Concours commun Mines-Ponts, France

Session 2018, filière MP

Informatique option MP

Durée : 3 heures

Ce que traite le sujet

- 1 Recherche naïve d'un motif
- 2 Automates finis déterministes à repli
- 3 Automate de Knuth-Morris-Pratt
- 4 Ensemble de motifs et automates à repli arborescents

Sujet reproduit sans retouche à partir de la page suivante.

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

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

A2018 – INFO MP

ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT Atlantique, ENSAE PARISTECH.

Concours Centrale-Supélec (Cycle International),
Concours Mines-Télécom, Concours Commun TPE/EIVP.

CONCOURS 2018

ÉPREUVE D'INFORMATIQUE MP

Durée de l'épreuve : 3 heures

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve concerne uniquement les candidats de la filière MP.

Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :

INFORMATIQUE - MP

L'énoncé de cette épreuve comporte 8 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

L'épreuve est composée d'un unique problème, comportant 24 questions. Après un préliminaire et des définitions générales, ce problème est divisé en 4 parties ; la partie 2 est indépendante de la partie 1.

Le but du problème est d'étudier des algorithmes permettant de rechercher efficacement des occurrences d'une ou plusieurs chaînes de caractères (appelées *motifs*) à l'intérieur d'une longue chaîne de caractères.

Préliminaire concernant la programmation

Il faudra coder des fonctions à l'aide du langage de programmation Caml, tout autre langage étant exclu. Lorsque le candidat écrira une fonction, il pourra faire appel à d'autres fonctions définies dans les questions précédentes ; il pourra aussi définir des fonctions auxiliaires. Quand l'énoncé demande de coder une fonction, il n'est pas nécessaire de justifier que celle-ci est correcte, sauf si l'énoncé le demande explicitement. Enfin, si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction de tester si les hypothèses sont bien vérifiées.

Dans les questions du problème, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple `n`).

Définitions générales

On fixe un *alphabet* Σ, c'est-à-dire un ensemble fini, dont les éléments sont appelés *symboles*. On note $\lambda > 0$ le nombre d'éléments de cet alphabet, et on suppose que cette taille λ est disponible depuis les fonctions Caml à implémenter sous la forme d'une constante globale `lambda`. Par exemple :

```
let lambda = 5;;
```

On supposera que les éléments de Σ sont ordonnés, par exemple par ordre alphabétique, et on les note dans l'ordre $\alpha_0 < \cdots < \alpha_{\lambda-1}$. On dit que le *code* d'un symbole α de l'alphabet est l'entier i tel que $\alpha = \alpha_i$ $(0 \leqslant i \leqslant \lambda - 1)$.

Une *chaîne de caractères* s est une suite finie *non vide* $u_1 \dots u_k$ de symboles de Σ. La *longueur* de s est son nombre de symboles, c'est-à-dire, ici, k. On représentera en Caml les chaînes de caractères par des listes d'entiers (`int list`), chaque entier étant le code d'un symbole. Ainsi, si l'alphabet est $\Sigma = \{a, b, c\}$, dans cet ordre, la chaîne de caractères « *abac* » sera représentée par la liste `[0; 1; 0; 2]`. En particulier, on n'utilisera jamais le type `string` de Caml.

1 Recherche naïve d'un motif

Une *occurrence* d'une chaîne de caractères $s = \alpha_1 \dots \alpha_k$ (aussi appelée un *motif*) dans une autre chaîne de caractères $t = \beta_1 \dots \beta_n$ est un entier naturel y avec $1 \leqslant y \leqslant n$ tel que, pour tout entier i tel que $0 \leqslant i \leqslant k-1$, $\alpha_{k-i} = \beta_{y-i}$. En d'autres termes, l'occurrence indique la position du dernier caractère du motif s au sein de la chaîne de caractères t (les positions commençant à 1).

Par exemple, si $\Sigma = \{a, b, c, d, e\}$, s est le motif « *abc* » et t la chaîne de caractères « *abcabcdababcdabcdabde* », il y a quatre occurrences de s dans t, à savoir :

— 3

— 6

— 12

— 16

☐ 1 – Soit s, t deux chaînes de caractères. Est-il possible d'avoir deux occurrences y et y' de s dans t avec $y < y'$ et $y \geqslant y' - k + 1$? Si oui, donner un exemple ; sinon, le prouver.

☐ 2 – Quel est le nombre maximal d'occurrences d'un motif de longueur k dans une chaîne de caractères de longueur $n \geqslant k$? Prouver que cette borne est toujours respectée, et que cette borne est atteinte.

☐ 3 – Programmer en Caml une fonction `longueur : 'a list -> int` qui prend en entrée une liste et qui renvoie la longueur de cette liste. Quelle est la complexité de cette fonction, en terme du nombre n d'éléments de la liste ?

☐ 4 – Programmer en Caml une fonction `int list -> int list -> bool` prenant en entrée deux listes d'entiers s et t codant des chaînes de caractères et testant si s est un préfixe de t. Quelle est la complexité de cette fonction, en terme des longueurs k et n de s et t ?

☐ 5 – À l'aide des fonctions `longueur` et `prefixe`, programmer une fonction `recherche_naive : string -> string -> int list`, la plus directe possible, telle que si s est un motif et t une chaîne de caractères, `recherche_naive s t` renvoie la liste des entiers y qui sont des occurrences de s dans t, triés par ordre croissant.

☐ 6 – Quelle est la complexité de la fonction `recherche_naive`, en terme des longueurs k et n de ses deux paramètres s et t ?

2 Automates finis déterministes à repli

Un *automate fini déterministe à repli* (ou *AFDR*) sur l'alphabet Σ est un quadruplet $\mathcal{A} = (k, F, \delta, \rho)$ tel que :

— $k \in \mathbb{N}$ est un entier strictement positif représentant le *nombre d'états* de $\mathcal{A}$; l'ensemble des *états* de $\mathcal{A}$ est $Q_\mathcal{A} = \{0, 1, \ldots, k-1\}$ et 0 est appelé l'*état initial*.

— $F \subseteq Q_\mathcal{A}$ est un ensemble d'états, appelés *finals*.

— $\delta : Q_\mathcal{A} \times \Sigma \to Q_\mathcal{A}$ est une *fonction partielle de transition* (c'est-à-dire, une fonction dont le domaine de définition est un sous-ensemble de $Q_\mathcal{A} \times \Sigma$) ; on impose que $\delta(0, \alpha)$ soit défini pour tout $\alpha \in \Sigma$.

— $\rho : Q_\mathcal{A} \setminus \{0\} \to Q_\mathcal{A}$ est une application (c'est-à-dire, une fonction totale), appelée *fonction de repli*, telle que pour tout $q \in Q_\mathcal{A}$ avec $q \neq 0$, $\rho(q) < q$. On prolonge ρ en convenant $\rho(0) = 0$.

Un AFDR est représenté en Caml par le type enregistrement suivant :

```
type afdr = {
  final : bool vect;
  transition : int vect vect;
  repli: int vect;
};;
```

où :

— `final` est un tableau de taille k de booléens tel que si $q \in Q_\mathcal{A}$, `final.(q)` contient `true` si et seulement si $q \in F$;

— `transition` est un tableau de taille k de tableaux de taille λ d'entiers, tel que si $q \in Q_\mathcal{A}$ et $\alpha_i \in \Sigma$ de code i, `transition.(q).(i)` contient -1 si $\delta(q, \alpha_i)$ n'est pas défini, et contient $\delta(q, \alpha_i)$ sinon ;

— `repli` est un tableau de taille k d'entiers tel que `repli.(0)` contient 0 et `repli.(q)` pour $q \in Q_\mathcal{A} \setminus \{0\}$ contient $\rho(q)$.

On observe que le type `afdr` ne code pas explicitement la valeur de k, mais k est par exemple la longueur du tableau `final`.

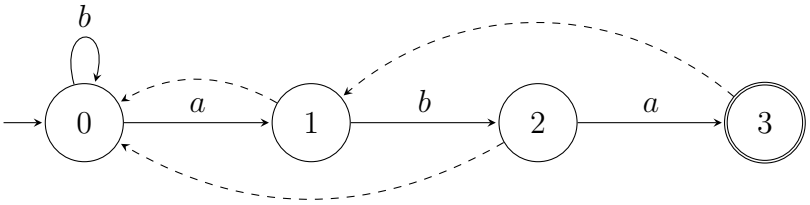
On dessine un AFDR de manière similaire au dessin d'un automate fini déterministe classique, en faisant figurer en pointillés une flèche indiquant les replis d'un état vers un autre. Les états finals sont figurés par un double cercle.

Considérons par exemple l'automate $\mathcal{A}_1 = (k_1, F_1, \delta_1, \rho_1)$ sur l'alphabet $\Sigma = \{a, b\}$ défini par $k_1 = 4$, $F_1 = \{3\}$, et les fonctions δ_1 et ρ_1 suivantes :

δ_1		
q	$\delta_1(q,a)$	$\delta_1(q,b)$
0	1	0
1		2
2	3	
3		

ρ_1	
q	$\rho_1(q)$
0	
1	0
2	0
3	1

$\mathcal{A}_1$ peut être dessiné comme suit :



☐ 7 – Soit $\mathcal{A} = (k, F, \delta, \rho)$ un AFDR. Pour tout $q \in Q_{\mathcal{A}}$, on note $\rho^j(q)$ pour $j \in \mathbb{N}$ l'application répétée j fois de la fonction ρ à q, c'est-à-dire $\underbrace{\rho(\rho(\ldots(\rho}_{j \text{ fois}}(q))\ldots))$. Par convention, pour tout état q, on note $\rho^0(q) = q$.
Montrer que pour tout $q \in Q_{\mathcal{A}}$, pour tout $\alpha \in \Sigma$, il existe $j \geqslant 0$ tel que $\delta(\rho^j(q), \alpha)$ est défini.

On dit qu'un AFDR $\mathcal{A} = (k, F, \delta, \rho)$ *accepte* un mot $u = u_1 \ldots u_p \in \Sigma^*$ s'il existe une suite finie $q_0, q'_1, q_1, q'_2, q_2, \ldots, q'_{p-1}, q_{p-1}, q'_p, q_p$ d'états de $\mathcal{A}$ avec :

— $q_0 = 0$;
— pour tout $1 \leqslant i \leqslant p$, $q'_i = \rho^j(q_{i-1})$ avec $j \geqslant 0$ le plus petit entier tel que $\delta(\rho^j(q_{i-1}), u_i)$ est défini ;
— pour tout $1 \leqslant i \leqslant p$, $q_i = \delta(q'_i, u_i)$;
— $q_p \in F$.

Le *langage* accepté par un AFDR est l'ensemble des mots acceptés.

Ainsi, $\mathcal{A}_1$ accepte le mot « *ababa* », comme le montre la suite d'états parcourus

$$0,\ 0, 1,\ 1, 2,\ 2, 3,\ 1, 2,\ 2, 3.$$

On remarque qu'un AFDR dont la fonction de transition δ est définie partout peut être vu comme un automate fini déterministe classique, et que cet automate fini déterministe est *complet* (ce qui signifie précisément que sa fonction de transition est définie partout). En effet, les puissances non nulles de la fonction de repli ρ ne sont utilisées que si la fonction de transition n'est pas définie. On appellera un tel automate fini déterministe complet un *AFDC*.

☐ 8 – Construire (sans justification) un AFDC sur l'alphabet $\{a, b\}$ reconnaissant le même langage que $\mathcal{A}_1$ et ayant le même nombre d'états que $\mathcal{A}_1$.

☐ 9 – Donner (sans justification) une description concise du langage reconnu par l'AFDR $\mathcal{A}_1$.

☐ 10 – Programmer en Caml une fonction `copie_afdr : afdr -> afdr` qui renvoie un nouvel AFDR identique à l'AFDR fourni en entrée, mais ne partageant aucune donnée avec lui. On pourra utiliser la fonction standard `copie_vect : 'a vect -> 'a vect` qui renvoie un nouveau tableau contenant les mêmes éléments que les éléments d'entrée et s'exécute en un temps proportionnel au nombre d'éléments du tableau d'entrée.

☐ 11 – En s'inspirant de la réponse aux questions 7 et 8, et en utilisant la fonction `copie_afdr`, programmer une fonction `enleve_repli : afdr -> afdr` telle que, si $\mathcal{A}$ est un AFDR, `enleve_repli A` renvoie un AFDC reconnaissant le même langage. On souhaite que cette fonction s'exécute en temps $O(k \times \lambda)$, où k est le nombre d'états de $\mathcal{A}$ et λ est la taille de l'alphabet. Montrer que la fonction proposée a bien cette complexité.

☐ 12 – Étant donné un AFDC $\mathcal{A}$ et un mot $u = u_1 \dots u_n$ sur Σ, proposer un algorithme (pas un programme Caml) en $O(n)$ pour calculer la liste triée des entiers i avec $1 \leqslant i \leqslant n$ tels que le préfixe $u_1 \dots u_i$ de u est accepté par $\mathcal{A}$.

☐ 13 – Implémenter en Caml l'algorithme de la question précédente : programmer une fonction `occurrences : afdr -> int list -> int list` telle que, si $\mathcal{A}$ est un AFDC et *liste* une liste d'entiers $j_1, \dots, j_n$ codant des symboles $\alpha_{j_1}, \dots, \alpha_{j_n}$ de l'alphabet Σ, `occurrences A liste` renvoie la liste des entiers i avec $1 \leqslant i \leqslant n$ tels que le mot $\alpha_{j_1} \dots \alpha_{j_i}$ est reconnu par $\mathcal{A}$. Quelle est la complexité de cette fonction en terme de la longueur n de la liste d'entiers, du nombre k d'états de l'automate $\mathcal{A}$ et de λ ?

3 Automate de Knuth–Morris–Pratt

L'*automate de Knuth–Morris–Pratt* (ou *automate KMP*) associé à un motif $s = u_1 \dots u_k$ sur l'alphabet Σ est un AFDR $\mathcal{A}_s^{\text{KMP}} = (k', F, \delta, \rho)$ sur Σ avec :

— $k' = k + 1$.

— $F = \{k\}$.

— Pour tout $1 \leqslant i \leqslant k$, $\delta(i-1, u_i) = i$ et, pour tout $\alpha \in \Sigma \setminus \{u_1\}$, $\delta(0, \alpha) = 0$; aucune autre transition n'est définie.

— Pour tout $1 \leqslant i \leqslant k$, $\rho(i)$ est le plus grand entier $0 \leqslant j < i$ tel que $u_1 \dots u_j$ est un *suffixe* de $u_1 \dots u_i$.

On peut ainsi vérifier que l'automate $\mathcal{A}_1$ de la question précédente est l'automate KMP associé à « *aba* »sur l'alphabet $\Sigma = \{a, b\}$.

☐ 14 – Construire (sans justification) l'automate KMP associé à « *ababc* »sur l'alphabet $\{a, b, c\}$.

☐ 15 – Donner (sans justification) une description concise du langage reconnu par l'AFDR $\mathcal{A}_s^{\text{KMP}}$ pour un motif s arbitraire.

☐ 16 – Montrer que si $s = u_1 \dots u_k$ est un motif et $(k+1, F, \delta, \rho)$ l'automate KMP associé à s, alors pour tout $1 \leqslant i \leqslant k$, si $j \geqslant 0$ est le plus petit entier tel que $\delta(\rho^j(\rho(i-1)), u_i)$ est défini (qui existe d'après la question 7), alors $\rho(i) = \delta(\rho^j(\rho(i-1)), u_i)$.

☐ 17 – En utilisant la caractérisation de la question 16, programmer une fonction `automate_kmp : int list -> afdr` qui prend en entrée une liste d'entiers `s` codant une chaîne de caractères s et qui renvoie l'AFDR $\mathcal{A}_s^{\text{KMP}}$.

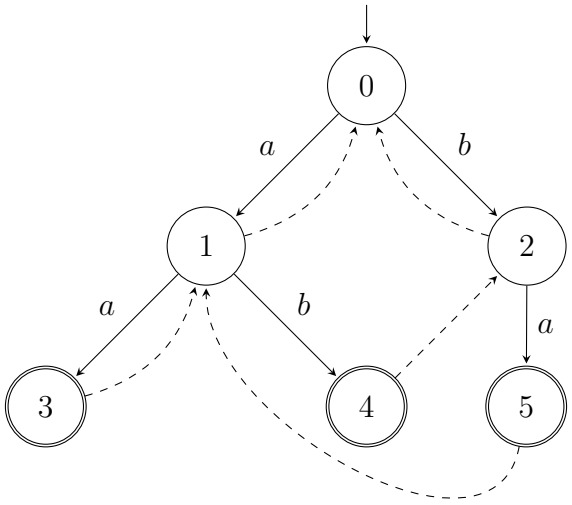
☐ 18 – Pour un motif $s = u_1 \dots u_k$ et pour tout $1 \leqslant i \leqslant k$, on note j_i le plus petit entier tel que $\delta(\rho^{j_i}(\rho(i-1)), u_i)$ est défini, comme à la question 16. Montrer que pour tout $1 \leqslant i \leqslant k$, $\rho(i) \leqslant \rho(i-1) + 1 - j_i$ et en déduire que $\sum_{i=1}^{k} j_i$ est en $O(k)$.

☐ 19 – Quelle est la complexité de la fonction `automate_kmp` en terme de la longueur k du motif s passé en argument et de λ ? Prouver cette affirmation.

☐ 20 – En s'appuyant sur les fonctions Caml `automate_kmp`, `enleve_repli` et `occurrences`, programmer une fonction Caml `recherche_kmp : string -> string -> int list` telle que, si s est un motif de longueur k et t une chaîne de caractères, `recherche_kmp s t` renvoie la liste des occurrences y de s dans t, triés par ordre croissant.
Quelle est la complexité de cette fonction en terme des longueurs k et n de s et t et de λ ? Comparer avec la complexité de la fonction `recherche_naive` obtenue en question 6.

4 Ensemble de motifs et automates à repli arborescents

On considère maintenant l'AFDR $\mathcal{A}_2$ suivant, sur l'alphabet $\Sigma = \{a, b\}$:



☐ 21 – Donner (sans justification) une description concise du langage reconnu par l'AFDR $\mathcal{A}_2$.

☐ 22 – On considère l'alphabet $\Sigma = \{a, b, c\}$. Construire (sans justification) un AFDR dont le langage reconnu est l'ensemble des mots dont un suffixe est « *baa* », « *bab* » ou « *bc* ». Indication : on cherchera un AFDR tel que si $\delta(q, \alpha)$ est défini et q est non nul, alors $\delta(q, \alpha) > q$.

On fixe maintenant un ensemble fini S de motifs. L'ensemble des *occurrences de S dans une chaîne de caractères t* est l'ensemble des occurrences de chacun des motifs de S dans t.

☐ 23 – En utilisant la fonction `recherche_kmp`, programmer une fonction `recherche_dictionnaire_kmp : int list list -> int list -> int list` qui est telle que, si S est un ensemble fini de motifs codé comme une liste de motifs, et t une chaîne de caractères, `recherche_dictionnaire_kmp S t` renvoie la liste des occurrences y d'un motif $s \in S$ dans t. On n'impose pas d'ordre particulier sur cette liste, et on autorise les doublons.

Quelle est la complexité de cette fonction, en terme du nombre $|S|$ de motifs, de la longueur k maximale d'un motif, de λ et de la longueur n de t ?

☐ 24 – En s'inspirant de l'automate $\mathcal{A}_2$, de la réponse à la question 22 et de la partie 3, proposer une approche pour calculer efficacement les occurrences d'un ensemble fini S de motifs dans une chaîne de caractères t. On ne demande

ni une formalisation complète de cette approche, ni une implémentation, mais une stratégie générale et les grandes étapes nécessaires à la résolution du problème. On pourra faire l'hypothèse, pour simplifier, que l'ensemble S ne contient pas deux mots qui sont préfixes l'un de l'autre.

Quelles difficultés sont à prévoir dans l'implémentation ? Quelle complexité peut-on espérer avec une telle approche, en terme du nombre $|S|$ de motifs, de la longueur k maximale d'un motif, de λ et de la longueur n de t ? Comparer avec la complexité obtenue en réponse à la question précédente.

Fin de l'épreuve

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

- Langage OCaml : expressions, types, filtrage, récursivité
- Chaînes de caractères et algorithmes sur les textes
- Complexité des algorithmes itératifs et récursifs
- Arbres binaires, arbres binaires de recherche et tas

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/mines/mp