Recherche d'occurrences d'une chaîne de caractères dans une autre

- Une chaîne longue le *texte* dans un tableau `T[1..n]`

- une chaîne courte la *cible* dans un tableau `C[1..m]`

- vérifier qu'une occurrence de `C` commence à `T[i]` prend temps m; donc algorithme simple en temps $O(mn)$

- Comment faire mieux ?

(1) L'algorithme de Knuth-Morris-Pratt

- Lecture de `T` de gauche à droite
- A chaque moment stocker la (taille de la) partie au début de `C` qui correspond à la partie de `T` qu'on vient de lire
- Cette taille peut être mise à jour sans retour en arrière
- Il faut savoir, pour chaque préfixe de `C`, quelle est sa plus longue (propre) suffixe qui est aussi une préfixe de `C`

Un automate qui effectue la recherche du suffixe-préfixe

- Un état pour chaque lettre de `C`, mettons $C = C_1, ..., C_m$

- Considérer un état C_i; on a déjà lu $C_1, ..., C_i$

- avancer si la prochaine lettre x est la bonne C_{i+1} (sauf si on a trouvé un `C` complet)

- sinon reculer

- mais par combien?
  à C_j où $C_1, ..., C_j$ est le plus long préfixe de C qui est aussi suffixe de $C_1, ..., C_i x$.

Le précalcul

(du plus long préfixe de C qui est aussi suffixe de $C_1, ..., C_i$)
Peut se faire facilement en temps $O(m^2)$ mais ce n'est pas optimal

- Une récurrence pour S_i, la longueur de ce préfixe-suffixe
  si $C_{i+1} = C_{S_i+1}$, $S_{i+1} = S_i + 1$,
  sinon, si $C_{i+1} = C_{S_{S_i}+1}$, $S_{i+1} = S_{S_i} + 1$,
  etc.

- Le calcul par programmation dynamique

- Le temps du précalcul : $O(m)$

- Le temps total $O(m + n)$

(2) L'algorithme de Boyer et Moore: l'idée

- Comment faire mieux qu'un algorithme qui considère chaque lettre une seule fois?

- Impossible dans certains cas au moins

- Mais souvent, si `C` est de taille moyenne, on peut exclure des sections de `T` sans les regarder !

- exemple trivial: si `C` est `aaaaaaaaaa`, et `T` ne contient pas de `a`, on peut trouver ce fait en ne regardant que chaque $\mathbf{10}$-ème lettre de `T`.

L'algorithme de Boyer et Moore: un peu de détail

Pour trouver les positions où une occurrence de `C` pourrait terminer

- regarder d'abord la lettre `T[m]`
- si cette lettre n'a pas d'occurrences en `C`, avancer par `m`
- si sa dernière occurrence en `C` est à position `j (j<m)`, avancer par `m-j`
- si sa dernière occurrence est à `C[m]`, chercher une occurrence complète (de droite à gauche)

Récupération après un échec

et aussi redémarrage après une réussite quand on veut trouver toutes les occurrences
On a déjà trouvé un suffixe de C (disons xyz) qui termine à une position p en T
mais les prochaines caractères à gauche dans C et T sont différents; il y a trois cas:

- il y a d'autres occurrences de xyz en C; avancer la distance qui aligne l'avant dernière occurrence de xyz avec l'occurrence déjà trouvée en T; c'est-à-dire incrémenter p par la différence des positions des deux occurrences en C

- aucune autre occurrence de xyz en `C` mais `C` commence avec un suffixe de xyz, par exemple yz; avancer la distance qui aligne ce suffixe avec son occurrence déjà trouvée en `T`; c'est-à-dire incrémenter p par m moins la longueur du suffixe-préfixe
  (yz dans l'exemple)

- sinon: incrémenter p par m.

Temps de calcul

- Précalcul des tableaux des distances pour les incréments de p: $O(m)$

- Pire cas de la recherche dans `T`: $O(n)$
  (avec une petite modification de l'algorithme pour traiter le cas de plusieurs occurrences chevauchées de `C`)

- Temps "moyen" $O(n/m)$ si on accepte l'idée que les deux chaînes sont aléatoires.
  attention : ici le m est le plus petit de longueur de la cible et taille de l'alphabet.