Concours Commun INP (CCINP), France

# Session 2018, filière MP

## Informatique

Durée : 4 heures

### Ce que traite le sujet

- Partie I : logique et calcul des propositions
- Partie II : automates
- Partie III : algorithmique et programmation

**Sujet reproduit sans retouche à partir de la page suivante.**

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/mp/2018/info

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

CONCOURS COMMUNS
POLYTECHNIQUES

**ÉPREUVE SPÉCIFIQUE - FILIÈRE MP**

---

# INFORMATIQUE

**Jeudi 3 mai : 14 h - 18 h**

---

*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.*

---

**Les calculatrices sont interdites**

**Le sujet est composé de trois parties, toutes indépendantes.**

# Partie I - Logique et calcul des propositions

Vous avez été sélectionné(e) pour participer au jeu "Cherchez les Clés du Paradis (CCP)". Le jeu se déroule en trois épreuves, au cours desquelles vous devez collecter des clés vertes. À l'issue de chacune d'entre elles, vous passez à l'épreuve suivante en cas de succès et êtes éliminé(e) en cas d'échec.

### I.1 - Première épreuve

Jean-Pierre Pendule, le célèbre animateur, vous accueille pour la première épreuve. Il vous explique la règle du jeu. Devant vous, deux boîtes et sur chacune d'entre elles une inscription. Chacune des boîtes contient soit une clé verte, soit une clé rouge. Vous devez choisir l'une des boîtes : si le résultat est une clé rouge, alors vous quittez le jeu, si c'est une clé verte vous êtes qualifié(e) pour l'épreuve suivante.
Jean-Pierre Pendule dévoile les inscriptions sur chacune des boîtes et vous affirme qu'elles sont soit vraies toutes les deux, soit fausses toutes les deux :

— sur la boîte 1, il est écrit : "Une au moins des deux boîtes contient une clé verte" ;
— sur la boîte 2, il est écrit : "Il y a une clé rouge dans l'autre boîte".

Dans toute cette partie, on note $P_i$ la proposition affirmant qu'il y a une clé verte dans la boîte $i$.

**Q1.** Donner une formule de la logique des propositions représentant la phrase écrite sur la boîte 1.

**Q2.** Donner de même une formule de la logique des propositions pour l'inscription de la boîte 2.

**Q3.** Donner une formule représentant l'affirmation de l'animateur. Simplifier cette formule de sorte à n'obtenir qu'une seule occurrence de chaque $P_i$.

**Q4.** Quel choix devez-vous faire pour continuer le jeu à coup sûr ?

### I.2 - Deuxième épreuve

Bravo, vous avez obtenu la première clé verte. Jean-Pierre Pendule vous félicite et vous annonce que cette première épreuve n'était qu'une mise en jambe. Avec les mêmes règles du jeu, l'animateur vous propose alors deux nouvelles boîtes portant les inscriptions suivantes :

— sur la boîte 1, il est écrit : "Il y a une clé rouge dans cette boîte, ou bien il y a une clé verte dans la boîte 2" ;
— sur la boîte 2, il est écrit : "Il y a une clé verte dans la boîte 1".

**Q5.** Donner une formule de la logique des propositions pour chaque affirmation.

**Q6.** Sachant qu'encore une fois les deux affirmations sont soit vraies toutes les deux, soit fausses toutes les deux, donner le contenu de chaque boîte. En déduire votre choix pour remporter la deuxième clé verte.

### I.3 - Troisième épreuve

Le suspens est à son comble, vous voici arrivé(e) à la dernière épreuve. À votre grande surprise, Jean-Pierre Pendule vous dévoile une troisième boîte et vous explique les règles du jeu. Dans une des boîtes se cache la clé verte qui vous permet de remporter la victoire finale. Dans une autre boîte se cache une clé rouge qui vous fait tout perdre. La dernière boîte est vide. Encore une fois, chacune des boîtes porte une inscription :

— sur la boîte 1, il est écrit : "La boîte 3 est vide" ;
— sur la boîte 2, il est écrit : "La clé rouge est dans la boîte 1" ;
— sur la boîte 3, il est écrit : "Cette boîte est vide".

L'animateur affirme que l'inscription portée sur la boîte contenant la clé verte est vraie, celle portée par la boîte contenant la clé rouge est fausse. L'inscription affichée sur la boîte vide est aussi vraie.

**Q7.** Donner une formule de logique des propositions pour chaque inscription.

**Q8.** Donner une formule de logique des propositions synthétisant l'information que vous a apportée l'animateur.

**Q9.** En supposant que la clé verte est dans la boîte 2, montrer par l'absurde que l'on aboutit à une incohérence.

**Q10.** Donner alors la composition des trois boîtes.

## Partie II - Automates

Un langage est régulier si et seulement si il est accepté par un automate fini (en particulier déterministe). Cependant, plusieurs automates peuvent accepter le même langage. L'objectif de cette partie est de montrer que tout langage reconnaissable $L$ est reconnu par un unique (au renommage près des états) automate déterministe, tel que tout automate déterministe reconnaissant $L$ a au moins autant d'états que lui. Cet unique automate est appelé automate minimal reconnaissant $L$.

### II.1 - Définitions

**Définition 1.** Automate déterministe
Un automate déterministe $A$ est un quintuplet $A = (Q, \Sigma, q_0, F, \delta)$, avec :

— $Q$ un ensemble d'états,
— $\Sigma$ un alphabet,
— $q_0$ l'état initial,
— $F \subseteq Q$ un ensemble d'états finaux,
— $\delta : Q \times \Sigma \to Q$ une application de transition définie sur $Q \times \Sigma$ tout entier.

**Définition 2.** Soit $\Sigma$ un alphabet. $\Sigma^*$ est l'ensemble des mots construits à partir de $\Sigma$, $\Lambda$ le mot vide et $|u|_a$ le nombre d'occurrences de la lettre $a \in \Sigma$ dans le mot $u$.

**Définition 3.** Résiduel d'un langage par rapport à un mot
Soient $\Sigma$ un alphabet, $L \subseteq \Sigma^*$ un langage et $u \in \Sigma^*$. Le résiduel à gauche de $L$ par rapport à $u$ est le langage :

$$u^{-1}L = \{v \in \Sigma^*, uv \in L\}.$$

**Q11.** Pour $L = \{ab, ba, aab\}$, donner le résiduel de $L$ par rapport à $a$.

Définissons alors une relation $\sim_L$ sur $\Sigma^*$, dite congruence de Nerode, de la manière suivante : pour tous $u, v \in \Sigma^*$ :

$$u \sim_L v \Leftrightarrow u^{-1}L = v^{-1}L.$$

**Q12.** Montrer que $\sim_L$ est une relation d'équivalence et que : $\forall u, v, w \in \Sigma^* \quad (u \sim_L v) \Rightarrow (uw \sim_L vw)$.

**Q13.** Posons $L = \{u \in \Sigma^*, |u|_a = 0 \bmod 3\}$ le langage basé sur $\Sigma = \{a, b\}$, composé des mots de $\Sigma^*$ ayant un nombre de $a$ multiple de 3. Pour chacun des cas suivants, déterminer si les deux mots sont équivalents par $\sim_L$ :

(i). $b$ et $ab$,

(ii). $aba$ et $bab$,

(iii). $abbaba$ et $aaa$.

**Définition 4.** Soit $A = (Q, \Sigma, q_0, F, \delta)$ un automate déterministe. La fonction de transition $\delta^*$ étendue aux mots est définie de manière récursive par :

— $\forall q \in Q \quad \delta^*(q, \Lambda) = q$,

— $\forall a \in \Sigma, \forall m \in \Sigma^*, \forall q \in Q \quad \delta^*(q, m.a) = \delta(\delta^*(q, m), a)$.

**Définition 5.** Soit $A = (Q, \Sigma, q_0, F, \delta)$ un automate déterministe. Pour $q \in Q$ et $G \subseteq Q$, on note :

$$q^{-1}G = \{u \in \Sigma^*, \delta^*(q, u) \in G\}.$$

$q^{-1}G$ est donc l'ensemble des mots qui correspondent à des chemins débutant en $q$ et aboutissant dans un état de $G$.
Une relation d'équivalence $\sim_A$ est également définie sur $Q$ par :

$$p \sim_A q \Leftrightarrow p^{-1}F = q^{-1}F.$$

Si $A = (Q, \Sigma, q_0, F, \delta)$ est un automate déterministe acceptant un langage $L = q_0^{-1}F$ et si $q \in Q$ et $u \in \Sigma^*$ sont tels que $\delta^*(q_0, u) = q$ alors on peut montrer que $q^{-1}F = u^{-1}L$. Ainsi, la congruence de Nerode peut être utilisée pour définir un automate particulier, appelé automate minimal de $L$.

**Définition 6.** Automate minimal
Soit $L$ un langage. L'automate minimal de $L$ est défini par le quintuplet $A_L = (Q_L, \Sigma, q_{L_0}, F_L, \delta_L)$, avec :

— $Q_L = \left\{u^{-1}L, u \in \Sigma^*\right\}$,

— $q_{L_0} = \Lambda^{-1}L = L$,

— $F_L = \left\{u^{-1}L, u \in L\right\} = \{q \in Q_L, \Lambda \in q\}$,

— $\forall q \in Q_L, \forall a \in \Sigma \quad \delta_L(q, a) = a^{-1}q$.

On admettra qu'un automate minimal définit bien un automate.

**Q14.** Montrer que l'automate minimal d'un langage régulier $L \subseteq \Sigma^*$ est un automate fini, c'est-à-dire un automate possédant un nombre fini d'états.

**Définition 7.** Automate accessible
Un automate déterministe $A = (Q, \Sigma, q_0, F, \delta)$ est accessible si pour tout $q \in Q$, il existe $u \in \Sigma^*$, tel que $\delta^*(q_0, u) = q$.

**Définition 8.** Automate réduit
Un automate déterministe $A = (Q, \Sigma, q_0, F, \delta)$ est réduit si pour tous $p, q \in Q, (p^{-1}F = q^{-1}F) \Rightarrow p = q$.
$A$ est donc réduit si les langages acceptés depuis deux états distincts sont distincts, ou encore si chaque classe d'équivalence pour la relation $\sim_A$ sur $Q$ est un singleton.

Pour $L \subset \Sigma^*$, il est possible de montrer que l'automate minimal $A_L$ de $A$ est accessible et réduit. Le paragraphe suivant s'intéresse à la construction de l'automate minimal d'un automate $A$ donné, exploitant cette propriété.

### II.2 - Construction de l'automate minimal

Soit $A = (Q, \Sigma, q_0, F, \delta)$ un automate déterministe acceptant le langage $L \in \Sigma^*$. Trouver l'automate minimal $A_L$ de $A$ revient à trouver un automate fini déterministe accessible et réduit équivalent.
Pour trouver un automate accessible, il suffit par exemple de visiter les états qui peuvent être atteints par $A$ depuis $q_0$ et d'éliminer les autres états. Il reste donc à trouver une méthode pour rendre $A$ réduit.
Par définition de la relation $\sim_A$ sur $Q$, $A$ est réduit si pour tout couple $(p, q) \in Q^2$, avec $p \neq q$, $p \nsim_A q$.

En particulier, $p \not\sim_A q$ s'il existe $u \in \Sigma^*$, tel que $\delta(p,u) \in F$ et $\delta(q,u) \notin F$ ou $\delta(q,u) \in F$ et $\delta(p,u) \notin F$. On dit alors que $u$ distingue $p$ et $q$ ou que le couple $(p,q)$ est distingué par $u$.
L'algorithme 1 est un algorithme de réduction d'un automate $A$ utilisant ces notions. Dans la suite, $N_k$ désigne l'ensemble des couples d'états de $Q$ qui sont distingués par un mot de longueur $k$ et qui ne sont distingués par aucun autre mot plus court.

**Algorithme 1:** Algorithme de recherche des états équivalents

**Entrée** : un automate déterministe $A = (Q, \Sigma, q_0, F, \delta)$
**Sortie** : les ensembles d'états équivalents
$k \leftarrow 0$
********** Initialisation **********
$N_0 \leftarrow \emptyset$
**pour** *tous* $p \in F$ *et* $q \in Q \setminus F$ **faire**
    ***La paire (p,q) est distinguée.***
    $N_0 \leftarrow N_0 \cup \{(p,q)\}$
**tant que** $N_k \neq \emptyset$ **faire**
    ****Construction de $N_{k+1}$****
    $N_{k+1} \leftarrow \emptyset$
    **pour** *chaque paire* $(p,q) \in N_k$ **faire**
        **pour** *chaque* $a \in \Sigma$ **faire**
            **pour** *chaque* $(r,s) \in Q^2$ *tel que* $\delta(r,a) = p$, $\delta(s,a) = q$ **faire**
                **si** $(r,s) \notin \bigcup_{i=0}^{k} N_i$ **alors**
                    $N_{k+1} \leftarrow N_{k+1} \cup \{(r,s),(s,r)\}$
    $k \leftarrow k+1$

**Q15.** Montrer pourquoi, dans la phase d'initialisation, la paire $(p,q)$ est distinguée.

**Q16.** Montrer pourquoi, si $N_i = \emptyset$ alors $\forall j > i, N_j = \emptyset$.

Soit alors $A = (Q, \Sigma, q_0, F, \delta)$ l'automate déterministe suivant :
— $Q = \{1,2,3,4,5,6\}$
— $\Sigma = \{a,b\}$
— $q_0 = \{4\}$
— $F = \{2,5\}$
— $\delta$ :

| | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|
| **a** | 2 | 4 | 6 | 5 | 1 | 6 |
| **b** | 1 | 3 | 2 | 1 | 6 | 2 |

où $\delta(i,j)$ représente donc l'état atteint à partir de l'état $i$ lorsque le symbole $j$ est présenté.

**Q17.** Représenter graphiquement l'automate $A$. La représentation suivra les contraintes suivantes :
— les états sont des cercles, le nom de l'état est écrit à l'intérieur du cercle ;
— les transitions sont représentées par des flèches partant de l'état de départ et pointant sur l'état d'arrivée. Le symbole définissant la transition est indiqué au milieu de la flèche ;
— l'état initial est signalé par une flèche sans étiquette pointant sur cet état ;
— les états finaux sont entourés d'un deuxième cercle, externe au premier.

**Q18.** Appliquer l'algorithme 1 pour trouver l'ensemble des états équivalents. Pour chaque itération $k$, la trace de l'algorithme sera donnée par une matrice carrée $T$ de taille $|Q| \times |Q|$, avec $T(i, j)$ = longueur d'un chemin, s'il existe, qui distingue le couple $(i, j)$ ($T(i, j)$ vide sinon). En déduire les classes d'équivalence des états de $A$.

Un théorème, non détaillé ici, permet alors de projeter $A$ sur $A_L$ et de préciser états et transitions de cet automate minimal. Il permet en particulier de définir les états de $A_L$ comme étant les classes d'équivalence issues de l'algorithme précédent. Il permet également d'affirmer que si un état de $A_L$ correspond à une classe d'équivalence $[q]_A$ pour la relation $\sim_A$, alors la lecture d'un symbole $a \in \Sigma$ depuis cet état dans $A_L$ conduit à l'état correspondant à la classe $[\delta(q, a)]_A$.

**Q19.** Représenter graphiquement l'automate minimal de la question précédente, avec ses états et ses transitions.

## Partie III - Algorithmique et programmation

Nous proposons dans cette partie d'étudier une méthode de compression de données. L'algorithme proposé ici implémente plusieurs couches d'arrangement de données et de compression successives, utilisées dans l'ordre suivant pour la compression et l'ordre inverse pour la décompression :

(i). Transformation de Burrows-Wheeler (BWT),

(ii). Codage par plages (RLE),

(iii). Codage de Huffman.

**Définition 9.** Soit $\Sigma$ un alphabet de symboles, de cardinal $|\Sigma| = h$. On munit $\Sigma$ d'une relation d'ordre $\leq$. $\Sigma^k$ est l'ensemble des mots de longueur $k$ construits à partir de $\Sigma$. $\Sigma^k$ est muni d'une relation d'ordre lexicographique induite par la relation d'ordre $\leq$.
Pour $\mu \in \Sigma^k$, on note $|\mu| = k$ la taille de $\mu$ et $|\mu|_a$ le nombre d'occurrences de $a \in \Sigma$ dans $\mu$.

Dans toute cette partie, lorsqu'il s'agira de coder une fonction CAML, un mot $\mu \in \Sigma^k$ sera représenté par une liste de caractères en CAML (`char list`).

Les paragraphes suivants étudient les algorithmes et propriétés de ces phases, chacun d'entre eux pouvant être abordé **de manière indépendante**.

### III.1 - Transformation de Burrows-Wheeler (BWT)

Soit $\mu \in \Sigma^k$ un mot. La transformation BWT réalise une permutation des symboles de $\mu$ de sorte que les symboles identiques sont regroupés dans de longues séquences. Cette transformation n'effectue pas de compression, mais prépare donc à une compression plus efficace.
Dans la suite, nous étudions le codage et le décodage d'un mot transformé par cette opération.

**– Phase de codage –**

On rajoute à la fin de $\mu$ un marqueur de fin (par convention noté |, inférieur par $\leq$ à tous les autres symboles de $\Sigma$). Dans toute la suite $\hat{\mu}$ désigne le mot auquel on a ajouté le symbole |.

**Q20.** Pour $\mu = turlututu$, construire une matrice $M$ dont les lignes sont les différentes permutations circulaires successives du mot $\hat{\mu}$. Les permutations seront ici envisagées par décalage à droite des caractères.

**Q21.** Écrire une fonction récursive CAML `circulaire : 'a list -> 'a list` qui réalise une permutation à droite d'un mot $\mu$ donné en entrée.

**Q22.** Écrire une fonction CAML `matrice_mot : 'a list -> 'a list list` qui construit la matrice $M$ à partir d'un mot passé en entrée. La valeur de retour est une liste de liste de symboles (une liste de mots). Cette fonction utilisera la fonction `circulaire : 'a list -> 'a list`.

Une permutation des lignes de $M$ est alors effectuée, de sorte à classer les lignes par ordre lexicographique. On note $M' = P.M$ la matrice obtenue, $P$ étant la matrice de permutation.

**Q23.** Donner les matrices $P$ et $M'$ dans le cas du mot $\hat{\mu}$, pour $\mu = turlututu$.

Pour construire la matrice de permutation $P$, il faut trier la liste des mots définissant $M$. La méthode de tri choisie ici est le tri par insertion.

**Q24.** Écrire une fonction récursive CAML `tri : 'a list -> 'a list` qui réalise le tri par insertion d'une liste d'éléments.

**Q25.** En déduire une fonction `matrice_mot_triee : 'a list -> 'a list list` qui construit $M'$ à partir de $M$.

**Q26.** Pour $\mu \in \Sigma^k$, donner le nombre de comparaisons de symboles nécessaires au pire des cas, pour trier deux permutations circulaires du mot $\mu$.

**Q27.** En déduire la complexité dans le pire des cas pour le tri des $k$ permutations circulaires d'un mot $\mu \in \Sigma^k$ (exprimée en nombre de comparaisons de symboles).

La transformation BWT consiste alors à coder le mot $\mu$ par la dernière colonne de la matrice $M'$ obtenue à l'aide de $\hat{\mu}$. On note $BWT(\mu)$ ce codage.

**Q28.** Écrire alors une fonction `codageBWT : char list -> char list` qui encode un mot passé en entrée. On utilisera une fonction récursive permettant de récupérer le dernier symbole de chacun des mots de $M'$. Donner le codage du mot $\mu = turlututu$.

**– Phase de décodage –**

Pour décoder un mot codé par BWT, il est nécessaire de reconstruire itérativement $M'$ à partir de la seule donnée du mot codé $BWT(\mu)$. Par construction, $BWT(\mu)$ est la dernière colonne de $M'$.
On pose ici comme exemple $BWT(\mu) = ednngvnea|$.

**Q29.** Construire, à partir de la seule donnée de $BWT(\mu)$, la première colonne de $M'$. Justifier le principe de construction.

La dernière et la première colonne de $M'$ donnent alors tous les sous-mots de longueur 2 du mot $\mu$.

**Q30.** Proposer un algorithme permettant d'obtenir la deuxième colonne de $M'$. Donner cette deuxième colonne pour $BWT(\mu) = ednngvnea|$.

**Q31.** On dispose à l'itération $n$ des $(n-1)$ premières colonnes de $M'$ et de sa dernière colonne. Proposer un principe algorithmique permettant de construire la $n$-ème colonne de $M'$.

**Q32.** En déduire un algorithme itératif permettant de reconstruire $M'$.

**Q33.** Quel décodage obtient-on pour le mot $BWT(\mu)$ proposé ?

### III.2 - Codage par plages RLE [Informatique pour tous]

Le codage RLE (Run Length Coding), ou codage par plages, est une méthode de compression dont le principe est de remplacer dans une chaîne de symboles une sous-chaîne de symboles identiques par le

couple constitué du nombre de symboles identiques et du symbole lui-même. Par exemple, la chaîne "aaababb" est compressée en [(3,'a'),(1,'b'),(1,'a'),(2,'b')].

**Q34.** Proposer un type naturel Python pour la compression RLE, qui permet de représenter le résultat comme indiqué précédemment.

**– Phase de codage –**

On s'intéresse tout d'abord au codage RLE d'un mot.

**Q35.** Écrire une fonction itérative en Python `def RLE (mot):` qui code un mot passé en entrée par codage RLE.

**– Phase de décodage –**

On s'intéresse maintenant au décodage d'une liste.

**Q36.** Écrire une fonction itérative en Python `def decodeRLE (codeRLE):` qui décode une liste`codeRLE` issue du codage RLE d'un mot.

### III.3 - Codage de Huffman [Informatique pour tous]

La dernière étape de l'algorithme proposé implémente le codage de Huffman, qui utilise la structure d'arbre binaire. Le principe est de coder un symbole de manière d'autant plus courte que son nombre d'occurrences dans le mot est élevé. L'arbre de Huffman se construit à l'aide de l'algorithme 2.

**Algorithme 2:** Codage de Huffman

**Entrée** : $\mu$ un mot de taille $|\mu|$
**Sortie** : `Huffman`($\mu$) le codage de Huffman de $\mu$
**pour** $a \in \Sigma$ **faire**
  **si** $|\mu|_a > 0$ **alors**
    créer un noeud $(a, |\mu|_a)$
$\mathcal{L} \leftarrow$ liste des noeuds dans l'ordre croissant des poids
$\mathcal{A} \leftarrow$ liste vide
**tant que** *(longueur($\mathcal{L}$)+longueur ($\mathcal{A}$)>1)* **faire**
  $(g, d) \leftarrow$ deux noeuds de plus faible poids parmi les 2 premiers noeuds de $\mathcal{L}$ et les 2 premiers noeuds de $\mathcal{A}$
  Créer un noeud $t$
  $n_t \leftarrow n_g + n_d$
  $gauche(t) \leftarrow g$
  Coder la branche de $t$ à $g$ par 0
  $droite(t) \leftarrow d$
  Coder la branche de $t$ à $d$ par 1
  Insérer $t$ à la fin de $\mathcal{A}$
  Retirer $g$ et $d$ de $\mathcal{L}$ ou de $\mathcal{A}$
`Huffman`($\mu$)$\leftarrow \mathcal{A}$

**Q37.** Construire le codage de Huffman du mot $\mu$ = "*turlututu*" en utilisant l'algorithme 2. Vous expliciterez par des dessins d'arbres chacune des étapes de construction de `Huffman`($\mu$).

**Q38.** Quelle est la forme de l'arbre de Huffman dans un mot où tous les symboles ont le même nombre d'occurrences ?

**FIN**



IMPRIMERIE NATIONALE – 18 1057 – D'après documents fournis

# Travailler les chapitres de ce sujet

Prepa Help suit le programme de MP 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

- Logique, ensembles et applications
- Langage OCaml : expressions, types, filtrage, récursivité
- Tris par sélection, à bulles, par insertion et par comptage
- Chaînes de caractères et algorithmes sur les textes
- Complexité des algorithmes itératifs et récursifs

## 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/mp