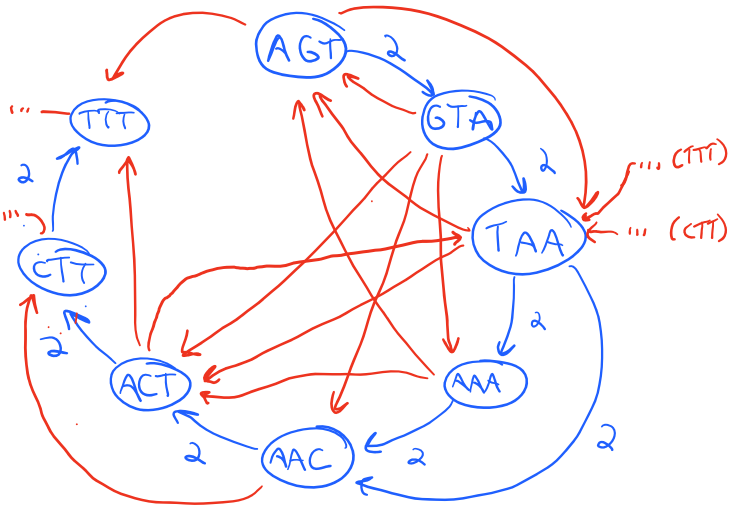
BIN702 - Série d'exercices sur le séquençage

Manuel Lafond

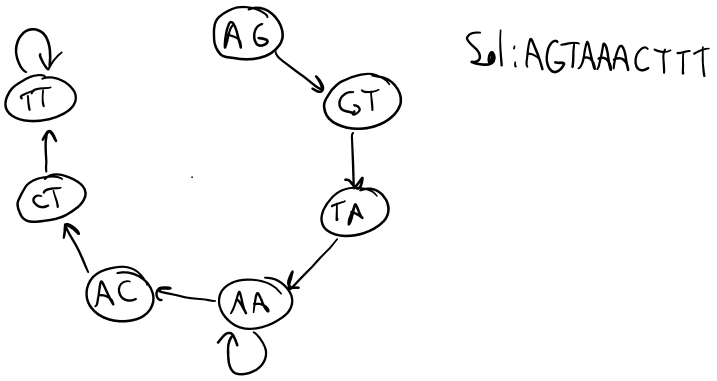
Exercice 1: Soit l'ensemble des 3-mers
$\{AGT, AAA, ACT, AAC, CTT, GTA, TTT, TAA\}$.

- Construisez le graphe de "overlap" et trouvez la plus courte superséquence à l'aide d'un chemin Hamiltonien (i.e. un chemin qui passe exactement une fois par sommet).
- Construisez le graphe de De Bruijn pour ces 3-mers. Est-ce qu'il existe un chemin Eulérien ? Si oui, comparez la superséquence résultante avec celle obtenue à la question précédente.

Solution. D'abord, le graphe d'overlap. Cet exercice a été pris dans la référence 2 du plan de cours. Je vois maintenant que c'est un peu sadique de demander de construire TOUT ce graphe. En rouge, les arêtes de poids 1 (si j'en ai oublié, laissez-moi savoir). Une solution optimale prendra seulement les arêtes de poids 2 qui parcourt le graphe dans le sens horaire, en démarrant à AGT. Ceci donne la solution $AGTAAACTTT$.



Voici le graphe de De Brujin. Beaucoup plus facile à reconstruire.



□

Exercice 2: Considérez l'algorithme glouton pour résoudre le problème du commis voyageur sur le graphe d'overlap.

- On trouve l'arête (u, v) de poids maximum et on initialise un chemin $P = (u, v)$;
- tant que la taille de P est plus petite que n :
  - soit w_1 le *premier* sommet de P. On trouve le voisin entrant z_1 de w_1 tel que z_1 n'est pas déjà dans P, et tel que (z_1, w_1) est de poids maximum.
  - soit w_p le *dernier* sommet de P. On trouve le voisin sortant z_p de w_p tel que z_p n'est pas déjà dans P, et tel que (w_p, z_p) est de poids maximum ;
  - si le poids de (z_1, w_1) est plus grand que celui de (w_p, z_p), on ajoute z_1 en début de P. Sinon, on ajoute z_p en fin de P.

Montrez que cet algorithme peut retourner une solution sous-optimale. Il suffit de donner un exemple d'instance.

Solution. Essayez avec les séquences $CCAAA, AAATT, AAGG, GGAA$. L'algo glouton ferait

$$CCAAA \to AAATT \to AAGG \to GGAA$$

menant à la séquence $CCAAATTAAGGAA$.

Toutefois, il existe une meilleure solution :

$$CCAAA \to AAGG \to GGAA \to AAATT$$

correspondant à $CCAAAGGAAATT$. □

Exercice 3: Donnez un algorithme qui reconstruit un chemin Eulérien dans un graphe orienté, s'il en existe un.

Solution.
On dénote par $in(v)$ et $out(v)$ le nombre de voisins entrants et sortants, respectivement, d'un sommet v. On suppose qu'on a vérifié les conditions discutées en classe.
Supposons qu'il existe un sommet v tel que $out(v) = in(v) + 1$. On définit v comme sommet de départ.
On maintient cur, le noeud courant, qu'on initialise à $cur = v$. On initialise notre chemin $P = (cur)$ et, tant que P n'a pas tous les sommets :

- s'il y a un cycle C qui contient cur, on parcourt C et on ajoute chacune de ses arêtes à P. On retire ces arêtes du graphe. Et on reprend à cur.
- si aucun cycle ne contient cur, on a soit terminé, ou bien cur a un seul voisin sortant z. On ajoute l'arête (cur, z) à P et on la retire du graphe. On pose $cur = z$, et on reprend.

S'il n'existe pas de v tel que $out(v) = in(v)+1$, on peut démontrer que la même procédure fonctionne, mais en choisissant n'importe quel v comme sommet de départ. Bien sûr, il faut démontrer que tout ceci fonctionne, ce que je vous laisse comme réflexion.

□

Exercice 4: Soit $S = ACATACACAG$. Construisez la transformée Burrows-Wheeler de S.

Solution. Ajoutons un caractère de terminaison \$, pour ainsi obtenir $S = ACATACACAG\$$. En triant les rotations, on obtient

```
$ACATACACAG
ACACAG$ACAT
ACAG$ACATAC
ACATACACAG$
AG$ACATACAC
ATACACAG$AC
CACAG$ACATA
CAG$ACATACA
CATACACAG$A
G$ACATACACA
TACACAG$ACA
```

En prenant la dernière colonne, la transformée est donc $GTC\$CCAAAAA$. □

Exercice 5: Supposez que vous avez déjà l'arbre de suffixes d'une chaîne S. Montrez comment, de façon efficage, on peut construire la transformée de Burrows-Wheeler (i.e. la dernière colonne des rotations) à partir de l'arbre.

Suggestion. On peut visiter les suffixes en ordre lexicographique (i.e. alphabétique).

Solution. Réponse courte :
Soit A l'arbre de suffixes pour S. On peut parcourir les suffixes en ordre lexicographique. Pour ce faire, on fait une fouille en profondeur, mais lorsque nous sommes à un noeud v, on visite ses enfants en ordre lexicographique du premier symbol de la branche enfante. Notez que la première feuille trouvée ira toujours à \$.
Dans ce parcours, la i-ème feuille rencontrée correspond au i-ème suffixe en ordre alphabétique. Supposons que le i-ème feuille correspond au suffixe qui démarre à la position j de S. On remarque que la i-ème rotation peut être obtenue en prenant ce suffixe $S[j..n]$, puis en lui ajoutant $S[1..j-1]$ à la fin. La dernière lettre de cette rotation est $S[j-1]$. On sait donc que le i-ème caractère de la BWT est $S[j-1]$.
Le parcours traversera les n feuilles de A, et à chaque feuille rencontrée correspondant à la position j, il suffit d'ajouter le caractère $S[j-1]$ de S pour obtenir la BWT.

□

Exercice 6: On a mentionné que si tous les k-mers d'une séquence ne sont pas représentés dans l'entrée, alors la technique du chemin Eulérien ne fonctionne pas. Dans ce cas, on peut essayer de trouver le minimum de chemins tels que chaque arête apparaît exactement une fois sur un des chemins.
Donnez un algorithme qui trouve ce minimum de chemins.
Version facile : faites deux suppositions : (1) supposez que s'il y a un cycle C, alors il existe une solution optimale (i.e. un ensemble de chemins) qui contient C (i.e. il y a un chemin qui suit toutes les arêtes de C de façon consécutive) ; (2) supposez qu'il existe un algorithme boîte-noire qui trouve, en temps polynomial, une solution optimale si vous avez un graphe orienté acyclique. Avec ces deux suppositions, vous devriez être en mesure de concevoir un algorithme en temps polynomial.
Version difficile : démontrez ou infirmez les deux suppositions ci-haut. Sont-elles vraies ou fausses ?

Solution. Version facile. Avec les suppositions, on peut faire la chose suivante. L'idée est que s'il y a un cycle C, on l'enlève, on trouve une solution récursivement, puis on réintègre C dans un chemin de la solution.

fonction *getMinChemins(G)*
  si *G ne contient pas de cycle* **alors**
    Trouver un nombre minimum de chemins $\mathcal{C}$ qui contient chaque arête. Ceci est faisable en temps polynomial selon la supposition (2) ;
    Retourner $\mathcal{C}$;
  sinon
    Trouver un cycle C;
    Soit v un sommet de C qui a des arêtes en dehors de C;
    //si v n'existe pas, on G est un ensemble de cycles disjoints. On pourrait traiter ce cas trivial séparément;
    Soit G' obtenu à partir de G en retirant chaque arête de C;
    $\mathcal{C} = getMinChemins(G')$;
    Soit $P \in \mathcal{C}$ un chemin contenant v;
    Obtenir P' en remplaçant v par le cycle C;
    Retourner $(\mathcal{C} \setminus \{P\}) \cup \{P'\}$;
  fin

Pour être précis, si on a un cycle $C = (v_1, v_2, \ldots, v_k, v_1)$ et un chemin $P = (p_1, p_2, \ldots, p_i = v_1, p_{i+1}, \ldots, p_l)$ contenant v_1, remplacer v_1 par C dans P créé un nouveau chemin $P' = (p_1, p_2, \ldots, p_i = v_1, v_2, \ldots, v_k = p_i, p_{i+1}, \ldots, p_l)$.

Si on avait une solution optimale pour G', le remplacement de v par C dans P ne créé pas de nouveau chemin. Donc, la seule raison pour laquelle cet algorithme serait sous-optimal pour G serait si les arêtes de C n'étaient pas consécutives dans cette solution optimale. Mais avec la supposition (1), cet algorithme donne une solution optimale.

Pour ce qui est de la version difficile, je n'entre pas dans les détails et je vous laisse en suspens. Les deux suppositions devraient être vraies, mais les preuves deviennent un peu laborieuses. De plus, la réponse ne se trouve pas dans la littérature, étonnamment. L'idée de (2) est de faire un tri topologique (`https://fr.wikipedia.org/wiki/Tri_topologique`), de démarrer un chemin à partir du sommet minimal de degré non-zéro et de suivre les arêtes de façon gloutonne jusqu'à ce qu'on bloque. Ceci forme notre premier chemin, on retire ses arêtes et on répète jusqu'à ce que toutes les arêtes soient incluses.
Pour démontrer (1) formellement, il faut prendre une solution optimale et la modifier pour qu'elle inclue notre cycle C sans augmenter le nombre de chemins. Si vous êtes trop curieux ou curieuses et que vous voulez une démonstration, venez me voir en personne ! □