EPITA IPSA ÉCOLE D'INGÉNIEURS ESME

CONCOURS

CPGE

EPITA - IPSA - ESME

1 SEUL ET UNIQUE CONCOURS
POUR VOUS PRÉPARER À UNE MULTITUDE DE MÉTIERS

Samedi 12 avril 2025

OPTION SCIENCES NUMÉRIQUES

MP - PC - PSI - PT - TSI

DURÉE : 2 HEURES

Conditions particulières :

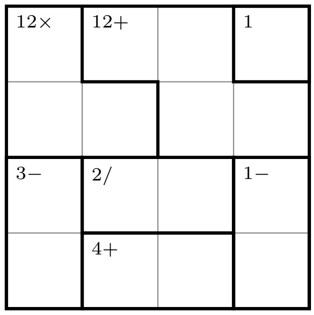
Calculatrice et documents interdits

Indiquez votre code candidat SCEI sur le QCM
qu'il faudra insérer dans votre copie d'examen

Le sujet se compose d'un problème et d'un questionnaire à choix multiples. Le questionnaire à choix multiples devra être inséré dans votre copie. Les fonctions à produire dans ce sujet devront être rédigées en langage Python et ne pas avoir recours à l'usage de bibliothèques. Il est possible d'écrire des fonctions auxiliaires non explicitement demandées à condition de les documenter et de les définir avant d'en faire usage.

Problème – Le KenKen

Le **KenKen** est un jeu constitué d'une grille carrée de n cases de coté (en général $n \geqslant 3$) ; n sera alors appelé **taille** du jeu. La grille comporte des ensembles de cases, délimités par un contour plus épais, que l'on appellera des **zones**. Dans chacune de ces zones est précisé une contrainte : un nombre, généralement suivi d'un opérateur. Ces contraintes et leur format seront détaillés plus loin. Voici l'exemple d'une grille de taille 4 :



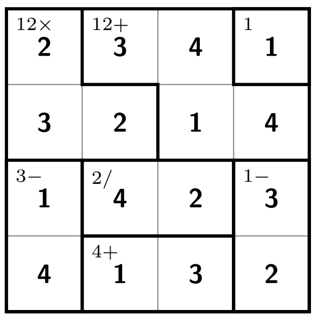
Le but du jeu est de remplir chaque case de la grille avec un entier, de sorte que chaque ligne et chaque colonne comporte chaque entier de 1 à n et de façon à ce que les entiers présents dans les cases d'une même zone respectent la **contrainte de la zone** définie comme suit :

— si la zone contient une seule case : la contrainte est un simple nombre indiquant la valeur que doit contenir cette case ;

— si la zone comporte au moins deux cases : la contrainte est un chiffre r suivi d'un opérateur parmi $+$, $\times$, $-$ et $/$ dont la signification est donnée par la tableau ci-après :

opérateur	**description de la contrainte**
$+$	la somme des entiers dans les cases de la zone doit donner r.
$\times$	le produit des entiers dans les cases de la zone doit donner r.
$-$	la différence des deux entiers des cases de la zone doit donner r (dans un sens ou dans l'autre). *Attention, cette contrainte ne s'applique qu'à des zones de 2 cases.*
$/$	le quotient des deux entiers des cases de la zone doit donner r (dans un sens ou dans l'autre). *Attention, cette contrainte ne s'applique qu'à des zones de 2 cases.*

Une grille de KenKen bien formée ne comporte qu'une seule solution satisfaisant ces contraintes. Voici la solution de l'exemple précédent :



On choisit de représenter chaque zone par un dictionnaire contenant les champs `val`, `op` et `cases` ; `val` et `op` contiennent respectivement le nombre et l'opérateur associés à la zone tandis que `cases` est la liste des cases de la zone, représentées par le couple de leurs coordonnées. On choisit de repérer les cases en ligne puis colonne en commençant par la case supérieure gauche. Ainsi, pour une grille de taille n, la case supérieure gauche aura pour coordonnées $(0, 0)$, la case inférieure gauche $(n-1, 0)$ et la case inférieure droite : $(n-1, n-1)$. Pour les zones ne comportant qu'une seule case, le champ `op` sera une chaîne de caractères vide.

Dans l'exemple précédent, la zone comportant la case de coordonnées $(0, 0)$ pourra être représentée par le dictionnaire `{"val":12, "op":"×", "cases": [(0,0),(1,0),(1,1)]}` et la zone réduite à la case de coordonnées $(0, 3)$ pourra être représentée par : `{"val":1, "op":"", "cases": [(0,3)]}`.

Un jeu de KenKen sera alors intégralement décrit par sa taille n et par la liste des zones qui le composent. De plus, on représente l'état du remplissage d'une grille par une liste de listes d'entiers de sorte que la sous-liste d'indice k corresponde à la valeur des cases situées sur la ligne k (parcourues de gauche à droite). Les cases non remplies de la grille se verront attribuer la valeur 0. Dès lors, la grille de l'exemple dont on aurait effectué un remplissage partiel (en ne remplissant que la première ligne) est représentée par la liste `[[2,3,4,1],[0,0,0,0],[0,0,0,0],[0,0,0,0]]`.

Autour des contraintes

Question 1 Écrire une fonction `test_somme` qui, étant donné une liste d'entiers et un entier `v`, retourne `True` si la somme des éléments de la liste vaut `v` et `False` sinon.

Question 2 Écrire une fonction `test_quotient` qui, étant donné une liste de deux entiers non nuls et un entier `v`, retourne `True` si la valeur `v` peut-être obtenue en calculant le quotient des deux entiers (dans un sens ou dans l'autre) et `False` sinon.

On considère à présent disposer des fonctions suivantes :

- `test_produit` qui, étant donné une liste d'entiers et un entier `v`, retourne `True` si le produit des éléments de la liste vaut `v` et `False` sinon ;
- `test_difference` qui, étant donné une liste de deux entiers et un entier `v`, retourne `True` si la valeur `v` peut-être obtenue en calculant la différence des deux entiers (dans un sens ou dans l'autre) et `False` sinon.

Question 3 Écrire une fonction `test_zone` qui, étant donné une grille remplie et une zone, retourne `True` si la contrainte de zone est respectée et `False` sinon.

Question 4 Écrire une fonction `test_grille` qui, étant donné une grille (complètement ou partiellement remplie) et une liste de zones, retourne `True` si la grille respecte les contraintes de chacune des zones, et `False` sinon. On considèrera que cette fonction ne sera appelée que pour des zones portant sur des cases remplies de la grille.

Question 5 Écrire une fonction `test_conflit` qui, étant donné une grille dont les i premières lignes sont supposées remplies et l'entier i, retourne `False` si l'un au moins des éléments de la ligne i est déjà présent parmi les éléments des lignes précédentes situés sur la même colonne que lui. Dans le cas contraire la fonction renverra `True`.

On souhaite à présent pouvoir regrouper les contraintes de zones selon le plus grand numéro de ligne des cases qu'elles comportent.

Question 6 Écrire une fonction `ligne_max` qui, étant donné une zone, retourne le plus grand numéro de ligne parmi ceux des cases qu'elle comporte. On interdit pour cette question l'usage des fonctions `min` et `max` de Python.

Question 7 Écrire une fonction `zones_par_ligne_max` qui, étant donné la taille n d'un jeu et la liste de ses zones, retourne une liste comportant n sous-listes. La sous-liste d'indice k sera la liste des zones du jeu telles que le numéro de ligne maximal des cases qui la composent est égal à k.

Afin de vérifier si les zones sont bien formées, on souhaite construire une fonction permettant de déterminer si une liste de cases est **connexe** ; c'est-à-dire s'il est possible de se déplacer entre n'importe quelles cases de la liste en passant uniquement par des cases de cette liste et au moyen de déplacements horizontaux et/ou verticaux.

Question 8 Écrire une fonction `test_connexite` qui, étant donné une liste de cases (exprimées par leurs coordonnées), retourne `True` lorsqu'elle est connexe et `False` sinon. Cette fonction pourra opérer comme si elle parcourait le graphe formé par les cases de la liste dont seraient adjacentes les cases se touchant horizontalement ou verticalement.

Itérateur de permutations

On rappelle qu'une **permutation** de $[\![1,n]\!]$ est une bijection de $[\![1,n]\!]$ dans $[\![1,n]\!]$. On choisit de représenter une permutation σ de $[\![1,n]\!]$ par la liste d'entiers $[\sigma(1),\cdots,\sigma(n)]$. Ainsi les listes comportant une et une seule fois chaque entier de 1 à n représentent toutes les permutations de $[\![1,n]\!]$.

On considère l'ordre des permutations de $[\![1,n]\!]$ comme étant l'**ordre lexicographique** des listes qui les représentent :

$$[a_1] < [b_1] \Leftrightarrow a_1 < b_1$$

$$\forall p \geqslant 2,\ [a_1,\cdots,a_p] < [b_1,\cdots,b_p] \Leftrightarrow a_1 < b_1 \text{ ou } \begin{cases} a_1 = b_1 \\ [a_2,\cdots,a_p] < [b_2,\cdots,b_p] \end{cases}$$

Ainsi $[2,4,1,3] < [2,4,3,1] < [3,1,2,4] < [3,1,4,2]$.

Question 9 Donner les 3 permutations qui suivent $[3,1,4,2]$ dans l'ordre lexicographique.

Question 10 Écrire une fonction `identite` prenant en paramètre un entier `n` et retournant la permutation $[1,\ldots,n]$.

Question 11 Écrire une fonction `inverse` prenant en paramètres une permutation `p` de $[\![1,n]\!]$ et 2 entiers `i` et `j`. Cette fonction modifiera la permutation `p` en inversant l'ordre des éléments dont l'indice des cases appartient à $[\![i,j]\!]$.

Cette fonction ne fera rien dans le cas où $i \geqslant j$ et on supposera que cette fonction n'est appelée que lorsque i et j sont dans $[\![1,n]\!]$.

Question 12 Écrire une fonction `indice_pps` prenant comme paramètres une permutation `p` de $[\![1,n]\!]$ et un entier i de $[\![1,n]\!]$. Cette fonction renverra, parmi les cases d'indice strictement supérieur à i et de valeur strictement supérieure à $p[i]$, l'indice de celle de valeur minimale. On suppose que cette fonction ne sera appelée que lorsqu'un tel indice existe.

Question 13 Écrire une fonction `suivante` prenant en paramètre une permutation `p` de $[\![1,n]\!]$. Cette fonction modifiera la permutation pour la transformer par la suivante dans l'ordre lexicographique. Lorsque la permutation suivante existe, la fonction renverra la valeur `True` ; dans le cas contraire, elle renverra la valeur `False` sans modifier la permutation.

Résolution

On cherche à présent à résoudre des jeux de KenKen. On propose d'essayer de remplir ligne après ligne la grille en commençant par attribuer à la première ligne, la première permutation des entiers de $[\![1,n]\!]$ dans l'ordre lexicographique. On fait de même pour la seconde ligne, puis les suivantes, en veillant à vérifier à chaque étape que le remplissage partiel de la grille est conforme aux contraintes du jeu : les zones portant sur les cases remplies doivent voir leurs contraintes vérifiées et 2 mêmes entiers ne peuvent se trouver sur une même colonne. En cas d'échec, on essaie la permutation suivante de la ligne faisant défaut. Si aucune des permutations ne convient, c'est que l'une des lignes précédentes est en cause et on essaie alors la permutation suivante de la ligne qui précède celle faisant défaut.

Question 14 Écrire une fonction récursive `continue_remplissage` qui, étant donné une grille dont les i premières lignes sont intégralement remplies, la liste des listes de zones rangées par indice de ligne maximal de ses cases, et l'entier i, essaie tous les remplissages possibles des lignes suivant i jusqu'à aboutir à une solution. Cette fonction modifiera la grille en la remplissant par la solution obtenue. La fonction renverra `True` si elle aboutit à un remplissage valide et `False` sinon.

Question 15 Écrire une fonction `resoudre` prenant en paramètres la taille `n` d'un jeu et la liste de ses zones. Cette fonction retournera l'unique solution au problème et `None` si elle ne l'a pas trouvée.

Question 16 Décrire le pire des cas en termes de découpage des formes des zones pour cette stratégie de résolution.

Question 17 Écrire une fonction `nb_solutions` qui, étant donné la taille `n` d'un jeu possiblement mal formé (en ce sens qu'elle pourrait admettre plusieurs solutions) et la liste de ses zones, retourne le nombre de solutions respectant les contraintes du jeu.

Gestion d'un site en ligne

On cherche à présent à élaborer la base de données d'un site internet proposant des grilles de KenKen. On choisit de stocker les informations des comptes utilisateurs, la liste des grilles disponibles et les informations des temps de résolution des grilles par les utilisateurs.

On répartit ces informations en 3 tables dont on présente des extraits ci-après. La première ligne représente les noms des champs et les suivantes les enregistrements. La clé primaire de chaque table sera indiquée en première colonne. Les clés étrangères seront soulignées.

Une première table `utilisateurs` permet de stocker les informations personnelles des utilisateurs :

idUtilisateur	**nom**	**prenom**	**surnom**	**dateDeNaissance**	⋯
⋮	⋮	⋮	⋮	⋮	⋮
42	Page	Larry	DeepMind	1973-03-26	⋯
⋮	⋮	⋮	⋮	⋮	⋮
314	Harmon	Elizabeth	Beth	1948-11-02	⋯
⋮	⋮	⋮	⋮	⋮	⋮

Une seconde table `grilles` permet de stocker les grilles disponibles pour les joueurs, en particulier leur taille et leur niveau de difficulté.

idGrille	**taille**	**difficulte**	⋯
⋮	⋮	⋮	⋮
159	4	2	⋯
⋮	⋮	⋮	⋮

Une troisième table `parties` permet de déterminer les temps de résolution des grilles par les joueurs. On stocke l'identifiant `idUtilisateur` de l'utilisateur concerné dans `u_idUtilisateur` qui sera donc une clé étrangère ; de même pour la clé étrangère `g_idGrille` permettant d'identifier la grille concernée. Le champ `resolue` contient 0 pour une grille non encore résolue et 1 sinon tandis que `debut` (respectivement `fin`) contient l'instant de début (respectivement de fin) de résolution de la grille par l'utilisateur (exprimé en secondes écoulées depuis le 1er janvier 1970 à minuit). Enfin le champ `note` comporte un indice de satisfaction de la grille que fournit le joueur après avoir résolu la grille. Les champs non remplis se verront attribuer la valeur `NULL`.

idPartie	**u_idUtilisateur**	**g_idGrille**	**resolue**	**debut**	**fin**	**note**
⋮	⋮	⋮	⋮	⋮	⋮	⋮
421	314	159	1	1735080872	1735081199	3
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2718	42	159	0	1744461130	NULL	NULL
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Question 18 Écrire une requête SQL retournant le meilleur temps de résolution (en secondes) obtenu pour la grille d'identifiant 159.

Question 19 Écrire une requête SQL qui donne la meilleure note attribuée à une grille par la joueuse « Elizabeth Harmon ».

Question 20 Écrire une requête SQL donnant les surnoms et les temps de résolution des trois joueurs les plus rapides parmi ceux ayant résolu la grille d'identifiant 421. On ordonnera les réponses par temps croissant de résolution de la grille.

Question 21 Écrire une requête SQL donnant les noms, prénoms et surnoms des joueurs ayant donné la meilleure note à une grille de difficulté 2 (plus une note est élevée, meilleure elle est).

QCM – MP-PC-PSI-PT-TSI

Ce document est à insérer dans votre copie en fin d'épreuve. Vous devez y reporter votre numéro de candidat, votre centre d'écrits et votre numéro de table.

Numéro SCEI :
Centre d'écrits :
Numéro de table :

Dans ce questionnaire à choix multiples, chaque question comporte une ou plusieurs bonnes réponses.
Chaque réponse correcte fait gagner des points, mais chaque réponse fausse annule tous les points de la question.
Les questions peuvent-être formulées au pluriel par commodité d'expression. Cela n'implique pas nécessairement qu'elles admettent plusieurs réponses correctes.

On considère la fonction Python suivante :

```python
def mystere(m,n):
    t=1
    if n>0:
        t=mystere(m,n//2)
        t=t*t
        if n%2==1:
            t=m*t
    return t
```

1. La fonction `mystere` :
   - ☐ effectue un nombre linéaire d'appels récursifs en `n` quand `n` est une puissance de 2 ;
   - ☐ effectue un nombre quadratique d'appels récursifs en `n` quand `n` est une puissance de 2 ;
   - ☐ permet de calculer les puissances d'un nombre ;
   - ☐ permet de calculer le produit de deux nombres.

2. Quels sont les 4 derniers bits de la représentation de l'entier 2000 en binaire non signé de l'entier ?
   On considèrera placé à droite le bit de poids le plus faible.
   - ☐ 0000
   - ☐ 1100
   - ☐ 0011
   - ☐ 1111

3. On dispose des 4 derniers bits de la représentation binaire d'un entier non signé (le bit de poids le plus faible est à droite). Parmi les quatre entiers suivants, quels sont ceux qui ont 1001 pour derniers bits de leur représentation ?
   - ☐ 25
   - ☐ 2048
   - ☐ 2025
   - ☐ 2345

4. On considère un graphe non orienté dont les sommets sont étiquetés par les représentations binaires (non signées) à trois bits des entiers de 0 à 7. Deux sommets sont reliés par une arête si et seulement si leurs étiquettes diffèrent d'exactement un bit. Par exemple : 101 et 111 sont reliés, mais 101 et 110 ne le sont pas. Parmi les assertions suivantes, indiquer celles qui sont correctes :
   - ☐ Les sommets du graphe sont tous de même degré ;
   - ☐ Le graphe ne contient pas de cycles ;
   - ☐ Il existe un chemin entre tout couple de sommets du graphe ;
   - ☐ Il est possible d'attribuer une couleur à chaque sommet du graphe de sorte que deux sommets reliés aient toujours des couleurs différentes ; ce en utilisant seulement 2 couleurs.

5. On considère un jeu à deux joueurs dans lequel chaque joueur retire à tour de rôle 1, 2 ou 3 bâtons parmi un ensemble initial de 42 bâtons. Le perdant est le joueur qui retire le dernier bâton. Parmi les assertions suivantes, indiquer celles qui sont correctes :
   - ☐ Le premier joueur a une stratégie gagnante ;
   - ☐ Le premier joueur n'a pas de stratégie gagnante ;
   - ☐ Si le premier joueur retire 1 bâton au premier tour alors le second joueur a une stratégie gagnante ;
   - ☐ Si le premier joueur retire au moins 2 bâtons au premier tour alors le second joueur a une stratégie gagnante.