Knuth-Morris-Pratt.

Leçons : 907, 927.

Ref :

Théorème

L'algorithme de recherche (K)MP est correct et termine en $O(m)+n)$, où l'on cherche un mot de longueur m dans un mot de longueur n.

Résumé

- On cherche $u \in \Sigma^*$ dans $t \in \Sigma^*$ sur un alphabet Σ.

I - Déroulement de l'algorithme sur un exemple significatif.

II - Définition de l'automate des occurences, reconnaissant $\Sigma^* u$.

III - Étude de la fonction Bord pour un calcul efficace de l'automate.

IV - Algorithme de Morris-Pratt + l'amélioration de Knuth.

I | • Prenons $t = ABADABABAC$

$u = ABAC$.

• L'algorithme de Morris-Pratt donne :

$t = A\ B\ A\ D\ A\ B\ A\ B\ A\ C \ \ldots$

1) $A^{\checkmark}\ B^{\checkmark}\ A^{\checkmark}\ C^{\times}$

2) $A^{\checkmark}\ B^{\times}$

3) $A^{\times}$

4) $A^{\checkmark}\ B^{\checkmark}\ A^{\checkmark}\ C^{\checkmark}$

• L'idée, c'est que quand il y a un problème, on peut décaler la fenêtre de plus d'un caractère.

• On se rend compte que le motif déjà matché avant décalage, c'est à la fois un préfixe de u et un suffixe de $t[k+1]$.

II] · Le déroulement de KMP peut être vu comme le parcours de l'automate suivant : $A = (Q, \Sigma, u, \delta)$ où :

* Q est l'ensemble $Pref(u)$ des préfixes de u. "ce qu'on a déjà matché".
* ε (le mot vide) est l'état initial "on n'a encore rien matché".
* la fonction de transition δ est :

$$\delta(v,a) = \begin{cases} va & \text{si } va \in Pref(u) \\ Bord(va) & \text{sinon.} \end{cases} \quad \text{où } Bord(w) \text{ est le plus long bord strict de } w.$$

* u l'état final : "on a reconnu le mot entier".

· Ce qu'il se passe, c'est que l'état indique la partie du mot qu'on a déjà matché, puis :

→ soit on lit la lettre a qui suit v dans u, et on continue vers va.

→ soit on lit une autre lettre a, et là, on veut éviter de recommencer depuis ε, donc on recommence après un préfixe de va, qu'on aurait déjà reconnu à la fin de ce que l'on vient de lire, c'est-à-dire à la fin de va. (sans que ça soit va !)
C'est donc $Bord(va)$ = le plus long bord strict de va.

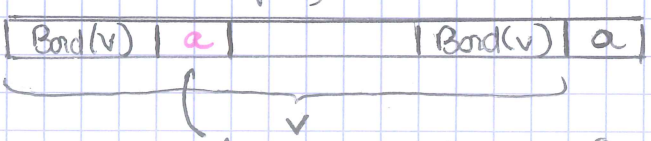
III] · Mais comment calculer $Bord(va)$ efficacement ?

· Grâce au lemme :

$$Bord(va) = \begin{cases} Bord(v)\,a & \text{si } Bord(v)\,a \in Pref(u) \\ Bord(Bord(v)\,a) & \text{sinon.} \end{cases}$$

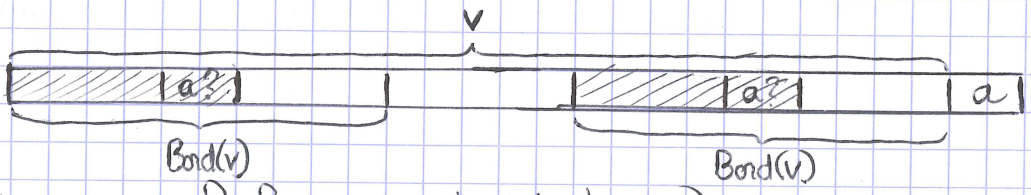
· Intuitivement, on a : → si $va \in Pref(u)$ on n'a qu'à continuer.

→ si $Bord(v)\,a \in Pref(u)$, on a :



il y a un a ici car $Bord(v)\,a \in Pref(u)$ et $v \in Pref(u)$.

→ sinon, $Bord(v)a \notin Pref(u)$, et on cherche un préfixe de u qui finisse par a. C'est-à-dire un mot au début de v, suivi d'un a, qui apparait aussi à la fin de va ... et donc de $Bord(v)a$. (*)



(Les deux zones hachurées sont identiques)

(*) Si ce n'était pas dans $Bord(v)a$, on serait dans le premier cas car on aurait un mot, bord de va, finissant par a, et donc un bord de v plus long que $Bord(v)$, c'est-à-dire ... $Bord(v)$!

• On peut maintenant prouver le lemme :

- soit $w = Bord(va)$, il s'écrit $w'a = w$ car c'est un suffixe de va.
  → $w \in Pref(va)$ donc $w' \in Pref(v)$
  → $w \in Suff(va)$ donc $w' \in Suff(v)$
  ⇒ de plus, w est maximal donc w' est le plus grand bord de v tel que $w'a$ est un préfixe de u.
- cela laisse deux possibilités :
  1 - $Bord(v)a \in Pref(u)$, auquel cas $w' = Bord(v)$ par maximalité.
  2 - w' est un préfixe strict de $Bord(v)$, et comme $w'a = Bord(va)$, w' est aussi à la fin de v et donc de $Bord(v)$.
  De là, w' est un préfixe de $Bord(Bord(v))$, d'où $w = Bord(Bord(v)a)$.

• En itérant ce lemme, on obtient

$$\text{Bord}(va) = \begin{cases} \text{Bord}(v)a & \text{si } \text{Bord}(v)a \in \text{Pref}(u) \\ \text{Bord}(\text{Bord}(v))a & \text{sinon si } \text{Bord}(\text{Bord}(v))a \in \text{Pref}(u) \\ \cdots & \\ a & \text{sinon si } a \in \text{Pref}(u) \\ \varepsilon & \text{sinon tout court.} \end{cases}$$

III] • On peut traduire tout ça en algorithme directement, mais pour cela, on va seulement garder en mémoire la taille du bord des préfixes de la forme $u[1 \dots k]$.
On définit aussi $\pi(k) = |\text{Bord}(u[1 \dots k])|$.

```
CalculPi(u) :
    π(1) ← 0        k ← 0.
    Pour i = 2 ... |u| :
        Tant que k > 0 et u[k+1] ≠ u[i] :
            k ← π(k).
        Si u[k+1] = u[i] :
            k ← k+1
        π(i) ← k.
```

• Terminaison : on a $\pi(k) < k$ donc le while termine et CalculPi aussi.
• Correction : c'est le lemme 1, on calcule bien les bords !
• Complexité : on remarque que k est toujours positif, le while ne peut pas désincrémenter k de plus que sa valeur.
De plus, k augmente de 1 au plus une fois par passage dans le for, dans lequel on passe n fois. On passe donc moins de n fois dans le while, d'où une complexité en $O(n)$, avec $n = |u|$.

- On peut (enfin) écrire l'algorithme de recherche : il suffit de chercher un bord de $u\#t$, pour $\# \notin \Sigma$, de taille $|u|$.

On a en effet :

> $\forall k \geq 0$, $\pi(k) \leq |u|$ et $\pi(k) = u$ ssi u apparaît dans t en position $k - 2|u| - 1$.

- Car, d'une part, $\# \notin \Sigma$, donc $\pi(k) \leq |u|$ sinon on aurait une occurence de $\#$ dans t.

- D'autre part, si $\pi(k) = |u|$, alors u est un préfixe et... un suffixe de $u\#t[1 \ldots k]$. C'est donc un suffixe ~~de t~~ de $t[1 \ldots k - |u| - 1]$ en position $k - 2|u| - 1$.

→ on trouve donc une occurence de u dans t en :

$$O(|u| + |t| + 1) = O(|u| + |t|).$$