



**RÉGION ACADÉMIQUE
LA RÉUNION**

*Liberté
Égalité
Fraternité*



Olympiades d'informatique de l'Océan Indien

**Numérique et Sciences Informatiques (NSI)
classes de première**

Académie de La Réunion - 26 mars 2025

Durée de l'épreuve : 3 heures

Cette épreuve individuelle se déroule sur poste informatique
déconnecté du réseau local de l'établissement et du réseau Internet.

Document autorisé : "Mémento Python 3" fourni.

Le document comporte 10 pages.

Vérifiez que votre exemplaire est complet avant de commencer.

Certaines énigmes nécessitent d'exploiter des fichiers. Ces fichiers sont fournis avec le sujet.

La personne candidate indique ses réponses aux énigmes à la page 10.

Seule la page 10 est à remettre à la fin de l'épreuve.

01. Message secret (10 pts)

Bob envoie un message à Alice et pour brouiller les pistes, intercale entre chacune des lettres du message initial un certain nombre de lettres majuscules. Par exemple, le message suivant :

FSABBNUPLSEAUGSOEPSI UOZLFYBMQPPIKACDHEXSL

a été créé en intercalant 1 lettre majuscule entre chaque caractère du message original. Ainsi, en ne retenant qu'un caractère sur deux, on retrouve ce message :

FABULEUSES OLYMPIADES

Voici le début du message codé envoyé par Bob qui a intercalé 3 lettres (le message entier est fourni dans le fichier `codes_python.py`) :

RSBNEPSANGOPDIUZEFBQZPKC-HXLVBNEOTLMUNJVSIOF VJVALEOUYHO...

Question : Donne le nom de la rue qui figure dans le message.

02. Code binaire (15 pts)

Alice a cherché à transmettre un message discret à son ami Bob en transformant chaque caractère de son message. Elle a d'abord calculé le code du caractère avec la fonction `ord`, puis a converti ce nombre en binaire sur 8 bits et a ensuite mis tous les bits les uns à la suite des autres.

Bob a reçu le message dont voici le début (le message entier est fourni dans `codes_python.py`) :

'0100110000100111011001010111001101110011011001010110...'

Question : Aide Bob à décoder le message d'Alice.

03. Le spectacle (20 pts)

Vous allez assister à un spectacle dans une grande salle pouvant accueillir 6000 spectateurs. Toutes les places ont été réservées, et chacun des 6000 spectateurs possède un billet qui lui indique son numéro de place dans la salle entre 0 et 5999.

Heureusement vous avez réussi à vous procurer un billet pour la représentation de ce soir : la place numéro 2025 vous est réservée ! Vous arrivez néanmoins avec beaucoup de retard, et lorsque vous entrez dans la salle vous vous apercevez que vous êtes le dernier arrivé...

Pire que cela, quelqu'un est assis à votre place... il semble que les spectateurs se soient assis sans tenir compte de leur numéro de réservation. Le fichier `salle.txt` représente la répartition de la salle. Dans ce fichier, chaque ligne correspond à une place dans la salle : la ligne 0 correspond à la place 0, la ligne 1 à la place 1, etc. La valeur indiquée sur la ligne est le numéro de billet du spectateur assis à cette place. Les premières lignes du fichier sont :

4945
1661
2242
4259
264
...

ce qui signifie qu'en place 0 il est assis le spectateur qui possède le billet pour la place 4945, en place 1 celui qui possède une réservation en 1661, etc. Quelque part dans ce fichier il y a une seule place inoccupée notée -1 (mais ce n'est pas la place 2025 !)

Vous tenez énormément à vous asseoir à votre place fétiche numéro 2025. Vous allez donc gentiment demander à la personne assise à votre place d'aller s'asseoir au numéro qui lui a été attribué. La personne accepte, se dirige vers sa place qui n'est évidemment pas libre et demande alors à cette nouvelle personne d'aller à son siège réservé. Le processus se poursuit ainsi jusqu'à ce que la personne qui se déplace arrive sur le siège vide.

Si on prend un plus petit exemple d'une salle de 10 personnes : [4, 1, 8, -1, 3, 2, 9, 0, 6, 5] et en supposant que notre numéro de billet soit le 7.

- en place 7 il y a le spectateur 0 qui va se lever pour aller en 0
- en place 0 il y a le spectateur 4 qui va se lever pour aller en 4
- en place 4 il y a le spectateur 3 qui va se lever pour aller en 3
- en place 3 il n'y a personne

Au final 3 personnes se sont déplacées en tout, et la nouvelle salle est [0, 1, 8, 3, 4, 2, 9, 7, 6, 5]

Vous disposez du fichier `salle.txt` qui représente la salle à votre arrivée ainsi que le code Python permettant de charger le fichier de données `salle.txt` sous forme d'un tableau d'entiers `salle` (voir le fichier `codes_python.py`).

Question : combien de personnes au total vont devoir changer de siège pour que je puisse m'asseoir en place 2025 ?

04. Le plus grand nombre mystère (25 pts)

D'après un vieux parchemin que vous avez trouvé dans le grenier, il existerait une liste de nombres dits "mystérieux" qui respectent la règle suivante : un **nombre mystère** est un nombre qui est **égal à la somme des factorielles de ses chiffres**. Vous remarquez par ailleurs que quelqu'un a gribouillé l'inscription suivante sous le mot factorielle :

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1$$

Votre prédécesseur semble également avoir fait des tests sur certains nombres. Le nombre 145 fait partie de ces nombres mystérieux :

$$1! + 4! + 5! = 1 + 24 + 120 = 145$$

Par contre, le nombre 52 ne fait pas partie des heureux élus :

$$5! + 2! = 120 + 2 = 122$$

Très peu de nombres respectent cette propriété et il n'en existe aucun qui excède 7 chiffres.

Question : quel est le plus grand nombre mystère existant ?

05. La course (30 pts)

Dix cyclistes participent à une course, on les identifie par une lettre de l'alphabet (de A à J), les positions des concurrents dans la course sont données par une liste python `classement`. Par exemple, si `classement = ['I', 'F', 'G', 'J', 'B', 'A', 'C', 'H', 'D', 'E']`, le coureur I est premier, le coureur F second, le coureur G troisième...

Ce classement évolue en fonction des dépassements, par exemple si H dépasse le concurrent qui le précède, le classement donné ci-dessus devient : ['I', 'F', 'G', 'J', 'B', 'A', 'H', 'C', 'D', 'E']

Partie 1 (12 pts)

Étant donné un classement initial et une liste de dépassements, on cherche à déterminer le classement final. Par exemple avec :

```
classement = ['I', 'F', 'G', 'J', 'B', 'A', 'C', 'H', 'D', 'E']
depassements = ['H', 'D', 'H', 'F']
```

Le classement final est : ['F', 'I', 'G', 'J', 'B', 'H', 'A', 'D', 'C', 'E']

En effet, on considère les dépassements de gauche à droite :

- après le dépassement de 'H', le classement est : ['I', 'F', 'G', 'J', 'B', 'A', 'H', 'C', 'D', 'E']
- après celui de 'D' : ['I', 'F', 'G', 'J', 'B', 'A', 'H', 'D', 'C', 'E']
- 'H' dépasse à nouveau un concurrent, le nouveau classement est : ['I', 'F', 'G', 'J', 'B', 'H', 'A', 'D', 'C', 'E']
- enfin 'F' double le premier concurrent : ['F', 'I', 'G', 'J', 'B', 'H', 'A', 'D', 'C', 'E']

Question : Dans le fichier `codes_python.py` un classement initial de 15 coureurs et une liste Q1 de 50 dépassements sont fournis, quel est le classement final ? La réponse est à donner sous la forme d'une suite de lettres, par exemple le classement final de l'exemple précédent est : FIGJBHADCE

Partie 2 (18 pts)

On considère maintenant qu'un coureur peut dépasser plusieurs concurrents à la fois. Par exemple si H dépasse 3 concurrents, ce qu'on représente par le tuple ('H', 3), le classement initial ['I', 'F', 'G', 'J', 'B', 'A', 'C', 'H', 'D', 'E'] devient : ['I', 'F', 'G', 'J', 'H', 'B', 'A', 'C', 'D', 'E'].

Par exemple si le classement initial est : `classement = ['G', 'E', 'D', 'B', 'H', 'I', 'J', 'A', 'F', 'C']` et que la liste de dépassements est : `depassements = [('B', 2), ('F', 4), ('E', 1), ('J', 3)]`

L'évolution du classement sera :

- positions initiales ['G', 'E', 'D', 'B', 'H', 'I', 'J', 'A', 'F', 'C']
- après le dépassement ('B', 2) : ['G', 'B', 'E', 'D', 'H', 'I', 'J', 'A', 'F', 'C']
- après le dépassement ('F', 4) : ['G', 'B', 'E', 'D', 'F', 'H', 'I', 'J', 'A', 'C']
- après le dépassement ('E', 1) : ['G', 'E', 'B', 'D', 'F', 'H', 'I', 'J', 'A', 'C']
- après le dépassement ('J', 3) : ['G', 'E', 'B', 'D', 'J', 'F', 'H', 'I', 'A', 'C']

Donc le classement final de la course est GEBDJFHIAC

On garantit que la liste de dépassements est valide, c'est-à-dire qu'un concurrent ne dépasse jamais plus que le nombre de coureurs qui le précède.

Question : pour le même classement initial que la question 1 fournit dans le fichier `codes_python.py` et la liste Q2 de dépassements avec ce nouveau format, vous devez déterminer le classement final de la course.

06. Validation d'un numéro de carte de crédit (30 pts)

Un algorithme (appelé algorithme de Luhn), permet de vérifier qu'un numéro de carte de crédit est valide. Les étapes sont les suivantes :

1. on commence par numéroter les chiffres de droite à gauche, en commençant les rangs à 1 ; par exemple pour le nombre 437716 cela donne :

```
chiffres : 4 3 7 7 1 6
rangs    : 6 5 4 3 2 1
```

2. on extrait du numéro la liste des chiffres de rang impair ainsi que celle des chiffres de rang pair. Par exemple, sur le numéro 437716 cette procédure donne [3, 7, 6] pour les chiffres de rang impair et [4, 7, 1] pour ceux de rang pair.
3. on double ensuite chaque chiffre de la liste des rangs pairs et si on obtient un chiffre plus grand que 9, on le remplace par la somme des deux chiffres qui le composent. Dans l'exemple précédent, la liste des chiffres de rang pair [4, 7, 1] devient donc [8, 5, 2] car $2 * 7$ donne 14 qui est remplacé par la somme de ses chiffres donc 5.
4. on calcule ensuite la somme des chiffres des deux listes, si le résultat obtenu est divisible par 10 alors le numéro de la carte de crédit est valide. Dans l'exemple précédent, on calcule donc : $3 + 7 + 6 + 8 + 5 + 2 = 33$ et comme 33 n'est pas divisible par 10, le numéro n'est pas valide.

Prenons en deuxième exemple fictif : 4762. La procédure donne :

- La liste des chiffres de rang impair : [7, 2]
- La liste des chiffres de rang pair : [4, 6] qui donne [8, 3] après l'étape 3
- La somme obtenue à l'étape 4 est $7 + 2 + 3 + 8 = 20$ qui est divisible par 10, donc le numéro est valide.

Question : dans le fichier `codes_python.py` une liste de 100 numéros de carte de crédit est fournie. Un seul de ces numéros n'est pas valide, lequel ?

07. Leçon d'(in)sécurité par Bob, l'expert auto-proclamé (35 pts)

Alice et Bob sont en pleine discussion autour d'un café, et comme toujours, Bob aime se donner des airs de grand expert en cybersécurité.

“Bob (fièrement, les bras croisés) : Alice, tu n'as AUCUNE chance de retrouver mon mot de passe. J'ai pris mes précautions !

- **Alice (intriguée) :** Ah oui ? Qu’est-ce que tu as fait ?
- **Bob (souriant, sûr de lui) :** J’ai généré un *hash* SHA256, ma chère. Tu peux essayer tant que tu veux, c’est irréversible. Même avec des supercalculateurs quantiques, tu n’y arriveras pas !
- **Alice (malicieuse, tapotant son clavier) :** Oh... intéressant ! Tu peux me montrer ton hash ?
- **Bob (toujours confiant, lui envoie par messagerie) :** Tiens, tu peux toujours essayer :
851d3fcf9fe60976956b0b644f56f852045435b42cc1797d5a4cc347e40bfae3
- Mais ça ne sert à rien, je te dis !
- **Alice (clavier cliquetant, souriant) :** Hmm... Ctrl+C... Ctrl+V... click ! Oh, tiens, voilà ton mot de passe.
- **Bob (blême, la mâchoire qui tombe) :** MAIS... COMMENT ?!?”

Partie 1 (14 pts)

Bob a malheureusement utilisé un mot du dictionnaire. Il suffit donc à Alice de calculer les empreintes de ces mots pour trouver le mot de passe.

Un dictionnaire vous est fourni dans le fichier `dico.txt` ainsi qu’un code Python permettant de charger ces mots dans un tableau (voir le fichier `codes_python.py`).

Question : quel est le mot de passe de Bob ?

L’échec cuisant de Bob, partie 2 (21 pts)

Après sa première défaite humiliante, Bob décide d’améliorer son mot de passe.

“**Bob (avec un sourire en coin) :** Ah, Alice ! Tu crois que tu es maligne, hein ? Mais cette fois, c’est du sérieux. J’ai renforcé mon mot de passe avec des substitutions ultra-sécurisées !

- **Alice (curieuse) :** Oh ? Et qu’as-tu fait exactement ?
- **Bob (fièrement) :** J’ai remplacé certaines lettres par des chiffres et des symboles. C’est impossible à deviner maintenant !
- **Alice (sourire malicieux, tapotant sur son clavier) :** Intéressant... Allez, donne-moi ton nouveau hash, juste pour voir.”

Bob, sûr de lui, lui envoie son hash SHA256 flambant neuf :

98619e8bc343b37368e0b567b9a575ba6dc2e6b5b22c6695eb6eff7445d56f6d

“**Alice (quelques clics plus tard, sourire en coin) :** Hmm... et si ton mot de passe, cette fois, c’était celui-là ?

- **Bob (désespéré, se prenant la tête dans les mains) :** Mais... mais... je pensais être un génie !”

Cette fois-ci, Bob a changé de mot de passe et il remplace quelques lettres :

- les o par des 0 (zéro)
- les i par des 1 (un)
- les a par des @ (arobase)

Son erreur ? C’est d’avoir encore utilisé un mot du dictionnaire ! Sacré Bob, il n’a décidément rien compris.

Question : mais alors, quel est le deuxième mot de passe de Bob ?

08. L’énigme du loup garou (40 pts)

200 randonneurs numérotés de 0 à 199 partent camper en forêt un soir de pleine de lune. À minuit pile, certains randonneurs entendent un hurlement terrifiant : un cri de loup-garou. Tous les randonneurs qui entendent le cri appellent immédiatement la police pour signaler l’incident.

L'inspectrice Alice prend en charge l'affaire. Elle commence par récupérer les données des téléphones portables des 200 randonneurs présents ce soir là et à l'aide des données GPS récupère pour chaque randonneur ses coordonnées cartésiennes (x, y) à minuit pile (l'unité est le mètre).

Les naturalistes ont établi avec certitude que la portée maximale du cri de loup garou est de 5km.

Question : en supposant que tous les randonneurs à portée du loup-garou ont prévenu la police et que le standard a recensé très exactement 32 appels signalant le cri, déterminer le numéro du randonneur qui est le loup-garou.

Ressources pour cette énigme

Les données GPS sont fournies sous forme d'un fichier texte `gps.txt` et une fonction `load_coordonnees` dans le fichier `codes_python.py` pour charger en mémoire la liste des coordonnées écrites dans le fichier `gps.txt`.

On rappelle que la distance qui sépare les points de coordonnées (x, y) et (a, b) est :

$$d = \sqrt{(x - a)^2 + (y - b)^2}$$

09. Gray and Anna's coding (45 pts)

Gray et Anna participent à une chasse au trésor. Arrivés à la dernière étape, ils découvrent un vieux parchemin sur lequel est inscrit un mystérieux code binaire et une question posée. Voici ce qui est inscrit :

Chaque "mot" du code secret est composé uniquement de lettres 0 et 1 appelées aussi *bits*. Tous les mots sont composés du même nombre de bits fixé à l'avance. Voici le code secret à 2 bits comportant ici 4 mots :

00
01
11
10

Voici un deuxième exemple, celui du code secret à 3 bits comportant ici 8 mots :

000
001
011
010
110
111
101
100

Après quelques minutes de réflexion, ils conjecturent tous les deux que le nombre de mots du code à n lettres est 2^n .

Puis, Anna trouve une façon de construire ce code :

- on construit les mots un par un en les écrivant de haut en bas
- par convention, le premier mot est constitué uniquement de bits 0
- deux mots consécutifs ne diffèrent que d'une seule lettre
- et pour passer d'une ligne à la suivante, et donc construire un nouveau mot à partir du précédent : on inverse (un 0 devient 1, et réciproquement) le bit le plus à droite possible **conduisant à un nouveau mot**

Par exemple, dans le code secret à 3 bits vu plus haut, pour construire le 7^e mot à partir du 6^e qui est 111, on procède de la façon suivante :

- On change le premier bit le plus à droite de 111, le mot obtenu est 110
- mais comme il s'agit déjà d'un mot du code (le 5^e mot), on poursuit la recherche et on inverse le deuxième bit le plus à droite de 111, ce qui donne 101 qui est bien un nouveau mot, c'est donc lui le 7^e.

Question : On considère ce code secret fixé cette fois-ci à 17 bits. Quel est le 2025^e mot ?

10. Puzzle 2025 pièces (50 pts)

On dispose d'un ensemble de pièces de quatre types différents : A, B, C, ou D. Ces pièces sont représentées sur la figure 1.

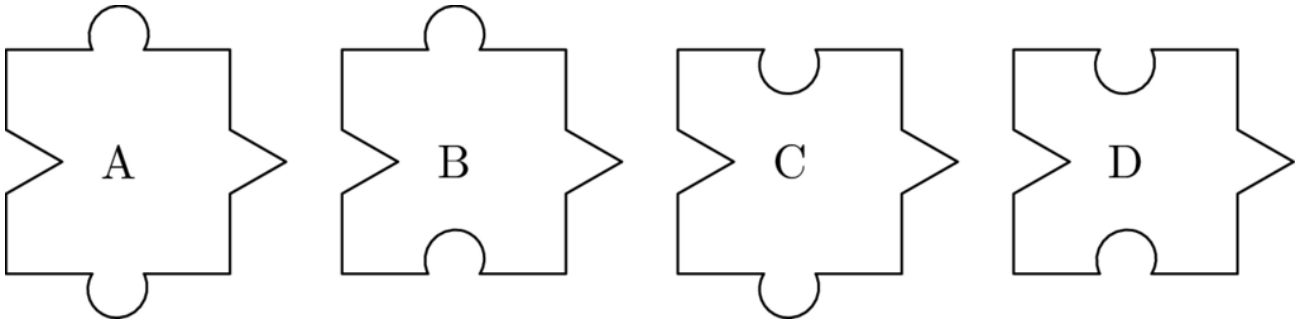


Figure 1: Types de pièces

On se propose de savoir s'il est possible de remplir un puzzle rectangulaire de n lignes et m colonnes avec une liste de pièces données. Les pièces sont données dans l'ordre suivant :

- on procède de la ligne du haut à la ligne du bas,
- puis pour chaque ligne on liste les pièces de gauche à droite.

Par exemple pour un puzzle $n = 3 \times m = 4$ avec la liste de pièces $L = ['A', 'A', 'A', 'A', 'C', 'D', 'C', 'D', 'C', 'A', 'D', 'B']$, la figure 2 montre le remplissage correct.

Partie 1 : pièces incorrectes (32 pts)

Dans la première partie du problème, on se fixe les dimensions du puzzle et on se donne une liste de pièces à placer. On s'imagine d'abord qu'on place toutes les pièces dans l'ordre donné. On compte ensuite le nombre de pièces incorrectes, c'est-à-dire qui sont en conflit avec l'une de leurs voisines au moins.

Par exemple, si les dimensions sont $n = 3 \times m = 4$ et que la liste des pièces est $['A', 'A', 'A', 'A', 'C', 'B', 'C', 'D', 'C', 'A', 'D', 'D']$ on obtient l'arrangement de la figure 3 qui contient 4 pièces incorrectes (marquées par un cercle).

Question : en utilisant la liste `PIECES` de 2025 pièces de votre fichier `codes_python.py`, déterminer le nombre de pièces incorrectes lorsque les dimensions sont $n = 45 \times m = 45$.

Partie 2 : les bonnes dimensions (18 pts)

Il y a beaucoup de pièces incorrectes ! Vous vous rendez compte que les dimensions choisies ne sont certainement pas les bonnes.

Question : déterminer les dimensions du puzzle $n \times m$ (avec $n > 1$ et $m > 1$) qu'il faut choisir pour que les 2025 pièces soient correctement placées. On ne modifiera pas l'ordre des pièces.

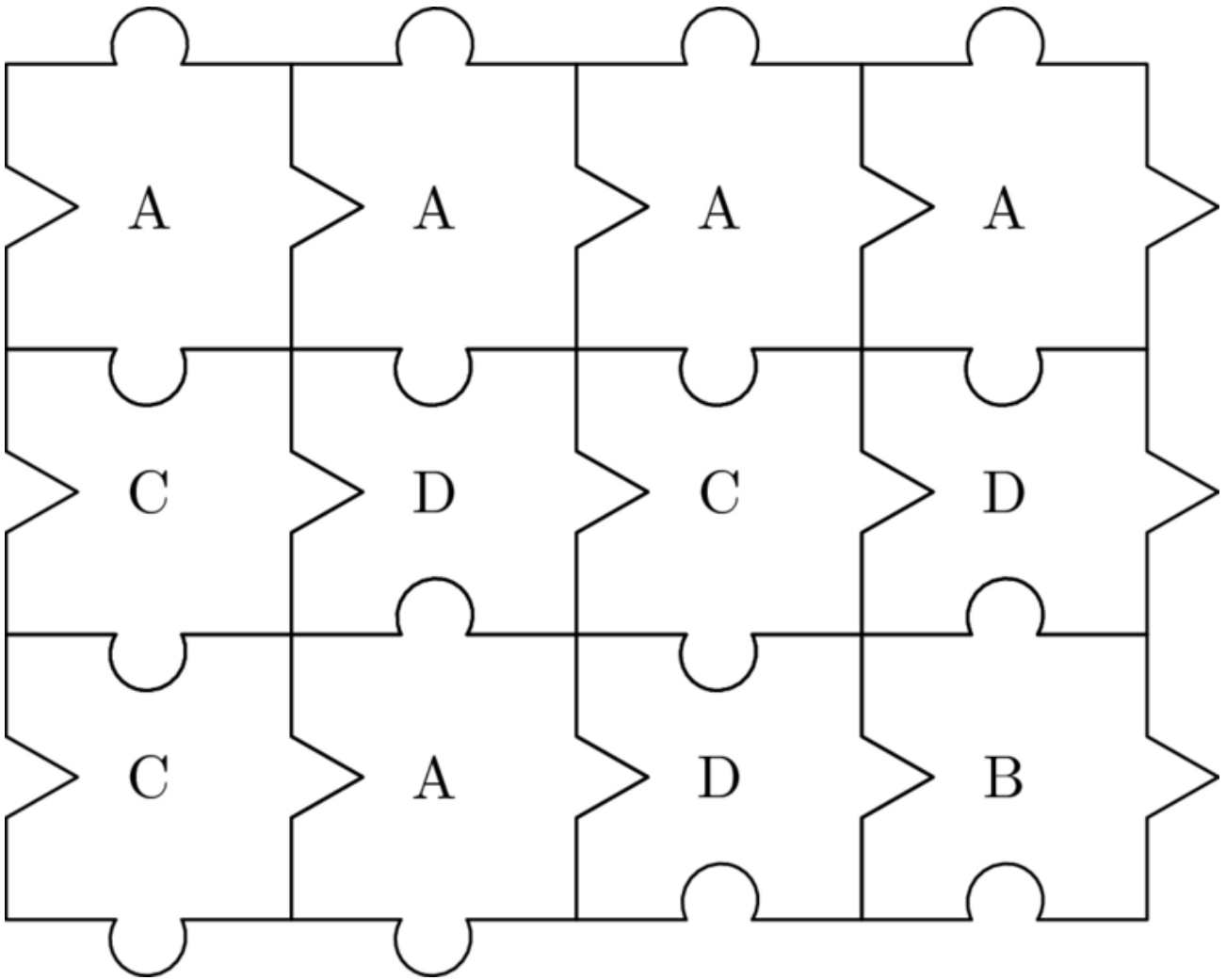


Figure 2: Puzzle correct

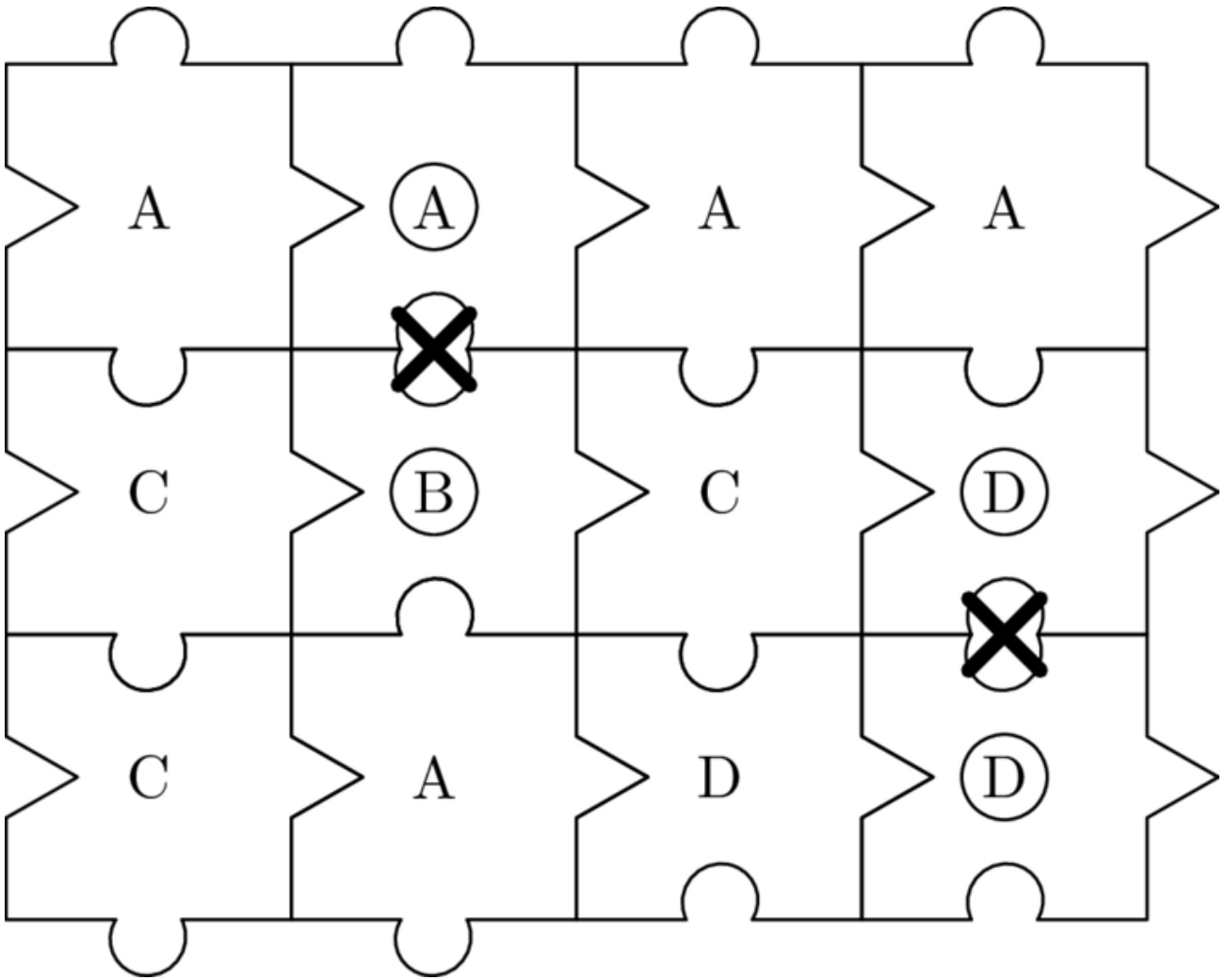


Figure 3: Puzzle incorrect

