CONCOURS BLANC • MPSI/PCSI • 2022–2023

ÉPREUVE D'INFORMATIQUE COMMUNE

Durée : 2 heures

L'usage de calculatrices est interdit. — Aucun document n'est autorisé.

La présentation, la lisibilité, l'orthographe, la qualité de la rédaction, la clarté et la précision des raisonnements entreront pour une part importante dans l'appréciation des copies. En particulier, les candidats devront apporter les commentaires suffisants à la compréhension de leurs programmes et veilleront à utiliser des noms de variables explicites. 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.

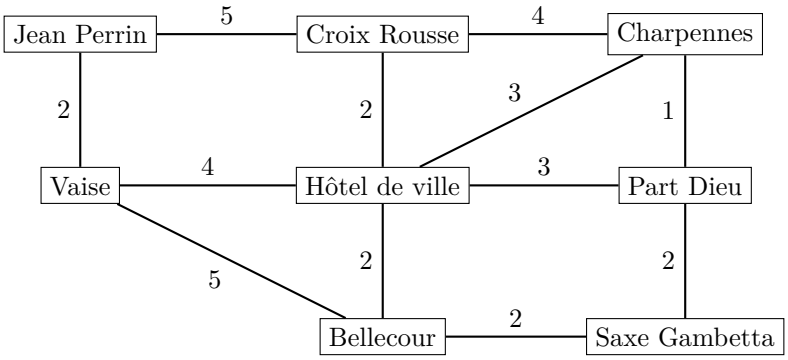
Le sujet est composé de quatre exercices indépendants.

Gestion du temps — En admettant une durée de 5 minutes pour la lecture et l'assimilation du sujet, il est vivement conseillé de consacrer environ 10 minutes sur l'exercice 1, 15 minutes sur l'exercice 2, 35 minutes sur l'exercice 3 et 55 minutes sur l'exercice 4.

Vous rédigerez sur 2 copies séparées les exercices {1, 2} et {3, 4}.

Exercice 1 – *Tous les chemins mènent à Jean Perrin*

Un étudiant arrive à la gare Part Dieu et souhaite faire le trajet jusqu'au lycée Jean Perrin. On modélise le plan de Lyon par le graphe suivant (où les distances sont indiquées en kilomètres) :



Question 1.1.

1. *Récapituler les différentes étapes de l'algorithme de Dijkstra dans un tableau.*
   On classera les colonnes par ordre alphabétique : Bellecour - Charpennes - Croix Rousse - Hôtel de ville - Jean Perrin - Part Dieu - Saxe Gambetta - Vaise.
2. *Quel est le plus court chemin de la Part Dieu au lycée Jean Perrin, et quelle est sa longueur ?*

Question 1.2. *On souhaite maintenant améliorer cet algorithme. Pour cela, on utilise l'heuristique suivante : on associe à chaque sommet sa distance au lycée.*
Les valeurs associées sont données par le tableau ci-dessous :

Bellecour	*Charpennes*	*Croix Rousse*	*Hôtel de ville*	*Jean Perrin*	*Part Dieu*	*Saxe Gambetta*	*Vaise*
6	7	5	5	0	7	7	2

1. *Recopier le graphe de Lyon, en effectuant les ajustements nécessaires sur les arêtes pour correspondre aux pseudo-poids de l'heuristique étudiée.*
2. *Récapituler les différentes étapes de l'algorithme A^* associé dans un tableau. En cas d'égalité entre les poids de deux sommets, on étudiera prioritairement celui dont l'heuristique est la plus faible.*
3. *Que peut-on conclure ?*

Exercice 2 – *La multiplication russe*

On donne ci-dessous un algorithme de multiplication de deux entiers connu sous le nom de *technique de multiplication russe.*

```
mult(x,y)
    r prend la valeur 0
    Tant que x >0
        Si x est impair
            r prend la valeur r+y
            x prend la valeur x-1
        Fin si
        x prend la valeur x/2
        y prend la valeur y*2
    Fin Tant que
    retourner r
```

Le but le l'exercice est d'étudier la terminaison, la correction et la complexité de cet algorithme.

Question 2.1. *Proposer des spécifications d'entrée et de sortie pour l'algorithme.*

Question 2.2. *Montrer que l'algorithme termine relativement aux préconditions choisies.*

Question 2.3. *On note* $x = \overline{b_{n-1}b_{n-2}\dots b_0}^2$ *le développement en binaire de* x.
On rappelle que les coefficients $(b_i)_{0\leq i\leq n-1}$ *sont définis par :*

- $\forall i \in [\![0; n-1]\!]$, $b_i \in \{0; 1\}$
- $b_{n-1} \neq 0$
- $x = \sum_{i=0}^{n-1} b_i 2^i$

On note de plus x_0 *et* y_0 *les valeurs initiales des paramètres* x *et* y.

1. *Montrer que l'invariant suivant est vérifié à la fin du* $k^{\text{ème}}$ *passage dans la boucle :*

$$\begin{cases} x &= \sum_{i=k}^{n-1} b_i 2^{i-k} \\ y &= 2^k y_0 \\ r &= \left(\sum_{i=0}^{k-1} b_i 2^i\right) y_0 \end{cases}$$

(On supposera par convention qu'une somme sur un ensemble d'indices vide est égale à 0.)

2. *En déduire la correction de l'algorithme.*

Question 2.4. *Donner une estimation approximative de la complexité de l'algorithme. On supposera que toutes les opérations élémentaires, comme l'addition et la multiplication des entiers, ont un coût constant.*

Exercice 3 – *Persistance multiplicative d'un entier*

Certains problèmes, qu'on explique en quelques secondes, semblent hors de portée des raisonnements abstraits et des capacités de calcul des plus puissants ordinateurs. Le problème de la persistance multiplicative des nombres en base 10, d'énoncé très simple, est l'une de ces redoutables énigmes où le blocage est total.

Considérons un nombre entier positif, par exemple 377. Multiplions ses chiffres : $3 \times 7 \times 7 = 147$. Opérons de même avec le résultat 147 : $1 \times 4 \times 7 = 28$. Recommençons : $2 \times 8 = 16$. Encore : $1 \times 6 = 6$. Arrivé à un nombre d'un seul chiffre, on ne peut plus rien faire : $377 \to 147 \to 28 \to 16 \to 6$.

Cette suite est la « suite multiplicative » de 377 et la « persistance multiplicative » p de 377 est le nombre de fois qu'il a fallu multiplier les chiffres avant d'arriver à un nombre à un seul chiffre ; ici, $p = 4$.

Question 3.1. *Écrire une fonction* `chiffres` *qui prend en argument un entier naturel* n *et qui renvoie la liste des chiffres composant l'écriture décimale d'un entier naturel donné* n.
Exemple : `chiffres(145)` *renverra* `[1,4,5]`.
Les méthodes nécessitant la conversion de n en chaine de caractères sont interdites.

Question 3.2. *Écrire une fonction* `prod` *qui prend en argument un entier naturel* n, *et qui renvoie le produit des chiffres composant* n *(écrit en base 10). Par exemple, si* $n = 128$, *alors* `prod(n)` *vaut 16.*

Question 3.3. *Implémenter une fonction* `suite` *qui prend en argument un entier naturel n, et qui renvoie la liste des valeurs de la « suite multiplicative » de n.*

Question 3.4. *Écrire une version itérative de la fonction* `persistance_iter` *qui prend en argument un entier naturel n, et qui renvoie la « persistance multiplicative » de n, sans utiliser la fonction* `suite` *ni stocker les différentes valeurs de la « suite multiplicative ».*

Question 3.5. *Proposer une version récursive* `persistance_rec` *de la fonction précédente.*

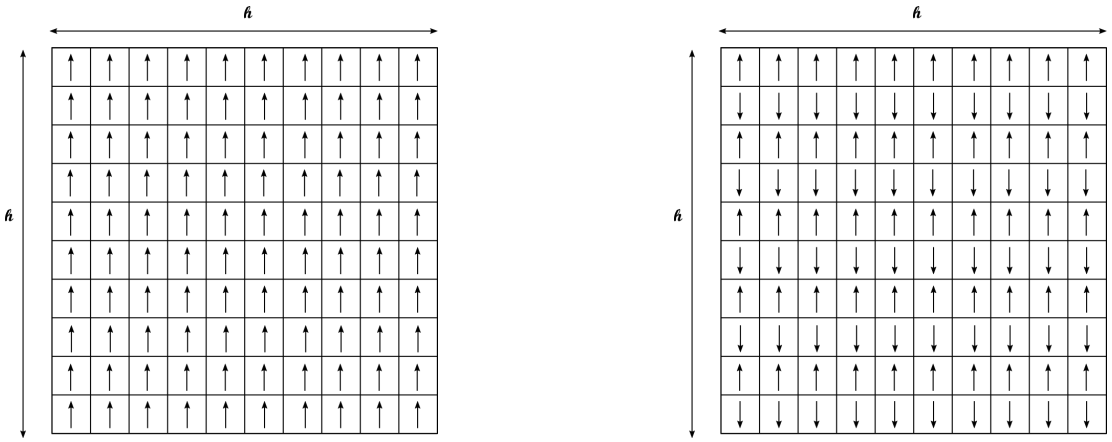
Jusqu'à présent, ni le raisonnement ni le calcul par ordinateur n'ont permis de trouver d'entiers ayant une persistance supérieure à 11.

Question 3.6. *Implémenter une fonction* `max_persistance` *qui prend en argument un entier naturel n, et qui renvoie le couple (j, p) où p est la valeur maximale de la « persistance multiplicative » dans l'intervalle $[\![1, n]\!]$ et j le plus petit entier pour lequel cette valeur de persistance est atteinte.*

Question 3.7. *Écrire une fonction* `persistances`*, prenant en argument un entier n, et qui revoie un dictionnaire dont les clés sont les entiers de l'intervalle $[\![1, n]\!]$. La valeur associée est le plus petit entier dont la persistance multiplicative est égale à la valeur de la clé.*

Exercice 4 – Modèle microscopique d'un matériau magnétique

Pour étudier l'effet du champ magnétique sur un matériau magnétique, on adopte une modélisation microscopique. On modélise les atomes par des sites portant chacun une grandeur physique, nommée *spin*, dont il n'est pas nécessaire de connaître les propriétés. L'échantillon modélisé est une zone carrée à deux dimensions possédant h *spins* régulièrement répartis dans chaque direction, donc formant une grille carrée de $n = h^2$ *spins*. Chaque *spin* ne possède que deux états *down* ou *up*, ce que l'on modélise par une variable $s_i \in \{-1, +1\}$.



(a) matériau ferromagnétique (b) matériau antiferromagnétique

FIGURE 1 – Modèles des *spins* dans un matériau ferromagnétique (a) et antiferromagnétique (b).

Pour implémenter ces configurations de *spins* décrivant l'état microscopique du matériau, on choisit de travailler sur une liste `s`, contenant $n = h^2$ entiers, chacun valant -1 ou 1. Un domaine d'aimantation uniforme (cf. figure 1a) sera donc représenté par une liste contenant n fois 1 (`[1,1,1,....1]`). Contrairement aux matériaux ferromagnétiques, dans les matériaux antiferromagnétiques, l'interaction d'échange entre les atomes voisins conduit à un alignement antiparallèle des moments magnétiques atomiques (cf. figure 1b). L'aimantation totale du matériau est alors nulle.

Question 4.1. *Définir une fonction* `initialisation` *qui prend comme argument un entier associé à une taille de grille* `h` *et un booléen* `f` *associé au ferromagnétisme (*`f` *est faux pour un matériau antiferromagnétique) et qui renvoie une liste de n entiers 1 ou -1.*

Question 4.2. *Définir une fonction* `carre` *qui prend comme argument une liste de $n = h^2$* spins `s` *et qui renvoie l'entier correspondant à h.*

Question 4.3. *Définir une fonction récursive* `aimantation` *qui prend comme argument une liste de* spins `s` *et qui renvoie le flottant correspondant à son aimantation moyenne.*

Dans la modélisation adoptée (sans champ magnétique extérieur), l'énergie d'une configuration, définie par l'ensemble des valeurs de tous les *spins*, est donnée par :

$$E = -\frac{J}{2} \sum_i \sum_{j \in V_i} s_i s_j$$

avec V_i l'ensemble des quatre voisins du *spin* i, noté s_i, situés juste au dessous, en dessous, à gauche et à droite de s_i (les seuls supposés capables d'interagir avec lui). J est l'intégrale d'échange qui modélise l'interaction entre deux *spins* voisins.

Malgré le caractère fini de l'échantillon, on peut utiliser une modélisation très utile pour faire comme s'il était infini en utilisant les conditions aux limites périodiques. Lorsque l'on considère un *spin* dans la colonne située la plus à droite (resp. gauche), il ne possède pas de plus proche voisin à droite (resp. gauche) : on convient de lui en affecter un, qui sera situé sur la même ligne complètement à gauche (resp. droite) de l'échantillon. De même, le plus proche voisin manquant d'un *spin* situé sur la première (resp. dernière) ligne sera situé sur la dernière (resp. première) ligne de l'échantillon (voir la figure 2).

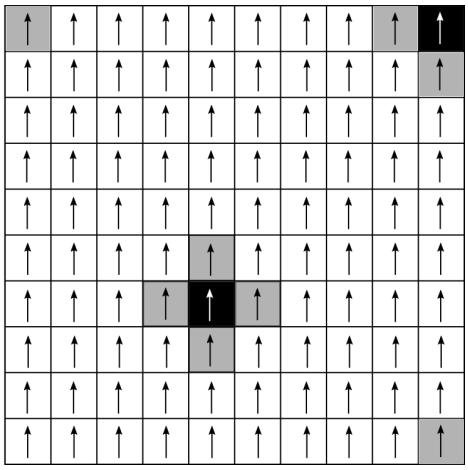


FIGURE 2 – Voisinage d'un *spin* (les voisins des *spins* sur les cases noires sont indiqués en gris).

Question 4.4. *Définir une fonction* `voisins` *qui prend comme arguments une liste de* spins `s` *et un indice de* spin `i` *et qui renvoie le quadruplet d'indices des* spins *voisins dans l'ordre gauche, droite, dessous et dessus.*

Question 4.5. *Définir une fonction* `energie` *qui prend comme argument une liste de* spins `s` *et comme argument optionnel une valeur de l'intégrale d'échange* `J` *(1 par défaut) et qui renvoie le flottant correspondant à son énergie E.*

Pour trouver une configuration stable pour les *spins*, il faut faire évoluer la liste vers une situation d'équilibre conformément aux principes de la physique statistique. On adopte une méthode probabiliste connue sous le nom de méthode de Monte-Carlo, dont le principe de fonctionnement est le suivant. À chaque étape :

— on choisit un *spin* au hasard dans l'échantillon ;

— on calcule la variation d'énergie ΔE qui résulterait d'un changement d'orientation de ce *spin* ;

— si $\Delta E \leqslant 0$, ce *spin* change de signe ;

— si $\Delta E > 0$, ce *spin* change de signe avec la probabilité donnée par la loi de Boltzmann :

$$p = \exp\left(\frac{-\Delta E}{k_B\, T}\right)$$

où T est la température (en K) et $k_B = 1{,}380\,649 \times 10^{-23}\ \mathrm{J{\cdot}K^{-1}}$ la constante de Boltzmann. Dans la suite, ΔE et le produit $k_B\, T$ seront désignées par les variables `E` et `T` et la fonction exponentielle prise dans la bibliothèque `math`. On pourra utiliser la fonction `random` de la bibliothèque `random` pour tirer un flottant dans l'intervalle $[0;1]$ avec une densité de probabilité uniforme et la fonction `randrange` sous la forme `random.randrange(n)` pour tirer un entier dans $[\![0, n-1]\!]$.

Question 4.6. *Définir une fonction* `boltzmann` *qui prend comme arguments deux flottants* `E` *et* `T` *et qui renvoie le booléen correspondant au changement de signe du* spin.

Juste après avoir sélectionné au hasard l'indice `i` d'un *spin* de la liste `s` à basculer éventuellement, pour évaluer l'écart d'énergie `E` entre les deux configurations avant/après, on propose deux solutions sous forme des deux fonctions ci-dessous.

```
def calcul_delta_e1(s, i):
    t = s[:]
    t[i] = -s[i]
    delta_e = energie(t)-energie(s)
    return(delta_e)
```

```
def calcul_delta_e2(s, i):
    delta_e = 0
    for j in voisins(s, i):
        delta_e = delta_e + 2*s[i]*s[j]
    return(delta_e)
```

où `s[i]` est le *spin* choisi pour être éventuellement retourné.

Question 4.7. *Indiquer la solution qui vous paraît la plus efficace pour minimiser le temps de calcul en justifiant votre réponse. Proposez éventuellement une optimisation de la fonction choisie.*

Question 4.8. *Définir une fonction* `MonteCarlo` *qui prend comme arguments une liste de* spins `s`*, un flottant* `T` *et un entier* `n` *correspondant au nombre de* spins *choisis au hasard et qui applique la méthode de Monte-Carlo en suivant les règles données.*

Une observation microscopique des matériaux magnétiques nous apprend que les zones magnétiques du matériau sont organisées en domaines, nommés domaines de Weiss. Dans un domaine de Weiss donné, tous les *spins* ont la même valeur.

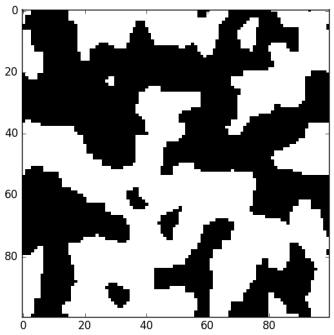


FIGURE 3 – Représentation des domaines de Weiss.

On souhaite dans la suite décrire les différents domaines de Weiss afin, par exemple, de les colorier d'une manière différente. Pour cela, on va construire une liste possédant exactement la même taille que `s` contenant initialement des -1 (cette valeur signifiant que le *spin* correspondant n'a encore été affecté à aucun domaine de Weiss).

Question 4.9. *Définir une fonction* `exploration` *qui prend comme arguments une liste de* spins `s`*, une liste de Weiss* `w`*, un indice* spin *de départ* `i` *et un entier* `n` *correspondant au numéro du* n^e *domaine de Weiss et qui modifie la liste* `w` *pour que tous les indices des* spins *du domaine de Weiss contiennent l'entier* `n`*.*

Question 4.10. *Définir une fonction* `weiss` *qui prend comme argument une liste de* spins `s` *et qui renvoie une liste d'entiers de même longueur correspondants aux domaines de Weiss de chaque* spin*.*

— Fin de l'énoncé —