Devoir maison n°4

À rendre mercredi 13/11

⋆ ⋆ ⋆

Recherche de motifs

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.

Définitions générales

On fixe un alphabet Σ et on note m le nombre d'éléments de cet alphabet. On supposera que les éléments de Σ sont exactement les entiers de 0 à $m-1$. Un mot $u \in \Sigma^*$ sera représenté par la liste de ses lettres (de type `int list`). Par exemple, le mot $u = 0102$ sera représenté par `[0; 1; 0; 2]`. En particulier, on n'utilisera jamais le type `string` de Caml.

1 Recherche naïve d'un motif

Une **occurrence** d'un mot $u = u_1 \dots u_k$ (appelée **motif**) dans un autre mot $v = v_1 \dots v_n$ est un entier naturel $i \in [\![1, n]\!]$ tel que pour $j \in [\![0, k-1]\!]$, $u_{k-j} = v_{i-j}$. En d'autres termes, l'occurrence indique la position de la dernière lettre du motif u au sein du mot v (les positions commençant à 1).

Par exemple, si $\Sigma = \{0, 1, 2, 3, 4\}$, $u = 012$ et $v = 012012301012301230134$, alors il y a quatre occurrences de u dans v, à savoir aux indices 3, 6, 12 et 16.

Question 1 Soit u et v deux mots. Est-il possible d'avoir deux occurrences i et i' de u dans v avec $i < i'$ et $i \geqslant i' - k + 1$. Si oui, donner un exemple ; sinon, le prouver.

Question 2 Quel est le nombre maximal d'occurrences d'un motif de longueur k dans un mot de longueur $n \geqslant k$. Prouver que cette borne est toujours respectée et qu'elle peut être atteinte.

Question 3 Écrire une fonction `prefixe : int list -> int list -> bool` telle que `prefixe u v` renvoie `true` si et seulement si u est un préfixe de v. Quelle est la complexité de cette fonction selon les longueurs k et n de u et v ?

Question 4 À l'aide de la fonction `prefixe`, écrire une fonction, sans chercher d'optimisation particulière, `recherche_naive : int list -> int list -> int list` telle que `recherche_naive u v` renvoie la liste des occurrences de u dans v, triée par ordre croissant.

Question 5 Déterminer la complexité de la fonction précédente en fonction des longueurs k et n de u et v.

2 Automates finis déterministes à repli

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

- $k \in \mathbb{N}^*$ représente le **nombre d'états** de $\mathcal{A}$; l'ensemble des **états** de $\mathcal{A}$ est $Q_{\mathcal{A}} = [\![0, k-1]\!]$ et 0 est appelé **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 $\delta(0, a)$ est défini pour tout $a \in \Sigma$;

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

Un AFDR est représenté en Caml par le type suivant :

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

où si `aut` est de type `afdr` représentant l'AFDR $\mathcal{A} = (k, F, \delta, \rho)$, alors :

– `aut.final` est un tableau de booléens de taille k tel que `aut.final.(q)` vaut `true` si et seulement si $q \in F$;
– `aut.transition` est une matrice d'entiers de taille $k \times m$ telle que `aut.transition.(q).(a)` contient `-1` si $\delta(q, a)$ n'est pas défini, et contient $\delta(q, a)$ sinon ;
– `aut.repli` est un tableau d'entiers de taille k tel que `aut.repli.(0)` contient `0` et `aut.repli.(q)` contient $\rho(q)$ si $q \neq 0$.

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

On dessine un AFDR comme un automate fini déterministe classique, en faisant figurer par des flèches en pointillés les replis d'un état vers un autre. Les états finals sont figurés par un double cercle.

Par exemple, si on définit l'automate $\mathcal{A}_1 = (4, \{3\}, \delta_1, \rho_1)$ sur l'alphabet $\Sigma = \{0, 1\}$ où les fonctions δ_1 et ρ_1 sont définies par :

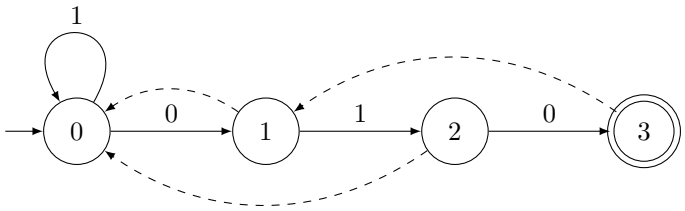
δ_1

q	$\delta_1(q,0)$	$\delta_1(q,1)$
0	1	0
1		2
2	3	
3		

ρ_1

q	$\rho_1(q)$
0	
1	0
2	0
3	1

Alors l'automate $\mathcal{A}_1$ pourra être représenté par :



Question 6 Soit $\mathcal{A} = (k, F, \delta, \rho)$ un AFDR. Montrer que $\forall (q, a) \in Q_{\mathcal{A}} \times \Sigma, \exists j \in \mathbb{N}, \delta(\rho^j(q), a)$ est défini (ρ^j est l'itérée j fois de ρ).

Pour $\mathcal{A} = (k, F, \delta, \rho)$, on note $\widetilde{\rho} : Q_{\mathcal{A}} \times \Sigma \to Q_{\mathcal{A}}$ l'application qui à un couple $(q, a) \in Q_{\mathcal{A}} \times \Sigma$ associe l'état $\rho^j(q)$ où j est le plus petit entier (éventuellement nul) tel que $\delta(\rho^j(q), a)$ est défini.

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

– $q_0 = 0$;
– pour $i \in [\![1, p]\!]$, $q'_i = \widetilde{\rho}(q_{i-1}, u_i)$;
– pour $i \in [\![1, p]\!]$, $q_i = \delta(q'_i, u_i)$;
– $q_p \in F$.

Le langage accepté par $\mathcal{A}$, noté $L(\mathcal{A})$ est l'ensemble des mots acceptés par $\mathcal{A}$.

Par exemple, l'AFDR $\mathcal{A}_1$ accepte le mot 01010 comme le montre la suite d'états $0, 0, 1, 1, 2, 2, 3, 1, 2, 2, 3$.

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 est **complet**. 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 partout. On notera un tel automate un AFDC.

Question 7 Représenter graphiquement sans justification un AFDC sur $\Sigma = \{0, 1\}$ acceptant le même langage que $\mathcal{A}_1$ et ayant le même nombre d'états que $\mathcal{A}_1$.

Question 8 Donner sans justification une description concise du langage accepté par $\mathcal{A}_1$.

Question 9 Écrire une fonction `copie_afdr : afdr -> afdr` qui copie complètement, c'est-à-dire **sans liaison de données**, un AFDR donné en argument. On pourra en particulier utiliser la fonction `Array.copy` qui prend en argument un tableau et renvoie un nouveau tableau contenant les mêmes éléments que celui donné en argument.

Question 10 Écrire une fonction `enleve_repli : afdr -> afdr` telle que si $\mathcal{A}$ est un AFDR codé par `aut`, alors `enleve_repli aut` renvoie un AFDC acceptant le même langage que $\mathcal{A}$. Cette fonction doit avoir une complexité temporelle en $\mathcal{O}(k \times m)$ où k est le nombre d'états de $\mathcal{A}$ et m la taille de l'alphabet et il faudra justifier cette complexité.

Question 11 Étant donné un AFDC $\mathcal{A}$ et un mot $u = u_1 \dots u_n \in \Sigma^*$, proposer un algorithme en pseudo-code ou en français, de complexité $\mathcal{O}(n)$, pour calculer la liste triée des entiers $i \in [\![1, n]\!]$ tels que le préfixe $u_1 \dots u_i$ est accepté par $\mathcal{A}$.

Question 12 Écrire l'algorithme précédent en Caml. Il prendra la forme d'une fonction `occurrences` de signature `afdr -> int list -> int list`.

3 Automate de Knuth-Morris-Pratt

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

- $k' = k + 1$;
- $F = \{k\}$;
- pour $i \in [\![1, k]\!]$, $\delta(i-1, u_i) = i$, pour $a \in \Sigma \setminus \{u_1\}$, $\delta(0, a) = 0$ et aucune autre transition n'est définie ;
- pour $i \in [\![1, k]\!]$, $\rho(i)$ est le plus grand entier $j \in [\![0, i-1]\!]$ tel que $u_1 \dots u_j$ est un **suffixe** de $u_1 \dots u_i$.

On peut vérifier que l'automate $\mathcal{A}_1$ de la question précédente est l'automate KMP associé à 010 sur l'alphabet $\Sigma = \{0, 1\}$.

Question 13 Représenter graphiquement sans justification l'automate KMP $\mathcal{A}_{01012}^{\text{KMP}}$ sur l'alphabet $\Sigma = \{0, 1, 2\}$.

Question 14 Donner sans justification une description concise du langage accepté par l'automate $\mathcal{A}_u^{\text{KMP}}$ pour un motif u quelconque.

Question 15 Si $u = u_1 \dots u_k$ est un motif et $\mathcal{A}_u^{\text{KMP}} = (k', F, \delta, \rho)$ est son automate KMP, alors pour $i \in [\![1, k]\!]$, on note j_i le plus petit entier tel que $\delta(\rho^{j_i}(\rho(i-1)), u_i)$ est défini. Montrer que pour $i \in [\![2, k]\!]$, $\rho(i) = \delta(\rho^{j_i}(\rho(i-1)), u_i)$.

Question 16 En déduire une fonction `kmp : int list -> int -> afdr` qui prend en entrée un motif et l'entier m correspondant à $|\Sigma|$ et renvoie l'automate KMP associé.

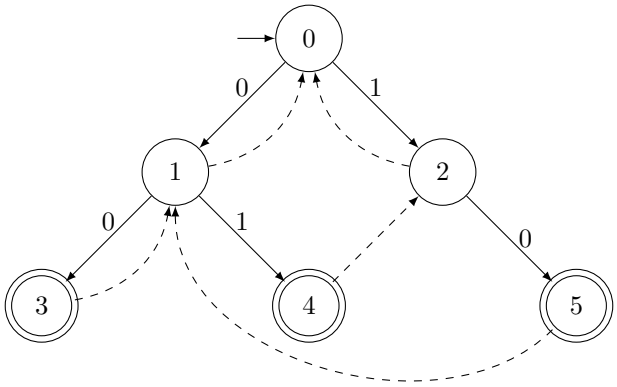
Question 17 Montrer que pour $i \in [\![1, k]\!]$, $\rho(i) \leqslant \rho(i-1) + 1 - j_i$ et en déduire que $\sum_{i=1}^{k} j_i = \mathcal{O}(k)$.

Question 18 Déterminer en justifiant la complexité de la fonction `kmp` en fonction de la longueur k du motif en argument et de m, la taille de l'alphabet.

Question 19 Écrire une fonction `recherche_kmp : int list -> int list -> int list` telle que l'appel `recherche_kmp u v` renvoie la liste des occurrences de u dans v, triée par ordre croissant. Déterminer sa complexité et comparer avec celle de la fonction `recherche_naive`.

4 Ensemble de motifs et automates à repli arborescents

On considère l'AFDR $\mathcal{A}_2$ suivant, sur l'alphabet $\Sigma = \{0, 1\}$:



Question 20 Donner sans justification une description concise du langage accepté par $\mathcal{A}_2$.

Question 21 On considère l'alphabet $\Sigma = \{0, 1, 2\}$. Représenter graphiquement sans justification un AFDR dont le langage accepté est l'ensemble des mots dont un suffixe est 100, 101 ou 12.

On fixe maintenant un ensemble fini D de motifs. L'ensemble des occurrences de D dans un mot v est l'ensemble des occurrences de chacun des motifs de D dans v.

Question 22 En utilisant la fonction `recherche_kmp`, écrire une fonction `recherche_dico_kmp`, de signature `int list list -> int list -> int list` telle que si D est un ensemble fini de motifs, codé par une liste de mots `dico`, et v est un mot, alors `recherche_dico_kmp dico v` renvoie la liste des occurrences d'un motif $u \in D$ dans v. On n'impose pas d'ordre particulier sur cette liste et on autorise les doublons. Déterminer la complexité de cette fonction selon le nombre $|D|$ de motifs, la longueur maximale k d'un motif, le nombre $m = |\Sigma|$ et de la longueur n de v.

Question 23 En s'inspirant de l'automate $\mathcal{A}_2$, de la question 22 et de la partie 3, proposer une méthode pour calculer efficacement les occurrences d'un ensemble fini D de motifs dans un mot v. On ne demande pas de formalisation complète, ni une implémentation. On pourra, pour simplifier, supposer que D 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 ? Comparer avec la question précédente.

⋆ ⋆ ⋆