Université de Liège

Recherche de motifs dans des séquences d'ADN

INFO0902-1 – Structure de données et algorithmes

François Rozet (s161024)

Bachelier Ingénieur civil

Année académique 2018-2019

Table des matières

1 Analyse théorique

1.1 Recherche dichotomique par force brute ou table de hachage

Pseudo-code 1 Recherche dichotomique par force brute

```
function DICHOTOMIC-SEARCH(X, Y)
    t = NIL
    p = 1; r = min(X.length, Y.length)
    while p ≤ r do
        q = ⌊(p+r)/2⌋
        s = BRUTE-FORCE-AUX(X, Y, q)
        if s ≠ NIL then
            p = q + 1
        else
            r = q − 1
            t = s
    return t
```

Pseudo-code 2 Recherche dichotomique par table de hachage

```
function HASH-TABLE-SEARCH(X, Y)
    t = NIL
    p = 1; r = min(X.length, Y.length)
    while p ≤ r do
        q = ⌊(p+r)/2⌋
        s = HASH-TABLE-AUX(X, Y, q)
        if s ≠ NIL then
            p = q + 1
        else
            r = q − 1
            t = s
    return t
```

Pseudo-code 3 Recherche auxiliaire par force brute

```
function BRUTE-FORCE-AUX(X, Y, l)
    for i = 1 to X.length − l + 1 do
        for j = 1 to Y.length − l + 1 do
            for k = 1 to l do
                if X[i + k − 1] ≠ Y[j + k − 1] then
                    k = 0
                    break
            if k ≠ 0 then
                return (l, i, j)
    return NIL
```

Pseudo-code 4 Recherche auxiliaire par table de hachage sans calcul incrémental

```
function HASH-TABLE-AUX1(X, Y, l)
    Let H be a new hash table
    for i = 1 to X.length − l + 1 do
        f = ENCODE(X, i, l)
        HASH-INSERT(H, f, i)
    for j = 1 to Y.length − l + 1 do
        f = ENCODE(Y, j, l)
        i = HASH-SEARCH(H, f)
        if i ≠ NIL then
            return (l, i, j)
    return NIL
```

Pseudo-code 5 Recherche auxiliaire par table de hachage avec calcul incrémental

```
function HASH-TABLE-AUX2(X, Y, l)
    Let H be a new hash table
    b = 4; p = b^(l−1)
    f = ENCODE(X, 1, l)
    HASH-INSERT(H, f, 1)
    for i = 2 to X.length − l + 1 do
        f = (f − X[i − 1] · p) · b + X[i + l − 1]
        HASH-INSERT(H, f, i)
    f = ENCODE(Y, 1, l)
    i = HASH-SEARCH(H, f)
    if i ≠ NIL then
        return (l, i, 1)
    for j = 2 to Y.length − l + 1 do
        f = (f − Y[j − 1] · p) · b + Y[j + l − 1]
        i = HASH-SEARCH(H, f)
        if i ≠ NIL then
            return (l, i, j)
    return NIL
```

Pseudo-code 6 Fonction d'encodage

```
function ENCODE(X, i, l)
    b = 4
    f = 0
    for j = 1 to l do
        f = f · b + X[i + j − 1]
    return f
```

1.2 Fonction de coût

$$C[i,j] = \begin{cases} C[i-1, j-1] + 1 & \text{si } i, j > 0 \text{ et } X[i] = Y[j] \\ 0 & \text{sinon} \end{cases} \tag{1}$$

1.3 Programmation dynamique

Pseudo-code 7 Recherche par programmation dynamique

```
function DYNAMIC-SEARCH(X, Y)
    Let C[0 ... X.length, 0 ... Y.length] be a new table
    l = 0
    for i = 0 to X.length do
        for j = 0 to Y.length do
            if i > 0 and j > 0 and X[i] == Y[j] then
                C[i, j] = C[i − 1, j − 1] + 1
                if C[i, j] > l then
                    l = C[i, j]
                    i_end = i
                    j_end = j
            else C[i, j] = 0
    if l ≠ 0 then
        return (l, i_end − l + 1, j_end − l + 1)
    return NIL
```

1.4 Complexité

Soit n la longueur des deux séquences et αn la longueur de leur sous-séquence contiguë commune la plus longue.

(a) La solution par force brute est composée de quatre boucles imbriquées. Les boucles initiale et intermédiaires exécutent toujours de l'ordre de $(1-\alpha)n$ itérations. La boucle finale, quant à elle, en exécute de l'ordre de αn. Les opérations à l'intérieur de ces boucles étant toutes $\mathcal{O}(1)$, la complexité en temps est

$$\mathcal{O}\left(\alpha(1-\alpha)^3 n^4\right) = \mathcal{O}\left(n^4\right) \tag{2}$$

Néanmoins, lorsque αn tend vers n ($\alpha \to 1$), les trois premières boucles voient leur nombre d'itérations tendre vers 1. Ainsi, la complexité en temps est $\Omega(n)$. On trouve aussi que $\alpha = \frac{1}{4}$ est le pire cas, bien qu'il soit équivalent au cas moyen en terme de complexité.

Concernant la complexité en espace, selon ce qu'il a été décidé de renvoyer, elle peut être $\Theta(1)$ ou $\Theta(\alpha n)$. Si la recherche renvoie un triplet (l, i, j) (l la longueur, i l'indice de départ dans $G1$ et j l'indice de départ dans $G2$), la complexité sera

$\Theta(1)$. À l'inverse, si la recherche renvoie la sous-séquence sous forme d'une table [1], la complexité sera $\Theta(\alpha n)$.

(b) Comme la solution par force brute, la recherche dichotomique est composée de quatre boucles imbriquées. Cependant, ici, la boucle initiale s'arrête lorsque un interval de taille n a été divisé *suffisamment* de fois, c.-à-d. $\log_2(n)$ fois. De plus, les boucles intermédiaires sont maintenant en $\mathcal{O}(n)$. Ainsi, la complexité en temps est

$$\mathcal{O}\left(\alpha n^3 \log(n)\right) = \mathcal{O}\left(n^3 \log(n)\right) \tag{3}$$

Pour cette solution, lorsque αn tend vers n, les boucles intermédiaires ne voient pas leur nombre d'itérations tendre vers 1. Cependant, lorsque $\alpha n = 0$, la boucle la plus profonde n'effectue plus qu'une seul itération. Ainsi, la complexité en temps est $\Omega(n^2 \log(n))$. Le pire cas est équivalent au cas moyen.

Concernant la complexité en espace, cette solution est équivalent à la force brute.

(c) La solution par table de hachage ne possède que deux boucles imbriquées qui correspondent aux deux premières de la recherche dichotomique. Néanmoins, sans calcul incrémental, l'opération d'encodage exécute $\mathcal{O}(n)$ opérations à chaque appel. On suppose aussi que l'insertion et la lecture dans la table de hachage sont faites en $\mathcal{O}(1)$. Dès lors, la complexité en temps est

$$\Theta\left(n^2 \log(n)\right) \tag{4}$$

Concernant la complexité en espace, à chaque appel de la fonction auxiliaire pour une taille l de sous-séquence, il est nécessaire d'initialiser une table de hachage sur un univers $U = \{0, 1, \ldots, 4^l - 1\}$ afin d'y stocker $n - l + 1$ sous-séquences. Pour conserver un accès (insertion, recherche et suppression) aux éléments de la table en $\mathcal{O}(1)$, il est nécessaire d'avoir $m = \Omega(n - l)$ où m est sa taille [2]. Or, la valeur de départ de l est $\frac{n}{2}$, dès lors, la complexité en espace de la solution par table de hachage est $\Theta(n)$.

(d) Avec le calcul incrémental, l'encodage ne demande plus que $\mathcal{O}(1)$ opérations par appel. Ainsi, la complexité en temps devient

$$\Theta\left(n \log(n)\right) \tag{5}$$

La complexité en espace n'a, quant à elle, pas changé.

(e) La solution par programmation dynamique n'utilise que deux boucles impbriquées itérant sur l'entièreté des séquences. Dès lors, la complexité en temps est

$$\Theta\left(n^2\right) \tag{6}$$

Concernant la complexité en espace, la matrice de coût possède $(n+1)^2$ éléments. La complexité spatiale est donc aussi $\Theta\left(n^2\right)$.

1. Les pseudo-codes ont été rédigés selon cette politique.
2. Dans le cas d'une table de hachage par adressage direct, la condition $m \geq n - l + 1$ doit aussi être respectée.