## 2.4 Recherche de motifs – Knuth-Morris-Pratt

```
entrée:  lalopalalali lala
sortie:        ^
```

**Définition** Étant donné une chaîne $s$ de longueur $n$ et un motif $t$ de longueur $m$, on souhaite déterminer le premier indice $i$ tel que $t$ est un facteur de $s$ à la position $i$. La réponse devrait être $-1$ si $t$ n'est pas un facteur de $s$.

**Complexité** $O(n+m)$, voir [19].

**Algorithme naïf** Cela consiste à tester tous les alignements possibles de $t$ en dessous de $s$ et pour chaque alignement $i$ vérifier caractère par caractère si $t$ correspond à $s[i..i+m-1]$. Il a une complexité de $O(nm)$ dans le pire cas. L'exemple suivant montre les comparaisons effectuées par l'algorithme sur un exemple. Chaque ligne correspond à un choix de $i$, et indique les caractères du motif qui correspondent, ou un × en cas de différence.

| | l | a | l | o | p | a | l | a | l | a | l | i |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | l | a | l | × | | | | | | | | |
| 1 | | × | | | | | | | | | | |
| 2 | | | l | × | | | | | | | | |
| 3 | | | | × | | | | | | | | |
| 4 | | | | | × | | | | | | | |
| 5 | | | | | | × | | | | | | |
| 6 | | | | | | | l | a | l | a | | |

On observe qu'après le traitement de $i$, on connaît déjà une bonne partie de la chaîne $s$. On pourrait utiliser cette information pour ne plus comparer $t[0]$ avec $s[1]$, comme dans l'exemple.

**Algorithme** Appelons le chevauchement de deux chaînes $x, y$ le plus long mot qui est à la fois suffixe strict de $y$ et préfixe strict de $x$. Au moment de détecter une différence entre $s[i]$ et $t[j]$ on pourrait alors décaler $t$ de sorte que $s[0\ldots i-1]$ chevauche $t$. Comme le suffixe de $s[0\ldots i-1]$ est $t[0\ldots j-1]$ — les $j$ dernières comparaisons ont montré égalité entre $s[i-j\ldots i-1]$ et $t[0\ldots j-1]$ — le décalage qu'on va appliquer à $t$ ne dépend que de $t$.

On peut donc déterminer dans un pré-calcul ce décalage. On note par $r[j]$ la valeur $j$ moins le chevauchement entre $t[0\ldots j-1]$ avec lui-même. L'implémentation est montrée ci-dessous. Pour analyser la complexité, on distingue la partie qui calcule $r$ avec la recherche du motif. La première partie coûte $\Theta(m)$ et la deuxième $\Theta(n)$ par l'argument suivant. Chaque égalité $s[i]=t[j]$ fait augmenter $j$ de 1 et chaque inégalité décroît strictement $j$, car $r[j]<j$. Or il y a au plus $n$ égalités, et comme $j$ ne devient jamais négatif, le nombre d'inégalités est aussi majoré par $n$.

```
def knuth_morris_pratt(s, t):
    assert t != ''
    len_s = len(s)
    len_t = len(t)
    r = [0] * len_t
    j = r[0] = -1
    for i in range(1, len_t):
        while j >= 0 and t[i - 1] != t[j]:
            j = r[j]
        j += 1
        r[i] = j
    j = 0
    for i in range(len_s):
        while j >= 0 and s[i] != t[j]:
            j = r[j]
        j += 1
        if j == len_t:
            return i - len_t + 1
    return -1
```

**Variantes** Avec une petite modification et sans augmenter la complexité on peut produire un tableau de booléens $p$ de taille $n$, qui indique pour chaque position $i$ si $t$ est un facteur de $s$ à la position $i$. De façon plus générale, on peut calculer un tableau entier $p$ qui détermine pour chaque position $i$ le plus grand $j$ tel que le préfixe de $t$ de longueur $j$ est un facteur de $s$ terminant en $i$. C'est l'objet de la section suivante.

## 2.5 Bords maximaux – Knuth-Morris-Pratt

Le problème de la recherche du bord maximal d'une chaîne permet de résoudre celui de la recherche du motifs et bien d'autres, exposés en fin de section. L'algorithme décrit ici repose sur les mêmes principes que l'algorithme de Knuth-Morris-Pratt, mais son implémentation est plus courte car repose sur de nombreuses astuces extrêmement élégantes.

**Définition** On appelle bord d'un mot $w$ un mot à la fois préfixe strict et suffixe strict de $w$, et bord maximal de $w$ noté $\beta(w)$ son plus long bord. Par exemple, *abaababa* a pour bords *aba*, *a* et le mot vide $\varepsilon$. Pour une chaîne $w = w_0 \cdots w_{n-1}$ donnée, on souhaite calculer les bords maximaux de chaque préfixe de $w$, ou de façon équivalente la longueur de ces bords, puisque tous les bords des préfixes de $w$ sont des préfixes de $w$. On cherche donc à construire efficacement la suite des longueurs $\ell_i = |\beta(w_0 \cdots w_{i-1})|$.

**Observation clé** On observe une structure récursive dans la notion de bord : si $u$ est bord de $v$ qui est bord de $w$, alors $u$ est également bord de $w$. L'application itérée de $\beta$ à un mot $w$ permet donc d'obtenir tous les bords de $w$. Par exemple, pour $w = abaababa$, on a $\beta(w) = aba$ puis $\beta(\beta(w)) = a$ et $\beta^3(w) = \varepsilon$.

**Algorithme** Supposons que l'on connaisse les bords maximaux pour les $i$ premiers préfixes de $w$, c'est-à-dire jusqu'au préfixe $u = w_0 \cdots w_{i-1}$. Considérons le préfixe

suivant $ux$ (pour plus de clarté, $x$ dénote la lettre $w_i$) : son bord maximal est forcément de la forme $vx$ où $v$ est bord de $u$, cf. figure 2.3. Notons $v_j = \beta^j(u)$ le $j$-ième bord de $u$ par taille décroissante et $k_j$ sa longueur. On va donc chercher $v$ parmi la suite $(v_j)$ des bords de $u$, en commençant par le bord maximal $v_1 = \beta(u)$. Pour savoir si un candidat $v_j x$, déjà suffixe de $ux$, est bord de $ux$, il reste à vérifier que $v_j x$ est préfixe de $ux$. Or, $v_j$ est déjà préfixe de $u$ (c'est un bord), donc il suffit de tester si la lettre qui suit immédiatement après $v_j$ (donc, à la position $k_j = |v_j|$) est $x$, ce qui revient à effectuer le test $w_{k_j} =^? x$. Si c'est le cas, on a trouvé $\beta(ux) = v_j x$ et le calcul du bord maximal pour le préfixe $ux$ est terminé. Sinon, on passe au bord suivant $v_{j+1} = \beta(v_j)$ de taille $k_{j+1} = |\beta(v_j)| = |\beta(w_0 \cdots w_{k_j-1})| = \ell_{k_j}$, puisque $v_j$ n'est autre que le préfixe de $w$ de longueur $k_j$. Si aucune comparaison n'a été fructueuse, on sait que $\beta(ux) = \varepsilon$. Comme à chaque itération, l'unique test qu'on effectue ainsi que le calcul de $k_{j+1}$ ne dépendent que de la longueur du bord en cours $k_j$, notre implémentation n'utilise qu'une seule variable $k$ pour déterminer le tableau $\ell$.

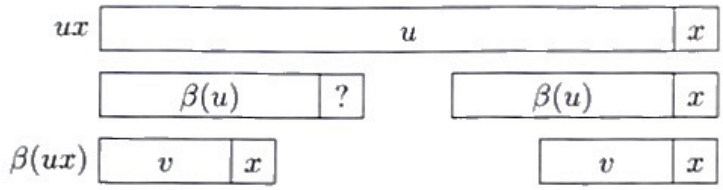


FIGURE 2.3 – Une étape de la variante de l'algorithme de Knuth-Morris-Pratt. Une fois les bords de $u$ connus, on sait que le bord maximal de $ux$ sera de la forme $vx$ où $v$ est bord de $u$. Si la lettre représentée par un point d'interrogation est un $x$, alors $v = \beta(u)$, sinon il faut chercher $v$ parmi les bords plus petits de $\beta(u)$.

```
def maximum_border_length(w):
    n = len(w)
    L = [0] * (n + 1)
    for i in range(1, n):
        k = L[i]
        while w[k] != w[i] and k > 0:
            k = L[k]
        if w[k] == w[i]:
            L[i + 1] = k + 1
        else:
            L[i + 1] = 0
    return L
```

**Complexité** Curieusement, cet algorithme a une complexité amortie linéaire : en effet, le nombre d'itérations de la boucle *while* ne dépasse jamais la longueur du bord courant $k$, qui augmente au plus d'un à chaque itération de la boucle *for*.

**Variantes** On peut se servir de cette liste de bords maximaux pour résoudre de nombreux problèmes sur les mots : déterminer les facteurs carrés, les préfixes palindromes, détecter si deux mots $x$ et $y$ sont conjugués (c'est-à-dire, s'ils s'écrivent

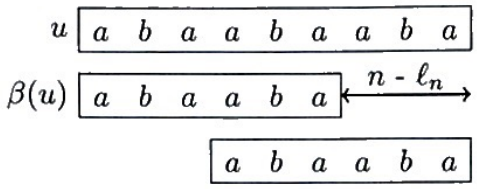


FIGURE 2.4 – Connaître le bord maximal d'un mot périodique permet de déterminer sa plus petite période. Si $n$ est la longueur de $u$ et si $n - \ell_n$ divise $n$, le mot est périodique et la plus grande valeur de $k$ pour laquelle on a $u = z^k$ pour un mot $z$ est $n/(n - \ell_n)$.

$x = uv$ et $y = vu$ pour des mots $u$ et $v$) ou encore la plus petite période d'un mot $x$, c'est-à-dire le plus grand $k$ tel que $x$ s'écrive $z^k$ pour un mot $z$.

**Observation clé** Si $u$ est périodique, le bord maximal de $u$ est exactement $z^{k-1}$ où $k$ est le plus grand entier pour lequel il existe un mot $z$ tel que $u = z^k$, cf. figure 2.4.

```
def powerstring_by_border(u):
    L = maximum_border_length(u)
    n = len(u)
    if n % (n - L[-1]) == 0:
        return n // (n - L[-1])
    return 1
```

**Application recherche du motif $t$ dans $s$** On choisit une lettre # qui n'est ni dans $t$ ni dans $s$ et on s'intéresse à la longueur des bords maximaux des préfixes de $t\#s$ : une première remarque est que cette longueur ne peut jamais dépasser la longueur de $t$, à cause du caractère #. Mais à chaque fois que cette longueur atteint $|t|$, cela signifie qu'on a trouvé une occurrence de $t$ dans $s$. On va donc déterminer par programmation dynamique la liste des $\ell_i = |\beta(u)|$ pour tous les préfixes $u$ de $t\#s$, cf. figure 2.5.

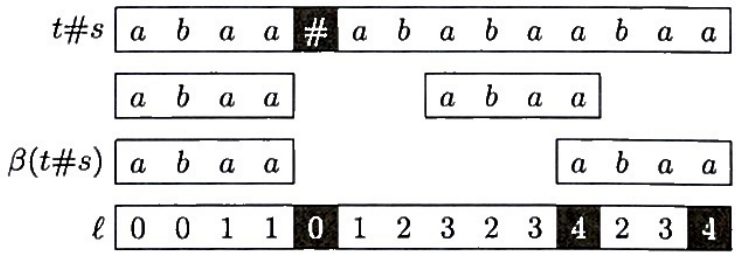


FIGURE 2.5 – Une itération de l'algorithme de recherche des bords maximaux. À chaque occurrence de $t$ dans $s$ correspond un bord de longueur $|t|$ dans le tableau des longueurs des bords maximaux $\ell$.