TD 3 :
Palindromes et fonction témoin – variations sur Boyer-Moore

Correction

1 Calcul d'une fonction témoin

Question 1 : Correction : Rappel $\mathit{Pref}(i) = \max\{j \mid 0 \leq j \leq m-i+1 \wedge P[1,j] = P[i,i+j-1]\}$ et $m = |P|$.
On démontre que si $\mathit{Pref}(i+1) = m-i$ alors $tem(i) = 0$, sinon $tem(i) = \mathit{Pref}(i+1)+1$. (Le $i+1$ vient du fait que l'indice de début de mot soit 1.)

- Si $\mathit{Pref}(i+1) = m-i$ alors, $\forall k \in [1, m-i]$ $P[i+k] = P[k]$ (par définition de *Pref*), donc, $tem(i) = 0$ (par définition de tem) .
- Si $\mathit{Pref}(i+1) \neq m-i$ alors $\mathit{Pref}(i+1) < m-i$ alors $P[\mathit{Pref}(i+1)+1] \neq P[i+\mathit{Pref}(i+1)+1]$ (par maximalité de *Pref*). On définit $tem(i) = \mathit{Pref}(i+1)+1$.

Comme *Pref* est calculé en $O(m)$, on obtient un algorithme pour tem en $O(m)$.

Question 2 : Correction : Soit i, j incohérentes avec $1 \leq i < j \leq n-m+1$ et $j-i < m$. Par absurde, on suppose que $T[i, i+m-1] = P = T[j, j+m-1]$.
Comme $tem(j-i) \neq 0$ alors $P[j-i+tem(j-i)] \neq P[tem(j-i)]$
Mais $P[j-i+tem(j-i)] = T[i-1+j-i+tem(j-i)]$ (par $T[i,i+m-1] = P$)
et $T[j-1+tem(j-i)] = P[tem(j-i)]$ (par $P = T[j, j+m-1]$), contradiction.

Question 3 : Correction : Soit $i < j < k$ trois positions telles que i, j cohérentes et j, k cohérentes et $m = |P|$.
Si $k-i \geq m$ alors i, k sont trivialement cohérentes.
Si $k-i < m$ alors $j-i < m$ et $k-j < m$ et $tem(j-i) = 0 = tem(k-j)$ (par cohérence),
donc $\forall u \in [1, m-(j-i)], P[j-i+u] = P[u]$ (par déf $tem(j-i) = 0$)
et $\forall v \in [1, m-(k-j)], P[k-j+v] = P[v]$ (par déf $tem(k-j) = 0$)
donc $\forall u \in [1, min(m-(j-i), m-(k-j))], P[k-i+u] = P[j-i+k-j+u] = P[k-j+u] = P[u]$
Or, $m-(j-i) > m-(k-i)$ et $m-(k-j) > m-(k-i)$
D'où $\forall u \in [1, m-(k-i)] P[k-i+u] = P[u]$.

Question 4 : Correction : L'invariant $I(k)$ à démontrer pour la boucle principale de l'algorithme est "(a) les positions de la pile sont cohérentes entre elles, et (b) elles sont triées en ordre croissant, et (c) elles sont toutes $< k$. "

A l'entrée de la boucle, l'invariant est trivialement satisfait car la pile est vide.

Montrons que si $I(k)$ est vraie pour $1 \leq k \leq n-m+1$, alors après l'exécution du corps de la boucle on retrouve $I(k+1)$. Soit π le contenu de la pile au début de la k-ème itération.
La première instruction de la boucle change π en $\pi.k$, ce qui satisfait les parties (b) et (c) de $I(k+1)$. Montrons que la boucle interne rétablit la partie (a) pour $I(k+1)$.

- Si $\pi = k$ alors cette boucle ne change pas la pile, et (a) est trivialement satisfaite.
- Si $\pi = \pi'.i.j$ alors $i < j$ (par hypothèse, et $j = k$ pour la première itération).
  - Si (i, j) sont cohérentes alors la boucle interne s'arrête ; par la transitivité de la relation de cohérence, π contient que des positions cohérentes.

— Si (i, j) sont incohérentes alors $j - i < m$, $tem(j - i) > 0$ et P ne peut pas commencer en i et en j (d'après question 2).
Si $T[j-1+tem(j-i)] \neq P[tem(j-i)]$ alors P ne peut pas commencer en j (car $tem(j-i) \geq 1$), donc on peut dépiler j ; la pile $\pi'.i$ contient que des positions cohérentes, donc la boucle interne s'arrête avec une pile qui satisfait (a).
Si $T[j-1+tem(j-i)] = P[tem(j-i)]$ (qui n'implique pas que P débute en j) alors supposons que $T[i, i+m-1] = P$, donc
$T[i-1+tem(j-i)] = P[tem(j-i)]$ (car $1 \leq tem(j-i)$)
$\neq P[j-i+tem(j-i)]$ (déf. $tem(j-i) > 0$)
$= T[i-1+j-i+tem(j-i)]$ (hypothèse $T[i, i+m-1] = P$ et $tem(j-i) \leq m-j-i$)
$= T[j-1+tem(j-i)]$, contradiction avec $T[j-1+tem(j-i)] = P[tem(j-i)]$. Donc P ne peut pas commencer en i et le contenu de la pile $\pi'.j$ est tel que π' contient que des positions cohérentes entre elles et j en tête de pile.

En conclusion, la boucle interne termine avec la pile contenant soit k (par élimination des autres positions), soit π' telle que toutes les positions sont cohérentes.

L'algorithme est en $O(n)$ car chaque position est ajoutée au plus une fois et retirée au plus une fois. Pour chaque ajout on a un test de cohérence et à chaque retrait un test de cohérence et de différence basés sur tem, donc en temps constant.

Question 5 : Correction : L'algorithme proposé s'écrit :

Algorithm 1: PrefPalNaif

Input : Un texte T de longueur n et *Possible* construit à la question 4.
Output: Un texte T' de longueur n contenant 0 et 1.

```
T' ← []*n
j ← (empty(Possible))  ? 0 : pop(Possible)
for i from n downto 1 do
    while i < j do
        j ← (empty(Possible))  ? 0 : pop(Possible)
    if j ≠ 0 and i − j < m and T[i] = P[i − j + 1] then
        T'[i] ← 1
    else
        T'[i] ← 0
return T'
```

On démontre que $T'[i, i+m-1] = 1^m$ ssi $T[i, i+m-1] = P$.

Si $T'[k, k+m-1] = 1^m$ avec $k \in Possible$[1] alors pendant m itérations à partir de $i = k+m-1$ à $i = k$ la condition du **if** a été satisfaite, donc $T[k, k+m-1] = P$. En effet, par la construction de *Possible*, toutes les positions $i \geq j$ avec j sur la pile sont cohérentes, donc pas de témoin de différence avec P sur la distance $i - j$.

Si $T[k, k+m-1] = P$ alors $k \in Possible$ (question précédente) et soit ℓ l'itération du dépilement de k de *Possible*.

Si $i = \ell \geq k+m-1$ alors les itérations de $k+m-1$ à k rempliront $T'[k, k+m-1] = 1^m$.

Si $k < \ell \leq k+m-1$ alors $T'[k, \ell] = 1^{\ell-k+1}$. On démontre que à l'itération précédente, donc $i = \ell+1$, on a $T'[\ell+1, k+m-1] = 1^{k+m-\ell-1}$. On a dépilé *Possible* quand $j = \ell+1$ et pour i entre $k+m-1$ et $\ell+1$ on a $T[i] = P[i-j+1]$ (car $T[k, k+m-1] = P$), donc $T'[i] = 1$ et $T'[\ell+1, k+m-1] = 1^{k+m-1-\ell}$.

1. La $k \in Possible$ condition est nécessaire, sinon considérez le contre-exemple $T = abaaba$ et $P = aba$.

L'algorithme est en $O(n)$ car il parcours T de n à 1 et *Possible* est trié (décroissant depuis la tête de pile) par construction.

Question 6 : Correction :

Algorithm 2: PrefPalNaif

```
Input  : Un texte T' de longueur n contenant des 0 et 1 et Possible.
Output: Une liste de positions i de Possible telles que T'[i, i+m-1] = 1^m
L ← []
while not empty(Possible) do
    i ← pop (Possible)
    c ← m
    while c > 0 and T'[i+c] = 1 do
        c ← c − 1
    if c = 0 then
        L ← L+[i]
return L
```

2 Recherche de palindromes pairs

Question 7 : Correction : L'algorithme naïf est de tester pour chaque position entre 1 et $\lfloor n/2 \rfloor$ si le préfixe $T[1, 2i]$ est un palindrome pair (centrée en i). Cet algorithme trouve le plus court prefpal, mais on peut l'adapter pour trouver tous les prefpal.

Algorithm 3: PrefPalNaif

```
Input  : Un texte T de longueur n.
Output: La longueur du plus petit prefpal de T.
for i = 1 to ⌊n/2⌋ do
    flag ← true
    j ← 1
    while j ≤ i and flag do
        if T[j] ≠ T[2*i+1−j] then
            flag ← false
        else
            j ← j + 1
    if flag then
        return 2*i
return 0
```

La complexité est majorée par $\Sigma_{i=1}^{n/2} i$ donc elle est en $O(n^2)$.

Question 8 : Correction : D'après la définition, la taille des prefpal d'un texte T est entre 0 (pas de prefpal) et $2 * \lfloor n/2 \rfloor$, donc elle est $\leq |T|$.
Soit le mot $T' = T\#\widetilde{T}$ avec # une lettre n'apparaissant pas dans T. La propriété suivante donne l'idée de la construction à utiliser pour l'algorithme :
Propriété : w est un préfixe et suffixe de T' t.q. $|w| = 2p \leq |T|$ ssi w est un prefpal de T.
Démonstration : w est un prefixe-suffixe de T' de taille $2p \leq |T|$ ssi

(def. préfixe-suffixe) pour tout $i \in [1, 2p]$ $w[i] = T'[i] = T[i] = T'[2 * |T| + 1 - |w| + i]$ ssi
(d'après $T' = T\#\widetilde{T}$) $i \in [1, p]$ $w[i] = \widetilde{T}[|T| - 2p + i]$ ssi
(d'après définition $\widetilde{T}$) $i \in [1, p]$ $w[i] = T[2p - i + 1] = w[2p - i + 1]$ ssi
(définition prefpal) w est prefpal de T.

Par conséquent, tout prefpal de T est un bord de T'. Or la fonction $\pi_{T'} : [1, |T'|] \to [0, |T'|-1]$ indique la taille maximale du bord de chaque préfixe de T'. Comme $\#$ n'est pas dans T, la longueur de ces bords est $\leq |T|$.

Proposition : Le plus long prefpal de T est $\max\{\pi^m_{T'}(|T'|) \mid 1 \leq m < n \text{ and } \pi^m_{T'}(|T|) \bmod 2 = 0\}$.
Le plus court prefpal est $\min\{\pi^m_{T'}(|T'|) \mid 1 \leq m < n \text{ and } \pi^m_{T'}(|T|) \bmod 2 = 0\}$.
Démonstration : $\pi^m_{T'}(|T'|)$ avec $m < n$ est un des bords des T' de longueur inférieure à n. En utilisant la propriété des bords (les bords non vides de x sont $Bord(x)$, $Bord^2(x)$, ..., $Bord^k(x)$ avec $Bord^{k+1}(x) = \epsilon$) appliquée à π, on obtient que les longueurs des bords de T' sont données dans les puissances de $\pi_{T'}(|T'|)$. Ils sont calculés en temps $O(|T'|)$ par l'algorithme de π, d'où la propriété.

Algorithm 4: PrefPalMaxMin

```
Input  : Un texte T de longueur n.
Output: Le plus long et le plus court prefpal de T.
T' ← T#T̃
π ← CalculPi(T')
j ← 2 * n + 1
max, min ← 0, n
while j > 0 do
    if π(j) pair then
        if π(j) > max then
            max ← π(j)
        if π(j) < min then
            min ← π(j)
if min = n then
    min ← 0
return max, min
```

Tous les pas de cet algorithme sont en $O(n)$.

Question 9 : Correction : Si $ray(i) = i$ alors $T[1, 2i]$ est un prefpal et il est pair. Donc le plus petit prefpal est celui de longueur $2k$ avec $k = min\{i \mid i = ray(i)\}$ et $k \neq 0$.

Question 10 : Correction : Soit $ray(i - k) \neq ray(i) - k$.
Cas 1 : $ray(i - k) < ray(i) - k$
$T[i - k - ray(i - k)] \neq T[i - k + 1 + ray(i - k)]$ (car $ray(i - k)$ est max)
Mais $T[i - k - ray(i - k)] = T[i + 1 + k + ray(i - k)]$ (car $ray(i) > ray(i - k) + k$)
Et $T[i + 1 + ray(i - k) - k] = T[i - ray(i - k) + k]$ (car $ray(i) > ray(i - k) - k + 1$)
donc $T[i + 1 + k + ray(i - k)] \neq T[i + k - ray(i - k)]$
donc $ray(i + k) \leq ray(i - k)$ (car ray est max)

Soit $j \leq ray(i - k)$ alors $T[i - k - (j - 1)] = T[i - k + 1 + j - 1]$
et $T[i + 1 - (k + j)] = T[i + k + j]$ (car $k + j \leq k + ray(i - k) < ray(i)$)
et $T[i - k + j] = T[i + 1 + k - j]$ (pour $k \leq j$ et $k - j < ray(i)$, pareil pour $j > k$)
donc $T[i + k + j] = T[i + k + 1 - j]$ et par conséquent $ray(i + k) \geq ray(i - k)$
Donc $ray(i + k) = ray(i - k)$

Cas 2 : Soit $ray(i-k) > ray(i) - k$ alors :
$T[i - ray(i)] \neq T[i + ray(i) + 1]$ (car $ray(i)$ est max)
Mais $T[i - k - ray(i) + k] = T[i - k + 1 + ray(i) - k]$ (car $ray(i) - k < ray(i-k)$)
Et $T[i + 1 + ray(i) - 2k] = T[i - (ray(i) - 2k)]$ (car $ray(i) > ray(i) - 2k$)
donc $T[i + k - ray(i) + k] \neq T[i + k + 1 + ray(i) - k]$
donc $ray(i + k) \leq ray(i) - k$ (car ray est max)

Soit $j \leq ray(i) - k$ alors $T[i + 1 + k + j - 1] = T[i - k - j + 1]$ (car $ray(i) \geq k + j$)
et $T[i - k - j + 1] = T[i - k + 1 + j - 1]$ (car $ray(i - k) > ray(i) - k \geq j$)
et $T[i - k + j] = T[i + 1 + k - j]$ (car $ray(i) \geq k - j$ pour $k > j$ sinon similaire)
donc $T[i + k + j] = T[i + k + 1 - j]$
donc $ray(i + k) \geq ray(i - k)$ (car ray est max)
Donc $ray(i + k) = ray(i) - k$

D'où l'égalité $ray(i + k) = min(ray(i) - k, ray(i + k))$ si $ray(i - k) \neq ray(i) - k$.

Question 11 : Correction : Si $ray(i - k) = ray(i) - k$ alors nous avons les mêmes égalités que dans l'exercice ci-dessus, c'est à dire :
pour tout $j \leq ray(i) - k$, $T[i + 1 + k + j] = T[i - k - j]$ (car $j + k \leq ray(i)$)
et $T[i - k - j] = T[i - k + 1 + j]$ (car $j \leq ray(i) - k = ray(i - k)$)
et $T[i + 1 + j - k] = T[i - (j - k)]$ (car $j - k \leq ray(i)$)

Donc $ray(i - k) = ray(i) - k \leq ray(i + k)$ et par conséquent $ray(i + k) = min\{j \mid T[i + ray(i) + j] \neq T[i + k + 1 - (ray(i) - k + j)]\}$, donc on n'a pas besoin de tester les lettres entre $i + k + 1$ et $i + ray(i)$.

Question 12 : Correction : On cherche $k = min\{i \mid i = ray(i)\}$. On va calculer $ray(j)$ pour les différentes positions j du mot T et on se sert des questions précédentes pour être plus efficaces :

- On va toujours garder en mémoire l'indice qui nous a permis de comparer le plus loin dans le mot. Il nous permettra de calculer les rayons de beaucoup d'indices uniquement en utilisant la propriété obtenue au point précédent [:-2].
- Lorsqu'on compare deux lettres, on compare en fait la lettre la plus à droite de la partie du mot visité avec une autre (distinction lettre comparée et comparant), et si la comparaison est fructueuse, on regarde la suivante à droite. Une lettre ne peut être vue de façon comparée et fructueuse qu'une seule fois. Le nombre de comparaison fructueuse est borné par n (après il n'y a plus de comparé). Lorsqu'une comparaison est infructueuse, on déplace automatiquement l'indice courant de un. L'indice courant est borné par n. Le nombre de comparaison infructueuse est aussi borné par n. Dans le pire des cas, il y a $2n$ comparaisons. S'il existe un prefpal, l'algorithme le trouve sans avoir visité ce qui se trouve à droite du prefpal. Si le prefpal existe, la recherche s'effectue en au pire $2s = O(s)$.

L'algorithme compare chaque fois soit des portions au delà de j et les portions comparées et égales on ne les touchera plus dans le test $T[j + k] = T[j + 1 - k]$ (car on compare au delà de $Ray[i]$). Ainsi, le nombre de comparaisons vaux n, d'où en algorithme en $O(s)$ si s est la taille du préfixe.

Question 13 : Correction : On double chaque lettre du mot T pour obtenir un mot T' de longueur $2n$. On montre alors qu'à tout sous-mot w' de T' qui est un palindrome pair, il correspond un sous-mot w de T qui est un palindrome.

Plus formellement, on pose le morphisme de mots φ défini sur les lettres par $\varphi : x \in \Sigma \mapsto x \cdot x$ puis $T' = \varphi(T)$.

Si σ est un sous-mot palindrome de $T = w_1 \sigma w_2$, alors $\sigma' = \varphi(\sigma)$ est un sous-mot de T' (morphisme) et $\widetilde{\varphi(\sigma)} = \varphi(\tilde{\sigma}) = \varphi(\sigma)$ est de longueur paire.

Réciproquement, si σ' est un sous-mot palindrome pair de T', on note $\sigma' = T'[i, j]$ Si i est

Algorithm 5: PrefPalMin

Input : Un texte T de longueur n.
Output: Le plus court prefpal de T.

$notfind \leftarrow true$
$Ray \leftarrow [0](n+1)$
$i \leftarrow 0$
for j *in* $[1, n]$ **do**
  if $j > i + Ray[i]$ **then**
    $k \leftarrow 1$
    while $k < min(n-j, j+1)$ ***and*** $T[j+k] = T[j-k+1]$ **do**
      $k \leftarrow k+1$
    $Ray[j] \leftarrow k-1$
    if $j = Ray[j]$ **then**
      return $2 * j$
    $i \leftarrow j$
  else
    $k \leftarrow j - i$
    if $Ray[i-k] \neq Ray[i] - k$ **then**
      $Ray[j] \leftarrow min(Ray[i-k], Ray[i]-k)$
    else
      $k \leftarrow Ray[i] - k + 1$
      while $k < min(n-i, i+1)$ ***and*** $T[j+k] = T[j+1-k]$ **do**
        $k++$
      $Ray[j] \leftarrow k-1$
      if $j = Ray[j]$ **then**
        return $2 * j$
      $i \leftarrow j$

pair, alors j est impair, donc $T[i-1] = T[i]$ et $T[j] = T[j+1]$, on peut donc supposer sans perte de généralité que $\sigma' = T'[i-1, j+1]$ car on a déjà (palindrome) $T[i] = T[j]$.

Ainsi, i est impair, et j est pair, ce qui permet d'écrire $\sigma' = \varphi(\sigma)$ avec $\sigma = T[(i+1)/2, j/2]$ et σ palindrome.