

CCP 2018 - Option informatique

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. A 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 .

- Q.1** Donner une formule de la logique des propositions représentant la phrase écrite sur la boîte 1.
- Q.2** Donner une même formule de la logique des propositions pour l’inscription de la boîte 2.
- Q.3** 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 .
- Q.4** 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”.

- Q.5** Donner une formule de la logique des propositions pour chaque affirmation.
- Q.6** 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. A 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 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.

Q.7 Donner une formule de la logique des propositions pour chaque inscription.

Q.8 Donner une formule de la logique des propositions synthétisant l'information que vous apportée l'animateur.

Q.9 En supposant que la clé verte est dans la boîte 2, montrer par l'absurde que l'on aboutit à une incohérence.

Q.10 Donner alors la composition des trois boîtes.

II. Automates

Le sujet présenté ici n'est pas exactement celui d'origine. Celui-ci comportait de nombreuses fautes d'énoncé et des notions étaient utilisées sans être définies. Les questions restent cependant les mêmes (et elles me semblent souvent mal posées).

Dans ce problème, on travaillera exclusivement avec des automates déterministes et complets que l'on appellera simplement automates.

Un langage est régulier si et seulement si il est accepté par un automate. 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, tel que tout automate reconnaissant L a au moins autant d'états que lui. Cet unique automate est appelé automate minimal reconnaissant L .

II.1 Définitions

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

- Q un ensemble fini non vide d'états,
- Σ un alphabet (ensemble fini non vide),
- q_0 est l'état initial,
- $F \subset Q$ un ensemble d'états finaux,
- $\delta : Q \times \Sigma \rightarrow Q$ une application de transition définie sur $Q \times \Sigma$ tout entier.

Il s'agit, comme évoqué plus haut, de la notion d'automate déterministe et complet.

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

Soit $L \subset \Sigma^*$ un langage et $u \in \Sigma^*$. Le **résiduel à gauche de L par rapport à u** est le langage

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

Q.11 Pour $L = \{ab, ba, aab\}$, donner le résiduel de L par rapport à a .

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

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

Q.12 Montrer que \sim_L est une relation d'équivalence et que $\forall u, v, w \in \Sigma^*$, $(u \sim_L v) \Rightarrow (uw \sim_L vw)$.

Q.13 Posons $L = \{u \in \Sigma^*, |u|_a = 0 \pmod{3}\}$ le langage basé sur $\Sigma = \{a, b\}$, composé des mots de Σ^* 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 .

Soit $A = (Q, \Sigma, q_0, F, \delta)$ un automate (au sens défini plus haut). La fonction de transition δ^* étendue aux mots est définie de manière récursive par

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

Pour $q \in Q$ et $G \subset Q$, 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 . En particulier, $q^{-1}F$ est l'ensemble menant de q à un état terminal. On dira que $q^{-1}F$ est l'ensemble des mots **accepté par l'automate depuis** q .

Une relation d'équivalence \sim_A est également définie sur Q par

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

L'automate A accepte le langage $L = q_0^{-1}F$. 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 Nérode peut être utilisée pour définir un automate particulier appelé automate minimal de L .

Soit L un langage. L'**automate minimal** de L est défini par le quintuplet $A_L = (Q_L, \Sigma, q_L, F_L, \delta_L)$ avec

- $Q_L = \{u^{-1}L, u \in \Sigma^*\}$
- $q_L = \Lambda^{-1}L = L$
- $F_L = \{u^{-1}L, u \in L\} = \{q \in Q_L, \Lambda \in q\}$
- $\forall q \in Q_L, \forall a \in \Sigma, \delta_L(q, a) = a^{-1}q$

Q.14 Montrer que si L est un langage régulier alors Q_L est fini.

Dans ce cas, l'automate minimal est alors bien un automate au sens de notre énoncé (ensemble fini d'état, caractère déterministe et complet).

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

Un automate $A = (Q, \Sigma, q_0, F, \delta)$ 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 $L \subset \Sigma^*$ rationnel, il est possible de montrer que l'automate minimal A_L de L est accessible et réduit. Le paragraphe suivant s'intéresse à la construction de l'automate minimal du langage reconnu par un automate fini A donné en exploitant cette propriété.

II.2 Construction de l'automate minimal

Soit $A = (Q, \Sigma, q_0, F, \delta)$ un automate fini acceptant le langage $L \subset \Sigma^*$. Trouver l'automate minimal A_L de L revient à trouver un automate fini accessible et réduit équivalent à A (c'est à dire reconnaissant le même langage).

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 $p \neq q$, $p \not\sim_A q$. Notons que $p \not\sim_A q$ a lieu s'il existe $u \in \Sigma^*$ tel que $\delta^*(p, u) \in F$ et $\delta^*(q, u) \notin F$ ou $\delta^*(q, v) \in F$ et $\delta^*(p, v) \notin F$. On dit alors que u **distingue** p et q ou que le couple (u, v) est **distingué** par u .

L'algorithme qui suit est un algorithme de réduction du langage associé à 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 strictement plus court.

Algorithme : 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 $\delta(r, a) = p$ et $\delta(s, a) = q$, **faire**

si $(r, s) \notin \bigcup_{i=0}^k N_i$, **alors**

$N_{k+1} \leftarrow N_{k+1} \cup \{(r, s), (s, r)\}$

$k \leftarrow k + 1$

Q.15 Montrer pourquoi, dans la phase d'initialisation, la paire (p, q) est distinguée.

Q.16 Montrer pourquoi, si $N_i = \emptyset$, alors $\forall j > i, N_j = \emptyset$.

Soit alors $A = (Q, \Sigma, q_0, F, \delta)$ l'automate fini suivant

— $Q = \{1, 2, 3, 4, 5, 6\}$

— $\Sigma = \{a, b\}$

— $q_0 = \{4\}$

— $F = \{2, 5\}$

— δ est représenté par la table

	1	2	3	4	5	6
a	2	4	6	5	1	6
b	1	3	2	1	6	2

Q.17 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

Q.18 Appliquer l'algorithme pour trouver l'ensemble des états équivalents. Pour chaque itération k , la trace de l'algorithme sera donnée par une matrice T de taille $|Q| \times |Q|$, avec $T(i, j)$ valant la 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 $[\delta(q, a)]_A$.

Q.19 Représenter graphiquement l'automate minimal de la question précédente, avec ses états et ses transitions.

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 compressions successives, utilisées dans l'ordre suivant pour la compression et l'ordre inverse pour la décompression :

- (i) Transformation de Burrows-Wheeler,
- (ii) Codage par plages (RLE),
- (iii) Codage de Huffman

Soit Σ un alphabet de symboles, de cardinal $|\Sigma| = h$. On munit Σ d'une relation d'ordre \leq . Σ^k est l'ensemble des mots de longueur k construits à partir de Σ . Σ^k est muni de la relation d'ordre lexicographique induite par la relation d'ordre \leq .

Pour $\mu \in \Sigma^k$, on note $|\mu| = k$ la taille de μ et $|\mu|_a$ le nombre d'occurrences de $a \in \Sigma$ dans μ .

Dans toute cette partie, lorsqu'il s'agira de coder une fonction `Ocaml`, un mot $\mu \in \Sigma^k$ sera représenté par une liste de caractères (`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 μ 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 μ un marqueur de fin (par convention noté $|$, inférieur par \leq à tous les autres symboles de Σ). Dans toute la suite, $\hat{\mu}$ désigne le mot auquel on a ajouté le symbole $|$.

Q.20 Pour $\mu = \textit{turlututu}$, 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.

Q.21 Ecrire une fonction récursive `circulaire` : `'a list` \rightarrow `'a list` qui réalise une permutation à droite d'un mot μ donné en entrée.

Q.22 Ecrire une fonction `matrice_mot` : `'a list` \rightarrow `'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`.

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

Q.23 Donner les matrices P et M' dans le cas du mot $\hat{\mu}$, pour $\mu = \textit{turlututu}$.

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.

- Q.24** Ecrire une fonction récursive `tri : 'a list → 'a list` qui réalise le tri par insertion d'une liste d'éléments.
- Q.25** En déduire une fonction `matrice_mot.triée : 'a list → 'a list list` qui construit M' à partir de M .
- Q.26** Pour $\mu \in \Sigma^k$, donner le nombre de comparaisons de symboles nécessaires au pire des cas, pour trier deux permutations circulaires du mot μ .
- Q.27** 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 μ par la dernière colonne de la matrice M' obtenue à partir de $\hat{\mu}$. On note $BWT(\mu)$ ce codage.

- Q.28** Ecrire 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 $\mu = \text{turlututu}$.

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 code $BWT(\mu)$. Par construction, $BWT(\mu)$ est la dernière colonne de M' .

On pose ici comme exemple $BWT(\mu) = \text{edngvnea}$.

- Q.29** 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 μ .
- Q.30** Proposer un algorithme permettant d'obtenir la deuxième colonne de M' . Donner cette deuxième colonne pour $BWT(\mu) = \text{edngvnea}$.
- Q.31** 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 -ième colonne de M' .
- Q.32** En déduire un algorithme itératif permettant de reconstruire M' .
- Q.33** 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 $[(3, 'a'), (1, 'b'), (1, 'a'), (2, 'b')]$.

- Q.34** 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.

- Q.35** Ecrire 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.

- Q.36** Ecrire 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)

Le texte qui suit n'est pas tout à fait celui d'origine car des notions n'étaient soit pas définies soit inutiles dans le cadre du sujet.

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 est élevé. Les arbres considérés sont étiquetés. Plus précisément, un noeud est soit une feuille soit un noeud interne à deux fils.

- Chaque feuille est étiquetée par un couple formé d'un caractère et d'un entier naturel. Ce dernier s'appelle le poids de la feuille (du noeud égal à cette feuille).
- Chaque noeud interne est étiqueté par un entier naturel qu'on appelle aussi le poids du noeud.

Si t est un noeud, on note n_t son poids. Ce poids se confond avec l'étiquette du noeud dans le cas d'un noeud interne.

L'arbre de Huffman se construit à l'aide de l'algorithme suivant.

Algorithme : Codage de Huffman

Entrée : μ un mot de taille $|\mu|$.

Sortie : $\text{Huffman}(\mu)$ le codage de Huffman de μ .

pour $a \in \Sigma$, **faire**

si $|\mu|_a > 0$ **alors**

 créer une feuille d'étiquette $(a, |\mu|_a)$.

$\mathcal{L} \leftarrow$ liste des noeuds créés dans l'ordre croissant des poids

$\mathcal{A} \leftarrow$ liste vide

tant que $(\text{longueur}(\mathcal{L}) + \text{longueur}(\mathcal{A}) > 1)$ **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 d'étiquette $n_t = n_g + n_d$ et tel que $\text{gauche}(t) = g$ et $\text{droite}(t) = d$

 Insérer t à la fin de \mathcal{A}

 Retirer g et d de \mathcal{L} ou \mathcal{A} (selon leur provenance)

$\text{Huffman}(\mu) \leftarrow \mathcal{A}$.

Q.37 Construire le codage de Huffman du mot $\mu = \text{turlututu}$ en utilisant l'algorithme 2. Vous explicitez par des dessins d'arbres chacune des étapes de construction de $\text{Huffman}(\mu)$.

Q.38 Quelle est la forme de l'arbre de Huffman dans un mot où tous les symboles ont le même nombre d'occurrences ?