CCP2019

Corrigé

Yann Salmon

14 août 2019

1 Inversions de permutations

1.1 Tri et inversions

Question 1. 1 a une inversion avec 0 ; 4 en a avec 0, 2 et 3 ; 0 aucune ; 2 aucune ; 5 avec 3. Soit un total de 5 inversions.

Question 2.

```
def tri_bulle(L) :
    for i in range(len(L)-1, 0, -1) :
        for j in range(len(L)-1, len(L)-i-1, -1) :
            if L[j] < L[j-1] :
                L[j], L[j-1] = L[j-1], L[j]
```

Question 3. L'algorithme échange $L[j]$ et $L[j-1]$ lorsque $L[j] < L[j-1]$: cela supprime une inversion. Montrons qu'on en n'a pas ainsi créé d'autre. La position relative des $L[i]$ pour $i < j-1$ par rapport à aux valeurs initialement en $j-1$ et j ne change pas, donc le nombre d'inversions non plus. De même pour les $L[i]$ avec $i > j$.

Question 4.

```
def nombre_inversions(L) :
    total = 0
    for i in range(len(L)-1, 0, -1) :
        for j in range(len(L)-1, len(L)-i-1, -1) :
            if L[j] < L[j-1] :
                L[j], L[j-1] = L[j-1], L[j]
                total = total + 1
    return total
```

1.2 Table d'inversion d'une permutation

Question 5. Il s'agit de $[1, 3, 0, 0, 1, 0]$.

Question 6. Soit σ une permutation de taille $n \geqslant 1$. Avec les notations de l'énoncé, pour tout $i \in [\![0\,,n[\![$,

$$\begin{aligned} \alpha_i &= \operatorname{Card}\left\{j \in [\![i+1\,,n[\![\;\middle|\; \sigma_j < \sigma_i\right\} \\ &\leqslant \operatorname{Card}[\![i+1\,,n[\![\\ &= n-i-1, \end{aligned}$$

donc

$$\alpha_i \in \{0, \dots, n-i-1\},$$

ce qui, avec le décalage d'indice, correspond au terme général du produit cartésien définissant E_n.

Question 7. On a $\alpha_0 = \text{Card}\left\{j \in [\![1\,,\,n[\![\;\middle|\; \sigma_j < \sigma_0\right\}$. Mais puisque $\sigma_0 \not< \sigma_0$, c'est aussi $\text{Card}\left\{j \in [\![0\,,\,n[\![\;\middle|\; \sigma_j < \sigma_0\right\}$. Puisque σ est une permutation, on a donc $\alpha_0 = \text{Card}\,\{j \in [\![0\,,\,n[\![\mid j < \sigma_0\}$, ce qui vaut σ_0 par construction des entiers naturels.

Question 8. Compte tenu de l'égalité des cardinaux, il suffit de montrer l'injectivité.

Soit $n \geqslant 1$ et soit σ et σ' deux permutations de taille n qui ont la même table α. On va montrer que $\sigma^{-1} = \sigma'^{-1}$.

En effet, l'indice du zéro est nécessairement $\min\{k \in [\![0\,,\,n[\![\mid \alpha_k = 0\}$: c'est forcément un tel k (ce qui assure qu'il existe de tels k), et si $\sigma_k = 0$ et $k' < k$ alors $\alpha_{k'} > 0$ à cause du zéro situé à droite de k'.

Notons donc $k_0 = \sigma_0^{-1} = \sigma_0'^{-1}$. S'il existe $k \in [\![0\,,\,k_0[\![$ tel que $\alpha_k = 1$ alors la plus petite de ces positions est $\sigma_1^{-1} = \sigma_1'^{-1}$; sinon cette position est $\min\{k \in [\![k_0 + 1\,,\,n[\![\mid \alpha_k = 0\}$.

On note $k_1 = \sigma_1^{-1} = \sigma_1'^{-1}$. En traitant comme ci-dessus les différentes possibilités, on montre de même que $\sigma_2^{-1} = \sigma_2'^{-1}$. De proche en proche on détermine $\sigma^{-1} = \sigma'^{-1}$ puis $\sigma = \sigma'$.

Question 9.

```
def permutation_vers_table(L) :
    n = len(L)
    res = [0] * n
    for i in range(0, n) :
        compte = 0
        for j in range(i+1, n) :
            if L[j] < L[i] :
                compte = compte + 1
        res[i] = compte
    return res
```

Question 10. On exploite le fait que pour tout naturel k, $k = \text{Card}\,[\![0\,,\,k[\![$. La liste L contient le nombre d'éléments plus petits après l'indice courant, mais on peut calculer le nombre d'éléments plus petits à gauche si on progresse de gauche à droite dans la construction de σ. On se donne un tableau de booléens pour savoir quelles sont les valeurs déjà prises par la permutation ; on compte $L[i]$ valeurs non-prises pour trouver la valeur de σ_i.

```
def table_vers_permutation(L) :
    n = len(L)
    sigma = [0] * n
    deja_mis = [False] * n
    for i in range(0, n) :
        k = 0
        compte = 0
        while deja_mis[k] or compte < L[i] :
            if not deja_mis[k] :
                compte = compte + 1
            k = k + 1
        sigma[i] = k
        deja_mis[k] = True
    return sigma
```

2 Automates et langages rationnels

Question 11. Clairement, $\{\mathtt{a}\}^* \cup \{\mathtt{b}\}^* \subseteq \sqrt{L}$. Soit u un mot contenant un $\mathtt{a}$ ainsi qu'un $\mathtt{b}$. Alors uu présente un $\mathtt{a}$ (dans le deuxième u) après un $\mathtt{b}$ (du premier u), donc n'est pas dans L. Finalement, $\sqrt{L} = \{\mathtt{a}\}^* \cup \{\mathtt{b}\}^*$.

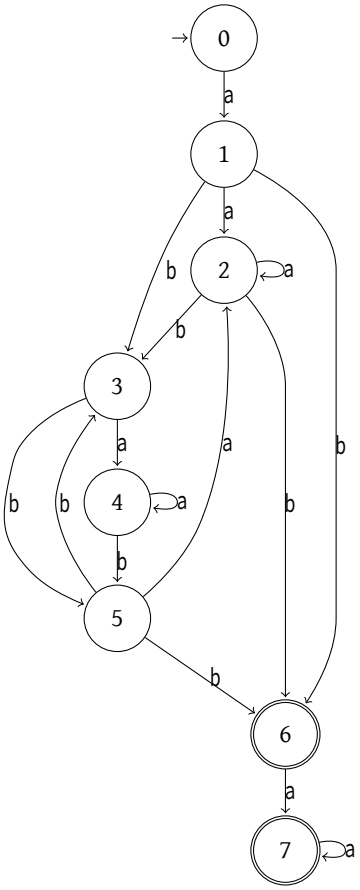
Question 12. De même que précédemment, $\sqrt{L} \cap \Sigma^*\{\mathtt{b}\}\Sigma^*\{\mathtt{a}\}\Sigma^* = \varnothing$ et $\sqrt{L} \cap \Sigma^*\{\mathtt{a}\}\Sigma^*\{\mathtt{b}\}\Sigma^* = \varnothing$. On trouve encore $\sqrt{L} = \{\mathtt{a}\}^* \cup \{\mathtt{b}\}^*$.

Question 13. Il suffit de prendre $A_{3,\{1\}}$, par définition.

Question 14. a. L' est décrit par l'expression régulière sur l'alphabet approprié $\mathtt{a}_1(\mathtt{a}_2 + \mathtt{b}_3\mathtt{a}_4^*\mathtt{b}_5)^*\mathtt{b}_6\mathtt{a}_7^*$.

b. $P = \{\mathtt{a}_1\}$, $S = \{\mathtt{b}_6, \mathtt{a}_7\}$, $F = \{\mathtt{a}_1\mathtt{a}_2, \mathtt{a}_1\mathtt{b}_3, \mathtt{a}_2\mathtt{a}_2, \mathtt{a}_2\mathtt{b}_3, \mathtt{b}_3\mathtt{a}_4, \mathtt{b}_3\mathtt{b}_5, \mathtt{a}_4\mathtt{a}_4, \mathtt{a}_4\mathtt{b}_5, \mathtt{b}_5\mathtt{a}_2, \mathtt{b}_5\mathtt{b}_3, \mathtt{a}_1\mathtt{b}_6, \mathtt{a}_2\mathtt{b}_6, \mathtt{b}_5\mathtt{b}_6, \mathtt{b}_6\mathtt{a}_7, \mathtt{a}_7\mathtt{a}_7\}$.

c. Le langage L' ne contient pas le mot vide, donc son automate de Glushkov est :



Question 15. On applique l'algorithme classique. Seuls les états accessibles sont considérés

	a	b
$\{0\}$	$\{1\}$	
$\{1\}$	$\{2\}$	$\{3,6\}$
$\{2\}$	$\{2\}$	$\{3,6\}$
$\{3,6\}$	$\{4,7\}$	$\{5\}$
$\{4,7\}$	$\{4,7\}$	$\{5\}$
$\{5\}$	$\{2\}$	$\{3,6\}$

Les états finals sont $\{3,6\}$ et $\{4,7\}$.

2.1 Propriétés de la racine carrée d'un langage rationnel

Question 16.

$$\begin{aligned} u \in \sqrt{L} &\Leftrightarrow uu \in L \\ &\Leftrightarrow \delta^*(q_0, uu) \in F \\ &\Leftrightarrow \delta^*(\delta^*(q_0, u), u) \in F \\ &\Leftrightarrow \exists q \in Q, \delta^*(q_0, u) = q \wedge \delta^*(q, u) \in F \\ &\Leftrightarrow \exists q \in Q, \delta^*(q_0, u) \in \{q\} \wedge \delta^*(q, u) \in F \\ &\Leftrightarrow u \in L_{q_0,\{q\}} \wedge u \in L_{q_0,F}. \end{aligned}$$

Question 17. On a alors $\sqrt{L} = \bigcup_{q \in Q} L_{q_0,\{q\}} \cap L_{q_0,F}$: c'est une union finie d'intersections finies de langages rationnels, c'est donc un langage rationnel.

Question 18. Soit $u \in \sqrt{L} \odot \sqrt{L}$. Il existe $v \in \sqrt{L}$ tel que $u = vv$. Mais $vv \in L$ par définition de $\sqrt{L}$, donc $u \in L$. D'où l'inclusion.

3 Algorithmique des mots sans facteur carré

3.1 Fonctions utiles sur les listes

Question 19.

```
let rec longueur = function
  | [] -> 0
  | _::t -> 1 + longueur t
```

Question 20.

```
let rec sous_liste l k long =
  if k > 0 then
    sous_liste (List.tl l) (k-1) long
  else if long = 0 then
    []
  else
    (List.hd l) :: sous_liste (List.tl l) 0 (long-1)
```

3.2 Un algorithme naïf

Question 21. 1 et 3 contiennent une répétition, pas les autres.

Question 22. On peut supposer que w commence par a, et que l'autre lettre est b. Pour être sans répétition, après a on ne peut poursuivre que par b, puis à nouveau par un a. Si on essaye d'ajouter une quatrième lettre, cela ne peut être a (sinon on aurait un facteur aa), mais pas b non plus car alors w commence par abab et a donc la répétition ab. D'où le résultat par contraposée.

Question 23. On coupe la liste en son milieu et on vérifie que les deux parties sont égales. J'exploite l'égalité de listes de OCaml ; c'est une comparaison terme à terme.

```
let estCarre w =
  let l = longueur w in
  if l mod 2 = 1 then
    false
  else let ls2 = l/2 in
    (sous_liste w 0 ls2) = (sous_liste w ls2 ls2)
```

Question 24. Dans le modèle de complexité proposé, le calcul de longueur et les extractions de sous-listes ne coutent rien et la comparaison est en $O(|w|/2)$. D'où une complexité linéaire.

Question 25. Il faut faire attention à ne pas se retrouver avec une liste de taille $< 2m$. Pour éviter de recalculer la longueur à chaque fois on utilise une fonction auxiliaire.

```
let contientRepetitionAux w m =
  let l = longueur w in
  let rec tester k w' =
    if l < k+2*m then
      false
    else match w' with
      | [] -> false
      | _::t -> estCarre (sous_liste w' 0 (2*m)) || tester (k+1) t
  in
  tester 0 w
```

Question 26. Soit c un facteur carré de w : il existe x, y, z tels que $w = yxxz$ et $c = xx$. On a $|w| = |y| + |z| + 2|x|$ donc $|x| \leqslant \frac{|w|}{2}$.

Question 27. On procède naïvement en testant tous les m possibles.

```
let contientRepetition w =
  let l = longueur w in
  let rec tester m =
    if m > l/2 then
      false
    else
      contientRepetitionAux w m || tester (m+1)
  in
  tester 1
```

Question 28. `contientRepetitionAux` est en $O(m|w|)$. D'où la complexité totale en $O(|w|^2)$.

3.3 Algorithme de Main–Lorentz

Question 29. Il s'agit de ababa.

Question 30. Supposons que uv contient un carré centré sur u. En reprenant les notations de l'énoncé, on a $w = w'w''$, $u = u'ww' = u'w'w''w'$ et $v = w''v''$. Notons i la position du début de w'' dans u, de sorte que $u'w' = u[0, i-1]$ et $w''w' = u[i, |u|-1]$. Par construction, w' est un suffixe commun à $u[0, i-1]$ et u, donc $|\operatorname{lcs}(u[0, i-1], u)| \geqslant |w'|$, et w'' est un préfixe commun à $u[i, |u|-1]$ et v, donc $|\operatorname{lcp}(u[i, |u|-1], v)| \geqslant |w''|$. Ainsi, la quantité étudiée est plus grande que $|w'| + |w''|$, qui vaut $|w|$, soit encore $|u[i, |u|-1]| = |u| - 1 - i + 1 = |u| - i$. D'où le sens direct.

Montrons la réciproque. Travaillons dans $u[i, |u|-1]$, mot de longueur $|u| - i$. Ce mot commence par les lettres de $\operatorname{lcp}(u[i, |u|-1], v)$ et s'achève par des lettres de $\operatorname{lcs}(u[0, i-1], u)$ (car c'est un suffixe de u). La relation sur les longueurs assure que ces deux mots comprennent toutes les lettres de $u[i, |u|-1]$: si l'égalité est réalisée, alors $u[i, |u|-1] = \operatorname{lcp}(u[i, |u|-1], v) \cdot \operatorname{lcs}(u[0, i-1], u)$; sinon, il y a chevauchement mais on peut l'éliminer en prenant un préfixe w'' et un suffixe w' un peu plus courts de sorte que $u[i, |u|-1] = w''w'$, $v = w''v''$ et $u[0, i-1] = u'w'$, ce qui établit le résultat.

Question 31. $\operatorname{pref}_u = [6, 1, 0, 0, 0, 1]$ et $\operatorname{pref}_{u,v} = [1, 3, 0, 0, 0, 1]$.

Question 32.

i	f	g	pref[i]
0	–	0	12
1	1	3	2
2	1	3	1
3	3	3	0
4	4	12	8
5	4	12	2
6	4	12	1
7	4	12	0
8	4	12	4
9	4	12	2
10	4	12	1
11	4	12	0

Question 33. Notons v le miroir de u. Soit i un indice. $v[0, i]$ est le miroir de $u[|u|-1-i, |u|-1]$, donc, puisque le passage au miroir change les préfixes en suffixes, on a $\operatorname{suff}_u[i] = \operatorname{pref}_v[|u|-1-i]$. On peut donc calculer suff_u en calculant v puis pref_v puis en renversant le tableau obtenu : les opérations nouvelles sont linéaires, donc on reste dans la même gamme de complexité que pour le calcul de pref_u.

Question 34. On calcule les tableaux suff_u et $\operatorname{pref}_{u,v}$ à l'aide des algorithmes précédents puis on vérifie par essais exhaustifs (avec une boucle) l'existence d'un i tel que $\operatorname{suff}_u[i-1] + \operatorname{pref}_{u,v}[i] \geqslant |u| - i$. Le cas $i = 0$ doit être géré à part car $\operatorname{suff}_u[-1]$ n'existe pas (conceptuellement c'est zéro).

Question 35. Le calcul des deux tableaux coute $O(|u|)$ (admis par l'énoncé). Le reste de l'algorithme ne fait aucune comparaison de caractère, d'où la complexité globale $O(|u|)$.

Question 36. Soit w le mot à étudier. Si $|w| \leqslant 1$, on renvoie Faux. Sinon on découpe $w = uv$ avec u et v de taille moitié (à un près). On étudie récursivement u, puis si nécessaire v. S'il n'y a aucun carré, on applique la procédure définie ci-dessus pour chercher un carré centré sur u, puis si besoin une procédure semblable pour un carré centré sur v.

Question 37. On a la relation de récurrence $C_n \approx 2\,C_{\frac{n}{2}} + O(n)$. C'est la même relation que celle du tri fusion, donc on a de même $C_n = O(n\,\log(n))$.