4MMAOD6

TP Alignement de séquences génétiques

Programmation et localité des accès mémoire

Équipe pédagogique AOD / Jean-Louis Roch & Antoine Frénoy – Octobre MMXXIV

Ce TP peut être fait en binôme : les 2 partenaires d'un binôme peuvent être de groupes de TD différents. Seuls les rendus sur teide seront évalués ; date de rendu attendue : 25/10/2024 à 19h ; date limite extrême : 4/11/2024 à 23h.

Le but de ce TP est d'écrire un programme *cache oblivious* qui calcule l'alignement global optimal de deux séquences génétiques (ADN) et d'analyser ses défauts de cache. Ce calcul est réalisé par programmation dynamique (algorithme de Needleman-Wunsch). Un programme vous est fourni qui implémente directement l'équation de Bellman de manière *top-down* par une fonction récursive avec mémoïsation dans un tableau bidimensionnel et sans prise en compte de la localité des accès aux données. Ce programme fonctionne sur de petites instances mais la place mémoire allouée est trop grande pour traiter les instances cibles. Vous devez écrire 3 autres fonctions implémentant respectivement :

— un algorithme itératif direct (*bottom-up*, en choisissant la manière de stocker et de parcourir le tableaui et en annalysant ses défauts de cache ;

— un algorithme cache aware qui utilise en entrée les paramètres Z et L d'un niveau de cache ;

— un algorithme cache oblivious qui n'utilise aucun paramètre sur le cache.

Les effets de cache seront émulés et mesurés avec `valgrind`.

Ce TP est aussi une introduction au domaine de la biologie computationnelle.

1 Séquence génétique

L'acide désoxyribonucléique (ADN) et l'acide ribonucléique (ARN) sont des macromolécules biologiques présentes dans presque toutes les cellules ainsi que chez de nombreux virus. Les molécules d'ADN des cellules vivantes sont formées de deux brins enroulés en une double hélice ; celles de l'ARN d'un seul brin. La séquence d'un brin d'acide nucléique - ADN ou ARN - est la succession des nucléotides qui le constituent. Cette succession contient l'information génétique, appelée génome, permettant le développement, le fonctionnement et la reproduction des êtres vivants. En génétique, le séquençage concerne la détermination de la séquence des gènes voire des chromosomes, voire du génome complet, ce qui techniquement revient à effectuer le séquençage de l'ADN constituant ces gènes ou ces chromosomes.

La nature d'un nucléotide est déterminée par la base nucléique qu'il contient. La séquence d'un brin d'ADN ou d'ARN peut donc se résumer à la succession des bases nucléiques présentes ; il y en a cinq principales appelées bases canoniques, chacune représentée par une lettre : l'adenine (A), la cytosine (C), la guanine (G), la thymine (T) uniquement dans l'ADN, l'uracile (U) uniquement dans l'ARN.

Chaque nouvelle cellule produite par un être vivant (comme votre corps) reçoit une copie du génome. Ce processus de copie, ainsi que l'usure naturelle, introduit un petit nombre de changements dans les séquences de nombreux gènes. Parmi les modifications les plus courantes figurent la substitution d'une base par une autre et la suppression d'une sous-chaîne de bases ; ces changements sont généralement appelés mutations ponctuelles. À la suite de ces mutations ponctuelles, le même gène séquencé à partir d'organismes étroitement apparentés présentera de légères différences.

Exemple de problème. Une biologiste trouve la séquence suivante d'un gène dans un organisme jusque-là non étudié : `A A C A G T T A C C`. Quelle est la fonction de la protéine codée par ce gène ? Il y a de fortes chances qu'il s'agisse d'une variante d'un gène connu dans un organisme déjà étudié. Or, les biologistes et les informaticiens ont laborieusement déterminé et publié les séquences génétiques de nombreux organismes (y compris les humains). Il s'agit donc de comparer la séquence génétique ci-dessus avec des séquences dont la fonction est connue comme par exemple : `T A A G G T C A`. Si les deux séquences génétiques sont suffisamment similaires, on pourrait s'attendre à ce qu'elles aient des fonctions similaires.

2 Distance d'édition.

Une des méthodes utilisées pour mesurer la similarité de deux séquences génétiques est leur *distance d'édition*[1] : Il s'agit d'*aligner* les deux séquences, i.e. les rendre égales, en s'autorisant à changer des caractères (en substituant un caractère par un autre dans une séquence)) ou à insérer des espaces dans l'une ou l'autre (par

1. La distance d'édition, ou *distance de Levenshtein*, mesure la distance entre deux chaînes de caractères. Introduite en 1965 dans le contexte de la théorie du codage, elle est maintenant largement utilisée, par exemple pour le suivi des versions d'un fichier (`git` par exemple ou en Unix `diff` et `patch`), la vérification orthographique, la reconnaissance vocale, la détection de plagiat, ou la linguistique computationnelle.

exemple, pour qu'elles aient la même longueur). Un coût est payé pour chaque espace inséré et également pour chaque paire de caractères qui ne correspondent pas dans l'alignement final. Intuitivement, ces coûts modélisent la probabilité relative de mutations ponctuelles résultant de la sibstitution ou de la suppression/insertion. Le tableau suivant donne des coûts largement utilisés dans les applications biologiques :

opération	coût
insérer un espace	2
aligner deux caractères qui ne correspondent pas (substitution)	1
aligner deux caractères qui correspondent	0

Voici deux alignements possibles des chaînes x="AACAGTTACC" et y="TAAGGTCA" :

x	y	coût
A	T	1
A	A	0
C	A	1
A	G	1
G	G	0
T	T	0
T	C	1
A	A	0
C	-	2
C	-	2
		8

x	y	coût
A	T	1
A	A	0
C	-	2
A	A	0
G	G	0
T	G	1
T	T	0
A	-	2
C	C	0
C	A	1
		7

L'alignement à gauche est de coût 8, l'autre 7. La *distance d'édition* est le coût minimal parmi tous les alignements possibles entre deux séquences. Dans cet exemple, la distance d'édition entre les deux chaînes est en fait de 7 et l'alignement de droite est donc optimal. Le calcul de la distance d'édition consiste à trouver le meilleur alignement parmi un nombre exponentiel d'alignements possibles. Par exemple, si les deux chaînes comportent 100 caractères, il existe plus de 10^{75} alignements possibles. A partir d'une caractérisation récursive du coût minimal et en éliminant les appels redondants par mémoïsation, la programmation dynamique permet de calculer la distance d'édition avec un nombre d'opérations polynomial en la longueur des séquences.

3 Caractérisation récursive de la distance d'édition

Notation : Soit $S = s_0 \dots s_{k-1}$ une séquence de k caractères ; pour $0 \leq i < j \leq k$, on note $S_{i,j}$ la sous-séquence $s_i \dots s_{j-1}$ de longueur $j - i$. Ainsi $S_{i,k} = s_i \dots s_{k-1}$ est le suffixe de S de longueur $k - i$.

Soient $X = x_0 \dots x_{M-1}$ et $Y = y_0 \dots y_{N-1}$ deux séquences de M et N caractères. La distance d'édition entre X et Y est caractérisée récursivement comme suit.
On note $\phi(i, j)$ la distance d'édition du suffixe $X_{i,M}$ de X avec le suffixe $Y_{j,N}$ de Y.
Il y a 3 cas pour un alignement optimal de $X_{i,M} = x_i \dots x_{M-1}$, avec $Y_{j,N} = y_j \dots y_{N-1}$.

Cas 1 l'alignement optimal fait correspondre x_i avec y_j : le coût de cette mise en correspondance est 0 ou 1 selon si $x_i = y_j$ ou non, soit $1 - \delta_{x_i,y_j}$; et en plus il faut aligner optimalement $x_{i+1,M}$ avec $y_{j+1,N}$; le coût total est donc $1 - \delta_{x_i,y_j} + \phi_(i + 1, j + 1)$.

Cas 2 l'alignement optimal fait correspondre x_i avec un espace inséré avant y_j ; dans ce cas, le coût est 2 pour l'insertion et en plus il faut aligner optimalement $x_{i+1,M}$ avec $y_{j,N}$; donc avec un coût total $2 + \phi_(i + 1, j)$.

Cas 3 l'alignement optimal fait correspondre y_j avec un espace inséré avant x_i ; dans ce cas, le coût est 2 pour l'insertion et en plus il faut aligner optimalement $x_{i,M}$ avec $y_{j+1,N}$; donc avec un coût total $2 + \phi_(i, j + 1)$.

D'où l'équation de Bellman pour calculer la distance d'édition $\phi(0, 0)$ entre X et Y :

$$\phi(i, j) = \min\left\{(1 - \delta_{x_i,y_j}) + \phi(i + 1, j + 1), 2 + \phi(i + 1, j), 2 + \phi(i, j + 1)\right\} \text{ pour } 0 \leq i < M, 0 \leq j < N \quad (1)$$

avec les conditions d'arrêt $\phi(i, N) = 2(M - i)$ et $\phi(M, j) = 2(N - j)$ pour $0 \leq i \leq M$ et $0 \leq j \leq N$ (car $X_{M,M}$ et $Y_{N,N}$ sont la chaîne vide).

4 Implémentation récursive de la distance d'édition

Les sources fournis (avec Makefile), à décompresser chez vous, sont sur les PCs de l'ensimag ici :

`/matieres/4MMAOD6/2023-10-TP-AOD-ADN-Docs-fournis/tp-ADN-distance.tgz`

Le programme `distanceEdition.c` fourni est une implémentation récursive avec mémoïsation de cette équation de Bellman :

```
distanceEdition file1 b1 L1 file2 b2 L2
```

prend en entrée deux noms de fichers de caractères `file1` (contenant la séquence X) et `file2` (contenant Y) chacun suivi de deux entiers b_1, L_1 et b_2, L_2 ; et il affiche sur la sortie standard la distance d'édition entre X_{b_1,b_1+L_1-1} et Y_{b_2,b_2+L_2-1} (i.e. commençant au caractère b_k de `filek` et de longueur L_k pour $k = 1, 2$).

Parcours des fichiers en lecture avec `mmap` : le programme fourni utilise la fonction système `mmap` pour projeter le fichier en mémoire virtuelle et le parcourir durectement comme un tableau de caractères, les accès aux éléments du tableau pouvant provoquer des défauts de cache. Cela évite de recopier le fichier dans un tableau auxiliaire (par exemple avec `read`), ce qui ajouterait des défauts de cache additionnels.

Format de fichiers en entrée : Le programme prend en entrée des séquences dans des fichiers texte, les bases canoniques étant représentées par les lettres A, C, G, T, U (majuscules ou minuscules). De plus, pour compatibilité avec les fichiers `.fna` au format FNA (*FASTA Format DNA and Protein Sequence Alignment file*) du site `https://www.ncbi.nlm.nih.gov/data-hub/genome/`, la séquence peut contenir :

- la lettre N (ou n en minuscule) correspondant à une base non identifiée (inconnue ou erronée) : la distance de N avec une autre base $b \in$ {A,C,G,T,U,N} – connue ou non – est ici fixée à $\delta_{N,b} = 1$;
- des caractères ne correspondant pas à des bases (comme le retour à la ligne '\n') : ils sont ignorés ;
- une ligne de commentaire, commençant par '>', avant une séquence.

Le fichier `characters_to_base.h` définit pour caractère z la base `CharToBase(`z`)` qui lui correspond ; :

- `isUnknownBase(`z`)` est vrai ssi z est la lettre N en minuscule ou majuscule ;
- `isBase(`z`)` est vrai ssi z est une des lettres {A, C, G, T, U, N} en minuscule ou majuscule (base connue ou inconnue) ; sinon, `isBase(`z`)` est faux, i.e. tous les autres caractères sont ignorés.

On définit : $\beta_x = 1$ si `isBase(`x`)` est vrai ; et 0 sinon. Et $\sigma_{x,y} = 1$ si $((x = N)$ ou $(x \neq y))$; et 0 sinon.

Caractérisation implémentée pour le format de fichier La formulation (1) est réécrite comme suit pour prendre en compte ces extensions pour $0 \leq i \leq M$ et $0 \leq j \leq N$:

$$\phi(i,j) = \min \begin{cases} 0 & \text{si } (i = M) \text{ et } (j = N) \\ 2 \times \beta_{y_j} + \phi(M, j+1) & \text{si } (i = M) \text{ et } (j < N) \\ 2 \times \beta_{x_i} + \phi(i+1, N) & \text{si } (i < M) \text{ et } (j = N) \\ \phi(i+1, j) & \text{si } (i < M) \text{ et } (\beta(x_i) = 0) \\ \phi(i, j+1) & \text{si } (j < N) \text{ et } (\beta(y_j) = 0) \\ \min\left\{\sigma_{x_i,y_j} + \phi(i+1, j+1); 2 + \phi(i+1, j); 2 + \phi(i, j+1)\right\} & \text{sinon.} \end{cases} \quad (2)$$

5 Jeux de données et mesures de performance

Vous mesurerez expérimentalement le nombre d'opérations et les défauts de cache avec **valgrind** ; pour cela vous compilerez le programme avec **gcc** et l'option **-g**.
Pour modifier les paramètres de la mémoire D1 (ou similairement D2)avec un cache de taille Z_1 octets, une ligne de cache de taille L_1 octets et une politique de remplacement associative à A_1 voies (approximation de LRU), la commande est :

`valgrind --tool=cachegrind --D1=`Z_1`,`A_1`,`L_1 `./nomprogramme <options>`

Ex : `valgrind --tool=cachegrind --D1=4096,4,64 ../install/distanceEdition XX.fna 0 10000 YY.fna 0 12340` (attention `--` ci-dessus est deux tirets consécutifs (cf `tests/Makefile-test`)

Le répertoire `tests` contient quelques tests de vérification (`make -f Makefile-test all`) et des tests de performance avec valgrind (`make -f Makefile-test all-valgrind`). Vous ajouterez vos propres tests qui doivent être lancés automatiquement.

Nota Bene. Vous remarquerez que pour des séquences de longueur assez grande, le programme récursif avec memoisation fourni s'arrête avec un message d'erreur (qui peut varier selon le système et les options de compilation) ; par exemple test5 plante sur les PCs de l'école. Quelle est la raison de cet arrêt ?

Le répertoire `/matieres/4MMAOD6/TP-AOD-ADN-Benchmark` contient des jeux réels de données de tailles réalistes issus de la base [2] [`https://www.ncbi.nlm.nih.gov/data-hub/genome`].

Deux génomes de *SARS-Cov-2* :

2. Remerciements à Antoine Frénoy pour les liens vers ces séquences.

— `wuhan_hu_1.fasta` : une séquence de la souche "originale" de Wuhan qui commence en position 116 et est de longueur 30331 ;

— `ba52_recent_omicron.fasta` : une séquence prise parmi des BA.5.2 (omicron) séquencés récemment qui commence en position 153 et est de longueur 30183 ;

Deux génomes de *Arabidopsis thaliana* (dont les positions et longueurs de séquences qui sont utilisées pour les tests sont données en annexe)

— `GCF_000001735.4_TAIR10.1_genomic.fna` : un génome de référence [3] contient 5 sequences, dont une en position 77328790 de longueur 20236404.

— `GCA_024498555.1_ASM2449855v1_genomic.fna` : un génome à classer [4] contient 7 sequences, dont une en position 30808129 de longueur 19944517.

Les positions et longueurs des séquences génétiques des fichiers fournis (qui incluent aussi des caractères à ignorer comme des retours à la ligne et des commentaires) sont données dans les tableaux en annexe (pages suivantes).

Dans un contexte réel, pour chacun de ces deux cas, les sous-séquences de ces fichiers peuvent être comparées entre elles. Pour les mesures de peroformance on comparera des sous-séquences de longueurs fixes (1000, 10000, 100000 etc) sans considérer le cadre réel.
Le modèle fourni pour le rapport précisera le barème et les tests qui seront faits pour l'évaluation de vos programmes.

6 Travail demandé et barême

On demande de remplir le questionnaire fourni dans `rapport-a-rendre/rapport-AOD-adn.tex`

Pour chacun de vos programmes, il est demandé d'analyser les défauts de cache sur les données manipulées en distinguant : le tableau mappé en mémoire X (resp. Y) et les données allouées en plus par votre programme (à la place du tableau ϕ de taille $N.M$).

1. (1 point) Préambule : le programme récursif avec mémoisation fourni alloue une mémoire de taille $N.M$. Pour le test 5, que vaut $N.M$ et pourquoi le programme fourni génère-t-il une erreur d'exécution ?
2. (5 points) Ecrire un programme itératif qui alloue un espace mémoire linéaire $O(N + M)$.
   Analyser le coût théorique de votre programme en fonction de N et M :
   (a) place mémoire allouée (ne pas compter les 2 séquences X et Y en mémoire via `mmap`) ;
   (b) travail (nombre d'opérations) ;
   (c) dur modèle CO, nombre de défauts de cache obligatoires (sur X, sur Y et sur vos tableaux) ;
   (d) nombre de défauts de cache capacitifs si $Z \ll \min(N, M)$ (sur X, sur Y et sur vos tableaux).
3. (3 points) Ecrire un programme cache aware. Expliquer très brièvement (2 à 5 lignes max) son principe, la mémoire utilisée, le sens de parcours des tableaux. Analyser son coût théorique comme ci-dessus (place memoire, travail et défauts de cache obligatoires et capacitifs).
4. (3 points) Ecrire un programme cache oblivious. Expliquer très brièvement (2 à 5 lignes max) son principe, la mémoire utilisée, le sens de parcours des tableaux. Analyser son coût théorique comme ci-dessus (place memoire, travail et défauts de cache).
5. (1 point) Comment choisissez-vous le seuil d'arrêt récursif du programme cache oblivious ? La réponse doit être justifiée. quelle valeur avez-vous fixée pour ce seuil sur les PCs de l'Ensimag ?
6. (7 points) Expérimentations : mesurer expérimentalement la performance de vos programmes : argumenter si ces mesures sont en accord ou en désaccord avec votre analyse théorique ci-dessus. (cf exemples de tableaux de mesures du questionnaire)
   — (3 points) par simulation avec valgrind (par exemple avec les paramètres `valgrind -tool=cachegrind -D1=4096,4,64`)
   — (3 points) en exécution sans valgrind (préciser les caractéristiques de la machine d'expérimentation)
   — (1 point) Extrapolation : lequel de vos programmes utiliseriez-vous pour la commande ci-dessous sur le PC utilisé pour vos mesures ? Quel temps et quelle énergie estimez-vous pour cette exécution ?

```
distanceEdition  GCA_024498555.1_ASM2449855v1_genomic.fna  77328790  20236404
                 GCF_000001735.4_TAIR10.1_genomic.fna      30808129  19944517
```

   Bonus : comment feriez-vous pour que cette commande s'exécute en moins de 1 minute ?

3. https://www.ncbi.nlm.nih.gov/data-hub/genome/GCF_000001735.4
4. https://www.ncbi.nlm.nih.gov/data-hub/genome/GCA_024498555.1

7 Annexe : Position et longueur maximale des séquences dans les fichiers fournis

FIGURE 1 – Sequences in file : /matieres/4MMAOD6/TP-AOD-ADN-Benchmark/GCA_024498555.1_ASM2449855v1_genomic.fna

n^o	Position	Length	First twenty chars	Last twenty chars
>CP086754.1 Arabidopsis thaliana ecotype 1254 chromosome 1 sequence				
1	68	32174572	aaccctaaaccctaacccta	agggtttagggtttagggtt
>CP086755.1 Arabidopsis thaliana ecotype 1254 chromosome 2 sequence				
2	32174709	20790648	GTGATCGGTTTTGTTGCCTT	agggtttagggttagggtaG
>CP086756.1 Arabidopsis thaliana ecotype 1254 chromosome 3 sequence				
3	52965426	24363295	aaccctaaaccctaaaccct	ggtttaggttaGGGTTTAGG
>CP086757.1 Arabidopsis thaliana ecotype 1254 chromosome 4 sequence				
4	77328790	20236404	CTTATCTGGGCTATTCAAGC	ggttaag ggtttagggtta
>CP086758.1 Arabidopsis thaliana ecotype 1254 chromosome 5 sequence				
5	97565263	28610666	aaccctaaaccctaaaccct	tttagggtttt taggGTTT

FIGURE 2 – Sequences in file : /matieres/4MMAOD6/TP-AOD-ADN-Benchmark/GCF_000001735.4_TAIR10.1_genomic.fna

n^o	Position	Length	First twenty chars	Last twenty chars
>NC_003070.9 Arabidopsis thaliana chromosome 1 sequence				
1	56	30808016	ccctaaaccctaaaccctaa	ttagggtttagggtttaggg
>NC_003071.7 Arabidopsis thaliana chromosome 2 sequence				
2	30808129	19944517	NNNNNNNNNNNNNNNNNNNN	ttagggtttagggtttaggg
>NC_003074.8 Arabidopsis thaliana chromosome 3 sequence				
3	50752703	23753077	NNNNNNNNNNNNNNNNNNNN	aaaccctaaaccctaaaccc
>NC_003075.7 Arabidopsis thaliana chromosome 4 sequence				
4	74505837	18817369	NNNNNNNNNNNNNNNNNNNN	tta gggtttagggtttagg
>NC_003076.8 Arabidopsis thaliana chromosome 5 sequence				
5	93323263	27312695	TATACCATGTACCCTCAAcc	ggatttagggtttttagatC
>NC_037304.1 Arabidopsis thaliana ecotype Col-0 mitochondrion, complete genome				
6	120636038	372405	AGAGGTCAGAGGTAACTTGT	TATATTTGCGGTCGTGAGAC
>NC_000932.1 Arabidopsis thaliana chloroplast, complete genome				
7	121008507	156408	ATGGGCGAACGACGGGAATT	ATAACTTGGTCCCGGGCATC

FIGURE 3 – Sequences in file : `wuhan_hu_1.fasta`

n^o	Position	Length	First twenty chars	Last twenty chars
>gi|1798174254|ref|NC_045512.2| Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome				
1	116	30331	ATTAAAGGTTTATACCTTCC	AAAAA AAAAAAAAAAAAA

FIGURE 4 – Sequences in file : `ba52_recent_omicron.fasta`

n^o	Position	Length	First twenty chars	Last twenty chars
>gi|2293206857|gb|OP341347.1| Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/humans [NCBI-Taxon :9606]/PAK/72613/2022, complete genome				
1	153	30183	CAGGTAACAAACCAACCAAC	ATTTTAGTAGT GCTATCC

FIGURE 5 – Sequences in file : `enonce-seq1`

n^o	Position	Length	First twenty chars	Last twenty chars
1	0	10	AAcaGTtACC	

FIGURE 6 – Sequences in file : `enonce-seq2`

n^o	Position	Length	First twenty chars	Last twenty chars
1	0	8	TAAGGTCA	

FIGURE 7 – Sequences in file : `f1.fna`

n^o	Position	Length	First twenty chars	Last twenty chars
1	0	9	CAcgT	

FIGURE 8 – Sequences in file : `f2.fna`

n^o	Position	Length	First twenty chars	Last twenty chars
1	0	55	<cette premiere lign	taire. acaCGTACNNNAT