TD 2 :
Recombinaisons et recherche en espace constant

Correction

1 Recombinaison de mots

Question 1 : Correction : Soient u et v deux mots avec $|u| = |v| = n$. La réponse est basée sur la propriété suivante (démonstration triviale) : v est une rotation circulaire de u si v est un motif de $u.u$. En appliquant l'algorithme de Knuth-Morris-Pratt ou celui de Simon, on obtient un automate du motif (v) en $\Theta(n)$ et la reconnaissance du motif en $O(2|u|)$, donc une complexité finale en $O(n)$.

Question 2 : Correction :

$\Leftarrow$ On suppose $u \sim v$. Par définition de $\sim$, il existe (u_1, u_2, u_3) un triplet de mots tel que $u = u_1u_2u_3$ et $v = u_1\widetilde{u_2}u_3$. Alors $au = au_1u_2u_3$ et $av = au_1\widetilde{u_2}u_3$, donc $au \sim av$.

$\Rightarrow$ On suppose $au \sim av$. Par définition de $\sim$, au se décompose en $u_1u_2u_3$ et av en $u_1\widetilde{u_2}u_3$.

— Si $u_1 \neq \epsilon$, alors $u_1 = au_1'$ et donc $u = u_1'u_2u_3$ et $v = u_1'\widetilde{u_2}u_3$, donc $u \sim v$.

— Si $u_1 = \epsilon$, alors $au = u_2u_3$ et $av = \widetilde{u_2}u_3$

  — Si $u_2 = \epsilon$ (car $u \sim v$ autorise $u = v$) ou $u_2 = a$, on a $u \sim v$.

  — Si $|u_2| > 1$, alors u_2 commence et se termine par a, donc $u_2 = au_2'a$ et $au = au_2'au_3$ et $av = \widetilde{au_2'}au_3 = a\widetilde{u_2'}au_3$. On déduit $u = u_2'au_3$ et $v = \widetilde{u_2'}au_3$, d'où $u \sim v$.

On peut généraliser cette propriété et démontrer que si $u = u_1u'u_3$ et $v = u_1v'u_3$ avec u' et v' ni ne commençant par la même lettre ni ne finissant par la même lettre, alors $u \sim v$ ssi $u' = \widetilde{v'}$.

Question 3 : Correction : Les entrées de l'algorithme sont u et v (de longueur n), la sortie est vraie ssi $u \sim v$.

```
i := 1; j := n;
tant que i <= n et u[i] = v[i] faire
  i := i+1;
tant que j > i et u[j] = v[j] faire
  j := j-1;
tant que j > i faire
  si u[i] != v[j] alors
    retourner faux
  sinon
    i := i+1; j := j-1
retourner vrai
```

La complexité est clairement en $O(n)$. La correction est une conséquence de la propriété de $\sim$ de la question précédente.

Question 4 : Correction : Supposons que $u \sim \widetilde{v}$. D'après la définition de $\sim$, u peut s'écrire $u_1u_2u_3$ et $\widetilde{v}$ peut s'écrire $u_1\widetilde{u_2}u_3$. D'après la définition du retournement, $v = \widetilde{u_3}u_2\widetilde{u_1}$. Or :

$$u_1u_2u_3 \circlearrowleft u_2u_3u_1 \sim u_2\widetilde{u_3u_1} = u_2\widetilde{u_1}\widetilde{u_3} \circlearrowleft \widetilde{u_3}u_2\widetilde{u_1} = v$$

Question 5 : Correction : Supposons que $u \sim\circlearrowleft v$. D'après les définitions de $\sim$ et $\circlearrowleft$, il existe u_1, u_2, u_3, v_1, v_2 tels que $u = u_1u_2u_3$, $u_1\widetilde{u_2}u_3 = v_1v_2$ et $v = v_2v_1$.

Les différents cas dépendent de la position relative de v_1 comme préfixe de $u_1\widetilde{u_2}u_3$:

1. Si $|v_1| \leq |u_1|$, alors u_1 est de la forme v_1v_1' pour un v_1' tel que $v_2 = v_1'\widetilde{u_2}u_3$, d'où $u = v_1v_1'u_2u_3 \circlearrowleft v_1'u_2u_3v_1 \sim v_1'\widetilde{u_2}u_3v_1 = v_2v_1 = v$.
2. Le cas où $|v_2| \leq |u_3|$ est similaire.
3. Le cas restant est $|u_1| < |v_1| < |u_1\widetilde{u_2}|$. On peut alors décomposer u_2 sous une forme $u_2'u_2''$ telle que $v_1 = u_1\widetilde{u_2''}$ et $v_2 = \widetilde{u_2'}u_3$. D'où on tire

$$u = u_1u_2'u_2''u_3 \circlearrowleft u_2''u_3u_1u_2' \sim u_2''\widetilde{u_3u_1}u_2' = (u_2''\widetilde{u_1})(\widetilde{u_3}u_2') = \widetilde{v_1}\widetilde{v_2} = \widetilde{v}.$$

Question 6 : Correction : Un algorithme naïf va énumérer toutes les rotations de u (en $O(n)$) et celles de v (en $O(n)$) et tester si elles se recombinent ($u \sim v$) en $O(n)$. Ça donne du $O(n^3)$.

Ce qui est un peu dommage c'est que la définition de $\approx$ (et l'algorithme qui en découle) introduit deux rotations et on sait bien que deux rotations ne mènent pas plus loin qu'une. Il se trouve qu'en faisant attention, on peut permuter la rotation et la recombinaison, ce qui permettra de fusionner les deux rotations en une seule. Car $u \circlearrowleft u$ et $u \circlearrowleft\circlearrowleft u'$ ssi $u \circlearrowleft u'$.

On démontre donc que $u \approx v$ ssi $u \circlearrowleft\sim v$ ou $u \circlearrowleft\sim \widetilde{v}$.

$\Rightarrow$ Supposons que $u \circlearrowleft\sim\circlearrowleft v$. D'après la question précédente, $u \circlearrowleft\circlearrowleft\sim v$ ou $u \circlearrowleft\circlearrowleft\sim \widetilde{v}$, qui est équivalent à (élimination de la double rotation) $u \circlearrowleft\sim v$ ou $u \circlearrowleft\sim \widetilde{v}$.

$\Leftarrow$ Si $u \circlearrowleft\sim v$, comme $v \circlearrowleft v$ alors $u \circlearrowleft\sim\circlearrowleft v$. Si $u \circlearrowleft\sim \widetilde{v}$, d'après la question 4 on a $u \circlearrowleft\circlearrowleft\sim\circlearrowleft v$, donc $u \circlearrowleft\sim\circlearrowleft v$, c.q.f.d.

En utilisant cette propriété, on obtient un algorithme plus efficace. Il énumère *que* les rotations de u (en $O(n)$) et puis teste si elles se recombinent avec v ou $\widetilde{v}$ (en $O(n)$). Donc une complexité globale de $O(n^2)$.

2 Recherche en espace constant

2.1 Recherche d'un motif auto-maximal

Question 7 : Correction : Prenons déjà des exemples pour la notation $\mathit{MaxSuf}(w)$. Soit l'alphabet $\Sigma = \{a, b\}$ et fixons l'ordre $a < b$. $\mathit{MaxSuf}(bab) = bab$ et $\mathit{MaxSuf}(abba) = bba$. Si l'ordre est $a > b$ alors $\mathit{MaxSuf}(bab) = ab$ et $\mathit{MaxSuf}(abba) = abba$. Pour ces deux mots, pour l'un des ordres le mot est auto-maximal.

Prenons le mot $abaa$. Pour l'ordre $a > b$ alors $\mathit{MaxSuf}(abaa) = aa$, et pour l'ordre $a < b$, $\mathit{MaxSuf}(abaa) = baa$.

Question 8 : Correction : Soit w tels que $\mathit{MaxSuf}(w) = w$ et soit w_1 un préfixe de w, ainsi $w = w_1w_2$.

Supposons par l'absurde que w_1 ne soit pas auto-maximal, c'est-à-dire qu'il existe $u_1 \sqsupseteq w_1$ tel quel $u_1 > w_1$. Comme u_1 est plus court que w_1, il existe nécessairement un indice i tel que $u_1[i] > w_1[i]$ et $u_1[1, i-1] = w_1[1, i-1]$. Mais en concaténant w_2 à u_1 on obtient $u_1w_2 > w_1w_2 = w$ et u_1w_2 est bien un suffixe de w, ce qui contredit l'hypothèse que w est auto-maximal.

```
periode_naive(w,j) :=
  pe := 1;
  pour i de 2 à j faire
    si w[i] ≠ w[i − pe] alors pe := i;
  retourner pe;
```

Question 9 : Correction : Pour démontrer que periode_naive(w,j) calcule $Period(w[1,j])$, il suffit de démontrer que

$$I(i): \quad pe = Period(w[1, i-1])$$

est un invariant de la boucle **pour**.

Pour cela, il faut démontrer deux propriétés :

— Condition initiale : $I(2)$, càd $1 = Period(w[1,1])$ est vrai. En effet, un mot de longueur 1 a la période 1.

— Préservation : Si $I(i)$ est vrai en début de la boucle alors après la condition et l'exécution du corps de la boucle $I(i+1)$ est vrai.

Supposons que $I(i)$ est vrai.

— Si $w[i] = w[i-pe]$ alors pe ne change pas et on démontre que pe est aussi période de $w[1,i]$ car pour tout $k \in [1, i-pe]$ $w[k] = w[k+pe]$ pour $k < i-pe$ (car $I(i)$) et $w[i-pe] = w[i]$. Par hypothèse d'induction, pe est minimal pour $w[1,i-1]$ donc aussi pour $w[1,i]$, donc pe est $Period(w[1,i])$, donc $I(i+1)$ est vrai.

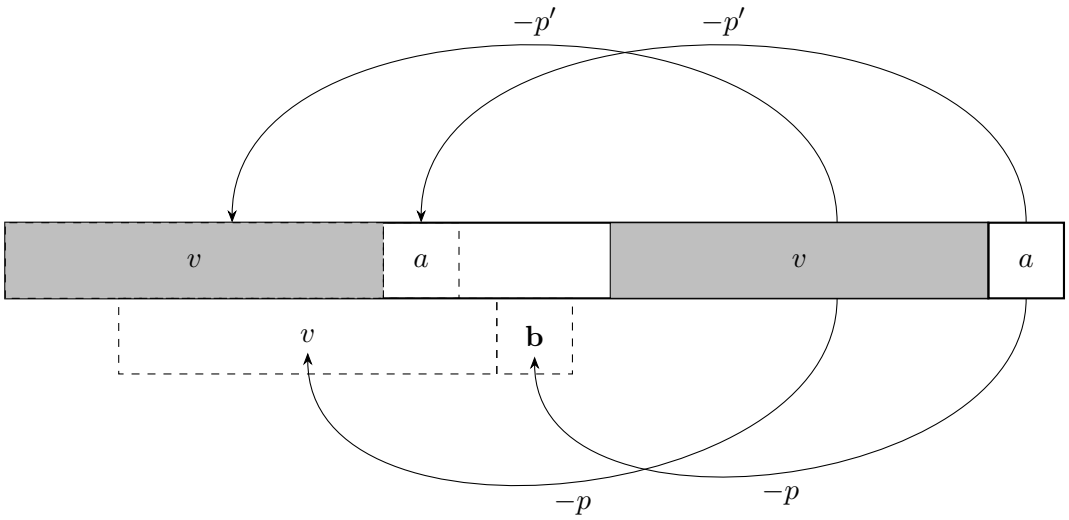
— Si $w[i] \neq w[i-pe]$, notons $a = w[i]$, $b = w[i-pe]$.

(A) On démontre que $a < b$ si w auto-maximal. Par hypothèse d'induction, $w[1,i-1] = w_1 b w_2$ avec $|w_1| = i-1-pe$ et $w_1 \sqsupset w[1,i-1]$. Donc $w[1,i] = w_1 b w_2 a$ et est auto-maximal car préfixe de w (propriété démontrée à la question précédente). Supposons par absurde que $a > b$. Comme w_1 est suffixe de $w[1,i-1]$, alors $w_1 a$ est un suffixe de $w[1,i]$, donc strictement supérieur à $w[1,i]$, contradiction avec $w[1,i]$ auto-maximal.

(B) On démontre que $Period(w[1,i]) = i$.[1] Supposons par absurde que $p' = Period(w[1,i]) < i$. Alors p' est également une période de $w[1,i-1]$, donc $p' \geq p = Period(w[1,i-1])$. Mais comme $w[i-p] \neq w[i] = w[i-p']$ on déduit que $p' > p$.

On suppose que $w[1,i-p'] = va$ et comme p' est une période de $w[1,i]$ alors $w[1,i] = w'_1 va$. Mais $p' > p = Period(w[1,i-1])$, alors $w[1,i-p'] \sqsubset w[1,i-1-p]$, donc $w[1,i-p] = w_1 vb = vaw_3$. Or $vaw_3 < vb$, contradiction avec $w[1,i-p]$ est auto-maximal (car préfixe de w). On déduit que $Period(w[1,i]) = i$, donc $I(i+1)$ est vrai après l'instruction conditionnelle.

Le schéma ci-dessous montre la seconde étape de la preuve, la duplication de va en vb par shift de $-p$ est toujours possible car $p < p'$ par hypothèse.



Question 10 : Correction : L'algorithme de Morris-Pratt (page 12 notes de cours) utilise la

1. Une preuve plus courte utilise le résultat du TD 01 sur les périodes.

fonction π pour calculer la position suivante dans le motif, j, quand $P[j+1] \neq T[i]$. Or la fonction π occupe $m = |P|$ entiers. Si on dispose de la période de $P[1, j]$ alors $\pi(j) = j - Period(P[1, j])$ (propriété démontré au TD précédent). Pour les motifs auto-maximaux, cette position est calculée avec periode_naive en $\Theta(j)$. On maintient donc *une* variable p contenant la période du préfixe courant $P[1, j]$, qui nécessite un recalcul lorsque p devient plus grand que j (sinon la période reste constante). Clairement, ce programme travaille avec un nombre fixé de variables.

```
MPAuto(P, T) : liste :=
  p := 1;
  j := 0;
  L := empty;
  pour i de 1 à |T| faire
    tant que j ≥ 0 ∧ P[j+1] ≠ T[i] faire
      j := j − p;
      si p > j alors p := periode_naive(P, j)
    si P[j+1] = T[i] alors
      j := j+1;
      si P[j] ≠ P[j−p] alors
        p := j;
    si j = |P| alors
      AjoutListe(L, i−|P|)
      j := j − p;
  retourner L
```

On pose l'invariant en début de boucle : $p = Period(P[1, j])$ avec $Period(\epsilon) = 1$. Ainsi, dès que $j \geq 0$, nous avons $\pi(j) = j - p$ et si $j - p \geq p$ alors $Period(P[1, j - p]) = p$. De plus, si $P[j] \neq P[j - p]$, nous avons démontré (question précédente) que pour P auto-maximal p est j. Ces propriétés assurent que l'invariant est préservé dans la boucle principale, donc celle ci se comporte comme l'algorithme de Morris-Pratt.

Prouvons désormais la complexité en temps : dans la boucle **tant que**, à chaque décrémentation de la variable j d'un offset p, un recalcul a lieu, de complexité j, uniquement si $j < p$.

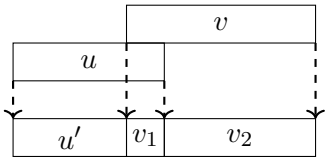
Rappel : Soit la fonction de potentiel *pot* définie sur les états de l'algorithme, la complexité amortie de l'opération *op* telle que $s \xrightarrow{op} s'$ est $a(op) = t(op) + pot(s') - pot(s)$. Pour un programme ayant une séquence d'opérations $\{op_i\}_{i \in [1,n]}$ en partant d'un état s_0, on obtient $\Sigma_{i=1}^{n} t(op_i) \leq \Sigma_{i=1}^{n} a(op_i) + pot(s_0)$.

Soit $pot(s)$ est la valeur de j en s. Pour une itération de la boucle **tant que**, $a(op) = t(op) + (-p)$ où $t(op)$ est le coût de calcul de la période, donc $t(op) = j < p$, d'où $a(op) \leq 0$. Pour une itération de la boucle **pour**, $a(op') = t(op') + pot(s_{i+1}) - pot(s_i)$ où $t(op')$ est le coût des opérations dans le corps de la boucle, donc $t(op') \leq 1$ et j augmente d'au plus 1. Donc $a(op') \leq 2$. Alors le temps d'exécution de la boucle est $\leq 2n + pot(j_0) = 2n$. D'où la complexité $O(n)$ en temps, par analyse amortie.

2.2 Recherche de texte en espace constant

Question 11 : <u>Correction</u> : Par l'absurde, si u apparaît avec un décalage $j \in \{i - |u| + 1, \ldots, i\}$, alors u commence avant v et fini après le début de v : u intersecte le début du mot v.

— Cas 1 : u termine avant v. Alors $u = u'v_1$ et $v = v_1v_2$. Alors $v = v_1v_2 > v_2$ par auto-maximalité mais alors $v_1v_1v_2 = v_1v$ est un suffixe de w plus grand que $v_1v_2 = v = MaxSuf(w)$, absurde.



— Cas 2 : u se termine après v. Alors $u = u_1 v u_2$, or $v u_2 v > v$ (les deux mots commencent par v et le premier est plus long) ce qui est absurde.