette liste `L` est de taille `n`.

Q7. Expliquer ce que traduit d'une part le respect de (a) et d'autre part le respect de (b) et (c) pour le problème considéré.

Q8. Écrire une fonction `tous_distincts` prenant en paramètre une liste `L` de positions et renvoyant `True` si tous ses éléments sont distincts, `False` sinon. Cette fonction procède comme suit :

- on initialise un dictionnaire `presents` vide ;
- on parcourt la liste `L`. Si on rencontre un élément `x` non encore présent parmi les clés du dictionnaire, on insère la clé `x` dans le dictionnaire avec comme valeur le booléen `True` (par exemple). Si par contre cet élément est déjà présent, on quitte la fonction en renvoyant le booléen `False` ;
- si le parcours complet de la liste n'a pas repéré de doublon, on renvoie `True`.

Un étudiant propose le code alternatif suivant qui comporte plusieurs erreurs :

```
0 def tous_distincts2(L):
1   n = len(L)
2   for i in range(n):
3     for j in range(i,n):
4       if L[i] == L[j]:
5         return True
6   return False
```

Q9. Écrire une version corrigée de la fonction `tous_distincts2`.

Comparer la complexité temporelle des codes `tous_distincts` et `tous_distincts2` (corrigé) dans le pire des cas, ceci pour une liste `L` de taille `n` donnée en paramètre. On justifiera en s'aidant des rappels sur la complexité donnés en **annexe 2**.

Q10. Écrire une fonction `est_possible` prenant en paramètre une liste `L` de positions et qui renvoie un booléen indiquant si `L` satisfait simultanément les trois conditions (a), (b) et (c), ou pas.

On pourra utiliser plusieurs appels à la fonction `tous_distincts`.

Par exemple, `est_possible([2, 4, 1, 7, 0, 6, 3, 5])` renvoie `True` et `est_possible([3, 4, 1, 7, 0, 6, 2, 5])` renvoie `False`.

Q11. Écrire une fonction `est_solution` prenant en paramètres une liste `L` de positions, un entier `n` représentant la taille de la grille et renvoyant un booléen indiquant si `L` satisfait simultanément les quatre conditions (a), (b), (c) et (d), ou pas.

On part du principe que `L` contient uniquement des entiers de {`0,1,…,n-1`}, il est donc inutile de le vérifier.

Nous disposons maintenant d'une fonction permettant de tester si une liste de positions permet de résoudre le problème pour une grille de taille $n \times n$.

I.4 - Une première approche par recherche aléatoire

Considérons une liste `L` d'éléments distincts. On appelle permutation de cette liste, toute liste de même taille contenant les éléments de `L` dans un ordre éventuellement différent.
Par exemple, si `L = [0, 1, 2]`, les permutations de `L` sont les 6 listes :

`[0, 1, 2]`, `[0, 2, 1]`, `[1, 0, 2]`, `[1, 2, 0]`, `[2, 0, 1]`, `[2, 1, 0]`.

Une première méthode pour trouver une configuration solution du problème des `n` reines consiste à générer aléatoirement des permutations de `[0, 1, …, n-1]`. Chaque permutation représente une configuration possible. Il suffit alors de tester si elle est valide. Si le nombre de permutations testées est grand et que `n` ne l'est pas trop, on peut espérer trouver une solution.

Q12. La fonction `sample` du module `random` peut renvoyer une permutation aléatoire d'une liste `L`. Il y a plusieurs manières d'importer cette fonction. Sur le **DR**, compléter les codes pour utiliser `sample` comme indiqué.

Par la suite, on considèrera avoir importé `sample` avec la méthode 2. Pour générer une permutation aléatoire d'une liste `L`, on saisira alors `sample(L, len(L))`.
Par exemple, si `L = [0, 1, 2, 3, 4]`, `sample(L, 5)` pourra renvoyer `[1, 0, 3, 4, 2]` ou toute autre liste permutation de `L`.

Q13. Écrire une fonction `recherche_aleatoire` prenant en paramètres deux entiers naturels non nuls `n` et `N`.

Cette fonction crée d'abord la liste `L = [0, 1, …, n-1]`. Ensuite, elle crée au maximum `N` permutations de `L` et teste si elles sont solutions du problème des `n` reines.

Dès qu'une liste solution est trouvée, elle est renvoyée. En cas d'échec après `N` essais, la liste vide `[]` est renvoyée.

En prenant `N = 100 000`, on trouve ainsi rapidement des configurations gagnantes jusqu'à `n = 12`.

À titre documentaire, on signale qu'il est possible d'accélérer cette recherche aléatoire en utilisant le fait que lorsqu'une configuration est fausse, l'erreur vient souvent du mauvais placement des premières reines. La **sous-partie I.7** exploitera partiellement cette remarque.

On se propose dans la sous-partie suivante de procéder à une recherche exhaustive des configurations solutions, quitte à se restreindre à des entiers `n` plutôt petits.

I.5 - Déterminer toutes les permutations de `L = [0, 1, …, n-1]`

Q14. Écrire une fonction `insertion` prenant pour paramètres un entier `x`, une liste `L`, un indice `i` et renvoyant une nouvelle liste obtenue en :

- recopiant les termes de la liste `L` d'indices `0` à `i-1` inclus ;
- puis en ajoutant `x` à la position `i` ;
- enfin, en recopiant les termes de la liste `L` depuis l'indice `i` jusqu'à l'indice `len(L)-1` inclus.

On renvoie donc une liste avec un élément de plus que la liste `L` : l'élément `x` inséré à l'indice `i`.

Par exemple, l'appel `insertion(5, [2, 3, 6, 4], 1)` renvoie la liste `[2, 5, 3, 6, 4]` et l'appel `insertion(5, [2, 3, 6, 4], 4)` renvoie la liste `[2, 3, 6, 4, 5]`.

Attention : l'usage d'une quelconque fonction Python faisant cette insertion est bien sûr interdit ici.

On considère maintenant le code suivant :

```
0 L = [[0, 1], [1, 0]]
1 v = 2
2 P = []
3 for l1 in L:
4     for i in range(v+1):
5         p1 = insertion(v, l1, i)
6         P.append(p1)
```

Q15. Quel est le contenu de `P` à l'issue du premier passage dans la première boucle ?

Donner le contenu de `P` après l'exécution complète du code ci-dessus.

(On demande à chaque fois de respecter l'ordre des différents éléments de `P`).

Expliquez exactement ce que fait ce code.

Q16. Compléter le code de la fonction `permutations` prenant en paramètre un entier `n` supérieur ou égal à 2 et renvoyant la liste de toutes les permutations possibles des entiers de `0` à `n-1`. (c'est-à-dire toutes les listes possibles contenant les entiers de `0` à `n-1` une fois et une seule).

Par exemple, `permutations(3)` renverra une liste de sous-listes de la forme suivante (les sous-listes n'étant pas forcément dans cet ordre) :

```
[[0, 1, 2], [0, 2, 1], [2, 0, 1], [2, 1, 0], [1, 0, 2], [1, 2, 0]].
```

Q17. En déduire une fonction `configurations` prenant en paramètre un entier `n` et renvoyant la liste de toutes les configurations satisfaisant les conditions imposées par le jeu.

Avec cette fonction, on peut vérifier qu'il y a 10 configurations possibles pour `n = 5` et 92 configurations possibles pour `n = 8`.

Cependant, on peut constater que certaines configurations sont similaires : elles peuvent se déduire les unes des autres par symétrie ou rotation. La sous-partie suivante détermine les configurations similaires.

I.6 - Classer les configurations similaires

Deux configurations sont dites similaires si l'on peut passer de l'une à l'autre par rotations successives d'un quart de tour, par symétrie axiale, ou par une composition des transformations précédentes.

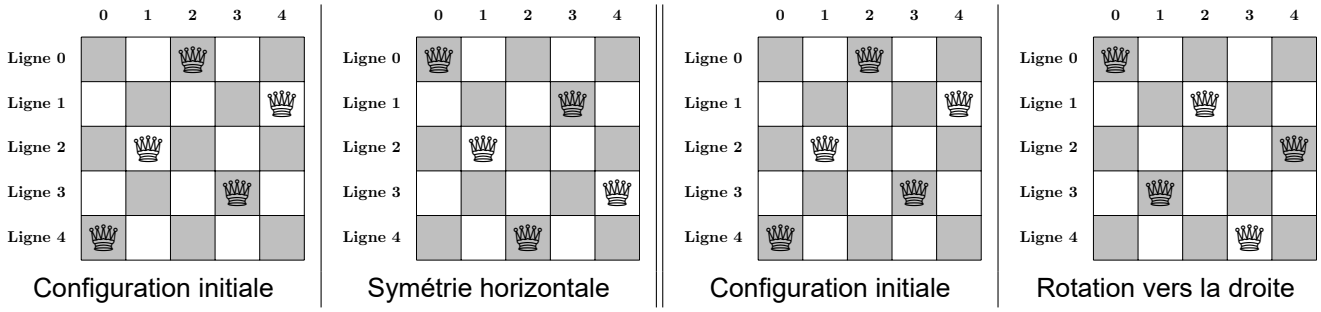


Figure 6 - Un exemple de symétrie horizontale, un exemple de rotation vers la droite

Q18. On se place dans une grille de taille 5×5 et on considère la configuration initiale représentée par la liste `L = [2, 4, 1, 3, 0]` (**figure 6**).

Donner, sous forme d'une liste d'entiers de 0 à 4, la liste `L1` correspondant à une symétrie horizontale de la grille et la liste `L2` correspondant à une rotation d'un quart de tour vers la droite de la grille.

De façon générale, comment obtient-on la liste `L1` correspondant à une symétrie horizontale à partir d'une liste `L` initiale ?

Q19. Écrire une fonction `symetrie` prenant en paramètre une liste `L` représentant une configuration et renvoyant la liste correspondant à une symétrie horizontale de la grille.

Par exemple, cette fonction appliquée à `[1, 0, 2]` renvoie `[2, 0, 1]`.

Attention : l'usage de la méthode `reverse`, de la fonction `reversed` ou du slicing `L[::-1]` est interdit ici.

Q20. Écrire une fonction `rotation` prenant en paramètre une liste `L` représentant une configuration et renvoyant la liste correspondant à la configuration obtenue par rotation d'un quart de tour vers la droite : chaque reine à la position ligne `i`/colonne `L[i]` se retrouve, après rotation, à la position ligne `L[i]`/colonne `n-1-i`.

Par exemple, cette fonction appliquée à `[2, 5, 3, 1, 7, 4, 6, 0]` renvoie `[0, 4, 7, 5, 2, 6, 1, 3]`.

On suppose que l'on dispose d'une fonction `similaires` prenant en paramètres deux listes `L1` et `L2` de taille `n` représentant des configurations. Cette fonction renvoie `True` si `L1` et `L2` sont des configurations similaires, `False` sinon. Cette fonction est basée sur les fonctions `symetrie` et `rotation` mais il n'est pas demandé de l'implémenter.

On donne alors le code d'une fonction `partition` qui prend en paramètre un entier `n`.

```
0 def partition(n):
1     C = configurations(n)
2     P = []
3     for c in C:
4         test = False
5         for elt in P:
6             if similaires(elt[0],c):
7                 elt.append(c)
8                 test = True
9         if test == False:
10            P.append([c])
11    return P
```

Q21. Que fait la ligne 1 de la fonction `partition` ?

Quel est le type de la variable renvoyée par l'appel `partition(n)` ? Décrire complètement la structure de cette variable.

Expliquer en quelques phrases comment fonctionne cette fonction `partition`.

I.7 - Des graphes pour aller plus vite et plus loin

Le test de toutes les permutations possibles fonctionne bien jusqu'à `n = 10`. Au delà, le nombre de permutations à tester est trop important ; on propose ici une autre approche basée sur le parcours du graphe implicite représentant les différents états possibles d'une grille.
La **figure 7** illustre ce point de vue.
La démarche consiste à parcourir (partiellement) ce graphe depuis la situation initiale (une grille vide). On place une reine, on regarde si la configuration obtenue est valide. Si c'est le cas, on place une nouvelle reine jusqu'à obtention éventuelle d'une solution.
En cas d'échec, on revient à l'étape précédente (c'est comme si on enlevait la dernière reine jouée) et on continue avec les autres coups à tester.
Le code mis en œuvre s'inspire ainsi d'un parcours de graphe classique (et récursif).
On note que :

- dès que l'on trouve une solution au problème, on arrête le parcours et on renvoie cette solution ;
- on revient en arrière (c'est-à-dire à l'étape précédente) quand on obtient une grille invalide ;
- en cas de parcours complet du graphe sans avoir trouvé de solution, on renvoie la valeur `-1` ;
- on ne visite *a priori* pas la totalité des sommets du graphe, ce qui est un facteur de rapidité.

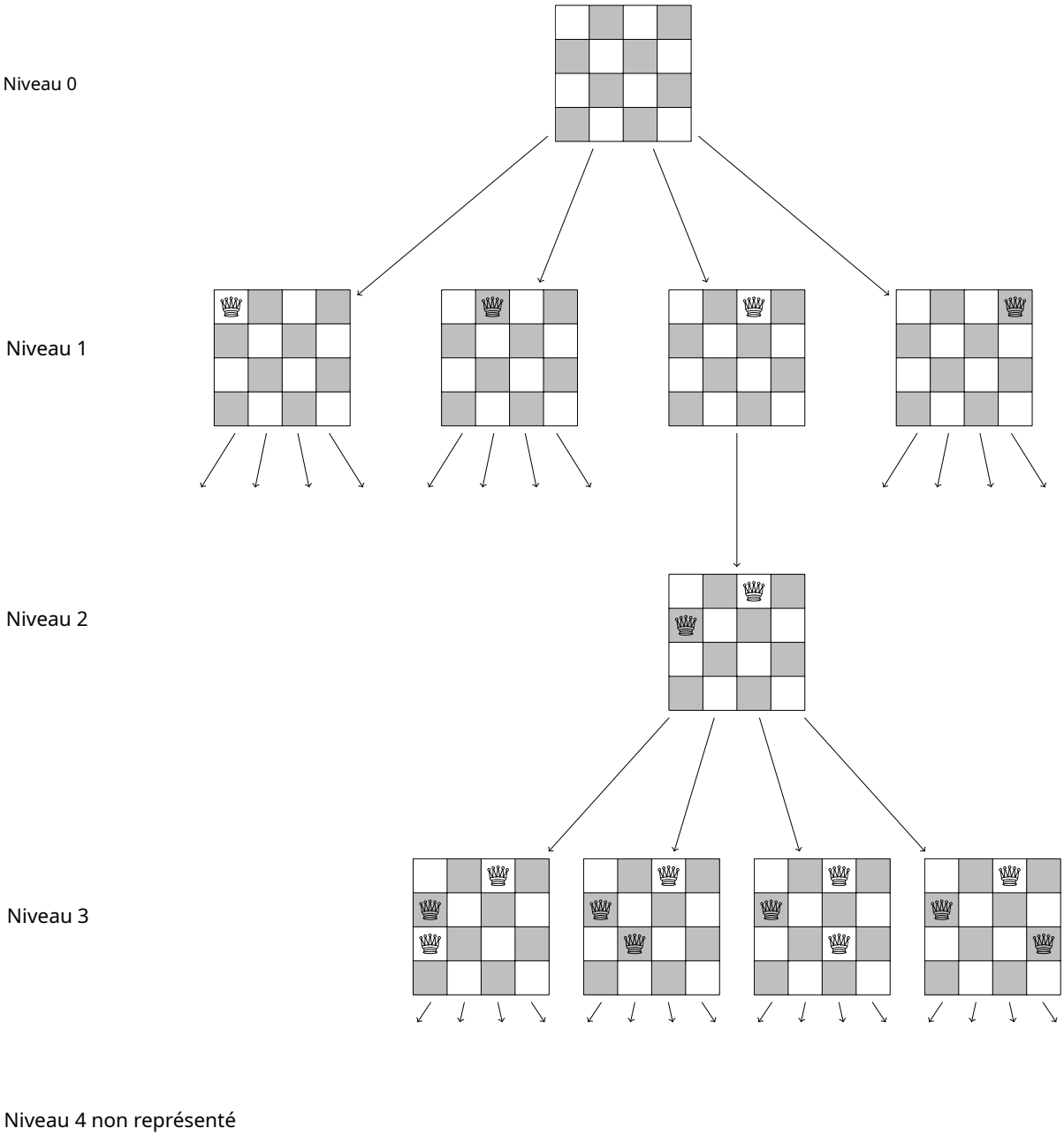


Figure 7 - Un extrait du graphe des coups pour une grille 4×4

Q22. Une fonction `resout_graphe` prend en paramètre un entier `n` $\geqslant 4$ correspondant à la taille de la grille et renvoie une liste gagnante de positions ou -1 si aucune solution n'a été trouvée. Elle utilise une fonction `explore`.
Compléter les zones en pointillés dans la fonction `explore` pour que la fonction `resout_graphe` effectue le travail demandé.

Le code précédent renvoie une solution pour `n = 26` en moins d'une minute sur un ordinateur bas de gamme.

Q23. L'exploration du graphe de la question **Q22** s'arrête dès qu'une solution est trouvée. On peut décider de continuer l'exploration du graphe pour déterminer les autres solutions au problème. On se propose ici de seulement compter le nombre de solutions grâce à une fonction `compte` prenant en paramètre un entier `n` $\geqslant 4$ correspondant à la taille de la grille.
Compléter les zones en pointillés dans la fonction `explore2` pour que la fonction `compte` effectue le travail demandé.

Partie II - Base de données d'un serveur de jeu en ligne

Le gestionnaire d'un site en ligne permettant de jouer à différents jeux a mis au point une base de données correspondant à ses besoins (**annexe 1**). Elle est constituée de 4 tables :

- la table `Clients` contient des informations sur les utilisateurs du site. Elle est constituée des attributs suivants :
  - `id_client` (clé primaire) : identifiant unique du client, nombre entier ;
  - `nom` : nom du client, chaîne de caractères ;
  - `prenom` : prénom du client, chaîne de caractères ;
  - `email` : email du client, chaîne de caractères ;
  - `date_inscription` : date d'inscription du client au format 'JJ-MM-AAAA' ;
  - `anniversaire` : jour anniversaire du client au format 'JJ-MM'.
- la table `Jeux` contient des informations sur les jeux disponibles. Elle est constituée des attributs suivants :
  - `id_jeu` (clé primaire) : identifiant unique du jeu, nombre entier ;
  - `nom_jeu` : nom du jeu, chaîne de caractères ;
  - `description` : brève description du jeu, chaîne de caractères ;
  - `prix` : prix du jeu, nombre flottant.
- la table `Factures` enregistre les transactions des achats de jeux. Elle est constituée des attributs suivants :
  - `id_facture` (clé primaire) : identifiant unique de la facture, nombre entier ;
  - `id_client` (clé étrangère) : référence vers la table `Clients`, nombre entier ;
  - `id_jeu` (clé étrangère) : référence vers la table `Jeux`, nombre entier ;
  - `date_achat` : date d'achat du jeu au format 'JJ-MM-AAAA' ;
  - `montant` : montant de la facture, nombre flottant.
- la table `Records` qui enregistre les records ou scores des différents utilisateurs dans les jeux qu'ils ont achetés. Elle est constituée des attributs suivants :
  - `id_record` (clé primaire) : identifiant unique du record, nombre entier ;
  - `id_client` (clé étrangère) : référence vers la table `Clients`, entier ;
  - `id_jeu` (clé étrangère) : référence vers la table `Jeux`, entier ;
  - `score` : score du joueur pour ce jeu, nombre entier ;
  - `date_record` : date du record au format 'JJ-MM-AAAA'.

Q24. Le gestionnaire veut souhaiter leur anniversaire à ses clients et leur proposer un bon d'achat à cette occasion. Écrire une requête `SQL` permettant de récupérer le nom, le prénom et l'email des clients ayant leur anniversaire le 3 juillet.

Q25. Le gestionnaire souhaite faire le récapitulatif des achats d'un client particulier. Écrire une requête `SQL` affichant le nom, le prénom et les dates des achats effectués par le client ayant pour email 'myriam.sarr@ccinp.edu'.

Q26. Le gestionnaire du site veut pouvoir identifier ses meilleurs clients. Écrire une requête `SQL` affichant le nom, l'email et le montant total dépensé pour chaque client du site.
Attention, un client peut avoir fait plusieurs achats, il faudra donc faire la somme de tous ses achats.

Q27. Le gestionnaire du site veut mettre à l'honneur les meilleurs scores pour chaque jeu, ceci sur la page d'accueil du site. Écrire une requête `SQL` affichant chaque nom de jeu avec le meilleur score pour ce jeu.

ANNEXE 1 - Base de données d'un site de jeux

Table Clients

id_client	nom	prenom	email	date_inscription	anniversaire
1	'Dupont'	'Jean'	'jean.dupont@enscm.com'	'15-01-2023'	'03-07'
2	'Martin'	'Sophie'	'sophie.martin@sccp.fr'	'22-05'2024'	'30-11'
3	'Cruise'	'Tom'	'tom.cruise@ccinp.edu'	'22-05-2024'	'03-07'
...	...	...	...	...	...

Table Jeux

id_jeu	nom_jeu	description	prix
1	'Pac Man'	'Mangez sans être mangé !'	0.99
2	'Space Invaders'	'Protégez la galaxie des envahisseurs'	1.99
...	...	...	...

Table Factures

id_facture	id_client	id_jeu	date_achat	montant
1	1	1	'15-06-2023'	2.99
2	2	2	'16-06-2023'	1.99
...	...	...	...	...

Table Records

id_record	id_client	id_jeu	score	date_record
1	1	1	1250	'20-06-2024'
2	327	8	2000	'21-06-2025'
...	...	...	...	...

ANNEXE 2 - Rappels de syntaxe Python

Création d'un dictionnaire vide	`dico = dict()`	Complexité `O(1)`
Nombre de clés d'un dictionnaire	`len(dico)`	Complexité `O(1)`
Test d'appartenance d'une clé à un dictionnaire	`x in dico`	Complexité `O(1)`
Création d'une liste vide	`L = []`	Complexité `O(1)`
Taille d'une liste	`len(L)`	Complexité `O(1)`
Test d'appartenance à une liste	`x in L`	Complexité `O(len(L))`

Ajouter un élément à la fin d'une liste	`>>> L = [1,2,3]`
	`>>> L`
	`[1,2,3]`
	`>>> L.append(5)`
	`>>> L`
	`[1,2,3,5]`
		Complexité `O(1)`

Supprimer et renvoyer le dernier élément d'une liste	`>>> L = [1,2,3]`
	`>>> a = L.pop()`
	`>>> a`
	`3`
	`>>> L`
	`[1,2]`
		Complexité `O(1)`

Affichage d'une variable x et retour à la ligne	`print(x)`	Complexité variable

FIN

IMPRIMERIE NATIONALE – 26 1031 – D'après documents fournis

CONCOURS COMMUN

Numéro d'inscription

Numéro de table

Né(e) le

Nom : _____

Prénom : _____

Emplacement QR Code

Filière : TPC

Session : 2026

Épreuve de : Informatique

Consignes

- *Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer*
- *Rédiger avec un stylo non effaçable bleu ou noir*
- *Ne rien écrire dans les marges (gauche et droite)*
- *Numéroter chaque page (cadre en bas à droite)*
- *Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre*

DOCUMENT RÉPONSE

Ce Document Réponse doit être rendu dans son intégralité.

Q1. Expliquer pourquoi la configuration perdante de la **figure 4** ne résout pas le problème.

Q2. Donner les listes décrivant les configurations de la **figure 4**.

Configuration gagnante :

Configuration perdante :

Q3. Fonction `ligne`.

NE RIEN ÉCRIRE DANS CE CADRE

Q4. Fonction `grille`.

Q5. Fonction `remplir`.

```
def remplir(L):
    # On récupère la taille de L
    n =
    # On crée la liste de listes G
    G =
    # On remplit G à l'aide de L
    for i in range(n):









    return
```

Q6. Fonction `affiche`.

Q7. Interprétations.

Qu'implique le respect de (a) ?

Qu'implique le respect de (b) et (c) ?

Q8. Fonction `tous_distincts`.

Q9. Version corrigée du code de la fonction `tous_distincts2`.

Comparaison de la complexité temporelle des deux codes.

Q10. Fonction `est_possible`.

Q11. Fonction `est_solution`.

CONCOURS COMMUN

Numéro d'inscription

Numéro de table

Né(e) le

Nom :

Prénom :

Emplacement QR Code

Filière : TPC

Session : 2026

Épreuve de : Informatique

Consignes

- *Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer*
- *Rédiger avec un stylo non effaçable bleu ou noir*
- *Ne rien écrire dans les marges (gauche et droite)*
- *Numéroter chaque page (cadre en bas à droite)*
- *Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre*

Q12. Différentes manières d'importer `sample`.

Méthode 1 :

.....
```
rd.sample(L, len(L))
```

Méthode 2 :

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

Méthode 3 :

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

Q13. Fonction `recherche_aleatoire`.

NE RIEN ÉCRIRE DANS CE CADRE

Q14. Fonction `insertion`.

Q15. Compréhension d'un code.

Contenu de `P` à l'issue du premier passage dans la première boucle.

Contenu de `P` après l'exécution complète du code.

Que fait exactement le code ?

Q16. Fonction `permutations`.

```
def permutations(n) :
    L = [[0]]
    for v in range(1, n) :
        P =
        for l1 in L :




        L = P
    return L
```

Q17. Fonction `configurations`.

Q18. Listes obtenues pour la symétrie horizontale et la rotation à droite.

L1 =

L2 =

Comment obtient-on en général la liste correspondant à une symétrie horizontale ?

Q19. Fonction `symetrie`.

Q20. Fonction `rotation`.

Q21. Compréhension d'un code.

Que fait la ligne 1 de la fonction `partition` ?

Quel est le type de la variable renvoyée par l'appel `partition(n)` ?

Décrire complètement la structure de cette variable.

Expliquer en quelques phrases comment fonctionne cette fonction `partition`.

Q22. Fonction `resout_graphe`.

```
def explore(L, n):
        if not est_possible(L):
            return ..........
        if len(L) == ..........: # si fin de recherche victorieuse
            return L
        for i in range(n):
            res = explore(L + [i], n)
            if res != -1: # si une solution a été trouvée
                return res
        return ..........

def resout_graphe(n):
    rep = explore([], n)
    return rep
```

NE RIEN ÉCRIRE DANS CE CADRE

Q23. Fonction `compte`.

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

def compte(n):
    rep = explore2([], n)
    return rep
```

Q24. Requête `SQL`.

Q25. Requête SQL.

Q26. Requête SQL.

Q27. Requête SQL.

FIN

Travailler les chapitres de ce sujet

Prepa Help suit le programme de TPC chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Bases de Python : expressions, types, instructions, fonctions
- Récursivité

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/ccinp/tpc