Banque MP inter-ENS – Session 2021

Rapport de jury de l'épreuve orale d'informatique fondamentale LCR

Membres du jury : V. Jugé, S. Le Roux, A. Pouly et F. Prost

Écoles partageant cette épreuve : ENS de Lyon, ENS Paris-Saclay, ENS Rennes

Coefficients (en pourcentage du total des points de chaque concours) **:**

ENS de Lyon		ENS Paris-Saclay		ENS Rennes	
MP option MI	Info	MP option MPI	Info	MP option MPI	Info
10,8 %	14,1 %	23,1 %	13,2 %	23,1 %	8,6 %

L'épreuve orale d'informatique fondamentale concerne les candidats aux ENS de Lyon, Paris-Saclay, et Rennes des concours MP (options MI et MPI) et Informatique.

Cette année, 205 candidates et candidats ont participé à l'épreuve. Leurs notes sont comprises entre 3 et 20, avec une moyenne de 10,99 et un écart-type de 3,67. L'histogramme de la figure 1 présente la distribution des notes. Les membres du jury tiennent cette année encore à souligner qu'ils ont été choqués et peinés de constater le très faible nombre de candidates – à peine 15 !

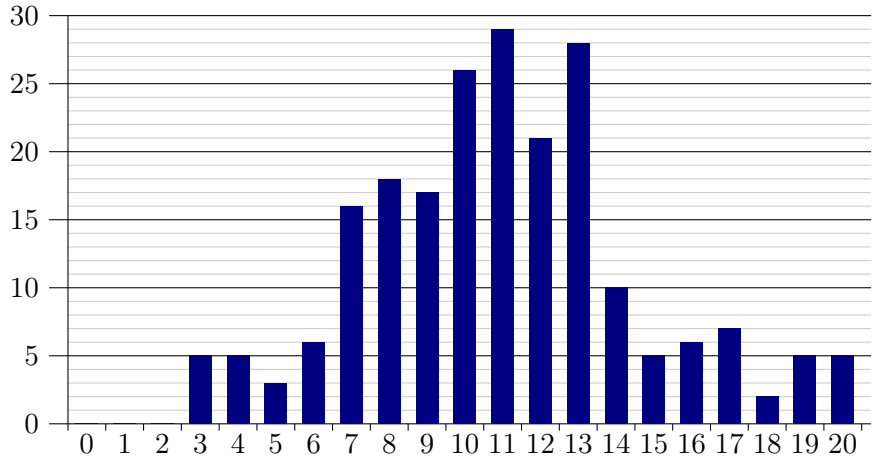


FIGURE 1 – Histogramme des notes de l'épreuve

Après avoir reçu un sujet, les candidats disposaient de 30 minutes de préparation, suivies de 27 minutes d'interrogation devant un des examinateurs. En effet, 3 minutes sont utilisées par l'examinateur pour aller chercher le candidat en salle de préparation et lui rappeler au candidat les conditions de l'évaluation : cette évaluation se base principalement sur la progression des candidats dans les questions, mais aussi sur la pédagogie de leur présentation et l'autonomie dont ils ont fait preuve pendant l'oral. À ce sujet, le jury a été favorablement impressionné par la pédagogie des exposés de nombreux candidats.

Le jury a proposé 21 sujets originaux, dont quatre exemples représentatifs sont présentés en annexe. Cela représente une moyenne de 10 candidats sur chaque sujet, ce qui permet aux membres du jury d'effectuer une meilleure harmonisation des évaluations, notamment via une pondération des questions selon leur difficulté.

Chaque sujet débute par un énoncé qui présente un problème d'informatique et introduit ses notations, puis comporte entre six et neuf questions de difficulté globalement croissante. Une ou deux premières questions relativement faciles d'accès permettent aux candidats de vérifier leur compréhension de l'énoncé et de « se lancer » dans l'interrogation orale. Les questions suivantes, progressivement plus difficiles, occupent la majeure partie du temps de préparation et d'interrogation, et visent à permettre de départager les candidats. Enfin, la plupart des

sujets terminent par une ou plusieurs questions considérées comme très difficiles. En général, il n'est pas attendu des candidats qu'ils traitent l'ensemble des questions dans le temps imparti.

Les sujets font appel aux différentes compétences nécessaires en science informatique : comprendre des concepts nouveaux, démontrer des résultats théoriques, et construire des solutions techniques telles que des algorithmes. Si l'épreuve d'informatique *fondamentale* met l'accent sur les concepts théoriques de l'informatique, on veille néanmoins à garder à l'esprit le sens des objets que l'on étudie.

Par contraste avec les constats formulés lors de la session précédente, en 2019, le jury a constaté avec satisfaction que la plupart des candidats affichait une bonne maîtrise des structures de données pertinentes, telle que des tableaux de listes chaînées, pour répondre aux questions qui leur étaient posées. Au contraire, quelques candidats ont fait montre d'une méconnaissance de constructions basiques sur les automates, comme l'automate reconnaissant l'intersection de deux langages : de telles lacunes s'avèrent souvent rédhibitoires pour le candidat, notamment parce qu'elles l'empêchent d'avancer sur les questions considérées comme faciles.

Les énoncés proposés cette année portaient sur des thèmes assez variés : algorithmique (la moitié des énoncés), automates, combinatoire des mots, graphes, modèles de calcul, relations d'ordre et probabilités.

Le jury adresse les conseils suivants aux futurs candidats :

— Si le principe même d'un concours nous impose de faire un classement et que l'épreuve orale peut sembler intimidante, nous tenons à rappeler que *tous* les candidats admissibles ont un excellent niveau et peuvent aborder l'oral avec confiance. Les candidats ne doivent en aucun cas se dévaloriser ou s'auto-censurer sur la base de leur parcours, de leurs notes en classes préparatoires, ou de ce qu'ils estiment être leur position dans le classement !

— L'objectif du jury est d'amener chaque candidat à réussir à traiter au mieux le maximum de questions. Il est donc dans l'intérêt des candidats d'écouter les demandes et les indications de l'examinateur et d'en tenir compte.

— Lors de la préparation, lire l'ensemble du sujet est indispensable : cela permet au candidat de comprendre la direction générale de l'exercice et d'identifier des questions éventuellement plus simples à traiter. Dès le début de l'interrogation, il faut ensuite préciser à l'examinateur quelles questions on a traitées, ne serait-ce que partiellement. Lors de l'interrogation, il ne faut pas non plus hésiter à proposer à l'examinateur de sauter ou remettre à plus tard des questions pour avoir le temps de présenter l'ensemble des résultats préparés. Cette année encore, plusieurs candidats sont arrivés au terme des 27 minutes imparties avant d'avoir pu présenter toutes leurs solutions.

— Deux écueils à éviter sont donc, d'une part, de traiter de manière trop superficielle les démonstrations et les algorithmes, en passant à côté de points techniques importants, et inversement, de détailler excessivement les réponses de manière trop formelle et manquer de temps pour traiter suffisamment de questions. Nous conseillons aux candidats de soigner la forme des réponses aux premières questions, plus faciles, quitte à prendre progressivement de la hauteur pour se concentrer sur le fond dans les questions difficiles. Dans tous les cas, on prendra soin de présenter brièvement la démarche ou l'intuition suivie avant de se lancer dans une démonstration, un calcul ou un algorithme.

— Lorsque l'on demande au candidat de proposer un algorithme ou une preuve, on attend de lui qu'il nous en présente d'abord la structure générale et les étapes principales, avant de détailler celles-ci si l'examinateur en fait la demande. En particulier, il n'est **jamais** attendu du candidat qu'il commence par écrire un pseudo-code au tableau.

— Les Écoles Normales Supérieures forment à la recherche, et le candidat ne doit donc pas hésiter à chercher au tableau : faire des dessins, traiter des exemples simples que suggèrent les premières questions ou que le candidat aura inventés lui-même, faire des essais qui seront soit concluants, soit porteurs d'une information nouvelle.

En annexe, nous présentons quatre exemples de sujet représentatifs de cette année :

— Attaques temporelles
— Coût automatique des opérations sur les langages
— Problèmes d'accords Byzantins
— Théorème de Dilworth et application

Attaques temporelles

Un hackeur essaye d'accéder à un serveur mais il ne connaît pas le mot de passe. Sans aucune information supplémentaire, il n'a pas d'autre choix que d'essayer toutes les combinaisons possibles. Dans ce sujet, nous allons voir qu'une information inattendue peut permettre de retrouver le mot de passe beaucoup plus rapidement : le temps mis par le serveur pour vérifier que le mot de passe est correct. Pour simplifier, on suppose dans ce sujet que toutes les opérations d'un programme prennent une unité de temps τ. On suppose par ailleurs que le hackeur peut comparer la durée d'exécution de deux programmes avec une précision de l'ordre de τ. Plus précisément, étant donné deux exécutions dont l'une est plus courte que l'autre d'au moins τ, on peut déterminer laquelle est la plus courte. Si les deux exécutions diffèrent d'un temps strictement inférieur à τ, on ne peut rien dire.

On suppose que le mot de passe est une chaîne de 0 et de 1 de longueur n. L'attaquant envoie le mot de passe au serveur qui répond soit par « correct », soit par « incorrect ».

Question 1. Si l'on a aucune information sur le mot de passe, combien d'essais faut-il pour trouver le mot de passe dans le pire cas ?

On suppose maintenant que le serveur compare le mot de passe x de l'attaquant au vrai mot de passe y en utilisant l'algorithme COMPARE ci-dessous.

Algorithme 1 COMPARE(x,y)

```
pour i de 1 à n faire
    si x_i ≠ y_i alors
        retourner « incorrect »
    fin si
fin pour
retourner « correct »
```

Algorithme 2 EXP(a,x,m)

```
res ← 1
base ← a
pour i de 1 à |x| faire
    si x_i = 1 alors
        res ← (res * base) mod m
    fin si
    base ← (base * base) mod m
fin pour
retourner res
```

Question 2. Montrer que l'on peut retrouver le mot de passe en au plus $2n$ essais en mesurant le temps d'exécution de l'algorithme.

Question 3. En réalité, le temps d'exécution de l'algorithme sur le serveur n'est pas déterministe : il peut y avoir plusieurs programmes en cours d'exécution, le temps pour envoyer et recevoir la réponse peut varier. En quoi est-ce un problème ? Comment peut-on y remédier en faisant plus d'essais ?

Question 4. Supposons maintenant que l'attaquant peut seulement mesurer des différences de l'ordre de 10τ. Que peut-on faire ? Quelle est la complexité de l'attaque ?

Question 5. Comment peut-on modifier COMPARE l'algorithme pour contrer ces attaques ? On proposera plusieurs solutions dont on discutera les avantages et inconvénients.

On s'intéresse à un cas plus réaliste d'un serveur qui stocke une valeur secrète s et utilise le protocole suivant. Le client et le serveur se sont mis d'accord à l'avance sur des entiers a et m publics. Le client envoie un entier x au serveur qui calcule $a^{x+s} \mod m$ en utilisant l'algorithme d'exponentiation rapide EXP ci-dessus. On suppose ensuite que le serveur prend une décision (« correct » ou « incorrect »), en temps constant, en fonction de la valeur res obtenue. On pourra supposer dans la suite que s est un entier sur au plus N bits.

Question 6. Donner le temps d'exécution de $\textsc{Exp}(a, x, m)$ en fonction de x. En déduire une attaque temporelle sur le protocole décrit ci-dessus pour retrouver le sercet s.

Question 7. On suppose maintenant que le serveur calcule $a^{s \times x} \mod m$ au lieu de $a^{s+x} \mod m$. Reprendre la question précédente dans ce nouveau cadre.

Coût automatique des opérations sur les langages

Soit $\mathcal{A} = \{a, b, c\}$ un alphabet fini. On note $\mathcal{A}^*$ l'ensemble des mots finis sur l'alphabet $\mathcal{A}$. Les sous-ensembles de $\mathcal{A}^*$ sont appelés *langages* sur $\mathcal{A}$. Si un langage $L \subseteq \mathcal{A}^*$ est rationnel, on note $\mathsf{c}(L)$ le plus petit nombre possible d'états d'un automate déterministe complet qui accepte L. Sinon, on pose $\mathsf{c}(L) = +\infty$.
Enfin, si $\varphi : \left(2^{\mathcal{A}^*}\right)^n \to 2^{\mathcal{A}^*}$ est une opération n-aire sur les langages, on note $\mathsf{c}(\varphi) : (\mathbb{N}_{\geqslant 1} \cup \{+\infty\})^n \to (\mathbb{N}_{\geqslant 1} \cup \{+\infty\})$ la fonction définie par

$$\mathsf{c}(\varphi)(\ell_1, \ell_2, \ldots, \ell_n) = \sup\{\mathsf{c}(\varphi(L_1, L_2, \ldots, L_n)) \colon \mathsf{c}(L_i) \leqslant \ell_i \text{ pour tout } i \leqslant n\}.$$

On dit qu'il s'agit du *coût automatique* de l'opération φ.

Question 1. Quel est le coût automatique de chacune des opérations $L \to L$, $L \to \{\varepsilon\}$ et $L \to \mathcal{A}^* \setminus L$?

Question 2. Soit k et ℓ deux entiers naturels non nuls. Démontrer que $\mathsf{c}(\cap)(k, \ell) \leqslant k\ell$ et que $\mathsf{c}(\cup)(k, \ell) \leqslant k\ell$.

Question 3. Démontrer que $\mathsf{c}(\cup) = \mathsf{c}(\cap)$.

Soit $L \subseteq \mathcal{A}^*$ un langage, rationnel ou non. On définit une relation d'équivalence $\overset{L}{\sim}$ sur $\mathcal{A}^*$ par :

$$u \overset{L}{\sim} v \text{ si et seulement si, pour tout mot } w \in \mathcal{A}^*, (uw \in L \Leftrightarrow vw \in L).$$

On note $[u]_L$ la classe à laquelle appartient le mot u. On dit qu'il s'agit d'une *classe résiduelle* de L.
Enfin, on appelle *automate résiduel* de L l'automate (non nécessairement fini) A dont les états sont les classes d'équivalence de $\overset{L}{\sim}$, l'état initial est $[\varepsilon]_L$, les états finaux sont les classes $[u]_L$ pour lesquelles $u \in L$, et la fonction de transition δ est définie par $\delta([u]_L, x) = [ux]_L$ pour tout mot $u \in \mathcal{A}^*$ et toute lettre $x \in \mathcal{A}$.

Question 4. Soit A un automate déterministe complet et L le langage qu'accepte A. Soit i l'état initial de A et δ sa fonction de transition étendue. Enfin, soit u et v deux mots tels que $\delta(i, u) = \delta(i, v)$. Démontrer que $u \overset{L}{\sim} v$.

En déduire, si L est rationnel, que la description ci-dessus définit bien un automate déterministe complet qui accepte L et possède $\mathsf{c}(L)$ états.

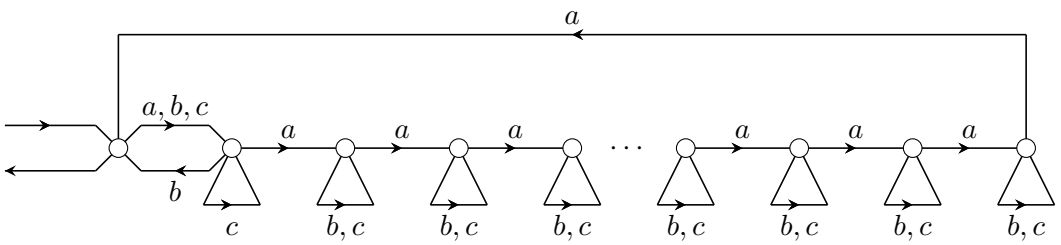
Question 5. Soit k et ℓ deux entiers naturels non nuls. Démontrer que $\mathsf{c}(\cup)(k, \ell) = k\ell$.

Question 6. On note $\mathsf{mir}(L)$ le langage *miroir* d'un langage L, c'est-à-dire le langage

$$\{u_1 u_2 \cdots u_n \colon u_n u_{n-1} \cdots u_1 \in L\}.$$

Soit $\ell \geqslant 2$ un entier naturel. Démontrer que $\mathsf{c}(\mathsf{mir})(\ell) = 2^\ell$.

On pourra s'intéresser au langage qu'accepte l'automate A *à* ℓ *états dessiné ci-dessous :*



Question 7. Soit $k \geqslant 2$ et $\ell \geqslant 2$ deux entiers, et $\cdot$ l'opération de concaténation. Démontrer que

$$\mathsf{c}(\cdot)(k, \ell) = (2^\ell - 1)(k - 1) + 2^{\ell - 1}.$$

Problèmes d'accords Byzantins

Des Généraux de l'armée byzantine campent autour d'une cité ennemie. Ils ne peuvent communiquer qu'à l'aide de messagers et doivent établir un plan de bataille commun, faute de quoi la défaite sera inévitable. Cependant, un certain nombre de ces Généraux peuvent s'avérer être des traîtres, qui se reconnaissent entre eux et essayeront de semer la confusion parmi les autres. Le problème est de trouver un algorithme pour s'assurer que les Généraux loyaux arrivent tout de même à se mettre d'accord sur un plan de bataille.

On se limite a un message binaire attaque/retraite. De plus, un Général sera désigné comme spécial (le Commandant) et les autres seront ses Lieutenants. On utilisera le terme Général pour désigner soit un Commandant soit un Lieutenant.

Les communications se font entre paires de Généraux : le Général A envoie un message au Général B, nul autre Général ne peut accéder à ce message. Un Général traître fait ce qu'il veut (en termes de communication et d'action). Plus précisément, on suppose que les conditions suivantes sont réunies :

M1 Tout message envoyé est correctement reçu (en négligeant le temps de communication).

M2 Celui qui reçoit un message sait qui le lui a envoyé.

M3 Lorsqu'un Général A doit envoyer un message au Général B il dispose d'un temps prédéterminé pour le faire, et s'il ne le fait pas le Général B sait qu'il ne recevra pas de message du Général A.

Formellement, le problème est alors le suivant : un Commandant doit envoyer un ordre à ses $n-1$ Lieutenants avec les conditions suivantes :

C1 Tous les Lieutenants loyaux doivent entreprendre la même action : soit ils attaquent tous, soit ils se retirent tous.

C2 Si le Commandant est loyal, chaque Lieutenant loyal doit obéir à l'ordre que le Commandant a envoyé.

Il faut maintenant établir un protocole de communication entre Généraux qui assurera la réalisation de ces deux conditions.

Question 1. Montrer que, s'il y a 3 Généraux (1 Commandant et 2 Lieutenants) dont un traître, le problème est insoluble.

Question 2. On suppose maintenant que l'on a un nombre quelconque de Généraux et qu'au moins un tiers d'entre eux sont des traîtres. Montrer que, sous ces hypothèses, le problème est insoluble.

L'algorithme $\texttt{OM}(m)$ résout le problème des Généraux Byzantins pour un Commandant et $n-1$ Lieutenants. Nous allons démontrer que $\texttt{OM}(m)$ fonctionne lorsque l'on a $3m+1$ Généraux, dont m traîtres ou moins. Cet algorithme repose sur une fonction `majorité` qui calcule la valeur majoritaire dans une liste de valeurs, et répond « retraite » en cas d'égalité. En outre, dans le cadre de cet algorithme, si un Lieutenant attend un message de son Commandant mais que que ce message n'arrive pas (car le Commandant est un traître), le Lieutenant considère qu'il a reçu l'ordre « retraite ».

— Algorithme $\texttt{OM}(0)$:

1. Le Commandant envoie sa valeur aux $n-1$ Lieutenants.
2. Chaque Lieutenant utilise la valeur reçue ou *retraite* s'il n'a rien reçu.

— Algorithme $\texttt{OM}(m), m > 0$:

1. Le Commandant envoie sa valeur aux $n-1$ Lieutenants.
2. Pour chaque i, soit v_i la valeur que i a reçu du Commandant (ou *retraite* s'il n'a rien reçu). Le Lieutenant i agit comme Commandant dans l'algorithme $\texttt{OM}(m-1)$ pour transmettre v_i au $n-2$ autres Lieutenants.
3. Pour chaque i, chaque $j \neq i$, soit v_j la valeur reçue par le Lieutenant i en provenance du Lieutenant j à l'étape (2) en utilisant l'algorithme $\texttt{OM}(m-1)$, ou alors retraite s'il n'a pas reçu de valeur. Le Lieutenant i utilise $\texttt{majorité}(v_1, \ldots, v_{n-1})$.

Question 3. Faites fonctionner l'algorithme pour $m = 1, n = 4$ avec deux scénarios :

1. Le Commandant est un traître.
2. Un Lieutenant est un traître.

Question 4. Montrer que, pour tous les entiers m et k, l'algorithme `OM(m)` satisfait **C2**, s'il y a au moins $2k+m+1$ Généraux et au plus k traîtres.

Question 5. Montrer que, pour tout m, l'algorithme `OM(m)` satisfait **C1** et **C2** s'il y a au moins $3m+1$ Généraux, dont m traîtres ou moins.

Nous rajoutons maintenant la possibilité pour les Généraux de signer les messages : cela empêche les traîtres de mentir car ils ne peuvent pas engendrer une fausse signature. On ajoute aux conditions **M1, M2, M3** la condition :

M4 (a) La signature d'un Général loyal ne peut pas être contrefaite et chaque altération d'un message signé peut être détectée.

(b) N'importe qui peut vérifier la signature d'un Général.

On écrira $m : g$ le message m avec la signature de m par g, et $m : g : h$ au lieu de $(m : g) : h$.

Aucune supposition n'est faite sur la signature d'un général traître. En particulier, sa signature peut être contrefaite par quelqu'un d'autre (ce qui permet de modéliser des collusions entre les traîtres).

Question 6. Montrer que le ratio traître/honnête calculé en question 2 n'est plus d'actualité et qu'on peut résoudre le problème des Généraux Byzantins avec plus d'un tiers de traîtres.

L'algorithme `SM` qui suit suppose implantée une fonction `choix` qui réduit un ensemble de valeurs en une seule. Les seules spécifications demandées sont les suivantes :

— `choix`$(\{attaque\}) = attaque$.
— `choix`$(E) = retraite$ sinon.

Algorithme `SM`(m) :
Initialement tous les $V_i = \emptyset$

1. Le commandant signe et envoie sa valeur à chaque Lieutenant.
2. Pour tout i :
   - 2.1 Si le Lieutenant i reçoit un message de la forme $v : 0$ du Commandant et qu'il n'a pas déjà reçu d'ordre :
     - 2.1.1 Il met V_i à la valeur $\{v\}$.
     - 2.1.2 Il envoie le message $v : 0 : i$ aux autres Lieutenants.
   - 2.2 Si le Lieutenant i reçoit un message de la forme $v : 0 : j_1 : \ldots : j_k$ et si v n'est pas dans V_i, alors :
     - 2.2.1 Il ajoute v à V_i
     - 2.2.2 Si $k < m$ alors il envoie le message $v : 0 : j_1 : \ldots : j_k : i$ à tous les Lieutenants autres que $j_1, \ldots, j_k$.
3. Pour tout i : quand le Lieutenant i sait qu'il ne va plus recevoir des messages, il obéit à l'ordre `choix`(V_i).

Question 7. Comment mieux spécifier 3 pour qu'un Lieutenant puisse déterminer qu'il ne va plus recevoir de messages à l'étape 3 ?

Question 8. Montrer que, pour tout m, l'algorithme `SM`(m) résout le problème des Généraux Byzantins s'il y a au plus m traîtres.

Théorème de Dilworth et application

Soit E un ensemble fini non vide et $\leq$ un ordre partiel sur E, c'est-à-dire une relation binaire réflexive, antisymétrique et transitive. Soit $<$ la partie irréflexive de $\leq$, c'est-à-dire $x < y$ si et seulement si $x \leq y$ et $x \neq y$.

Question 1. Pour tout $F \subseteq E$, on dit qu'un élément x de F est maximal dans F si x est le seul élément y de F tel que $x \leq y$. Soit $\mathrm{Max}(F)$ l'ensemble des éléments maximaux de F. Pour tout $x \in E$ soit $E_x := \{y \in E \mid x < y\}$.

1. Donner une condition nécessaire et suffisante pour que E_x soit vide.
2. Montrer que $\mathrm{Max}(E_x) \subseteq \mathrm{Max}(E)$.

Question 2. Montrer que $\mathrm{Max}(E)$ est non vide.

Une *chaîne* de E est une partie de E dont les éléments sont deux à deux comparables. Une *antichaîne* de E est une partie de E dont les éléments sont deux à deux incomparables.

Dans la suite du problème, on cherche à démontrer le théorème de Dilworth : si k est le maximum des cardinaux des antichaînes de E, alors E est l'union disjointe de k chaînes.

Question 3. Montrer que $\mathrm{Max}(E)$ est une antichaîne maximale.

Question 4. Démontrer le théorème de Dilworth lorsque $k = 1$.

Question 5. On cherche à montrer le théorème de Dilworth par récurrence sur k. On suppose maintenant que $k > 1$. Soient $z \in \mathrm{Max}(E)$, $F := E \setminus \{z\}$, et $n \in \mathbb{N}$ le maximum des cardinaux des antichaînes de F. Par hypothèse de récurrence, soient $C_1, \ldots, C_n$ des chaînes disjointes de F d'union F. Donner un encadrement de k en fonction de n.

Question 6. Pour tout entier i compris entre 1 et n, on note D_i l'ensemble des éléments de C_i qui appartiennent à une antichaîne de F de cardinal n.

1. Montrer que les D_i sont tous non vides.
2. Pour tout i, soit y_i un élément maximal de D_i. Montrer que $\{y_1, \ldots, y_n\}$ est une antichaîne de F.

Question 7. Montrer le théorème de Dilworth.

Question 8. Soient s, t deux entiers naturels non nuls et $\mathcal{I}$ une famille de $st + 1$ intervalles réels. Montrer qu'au moins une des deux propositions ci-dessous est vraie.

1. Il existe un réel qui appartient à $s + 1$ intervalles de I.
2. Il existe $t + 1$ intervalles de I qui sont disjoints deux à deux.