MITRO210 : Automates et données structurées

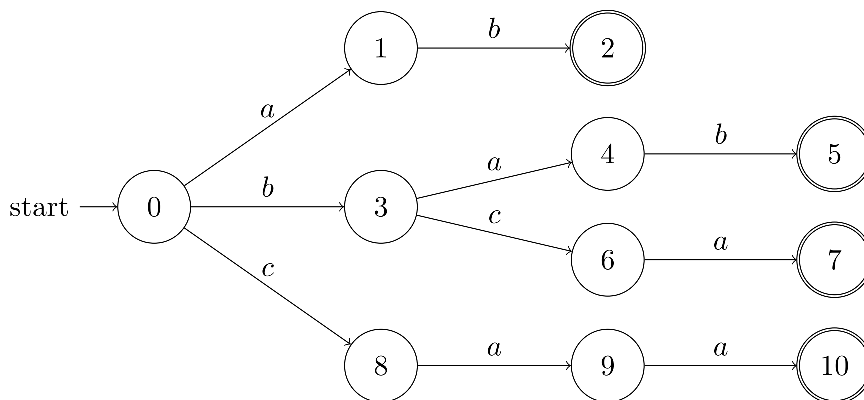
Feuille d'exercices 3 – Corrigé

Antoine Amarilli

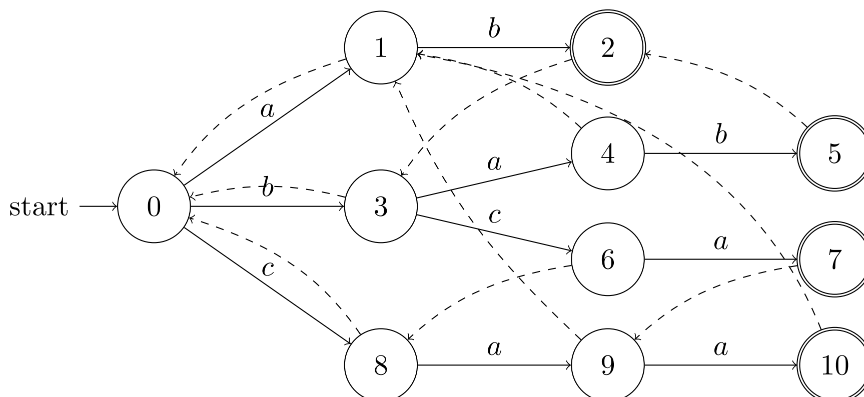
Avertissement : ce corrigé n'a été que peu relu. N'hésitez pas à me contacter pour toute question ou erreur.

1 Algorithme d'Aho-Corasick

Question 0. On propose l'automate suivant, qui correspond à un trie pour U_0.



Question 1. On propose ce qui suit. Intuitivement, en cas d'échec, si on identifie chaque état de l'automate avec un préfixe u' d'un mot de U_0, il faut tenter de reprendre depuis le plus grand suffixe strict de u' qui soit également préfixe d'un mot de U_0.



Question 2. On lit simplement le mot v dans l'automate. On voit qu'il y a une bijection entre les préfixes des mots de U_0 et les états de l'automate, et les préfixes de U_0. Du reste la construction

de l'automate assure que l'état où on se trouve correspond au plus long préfixe d'un mot de U_0 qui soit suffixe du mot lu jusqu'à présent. En effet, dès qu'on lit une lettre supplémentaire, le préfixe le plus long de U_0 qui soit suffixe de ce qui a été lu est soit l'extension du préfixe précédent avec cette lettre supplémentaire (correspondant à suivre une transition qui vient du trie), soit l'extension d'un préfixe plus petit ; et on peut essayer successivement tous les préfixes plus courts possibles avec des transitions d'échec.

La complexité nécessite une analyse de l'impact des transitions d'échec. Pour un U_0 fixé, bien sûr, cela n'ajoute qu'un facteur constant ; mais il faut le comprendre en vue des questions suivantes. On se rappelle de la notion d'automate bien ordonné avec transitions d'échec vue en cours : ici on fixe un ordre total sur les états qui respecte la profondeur dans le trie, c'est-à-dire l'ordre commence par la racine, puis tous les enfants de la racine dans le trie dans un ordre arbitraire, puis tous les enfants de ces enfants, etc. C'est alors bien le cas que chaque transition normale nous fait strictement avancer dans l'ordre, et chaque transition d'échec nous fait strictement reculer dans l'ordre. Si notre construction générale respecte cette condition, alors on peut borner la contribution du parcours des transitions d'échec au temps d'exécution, comme on l'a fait en cours pour l'algorithme KMP.

In fine, on a donc bien la complexité $O(|v|)$.

Question 3. On construit un trie représentant l'ensemble U, en temps linéaire, mais on veut également calculer les pointeurs d'échec. Pour ce faire, on remplace le tableau T de l'algorithme KMP par une fonction matérialisée par les pointeurs d'échec.

Quand on ajoute un nouveau mot au trie, on navigue dans le trie jusqu'à trouver l'endroit où ajouter de nouveaux nœuds. Le cas de base est l'ajout d'un enfant de la racine : le pointeur d'échec pointe alors toujours vers la racine.

Maintenant, à chaque nouveau nœud n créé, on regarde son prédécesseur n' (avec un pointeur d'échec déjà calculé vers un nœud p', car n' n'est pas la racine vu qu'on a exclu ce cas ci-dessus). Soit a la lettre de l'arête nouvellement créée de n' à n. Si p' a un successeur p pour la lettre a, alors le pointeur d'échec de n pointe vers p. Sinon, on suit le pointeur d'échec de p' et on essaie de lire a, jusqu'à finalement pointer vers la racine.

On étudie d'abord la complexité. À chaque fois qu'on insère un mot, les reculs du pointeur d'échec sont amortis par des navigations vers l'avant de la même façon que dans l'algorithme KMP ou dans la question précédente, donc la complexité est bien en $O(||U||)$ où $||U||$ désigne la taille totale des mots de U.

On montre ensuite que la construction est correcte. Pour les cas de base, c'est évident. Considérons la création du nœud n à partir de n' pour la lettre a, et p' le pointeur d'échec de n' (qui est défini car n' n'est pas la racine). Si on peut étendre p' avec a, alors c'est clairement le préfixe de U qui est de longueur maximale, sinon ceci contredirait la définition de p'. Sinon, on montre comme pour KMP que les préfixes des mots de U qui sont des suffixes du nœud n s'obtiennent en suivant les pointeurs d'échec, et le préfixe souhaité pour n' doit s'obtenir en étendant le plus grand tel suffixe qui peut être étendu avec a. Ainsi, on respecte bien l'invariant.

Question 4. L'argument donné en correction de la question 2 couvre déjà le cas général. On obtient bien une complexité de $O(|v|)$ pour la phase de lecture, soit $O(|U| + |v|)$ au total si $|U|$ dénote la longueur totale des mots de U.

Question 5. Si on prend par exemple $U = \{a, aa, ..., a^k\}$, le problème est qu'on peut trouver plusieurs résultats au même endroit : sur $v = a^k$ par exemple à la fin on trouve un résultat pour chaque mot de U. Or, on ne connaît pas ces mots.

Pour l'ensemble U_0 donné en exemple : si on lit le mot bab, il faut produire un résultat non seulement pour bab mais aussi pour ab. Du reste si on avait l'ensemble $U_0' = \{a, bac\}$ par exemple, à la lecture de ba on sait que le plus grand préfixe de U_0' qui soit suffixe est ba, or ba admet un autre suffixe strict qui est le mot $a \in U_0'$ qu'il faut donc produire.

En résumé, pour chaque état de l'automate correspondant à un préfixe v d'un mot de U, on souhaite donc savoir quel est le plus long mot *de* U qui est un suffixe strict de v, s'il existe. Noter que les transitions d'échec nous donnent déjà le plus long mot *préfixe de* U qui est un suffixe strict de v. Noter aussi que, si le préfixe v obtenu dans la lecture est lui-même dans U, alors il fournit aussi un résultat ; mais ses suffixes stricts qui sont également dans U fournissent d'autres résultats.

On va donc calculer des pointeurs supplémentaires, appelés *raccourcis*, qui nous diront pour chaque préfixe de U (correspondant à un état de l'automate) quel est leur suffixe strict le plus long qui soit dans U. On le calcule inductivement dans la construction. Pour le cas de base, la racine n'a pas de pointeur raccourci. Pour l'induction, quand on crée un nœud n depuis un nœud n' pour une lettre a, on regarde le pointeur d'échec p qu'on définit pour n dans les questions précédentes. On se rappelle que, identifiant les nœuds à des mots par abus de notation, on sait que p est le plus grand préfixe d'un mot de U qui est un suffixe strict du mot de n. Mais alors, le plus grand suffixe strict de n qui soit un mot de U est le plus grand suffixe (non nécessairement strict) de p qui soit un mot de U ; c'est évident que tout tel mot convient, et il ne peut pas être plus long par maximalité de p. Donc, si $p \in U$, alors le raccourci de n pointe vers p. Sinon, le raccourci de n pointe vers le raccourci de p (en particulier si p n'a pas de raccourci alors n non plus).

Ce calcul s'effectue en temps constant à chaque ajout de nœud, donc la complexité de construction de l'automate est inchangée : elle reste à $O(|U|)$.

Maintenant, pour utiliser les raccourcis, on lit le mot v dans l'automate, et à chaque lettre on va produire tous les mots de U qui finissent à cet endroit. L'automate nous donne le plus long préfixe d'un mot de U qui finit à cet endroit, identifié à un état n. Il faut donc produire n comme résultat si n est final, et ensuite trouver les suffixes stricts de n qui sont également dans U, en suivant les raccourcis, pour trouver ces suffixes les uns après les autres du plus grand au plus petit. Attention, on ne suit le raccourci que "temporairement", c'est-à-dire que l'automate ne change pas d'état pendant qu'on suit des raccourcis dans l'algorithme.

La complexité est en revanche altérée car on passe du temps supplémentaire à suivre les raccourcis, et c'est potentiellement inévitable : le nombre de résultats est potentiellement $O(|U| \times |v|)$, par exemple si on cherche $U = \{a, \ldots, a^k\}$ dans $v = a^n$ on a $O(k \times n)$ résultats. Mais on peut remarquer qu'à chaque fois qu'on suit un raccourci c'est pour produire un nouveau résultat. Ainsi, la complexité de la phase de lecture est bornée par $O(|v| + M)$ où M est le nombre total de résultats produits : c'est $O(|U| \times |v|)$ dans le pire cas mais c'est bien évidemment souvent meilleur en pratique. La complexité totale de l'algorithme est donc en $O(||U|| + |v| + M)$ où $||U||$ dénote la longueur totale des mots de U.