

Recherche des structures secondaires d'une chaîne d'ADN

Projet 2I003 réalisé sur l'ancien sujet

LI Mengda, GE Zhichun

5 décembre 2017

1 Exercice 1

1.1

Comme un nucléotide ne peut former une paire avec lui même, donc

$$\forall i \in \{1, \dots, n\}, \quad S_{i,i} = \{\} \text{ et } E_{i,i} = 0$$

1.2

1.2.1

Si ni i , ni j ne sont couplés dans $S_{i,j}$, alors

$$S_{i,j} = S_{i+1,j-1} \quad E_{i,j} = E_{i+1,j-1}$$

1.2.2

Si j n'est pas couplée dans $S_{i,j}$, alors

$$S_{i,j} = S_{i,j-1} \quad E_{i,j} = E_{i,j-1}$$

1.2.3

Si $(i, j) \in S_{i,j}$, alors

$$S_{i,j} = S_{i+1,j-1} \cup (i, j) \quad E_{i,j} = E_{i+1,j-1} + 1$$

1.2.4

Si $(k, j) \in S_{i,j}$ avec $k \in \{i+1, \dots, j-1\}$, alors

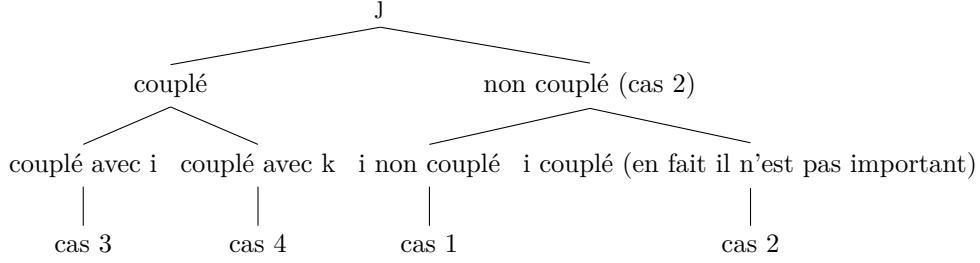
$$S_{i,j} = S_{i,k-1} \cup S_{k,j} \quad E_{i,j} = E_{i,k-1} + E_{k,j}$$

car, par hypothèse, il n'existe pas de nœud : lorsqu'on coupe en k , on ne coupe pas de couple dans ce cas là, il n'y a donc pas de perte. Sinon, on a $S_{i,j} \subset S_{i,k-1} \cup S_{k,j}$ dans tous les cas.

1.3

1.3.1

Par la question 1.2, résumons en distinguant tous les cas possibles :



1. Soit ni i , ni j ne sont couplés dans $S_{i,j}$, alors $E_{i,j} = E_{i+1,j-1}$,
et on a toujours $E_{i,j} \geq E_{i+1,j-1}$ dans tous les cas.
2. Soit j n'est pas couplée dans $S_{i,j}$, alors $E_{i,j} = E_{i,j-1}$,
et on a toujours $E_{i,j} \geq E_{i,j-1}$ dans tous les cas.
3. Soit j est couplé avec i , c'est à dire $(i,j) \in S_{i,j}$, alors $E_{i,j} = E_{i+1,j-1} + 1$
4. Soit j est couplé avec un $k \in \{i+1, \dots, j-1\}$, alors $E_{i,j} = E_{i,k-1} + E_{k,j}$,
sinon $E_{i,j} \geq E_{i,k-1} + E_{k,j}$ car $S_{i,j} \supset S_{i,k-1} \cup S_{k,j} \quad \forall k \in \{i+1, \dots, j-1\}$

Soit la fonction $e : \{1, \dots, n\} \times \{1, \dots, n\} \rightarrow \{0, 1\}$ qui vaut 1 ssi i et j peuvent être couplés.

Dans le cas 1 et cas 3, $E_{i,j} = E_{i+1,j-1} + e(i,j)$ et on a $E_{i,j} \geq E_{i+1,j-1} + e(i,j)$ dans tous les cas.

- si j est couplé avec i , $e(i,j) = 1$, $E_{i,j} = E_{i+1,j-1} + 1 = E_{i+1,j-1} + e(i,j)$,
- sinon $e(i,j) = 0$, on a toujours $E_{i,j} \geq E_{i+1,j-1} = E_{i+1,j-1} + e(i,j)$ par le 1.

Dans le cas 2, $E_{i,j} = E_{i,j-1}$ et on a $E_{i,j} \geq E_{i,j-1}$ dans tous les cas. car $S_{i,j} \supset S_{i,j-1}$

Par le cas 4, $\forall k \in \{i+1, \dots, j-1\}$ $E_{i,j} \geq E_{i,k-1} + E_{k,j}$. Donc $E_{i,j} \geq \max_{i < k < j} E_{i,k-1} + E_{k,j}$ dans tous les cas.

Et s'il existe $k_0 \in \{i+1, \dots, j-1\}$ tel que $(k_0, j) \in S_{i,j}$, $E_{i,j} = E_{i,k_0-1} + E_{k_0,j}$

Dans ce cas-là, le max est atteint :

$$E_{i,k_0-1} + E_{k_0,j} \geq \max_{i < k < j} E_{i,k-1} + E_{k,j} \text{ et } \max_{i < k < j} E_{i,k-1} + E_{k,j} \geq E_{i,k_0-1} + E_{k_0,j}$$

$$\text{D'où } E_{i,j} = E_{i,k_0-1} + E_{k_0,j} = \max_{i < k < j} E_{i,k-1} + E_{k,j} \text{ dans le cas 4}$$

En conclusion : $E_{i,j}$ est un majorant de l'ensemble $\{E_{i+1,j-1} + e(i,j), E_{i,j-1}, \max_{i < k < j} E_{i,k-1} + E_{k,j}\}$, et ce majorant est atteint : il y a toujours un cas où la condition d'égalité est vrai. Donc

$$E_{i,j} = \max\{E_{i+1,j-1} + e(i,j), E_{i,j-1}, \max_{i < k < j} E_{i,k-1} + E_{k,j}\}$$

...

1.3.2

Si $k = j$, $E_{k,j} = E_{j,j} = 0$ par 1.1, donc $E_{i,j-1} = (E_{i,j-1} + E_{j,j}) \in \{E_{i,k-1} + E_{k,j} \mid i < k \leq j\}$,
et

$$E_{i,j-1} \leq \max\{E_{i,k-1} + E_{k,j} \mid i < k \leq j\}$$

$$\text{On en déduit que } E_{i,j} = \max\{E_{i+1,j-1} + e(i,j), \max_{i < k \leq j} E_{i,k-1} + E_{k,j}\}$$

1.4 Code en Python3

```
1 def tailleMaxRec(a: str, i: int, j: int) -> int:
    n = len(a)
3     assert 1 <= i <= n and 1 <= j <= n, 'impossible de tronquer'

5     if a == '' or i >= j:
        return 0

7
8     def couple(i: int, j: int) -> bool:
9         return {a[i-1], a[j-1]} in [ {'A', 'T'}, {'C', 'G'} ]

11    def e(i: int, j: int) -> {0,1}:
12        if couple(i, j):
13            return 1
14        else:
15            return 0

17    return max(tailleMaxRec(a, i+1, j-1) + e(i, j),
18               max([tailleMaxRec(a, i, k-1) + tailleMaxRec(a, k, j)
19                    for k in range(i+1, j+1)]))
```

1.5

Cette fonction se termine car chaque fois on appelle récursivement sur une sous-partie de séquence $[i+1, j-1]$, le début de sous-séquence i est strictement croissant et la fin de sous-séquence j est strictement décroissante. Dans \mathbb{N} , l'ordre est bien fondé donc i, j se rencontre en certaine récursion. Lorsque i, j se rencontre, c'est à dire $i \leq j$, la fonction retourne une valeur et se termine en dépillant récursivement.

Cette fonction calcule exactement $\max\{E_{i+1,j-1} + e(i,j), \max_{i < k < j} E_{i,k-1} + E_{k,j}\}$, elle est valide selon la question 1.3.

1.6

1.6.1

Pour $u_0, (p=0) \Rightarrow (i=j)$. Donc l'appel de fonction retourne simplement 0, $u_0 = 1$

Pour $u_1, (p=1) \Rightarrow (i+1=j)$, donc cet appel retourne

```
1 max(tailleMaxRec(a, i+1, j-1) + e(i, j),
2     max([tailleMaxRec(a, i, k-1) + tailleMaxRec(a, k, j)
3          for k in range(i+1, j+1)]))
```

Vu que $i+1=j$, $\text{tailleMaxRec}(a, i+1, j-1)$ retourne simplement 0 car $i+1=j \geq j-1$.

Analysons la dernière itérable

```
[tailleMaxRec(a, i, k-1) + tailleMaxRec(a, k, j) for k in range(i+1, j+1)]
```

$\text{range}(j, j+1)$ ne contient que j , donc cette liste n'a qu'un élément, cela vaut dire qu'on appelle deux fois tailleMaxRec . En sommant, $u_1 = 3$

1.6.2

Soit $p \leq 2$, pour u_p

- On appelle d'abord une fois la fonction elle-même : $\text{tailleMaxRec}(a, i, j)$, on a 1 appel.
- Et puis on appelle $\text{tailleMaxRec}(a, i+1, j-1)$,
 $(j-1) - (i+1) = (j-i) - 2 = 0$, on a donc u_0 appel(s).
- Enfin, pour
 $[\text{tailleMaxRec}(a, i, k-1) + \text{tailleMaxRec}(a, k, j) \text{ for } k \text{ in } \text{range}(i+1, j+1)]$
 $\text{range}(i+1, j+1)$ équivaut à $\llbracket i+1, i+p \rrbracket$, car $p = j - i$, $j = i + p$
 Le nombre d'appel de $\text{tailleMaxRec}(a, i, k-1)$ équivaut u_{k-1-i} ,
 Le nombre d'appel de $\text{tailleMaxRec}(a, k, j)$ équivaut u_{p+i-k}
 On a donc $\sum_{k=i+1}^{i+p} u_{k-1-i} + \sum_{k=i+1}^{i+p} u_{p+i-k}$ d'appels.
 Pour la première, en effectuant un changement de variable $q = k - (i+1)$, on a bien $\sum_{q=0}^{p-1} u_q$
 Pour la deuxième, en effectuant un changement de variable $q = p + i - k$, on a $\sum_{q=p-1}^0 u_q$
 qui vaut en effet $\sum_{q=0}^{p-1} u_q$.
 En sommant les deux sommes, on a bien $2 \sum_{i=0}^{p-1} u_i$
 En conclusion, $u_p = u_0 + 1 + 2 \sum_{i=0}^{p-1} u_i$

1.6.3

- Démontrons par récurrence $u_n \geq 2^n$.
- Cas de base, $u_0 = 1 \geq 2^0$, $u_1 = 3 \geq 2^1$ vérifié dans question 1.
 Et $u_2 = u_0 + 1 + 2(u_0 + u_1) = 10 \geq 2^2$
 - Induction : par récurrence forte
 Soit $n \in \mathbb{N}, n \geq 2$ fixé, supposons on a $u_k \geq 2^k \forall k \in \{2, \dots, n\}$
 $u_{n+1} = u_0 + 1 + 2 \sum_{i=0}^n u_i \geq 2^0 + 2^0 + 2 \sum_{i=0}^n 2^i = 2^1 + 2^{\frac{1-2^{n+1}}{1-2}} = 2^{n+2} \geq 2^{n+1}$
 En conclusion, $\forall n \in \mathbb{N}, u_n \geq 2^n$

1.6.4

On en déduit que sa complexité est de $\mathcal{O}(2^n)$

1.7

Sa complexité est de $\Theta(n^3)$.

Il y a une boucle for simple et une boucle for triplé dans cet algorithme. Le reste est de temps constant.

La première boucle fait $(n-1)$ itération. Dans chaque itération on effectue un nombre constant d'opérations. Donc il est de complexité $\Theta(n)$.

Le cœur de l'analyse est au sein de la deuxième boucle composé en effet de trois boucle for, voici le code Python qui l'implémente :

```

2   for p in range(1, n):
3       for i in range(1, n-p+1):
4           E[(i, i+p)] = max( E[(i+1, i+p-1)] + e(a, i, i+p),
                             max ( E[(i, k-1)] + E[(k, i+p)]
                                   for k in range(i+1, i+p+1)))

```

Il fait en effet $\sum_{p=1}^{n-1} \sum_{i=1}^{n-p} \sum_{k=i+1}^{i+p}$ itérations dans laquelle on effectue un nombre fini d'opération.

$$\begin{aligned} \sum_{p=1}^{n-1} \sum_{i=1}^{n-p} \sum_{k=i+1}^{i+p} &= \sum_{p=1}^{n-1} \sum_{i=1}^{n-p} \frac{(i+1+i+p)p}{2} = \sum_{p=1}^{n-1} \frac{(\frac{(3+p)p}{2} + \frac{(2(n-p)+1+p)p}{2})(n-p)}{2} \\ &\Leftrightarrow \frac{(\frac{(3+1)1}{2} + \frac{(2(n-1)+1+1)1}{2})(n-1)}{2} + \frac{(\frac{(3+(n-1))(n-1)}{2} + \frac{(2(n-(n-1))+1+(n-1))(n-1)}{2})(n-(n-1))}{2} \\ &\Leftrightarrow \frac{(\frac{(n+\frac{3}{2})(n-1)}{2} + \frac{(\frac{(n+2)(n-1)}{2} + \frac{(n+2)(n-1)}{2})}{2})(n-1)}{2} \Leftrightarrow \frac{(n^2+n/2-3/2 + \frac{n^2+n-2}{2})(n-1)}{2} \Leftrightarrow \frac{(2n^2+3n/2-7/2)(n-1)}{4} \\ &\Leftrightarrow \frac{(4n+7)(n-1)(n-1)}{8} \in \Theta(n^3) \end{aligned}$$

2 Exercice 2

2.1 Code en Python3

```

1 a = 'TCGGCTGCATTTCGA'
2 def e(a: str, i: int, j: int) -> {0,1}:
3     """Retourner 1 si i,j sont couples, 0 sinon."""
4     def couple(a: str, i: int, j: int) -> bool:
5         return {a[i-1], a[j-1]} in [ {'A', 'T'}, {'C', 'G'} ]
6
7     if couple(a, i, j):
8         return 1
9     else:
10        return 0
11 def tailleMaxRec(a: str, i: int, j: int) -> int:
12     n= len(a)
13     assert 1 <= i <= n and 1 <= j <= n, 'impossible de tronquer'
14     if a == '' or i >= j:
15         return 0
16     return max(tailleMaxRec(a, i+1, j-1) + e(a,i,j),
17               max([tailleMaxRec(a, i, k-1) + tailleMaxRec(a, k, j)
18                   for k in range(i+1, j+1)]) )
19
20 def tailleMaxIter(a: str, i: int, j: int) -> int:
21     n= len(a)
22     E=dict()
23     E[(1,1)] = 0
24     for i in range(2, n+1):
25         E[(i,i)] = 0
26         E[(i,i-1)] = 0
27     for p in range(1, n):
28         for i in range(1, n-p+1):
29             E[(i,i+p)] = max( E[(i+1,i+p-1)] + e(a, i, i+p),
30                             max ( E[(i,k-1)] + E[(k,i+p)]
31                                 for k in range(i+1, i+p+1)))
32
33     return E[(i,j)]
34
35 if __name__ == '__main__':
36     import timeit
37     print("E_3_13 calcule par l'algorithme recursive =", tailleMaxRec(a, 3, 13))
38     print("E_3_13 calcule par l'algorithme iterative =", tailleMaxIter(a, 3, 13))
39     print('10 executions de la fonction recursive prennent',
40           timeit.timeit("tailleMaxRec(a, 3, 13)", number=10,
41                           setup="from __main__ import tailleMaxRec, a"), 'secondes')
42     print('10 executions de la fonction iterative prennent',
43           timeit.timeit("tailleMaxIter(a, 3, 13)", number=10,
44                           setup="from __main__ import tailleMaxIter, a"), 'secondes')

```

Le résultat de teste donne

```
E_3_13 calcule par l'algorithm recursive = 4
E_3_13 calcule par l'algorithm iterative = 4
10 executions de la fonction recursive prennent 1.213047794000886 secondes
10 executions de la fonction iterative prennent 0.004673413001000881 secondes
```

2.2

```
from random import randrange

def SeqAleatoire(n: int) -> str:
    return ''.join(['A', 'T', 'C', 'G'][randrange(4)] for i in range(n))
```

Quelques testes :

```
>>> SeqAleatoire (10)
'TAGACCATAG'
>>> SeqAleatoire (10)
'TCGACAGATT'
>>> SeqAleatoire (11)
'GGGGTGGGGCT'
>>> SeqAleatoire (11)
'CTTAATTCTGG'
```

2.3 Test expérimental : question 3, 4, 5

```
import timeit

def testRec() -> None:
    n = 1
    lt0 = 0
    while True:
        a = SeqAleatoire(n)
        t1 = timeit.timeit("tailleMaxRec(a, 1, n)", number=1,
                           setup="from __main__ import tailleMaxRec",
                           globals=globals())
        print('CRec({0:d}) = {1:7f}, log'(CRec({0:d})) = {2:7f}'.format(
            n, t1, log(t1) - lt0))
        print()
        if t1 > 600:
            break
        n = n + 1
        lt0 = log(t1)
```

```

1 def testIter() -> None:
2     n = 1
3     while True:
4         a = SeqAleatoire(n)
5         t2 = timeit.timeit("tailleMaxIter(a, 1, n)", number=1,
6                             setup="from __main__ import tailleMaxIter",
7                             globals=locals())
8         print('CIter({0:d}) = {1:7f}, CIter({0:d})/{0:d}^3 = {2:7f}'.format(
9             n, t2, t2 / n**3))
10        print()
11        if t2 > 300:
12            break
13        n = n + 1

```

2.3.1 Test de la fonction récursive

```

>>> testRec()
2 CRec(1) = 0.000003, log'(CRec(1)) = -12.782327
4 CRec(2) = 0.000085, log'(CRec(2)) = 3.406688
6 CRec(3) = 0.000040, log'(CRec(3)) = -0.739755
8 CRec(4) = 0.000066, log'(CRec(4)) = 0.490493
10 CRec(5) = 0.000169, log'(CRec(5)) = 0.939710
12 CRec(6) = 0.000681, log'(CRec(6)) = 1.392812
14 CRec(7) = 0.002577, log'(CRec(7)) = 1.331125
16 CRec(8) = 0.006125, log'(CRec(8)) = 0.865912
18 CRec(9) = 0.015648, log'(CRec(9)) = 0.937961
20 CRec(10) = 0.041903, log'(CRec(10)) = 0.984977
22 CRec(11) = 0.129540, log'(CRec(11)) = 1.128638
24 CRec(12) = 0.408717, log'(CRec(12)) = 1.149034
26 CRec(13) = 1.262126, log'(CRec(13)) = 1.127531
28 CRec(14) = 4.076094, log'(CRec(14)) = 1.172342
30 CRec(15) = 13.468090, log'(CRec(15)) = 1.195184
32 CRec(16) = 43.075104, log'(CRec(16)) = 1.162622
34 CRec(17) = 144.504570, log'(CRec(17)) = 1.210366
36 CRec(18) = 480.611199, log'(CRec(18)) = 1.201747
38 CRec(19) = 1446.607645, log'(CRec(19)) = 1.102334

```

On peut voir que le temps d'exécution de `tailleMaxRec` croît très vite en raison de sa complexité exponentielle. Lorsque $n = 18$, cela prend plus de 5 minutes pour l'exécuter et plus de 10 minutes lorsque $n = 19$. Donc la plus grande valeur de n que je peux traiter est 19.

La pente de la droite $\log' CRec(n) \approx 1$, qui vaut dire que cette fonction est affine à n .

2.3.2 Test de la fonction itérative

Le calcul par l'itération est très efficace donc la plus grande valeur que je peux tester est relativement très grande et pratiquement je ne peux pas afficher le résultat de test entier. Je donne un extrait de test ici. On peut voir que $\frac{CIter(n)}{n^3}$ vaut constamment presque nul.

```

1 CIter(1) = 0.000005, CIter(1)/1^3 = 0.000005
3 CIter(2) = 0.000040, CIter(2)/2^3 = 0.000005
5 CIter(3) = 0.000028, CIter(3)/3^3 = 0.000001
7 ...
9 CIter(25) = 0.001586, CIter(25)/25^3 = 0.000000
11 CIter(26) = 0.001813, CIter(26)/26^3 = 0.000000
13 CIter(27) = 0.001995, CIter(27)/27^3 = 0.000000
15 CIter(28) = 0.002584, CIter(28)/28^3 = 0.000000
17 ...
19 CIter(72) = 0.025478, CIter(72)/72^3 = 0.000000
21 CIter(73) = 0.026434, CIter(73)/73^3 = 0.000000
23 CIter(74) = 0.028332, CIter(74)/74^3 = 0.000000
25 CIter(75) = 0.029268, CIter(75)/75^3 = 0.000000
27 ...
29 CIter(100) = 0.062157, CIter(100)/100^3 = 0.000000
31 CIter(101) = 0.065108, CIter(101)/101^3 = 0.000000
33 CIter(102) = 0.069097, CIter(102)/102^3 = 0.000000
35 ...
37 CIter(200) = 0.467661, CIter(200)/200^3 = 0.000000
39 CIter(201) = 0.470119, CIter(201)/201^3 = 0.000000
41 CIter(202) = 0.517451, CIter(202)/202^3 = 0.000000
43 CIter(203) = 0.469706, CIter(203)/203^3 = 0.000000
45 ...
47 CIter(300) = 2.228376, CIter(300)/300^3 = 0.000000
CIter(301) = 2.223427, CIter(301)/301^3 = 0.000000...
```


Voici quelques recherche à la main pour trouver <la plus grande valeur>. Lorsque $n = 1500$, cela prend déjà plus de 5 min pour le calculer. Donc je choisis 1500 comme <la plus grande valeur>.

```
1 >>> n=500
2 >>> timeit.timeit("tailleMaxIter(SeqAleatoire(n), 1, n)", number=1,
3     setup="from __main__ import tailleMaxIter",
4     globals=globals())
5 15.326990880996163
6 >>> n = 700
7 >>> timeit.timeit("tailleMaxIter(SeqAleatoire(n), 1, n)", number=1,
8     setup="from __main__ import tailleMaxIter",
9     globals=globals())
10 39.779907284995716
11 >>> n = 1000
12 >>> timeit.timeit("tailleMaxIter(SeqAleatoire(n), 1, n)", number=1,
13     setup="from __main__ import tailleMaxIter",
14     globals=globals())
15 126.6036247910015
16 >>> n = 1500
17 >>> timeit.timeit("tailleMaxIter(SeqAleatoire(n), 1, n)", number=1,
18     setup="from __main__ import tailleMaxIter",
19     globals=globals())
20 468.63339252900187
```

Appendice

On peut ajouter une comparaison de $\frac{CIter(n)}{n^2}$ pour voir la complexité expérimentalement : $\frac{CIter(n)}{n^2}$ est en fait croissant !

Donc il n'est pas possible que la complexité de l'algorithme itérative soit de $\Theta(n^2)$.

2	CIter(1) = 0.000005, CIter(1)/1^3 = 0.000005, CIter(1)/1^2 = 0.000005
4	CIter(2) = 0.000028, CIter(2)/2^3 = 0.000004, CIter(2)/2^2 = 0.000007
6	CIter(3) = 0.000130, CIter(3)/3^3 = 0.000005, CIter(3)/3^2 = 0.000014
8	CIter(4) = 0.000057, CIter(4)/4^3 = 0.000001, CIter(4)/4^2 = 0.000004
10	...
12	CIter(100) = 0.071538, CIter(100)/100^3 = 0.000000, CIter(100)/100^2 = 0.000007
14	CIter(101) = 0.068498, CIter(101)/101^3 = 0.000000, CIter(101)/101^2 = 0.000007
16	CIter(102) = 0.067363, CIter(102)/102^3 = 0.000000, CIter(102)/102^2 = 0.000006
18	...
20	CIter(198) = 0.446628, CIter(198)/198^3 = 0.000000, CIter(198)/198^2 = 0.000011
22	CIter(199) = 0.462426, CIter(199)/199^3 = 0.000000, CIter(199)/199^2 = 0.000012
24	CIter(200) = 0.470033, CIter(200)/200^3 = 0.000000, CIter(200)/200^2 = 0.000012
26	...
28	CIter(250) = 1.016588, CIter(250)/250^3 = 0.000000, CIter(250)/250^2 = 0.000016
30	CIter(251) = 1.017460, CIter(251)/251^3 = 0.000000, CIter(251)/251^2 = 0.000016
32	CIter(252) = 1.049520, CIter(252)/252^3 = 0.000000, CIter(252)/252^2 = 0.000017
34	...
36	CIter(268) = 1.318201, CIter(268)/268^3 = 0.000000, CIter(268)/268^2 = 0.000018
38	CIter(269) = 1.327808, CIter(269)/269^3 = 0.000000, CIter(269)/269^2 = 0.000018
40	CIter(270) = 1.354491, CIter(270)/270^3 = 0.000000, CIter(270)/270^2 = 0.000019
42	CIter(271) = 1.368608, CIter(271)/271^3 = 0.000000, CIter(271)/271^2 = 0.000019
44	...
46	CIter(287) = 1.682067, CIter(287)/287^3 = 0.000000, CIter(287)/287^2 = 0.000020
48	CIter(288) = 1.745375, CIter(288)/288^3 = 0.000000, CIter(288)/288^2 = 0.000021
50	...
52	CIter(299) = 2.005110, CIter(299)/299^3 = 0.000000, CIter(299)/299^2 = 0.000022
54	CIter(300) = 2.021642, CIter(300)/300^3 = 0.000000, CIter(300)/300^2 = 0.000022
	CIter(301) = 2.101834, CIter(301)/301^3 = 0.000000, CIter(301)/301^2 = 0.000023

```

56 CIter(302) = 2.104127, CIter(302)/302^3 = 0.000000, CIter(302)/302^2 = 0.000023
58 ...
60 CIter(347) = 3.612243, CIter(347)/347^3 = 0.000000, CIter(347)/347^2 = 0.000030
62 CIter(348) = 3.701992, CIter(348)/348^3 = 0.000000, CIter(348)/348^2 = 0.000031
64 CIter(349) = 3.754785, CIter(349)/349^3 = 0.000000, CIter(349)/349^2 = 0.000031
66 ...
68 CIter(400) = 6.285140, CIter(400)/400^3 = 0.000000, CIter(400)/400^2 = 0.000039
70 CIter(401) = 6.664165, CIter(401)/401^3 = 0.000000, CIter(401)/401^2 = 0.000041
72 CIter(402) = 7.256984, CIter(402)/402^3 = 0.000000, CIter(402)/402^2 = 0.000045
74 CIter(403) = 6.513025, CIter(403)/403^3 = 0.000000, CIter(403)/403^2 = 0.000040
76 CIter(404) = 6.679665, CIter(404)/404^3 = 0.000000, CIter(404)/404^2 = 0.000041
78 ...
80 CIter(424) = 8.426929, CIter(424)/424^3 = 0.000000, CIter(424)/424^2 = 0.000047
82 CIter(425) = 8.165203, CIter(425)/425^3 = 0.000000, CIter(425)/425^2 = 0.000045
84 CIter(426) = 7.977312, CIter(426)/426^3 = 0.000000, CIter(426)/426^2 = 0.000044
86 CIter(427) = 8.363914, CIter(427)/427^3 = 0.000000, CIter(427)/427^2 = 0.000046
88 CIter(428) = 8.285690, CIter(428)/428^3 = 0.000000, CIter(428)/428^2 = 0.000045

```