UNIVERSITÉ DE MONTPELLIER
PLACE EUGÈNE BATAILLON 34095 MONTPELLIER CEDEX 5
Master Bioinformatique – 2023/2024

Développement opérationnel avancé
Travaux Pratiques n° 3

Réalisation d'une API pour la création et l'utilisation d'une table des suffixes

Alban MANCHERON

L'indexation de texte a pour objectif d'organiser l'information contenue dans un texte pour simplifier son analyse dans un second temps. Un index doit permettre de répondre efficacement à des questions du type :

- Est-ce qu'un motif donné apparaît dans le texte ?
- Le cas échéant, combien de fois et à quelles positions ?
- Quelle est le plus grand facteur répété du texte ?
- Quel est le plus petit facteur non répété dans le texte ?
- ⋮

Parmi les structures d'index utilisées en bioinformatique, la table des suffixes connaît un certain succès. En effet, cette structure est relativement économique en mémoire, elle peut se construire en temps linéaire et, doublée d'une table des plus long préfixes communs, permet de répondre très efficacement à bon nombre de questions comme celles énumérées précédemment.

Dans ce TP, il vous est demandé d'implémenter un algorithme simple (*i.e.*, non optimal) de construction de la table des suffixes d'un mot donné.

1 Table des suffixes

La table des suffixes, notée SA (*suffix array* en anglais), d'un mot de longueur n est un vecteur de $n+1$ entiers, dénotant les positions de début des $n+1$ suffixes du mot indexé, triés par ordre lexicographiques. Ainsi, pour le mot m de longueur n, étant donné deux entiers i et j tels que $1 \leq i \leq j \leq n$, alors $m\left[SA_m[i]..n\right]$ est plus petit (du point de vue lexicographique) que $m\left[SA_m[j]..n\right]$ (noté $m\left[SA_m[i]..n\right] \preceq m\left[SA_m[j]..n\right]$ dans la suite).

⚠ Ceci ne signifie pas que $SA_m[i] \leq SA_m[j]$ (bien évidemment 🙂).

Afin de visualiser le suffixe vide, il est d'usage de marquer la fin de la chaîne à indexer par un symbole qui n'apparaît jamais dans la chaîne. Généralement, en algorithmique du texte, la fin de chaîne est donc marquée par le symbole \$. Il est possible de réduire la table des suffixes à un vecteur de taille n puisque par définition du mot vide, la première case de la table a nécessairement la valeur n (qui correspond à la position du suffixe vide du mot indexé.

⚠ Dans le cas où l'on choisit de réduire la taille de la table des suffixes, il faut bien évidemment garder à l'esprit que le suffixe vide est tout de même présent dans le mot indexé, même s'il n'apparaît pas explicitement dans la table.

La recherche d'un mot dans la table des suffixes se fait par dichotomie. En effet, pour savoir si le facteur f apparaît dans le mot m de longueur n, il suffit de le comparer au mot $w = m\left[SA_m\left[\frac{n}{2}\right]..n\right]$. Si f est préfixe de w alors f est un facteur de m (car un préfixe d'un suffixe est un facteur...). Si f est plus petit que w dans l'ordre lexicographique, alors on recommence la recherche dans la première moitié de la table des suffixes (de l'indice 1 inclus à l'indice $\frac{n}{2}$ exclus). Enfin, si f est plus grand que w dans l'ordre lexicographique, alors on recommence la recherche dans la seconde moitié de la table des suffixes (de l'indice $\frac{n}{2}$ exclus à l'indice n inclus).

Pour faciliter davantage les recherches, il est possible de construire à partir de la table des suffixes, sa table des plus longs préfixes communs.

2 Table des plus longs préfixes communs

La table des plus longs préfixes communs, notée LCP (*Longest Common Prefixes* en anglais), est un vecteur d'entiers de la même taille que la table des suffixes à laquelle elle est associée. La table des plus longs préfixes communs d'un mot m de longueur n est telle que pour $2 \leq i \leq n$, la valeur $LCP_m[i]$ est la longueur du plus long préfixe commun à $m\big[SA[i-1]..n\big]$ et à $m\big[SA[i]..n\big]$.

De fait, comme les suffixes sont triés par ordre lexicographique croissant dans la table des suffixes, il est possible de calculer un certain nombre d'informations rapidement.

Par exemple, pour savoir s'il existe un facteur répété au moins k fois dans le mot m, il suffit de chercher dans la table des plus long préfixes communs s'il existe une suite consécutive d'au moins $k-1$ valeurs non nulles. La longueur d'un tel facteur, s'il existe, est égale au minimum de la suite.

3 Objectif du TP

Votre travail consiste à définir une API (*Application Programming Interface*) afin de permettre d'une part d'indexer une chaîne de caractère et d'autre part d'interroger l'index ainsi créé.

L'index d'un mot sera naturellement composé de sa table des suffixes et de sa table des plus longs préfixes communs.

3.1 Création d'un tableau ordonné des suffixes

Dans un premier temps, il vous est suggéré d'implémenter un programme qui, étant donné une chaîne de caractère, construit un vecteur [1] de tous ses suffixes, puis les trie par ordre lexicographique (en utilisant la méthode de tri `std::sort` [2] fournie par `C++`).

Concevez des tests unitaires pour ce programme.

3.2 Création d'une table des suffixes équivalente

Une fois le programme précédent fonctionnel, créez un nouveau programme qui cette fois construit un vecteur d'entiers initialisé de 1 à $n+1$ (ou de 0 à n, voire de 0 à $n-1$), définissez une fonction de comparaison qui étant donnés deux entiers i, j et une séquence s renvoie -1, 0 ou 1 selon que le $i^{ème}$ suffixe de s précède, est égal ou suit le $j^{ème}$ suffixe de s du point de vue lexicographique, et modifiez l'appel à la méthode de tri pour qu'elle utilise cette fonction de comparaison.

Adaptez les tests unitaires précédents à votre nouveau programme, sachant que le premier programme peut vous permettre de valider que les suffixes sont ordonnés de la même façon.

À l'aide des fonctions de mesure de la mémoire et du temps de calcul déjà présentées lors des TP précédents [3], comparez vos deux implémentations.

1. En `C++`, vous pouvez bien évidemment utiliser la classe paramétrique `std::vector<T>`.
2. Il s'agit d'un algorithme de tri complexe qui entremêle plusieurs algorithmes de tri, notamment le tri par tas et le tri par sélection, et qui s'exécute en effectuant $O\left(n \log n\right)$ comparaisons pour trier n éléments.
3. En `C++` (sous GNU/Linux), utilisation de `getrusage(int, struct rusage *)`.

3.3 Création d'une table des plus longs préfixes communs

Afin de créer cette table, il vous faudra définir une fonction qui, étant donnés deux entiers i, j et une séquence s, renvoie la longueur du plus grand préfixe commun au $i^{ème}$ et au $j^{ème}$ suffixe de s.

À partir de cette fonction, la création de la LCP ne devrait plus poser de problèmes. Toutefois (et pour vous en assurer), vous implémenterez également des tests unitaires.

Il n'est pas demandé d'implémenter une solution linéaire en temps de construction.

3.4 Réalisation de l'API de base

Après avoir encapsulé le travail précédent dans une (ou plusieurs) classes, il est nécessaire de définir des méthodes d'accès à l'information contenue dans l'index. Ces informations vont des plus basiques (connaître la longueur de la séquence, récupérer le $i^{ème}$ facteur de longueur k de la séquence, ...) à des questions plus subtiles (récupérer le facteur de longueur k qui a le plus d'occurrences dans la séquence, déterminer le nombre de facteurs distincts de longueur k présents dans la séquence, ...), voire assez compliquées (déterminer le plus petit mot qui n'est pas facteur de la séquence).

3.5 Amélioration de l'API

Pour celles ou ceux qui en auraient le temps, étendez l'API afin de pouvoir indexer non plus une séquence, mais un ensemble de séquence. Pour cela, la solution « classique » utilisée consiste à assembler toutes les séquences (en les séparant par un symbole spécial) et à indexer le résultat de cette concaténation.

Bien évidemment, dans ce cas, il faut ajouter ou modifier les méthodes afin de prendre en compte le fait que certains des symboles présents dans l'index correspondent à ces délimiteurs.

Dans tous les cas

Vous veillerez comme lors des TP précédents à vérifier que les solutions que vous proposerez se comportent en pratique comme attendu en théorie.

Bon Courage...