CONCOURS
COMMUN
INP

SUJET 0

ÉPREUVE SPÉCIFIQUE - FILIÈRE MPI

INFORMATIQUE

Durée : 4 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, 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 trois parties, toutes indépendantes.

Partie I - Logique

Un footballeur affirme à la presse :

(i). Le jour où je marque un but, je suis content et je fais la fête.

(ii). Le jour où mon équipe gagne, ou bien je suis content, ou bien je fais la fête ou les deux.

(iii). Le jour où mon équipe perd, ou bien je ne suis pas content, ou bien j'ai marqué un but ou les deux.

(iv). Le jour où je ne marque pas et je fais la fête, je suis content.

(v). Aujourd'hui, je ne suis pas content.

Q1. Définir les variables propositionnelles nécessaires à la modélisation de ce problème.

Q2. Modéliser chacune des assertions à l'aide de formules propositionnelles.

Q3. Mettre chacune de ces formules en forme normale conjonctive.

Q4. On souhaite savoir si le joueur a marqué, si son équipe a gagné et s'il a fait la fête. Donner la formule F permettant de répondre à ces questions. À quel problème classique est-on confronté ?

On rappelle l'algorithme de Quine (algorithme 1) : en notant
- C l'ensemble des clauses de F,
- x une variable propositionnelle,
- $C[x \leftarrow \top]$ l'ensemble des clauses obtenues en supprimant de C toutes les clauses contenant x, et en supprimant $\neg x$ de toutes les clauses contenant cette négation,
- $C[x \leftarrow \bot]$ l'ensemble des clauses obtenues en supprimant de C toutes les clauses contenant $\neg x$, et en supprimant x de toutes les clauses contenant cette variable.

l'algorithme s'écrit :

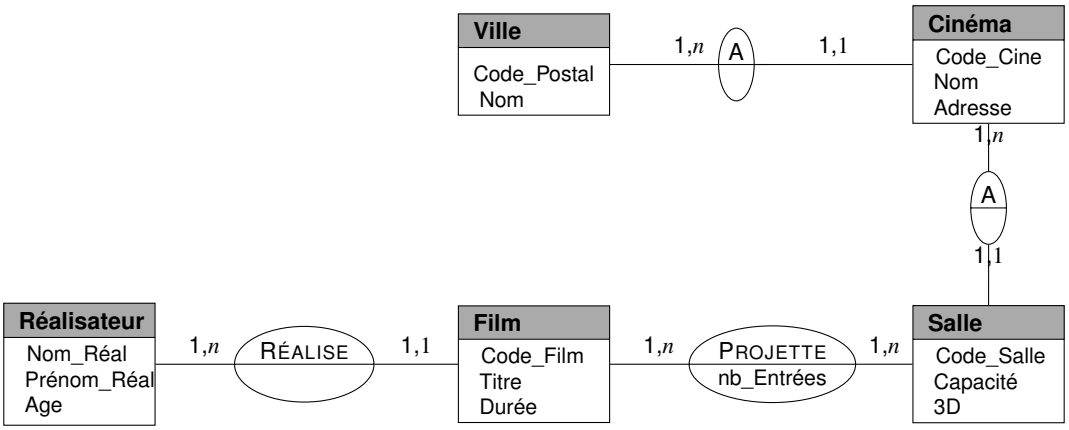
Algorithme 1 : Algorithme de Quine

Fonction `Quine`(C)
Entrées : C l'ensemble des clauses de F.
Sorties : Vrai si F est satisfsiable, Faux sinon.
début
  si $C = \emptyset$ **alors**
    retourner *Vrai*
  si $\bot \in C$ **alors**
    retourner *Faux*
  Choisir x apparaissant dans une clause de C
  si *Quine*$(C[x \leftarrow \bot])$ **alors**
    retourner *Vrai*
  sinon
    retourner *Quine*$(C[x \leftarrow \top])$

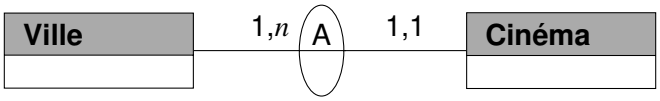
Q5. Appliquer cet algorithme et trouver une valuation qui rende F vraie.

Partie II - Bases de données

On considère la base de données décrite par le modèle entité-association suivant :



Dans cette représentation, on lie des entités par des associations avec des cardinalités. Ainsi par exemple



peut se lire comme “Une ville a 1 à n cinéma(s)“ et “un cinéma est présent dans 1 et une seule ville“. Parfois, l’association possède des propriétés (le nombre d’entrées d’un film projeté dans une salle par exemple).

On suppose que le champ 3D de l’entité Salle est un entier, valant 0 si la salle n’est pas en 3D, et 1 sinon. On suppose de plus que la durée des films est en heures. Enfin, on affirme qu’une ville peut être uniquement déterminée par son code postal et son nom.

Q6. Donner le schéma relationnel correspondant. Préciser les clés primaires et étrangères des relations. Les clés primaires peuvent être associées à plusieurs attributs.

Q7. Écrire les requêtes suivantes en langage SQL :

(i). Donner le titre des films durant moins de 2h.

(ii). Donner le nom et le prénom du réalisateur ayant réalisé le film “Matrix”.

(iii). Compter le nombre de cinémas à Nantes.

(iv). Donner l’adresse des cinémas contenant au moins une salle 3D.

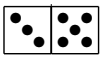
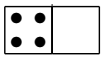
(v). Donner le code des films projetés dans toutes les salles.

(vi). Donner la liste des titres des films projetés dans le cinéma “Le Rio”.

Partie III - Algorithmique - Problèmes de dominos

Cette partie comporte des questions de programmation qui seront abordées en utilisant **exclusivement le langage C**. Les codes seront commentés de manière pertinente.
Dans toute la suite, on suppose disposer, via `stdbool.h`, d'un type `bool` avec deux constantes `true` et `false`.

Dans ce problème, on s'intéresse à quatre problèmes basés sur les dominos.

Un domino D est une pièce rectangulaire contenant deux chiffres, de 0 à N, matérialisés par des points. Par exemple,  ou  sont deux dominos, représentant les paires (3,5) et (4,0). Un domino est donc représenté par une paire d'entiers $(i, j), i, j \in [\![0, N]\!]$.

III.1 - Structures de données

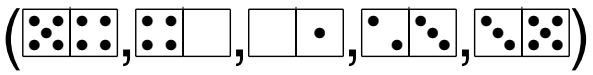
Dans cette section, on construit les structures de données utiles pour les deux premiers jeux.

On définit le type structuré `Domino` : `typedef struct { int x; int y; } Domino;`

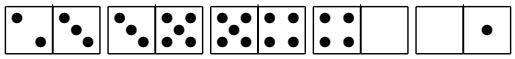
On dispose d'un sac $\mathcal{S}$ contenant n dominos $D_i = (i_k, j_k), k \in [\![1, n]\!], i_k, j_k \in [\![0, N]\!]$.

Définition 1 (Chaîne)
Une chaîne de dominos est une séquence de pièces telle que les chiffres voisins sur chaque paire de dominos consécutifs coïncident.

Par exemple, pour le sac



la séquence suivante est une chaîne de 5 dominos.



On souhaite gérer un sac et une chaîne comme une liste chaînée de dominos.

Q8. Définir en C un type `element` permettant de stocker un domino et un pointeur vers l'élément suivant de la liste chaînée.

Q9. Définir alors un type `chaîne`.

On représente les sacs de la même manière : on construit donc le type `sac` par `typedef chaine sac`.

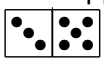
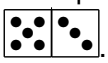
Q10. Écrire une fonction de prototype `element *ajoutElement(element *l,Domino d)` qui ajoute le domino `d` à la chaîne ou au sac `l`. Cet ajout se fera en fin de la liste chaînée. Par hypothèse, le domino n'est pas déjà dans la chaîne.

Q11. Écrire une fonction de prototype `element *retireElement(element *l,Domino d)` qui retire le domino `d` de la chaîne ou du sac `l`.

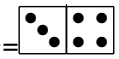
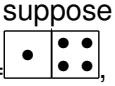
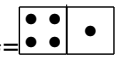
Q12. Écrire une fonction de prototype `bool rechercheElement(element *l, Domino d)` qui recherche si le domino `d` est déjà dans la chaîne ou le sac `l`. La fonction renvoie `true` si c'est le cas, `false` sinon.

III.2 - Existence d'une chaîne de taille n

Dans ce premier problème, on cherche une permutation des n dominos du sac formant une chaîne, si elle existe. Pour cela, on développe une approche de type retour sur trace (backtracking).

On suppose dans un premier temps que l'on ne peut pas effectuer de rotation des dominos. Ainsi,  ne pourra pas représenter le domino  .

Q13. Écrire une fonction de prototype `bool possible(int i,int j)` qui teste s'il est possible de placer D_i à droite de D_j. La fonction renvoie `true` si c'est le cas, `false` sinon.

On suppose maintenant qu'il est possible d'effectuer une rotation des dominos. Ainsi, si D_i=  et D_j=  , il n'est pas possible de placer directement D_j à droite de D_i, mais si on le retourne on obtient D_j=  et le placement devient possible. On souhaite donc écrire une fonction de prototype `bool possibleAvecRotation(Domino Di,Domino *Dj)`.

Q14. Pourquoi passe-t-on un pointeur sur D_j ?

Q15. Écrire la fonction `possibleAvecRotation`.

On définit une k-chaîne partielle comme une chaîne composée de $k \in [\![0, n]\!]$ dominos. Par convention, la 0-chaîne est la chaîne vide.
Un algorithme de backtracking peut alors être construit pour résoudre le premier problème.

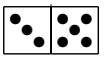
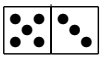
Q16. Comment passer d'une k-chaîne à une $k+1$-chaîne ?

Q17. Proposer alors une stratégie, fondée sur un principe de backtracking, permettant de rechercher, si elle existe, une chaîne qui utilise toutes les pièces.

Q18. Écrire l'algorithme correspondant en langage C.

Q19. Évaluer la complexité au pire des cas de votre algorithme.

III.3 - Nombre de chaînes de taille n

On souhaite ici compter le nombre de chaînes de taille n qu'il est possible de former à partir des dominos présents dans un sac contenant n pièces.
On considère pour $N \in \mathbb{N}$ un sac de dominos $\mathcal{S}_N$, contenant **tous** les dominos $(i, j), i, j \in [\![0, N]\!]$ sans doublons (les dominos  et  sont les mêmes).

Q20. Combien de dominos contient $\mathcal{S}_N$?

Dans la suite, on met les $N+1$ doubles de côté et on cherche à réaliser les chaînes avec les $n-N-1$ dominos restants (il suffit ensuite de rajouter les doubles à tous les emplacements possibles).
On construit un graphe non orienté $K_{N+1} = (S, A)$ où S est l'ensemble des sommets numérotés de 0 à N inclus et A est l'ensemble des arêtes, où a_{ij} est l'arête reliant le sommet i au sommet j, et représentant le domino (i, j). $\mathcal{S}$ contenant tous les dominos possibles, le graphe est complet.

Q21. Dessiner K_3.

Q22. Combien d'arêtes aboutissent à chaque sommet du graphe K_{N+1} ?

Réaliser une chaîne de dominos est équivalent à construire un chemin eulérien dans ce graphe.

Définition 2 (Chemin eulérien, cycle eulérien)
Un *chemin eulérien* dans un graphe non orienté est un chemin qui passe une fois et une seule par toutes les arêtes du graphe. Si ce chemin revient au sommet de départ, on parle de *cycle eulérien*.

On rappelle que le degré d'un sommet $s \in S$ dans un graphe est égal au nombre de sommets reliés à s par une arête.

Q23. Montrer que le nombre de sommets de degré impair d'un graphe est pair.

Il n'existe donc aucun graphe ayant un seul sommet de degré impair.
Soit alors G un graphe connexe.

Q24. Montrer qu'un chemin eulérien dans G est impossible si G comporte plus de deux sommets de degré impair.

On démontre maintenant qu'il suffit de montrer la condition réciproque sur un graphe dont tous les sommets sont de degré pair pour le prouver pour un graphe ayant deux sommets de degré impair. Supposons avoir montré qu'un graphe ayant tous ses sommets de degré pair admet au moins un chemin eulérien, et soit G un graphe connexe dont deux sommets seulement s_1 et s_2 sont de degré impair.

Q25. En construisant une arête entre s_1 et s_2, montrer qu'il existe un chemin eulérien dans G.

Dans G, graphe dont tous les sommets sont de degré pair, on considère alors le plus grand chemin C des arêtes de G, c'est-à-dire le chemin qui passe par le plus grand nombre d'arêtes du graphe. Si C contient toutes les arêtes de G, alors on a un chemin ou un cycle eulérien. Sinon G contient au moins une arête a non contenue dans C et deux cas sont possibles :

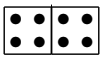
Q26. C est ouvert (ne forme pas une boucle). Démontrer par l'absurde que G ne contient pas une telle arête a.

Q27. C est fermé. Démontrer de même par l'absurde que G ne contient pas une telle arête a.

Les résultats précédents permettent alors d'affirmer qu'il n'est possible d'effectuer un chemin eulérien dans un graphe complet que si deux sommets au plus (donc 0 ou 2) sont de degré impair. Ils affirment également que si N est impair, alors il n'existe pas de chemin eulérien. Enfin, si tous les sommets sont pairs, on recherche des circuits, sinon des chemins.
Dans le reste des questions, N sera pair.

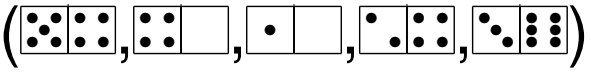
Q28. K_7 possède-t-il des chemins eulériens ? des cycles eulériens ?

Q29. En utilisant les questions 20 et 22, et en notant E_{N+1} le nombre de circuits eulériens de K_{N+1}, montrer le nombre de chaînes de taille n est égal à $\left(\binom{N+1}{2} + N + 1\right)\left(\frac{N}{2}\right)^{N+1} E_{N+1}$.

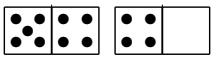
Il n'existe pas de formule connue donnant directement le nombre de circuits eulériens E_{N+1}. On trouve assez facilement $E_5 = 264$ ou encore $E_7 = 129976320$. La valeur de E_5 permet déjà d'apprécier que le nombre de chaînes des 15 dominos de  à  est égal à 126 730.

III.4 - Recherche de la plus longue sous-séquence.

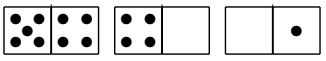
Ici, on s'intéresse à la recherche de la plus longue sous-séquence de n dominos d'un sac $\mathcal{S}$ qui forme une chaîne. Les dominos sont supposés ordonnés et ne peuvent être permutés. De plus, $\mathcal{S}$ ne contient pas nécessairement tous les dominos possibles.
Par exemple, pour le sac



la sous-séquence



est de taille maximale, égale à 2. Si on autorise les rotations de dominos, alors la plus longue sous-séquence est de taille 3 :



On aborde ce problème sous l'angle de la programmation dynamique. On suppose dans un premier temps que les rotations de dominos sont interdites.
Soit $l : \mathbb{N} \to \mathbb{N}$ la fonction qui donne la longueur de la plus longue chaîne sous-séquence de $D_1 \cdots D_i$, se terminant par le domino D_i.

Q30. Donner une formule de récurrence sur i pour la fonction l.

Q31. En déduire un algorithme en pseudo-code pour calculer la fonction l sur $[\![1, n]\!]$.

On autorise maintenant la rotation des dominos. On note l_{ij} la longueur de la plus longue chaîne sous-séquence de $D_1 \cdots D_i$, se terminant par le domino D_i, avec $j = 1$ si D_i a subi une rotation, et $j = 0$ sinon.

Q32. Donner la valeur de l_{10} et l_{11}.

Il est nécessaire de garder trace de la rotation des dominos lors du parcours du sac $\mathcal{S}$.

Q33. Proposer une modification du type structuré `Domino` permettant de prendre en compte cette contrainte.

À l'étape i, on regarde alors si D_i, tourné ou non, peut être accolé à D_{i-1}, lui aussi tourné ou non. Dans le cas d'une réponse positive, alors la longueur de la sous-séquence est incrémentée de 1. Dans le cas contraire elle est réinitialisée à 1. On retient finalement la valeur maximum entre les deux choix (D_i tourné ou non).

Q34. Proposer une formule de récurrence implémentant cette stratégie.

III.5 - Recherche du nombre de sous-séquences

Enfin, on souhaite compter le nombre de sous-séquences de taille quelconque que l'on peut former avec les dominos inclus dans un sac $\mathcal{S}$ contenant n dominos $D_i = (i_k, j_k), k \in [\![1, n]\!]$. Dans $\mathcal{S}$, un même domino (i, j) peut être présent plusieurs fois.

Définition 3 (matrice unitaire)
Pour $i, j \in [\![1, n]\!]$, on note :
- M_{ij} la matrice carrée de taille n dont le coefficient (i, j) vaut 1, les autres coefficients étant nuls.
- $(\forall i \neq j \in [\![1, n]\!]) \quad \bar{M}_{ij} = M_{ij} + M_{ji}$ et $\bar{M}_{ii} = M_{ii}$

Il est clair que $\mathcal{B} = \{\bar{M}_{ij}, i, j \in [\![1, n]\!]\}$ forme une base des matrices réelles symétriques de taille n.

Définition 4 (Produit)
Pour A, B matrices carrées de taille n, on définit le produit $A \bullet B$ par $A \bullet B = AB + BA$.

Q35. Montrer que : $(\forall i, j, k, l \in [\![1, n]\!])$

$$\bar{M}_{ij} \bullet \bar{M}_{kl} = \begin{cases} \bar{M}_{jl} & \text{si } i = k \text{ et } j \neq l \\ 2\bar{M}_{ii} + 2\bar{M}_{jj} & \text{si } i = k \text{ et } j = l \text{ et } i \neq j \\ 2\bar{M}_{ii} & \text{si } i = j = k = l \\ 0 & \text{si } \{i, j\} \cap \{k, l\} = \emptyset \end{cases}$$

Pour $n \in \mathbb{N}$ et S_n l'ensemble des permutation de $\{1 \cdots n\}$, on considère le polynôme

$$P_n(X_1 \cdots X_n) = \sum_{\sigma \in S_n} X_{\sigma(1)} \cdots X_{\sigma(n)}$$

En choisissant, pour tout $k \in [\![1, n]\!]$, $\bar{M}_k = \bar{M}_{i_k j_k}$ on peut calculer $P_n(\bar{M}_1 \cdots \bar{M}_n)$ et on sait puisque $\mathcal{B}$ est une base qu'il existe des réels α_{ij} tels que $P_n(\bar{M}_1 \cdots \bar{M}_n) = \sum_{i,j} \alpha_{ij} \bar{M}_{ij}$.

Dans la suite, on modélise le domino (i, j) du sac $\mathcal{S}$ à l'aide de la matrice $\bar{M}_{ij}$.

Q36. Pour $k \in [\![1, n]\!]$, montrer par induction sur k que les dominos $(i_1, j_1)(i_2, j_2) \cdots (i_k, j_k)$ forment une chaîne si et seulement si $\bar{M}_{i_1 j_1} \bullet \bar{M}_{i_2 j_2} \bullet \cdots \bullet \bar{M}_{i_k j_k} \neq 0$

Q37. En déduire qu'alors $\alpha_{i_k j_k} \neq 0$ si et seulement si on peut construire une chaîne commençant en i_k et terminant en j_k (ou vice-versa) en utilisant les dominos de $\mathcal{S}$.

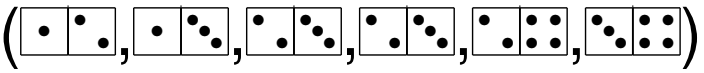
Étant donnés trois dominos $(i_1, j_1), (i_2, j_2), (i_3, j_3)$ formant une chaîne, on remarque qu'il est possible de construire cette dernière de plusieurs manières : d'abord en posant (i_1, j_1), puis (i_2, j_2) à sa droite, et enfin (i_3, j_3) à droite ; ou alors (i_2, j_2), puis (i_1, j_1) à sa gauche et (i_3, j_3) à sa droite... On note $N(k)$ le nombre de façons différentes de construire une même chaîne de longueur k.

Q38. Montrer que $N(k) = 2^{k-1}$.

Q39. Déduire des questions précédentes que, étant donné $\mathcal{S}$, il existe $\frac{\alpha_{i_k j_k}}{2^{n-1}}$ chaînes qui peuvent être construites démarrant de i_k et terminant en j_k (ou vice versa).

On reprend alors la modélisation de la sous-partie **III.3** : à partir du sac $\mathcal{S} = \{(i_k, j_k), k \in [\![1, n]\!]\}$, on construit un graphe non orienté $G = (S, A)$ où S est l'ensemble des sommets, un sommet par numéro rencontré sur les faces des dominos de $\mathcal{S}$ et A est l'ensemble des arêtes, où a_{ij} est l'arête reliant le sommet i au sommet j, et représentant le domino (i, j) de $\mathcal{S}$. Comme dans ce contexte, pour i et j fixés, le domino (i, j) peut être présent plusieurs fois dans $\mathcal{S}$, on s'autorise à faire de $G = (S, A)$ un *multigraphe* : A est un multi-ensemble d'arêtes. Ainsi, lorsque le domino (i, j) est présent plusieurs fois dans $\mathcal{S}$, l'arête correspondante apparait plusieurs fois dans le multi-ensemble A. Dans un dessin, on représente plusieurs traits entre les sommets i et j.

Q40. Dessiner le graphe correspondant au sac

$\mathcal{S} =$ 

Q41. Avec les notations précédentes, que représente $\sum_{(i,j)\in\mathcal{S}} \bar{M}_{ij}$?

Q42. En utilisant la question 39, proposer alors un moyen de calculer le nombre de chemins eulériens d'un graphe G commençant au sommet i et terminant au sommet j.

Q43. Sachant que

$$P_6(\bar{M}_{12}, \bar{M}_{13}, \bar{M}_{23}, \bar{M}_{22}, \bar{M}_{24}, \bar{M}_{34}) = 2^5(12\bar{M}_{11} + 24\bar{M}_{22} + 24\bar{M}_{33} + 12\bar{M}_{44})$$

donner le nombre de cycles eulériens du graphe de la question 40 commençant et terminant au sommet 2.

FIN