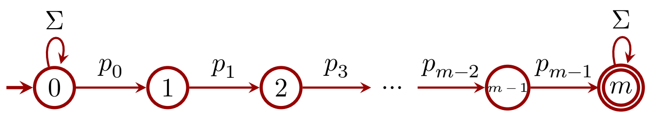
TP5 - Algorithme KMP (éléments de réponses)

Question 1. Si m est la taille du mot P, l'automate $\mathcal{A} = (Q, \Sigma, E, \{0\}, \{m\})$, où :

- $Q = \{0, \ldots, m\}$;
- $E = \{(0, x, 0), x \in \Sigma\} \cup \{(m, x, m), x \in \Sigma\} \cup \{(k, P[k], k+1), k \in \{0, \ldots, m-1\}\}$;

reconnaît le langage $\Sigma^* P \Sigma^*$.



Question 2. On propose deux versions de la fonction `sub_string`, l'une itérative, l'autre récursive, qui extrait une sous-chaîne de caractères d'une chaîne.

```
let sub_string string idx len =
  let n = String.length string
  and s_string = ref ""
  and i = ref idx in
  let idx_max = min n (idx+len) in
  while !i < idx_max do
    s_string := !s_string ^ (Char.escaped string.[!i]);
    incr i
  done;
  !s_string

let sub_string1 string idx len =
  let n = String.length string in
  let idx_max = min n (idx+len) in
  let rec aux acc_string = function
    | i when i = idx_max -> acc_string
    | i -> aux (acc_string ^ (Char.escaped string.[i])) (i+1)
  in
  aux "" idx
```

Question 3. La fonction `find_pattern` effectue la recherche d'un motif `p` dans un texte `t`.

```
let algo_find_pattern p t =
  let n = String.length t
  and m = String.length p
  and i = ref 0
  and found = ref false in
  while !i <= n - m && not !found do
    if sub_string t !i m = p then found := true;
    incr i;
  done;
  !found
```

Question 4. La complexité de la fonction `sub_string` est linéaire en la taille de la sous-chaîne à extraire. Celle de la fonction `find_pattern` est en $O(m \times n)$.

Question 5.

□ **5.1.** Par définition de la fonction π, pour tout entier q :

- $\pi[q] \geqslant 0$ puisque la fonction est à valeur dans $[\![0, m-1]\!]$;
- $\pi[q] < q - 1$ puisque $\pi[q]$ est un plus grand entier *strictement* à inférieur à q.

La double inégalité donne immédiatement $\pi[1] = 0$.

□ **5.2.** Pour construire le tableau demandé, il convient de bien comprendre le sens de la définition de $\pi[q]$. Pour une valeur de q fixée, cette définition fait appel à la connaissance des préfixes du motif p, notés $P_0 = \varepsilon$, $P_1 = p_0$, $P_2 = p_0p_1$, $P_3 = p_0p_1p_2$, ..., $P_q = p_0 \ldots p_{q-1}$. $\pi[q]$ est le plus grand entier k strictement plus petit que q tel que P_k soit un suffixe de P_q. On doit donc déterminer le plus grand suffixe de P_q qui soit également préfixe de P_q.

Par exemple, si le motif est $p = abababba$, si $q = 4$, on a :

$$P_4 = abab \qquad P_3 = aba \qquad P_2 = ab \qquad P_1 = a \qquad P_0 = \varepsilon$$

P_3 n'est pas suffixe de P_4. Mais P_2 l'est. Par conséquent, $\pi[4] = 2$.

En procédant ainsi avec toutes les valeurs de q de 1 à 8 puisque $|abababba| = 8$, on obtient le tableau suivant.

q	1	2	3	4	5	6	7	8
$\pi[q]$	0	0	1	2	3	4	0	1
$\pi^*[q]$	$\{1,0\}$	$\{2,0\}$	$\{3,1,0\}$	$\{4,2,0\}$	$\{5,3,1,0\}$	$\{6,4,2,0\}$	$\{7,0\}$	$\{8,1,0\}$

Question 6.
□ **6.1.** Par définition de $\sqsupseteq$, étant donnés deux mots u et v, on a $u \sqsupseteq v$ si et seulement s'il existe un mot w tel que $v = wu$. Si $P_{k+1} \sqsupseteq P_{q+1}$, alors $P_{q+1} = wP_{k+1}$. Deux mots égaux ont même dernière lettre, donc $p_{k-1} = p_{q-1}$. En simplifiant alors, il vient $P_q = wP_k$. Donc $P_k \sqsupseteq P_q$.
La réciproque se montre de façon analogue.
□ **6.2.** Si $\pi[q+1] = 0$, c'est évident.
Sinon, par définition, l'entier $\pi[q+1]$ vérifie $P_{\pi[q+1]} \sqsupseteq P_{q+1}$. D'après le résultat de la question précédente, en prenant $k = \pi[q+1] - 1$, il vient $P_{\pi[q+1]-1} \sqsupseteq P_q$. Or, par définition, $\pi[q]$ est le plus grand entier k tel que $P_k \sqsupseteq P_q$. Donc $\pi[q+1] - 1 \leqslant \pi[q]$. Ce qui établit le résultat.
□ **6.3.** C'est la transitivité de la relation $\sqsupseteq$.

Question 7.
□ **7.1.** L'ensemble des tels j' est non vide puisqu'il contient q et est inclus dans l'ensemble $\mathbb{N}$ des entiers naturels. Cet ensemble a donc un plus petit élément.
□ **7.2.** On a $P_q = wP_j = w'P_{j'}$. Ainsi, $P_j \sqsupseteq P_{j'}$ ou $P_{j'} \sqsupseteq P_j$. Puisque $j' > j$, on déduit le résultat demandé.
□ **7.3.** Soit $k < j'$ tel que $P_k \sqsupseteq P_{j'}$. Supposons $k > j$. Par maximalité de j, on a donc $k \in \pi^*[q]$. Mais par minimalité de j', on ne peut pas avoir $k > j$. Donc, $k = j$. Ainsi, si $k < j'$ et $P_k \sqsupseteq P_{j'}$, alors $k \leqslant j$. Donc, $j = \pi[j']$. Comme $j' = \pi^\alpha[q]$ (pour un α judicieux), on en déduit que $j = \pi^{\alpha+1}[q] \in \pi^*[q]$.

Question 8. Il s'agit d'une synthèse des résultats des questions précédentes.

Question 9. Fonction `calcul_prefixe`.

```
let prefix_array p =
  let m = String.length p in
  let pi = Array.make (m+1) 0 in
  for q = 2 to m do
    let k = ref pi.(q-1) in
    while !k > 0 && p.[!k] <> p.[q-1] do k := pi.(!k) done;
    if p.[!k] = p.[q-1] then k := !k + 1;
    pi.(q) <- !k;
  done;
  pi
```

Question 10. Fonction `kmp`.

```
let kmp text pattern =
  let n = String.length text
  and m = String.length pattern
  and pi = prefix_array pattern
  and pos = ref []
  and q = ref 0 in
  for i = 0 to n - 1 do
    while !q > 0 && pattern.[!q] <> text.[i] do q := pi.(!q) done;
    if pattern.[!q] = text.[i] then q := !q + 1;
    if !q = m then (
      pos := (i-m+1) :: !pos;
      q := pi.(!q);
    );
  done;
  List.rev !pos
```

Un exemple de résultat renvoyé par cette fonction est le suivant.

```
kmp "aababababaa" "ab";;
- : int list = [1; 3; 5]
```

Question 11. Le comportement de l'automate est donné en grande partie par le corps de la boucle `for` dans la fonction `kmp` ci-dessus. Si l'automate se trouve dans un état $q < m$, il peut soit passer à l'état $q+1$, soit à l'un des états $\pi[q]$, $\pi^2[q]$, …. On regarde pour cela le caractère courant de la chaîne lue. Si l'automate se trouve dans l'état m, il y reste. Le motif a déjà été découvert le motif et l'automate reste donc dans un état final. Le code suivant simule un automate complet avec un unique état acceptant, les états étant des entiers.

```
type automate = {
  start: int;
  delta: int -> char -> int;
  success: int
}

let accepte a t =
  let q = ref a.start
  and n = String.length t in
  for i = 0 to n - 1 do
    q := a.delta !q t.[i];
  done;
  !q = a.success
```

```
let auto p =
  let m = String.length p
  and pi = prefix_array p in
  let delt q x =
    if q = m then m
    else (
      let q1 = ref q in
        while !q1 > 0 && p.[!q1] <> x
          do
          q1 := pi.(!q1);
        done;
        if p.[!q1] = x
        then !q1 + 1 else 0
    )
  in {start=0; delta=delt; success=m}
```

La fonction suivante affiche la relation de transition `delta` associée à un motif.

```
let disp_auto_pattern p =
  let aut = auto p in
  let n_states = aut.success in
  let n = String.length p in
  for i = 0 to n-1 do
    for j = 0 to n_states do
      let c = p.[i] in
      Printf.printf "delta(%d, '%c') -> %d\n" j c (aut.delta j c)
    done;
  done
```

L'exécution `disp_auto_pattern "bit"` affiche les résultats suivants.

```
delta(1, 'b') -> 1
delta(2, 'b') -> 1
delta(3, 'b') -> 3
delta(0, 'i') -> 0
delta(1, 'i') -> 2
delta(2, 'i') -> 0
delta(3, 'i') -> 3
delta(0, 't') -> 0
delta(1, 't') -> 0
delta(2, 't') -> 3
delta(3, 't') -> 3
```