

HUANG Qijia
Yefangzhou

Project 21003

$$\text{Exz. Q1. } S_{i,i} = 0$$

$$E_{i,i} = 0.$$

$$\text{Q2. } \cancel{S_{i+1,j-1}} = S_{i,j} = S_{i+1,j-1}$$

$$1. E_{i,j} = \bar{E}_{i+1,j-1}$$

$$2. S_{i,j} = S_{i,j-1}$$

$$E_{i,j} = E_{i,j-1}$$

$$3. S_{i,j} = S_{i+1,j-1} \cup \{i,j\}$$

$$\bar{E}_{i,j} = \bar{E}_{i+1,j-1} + 1$$

$$4. S_{i,j} = S_{i,k-1} \cup S_{k,j}$$

$$\bar{E}_{i,j} = \bar{E}_{i,k-1} + \bar{E}_{k,j}$$

Q3- 1. On doit montrer que :

$$a. \bar{E}_{i,i+1} = \max(\bar{E}_{i+1,i} + e(i,i+1), \bar{E}_{i,i}, \max_{i < k < i+1} \bar{E}_{i,k-1} + \bar{E}_{k,j})$$

car $k \in \mathbb{N}$. donc k n'existe pas. De plus, $\bar{E}_{i+1,i} \neq 0$, $\bar{E}_{i,i} = 0$ (Q2).
donc il faut montrer que

$$\bar{E}_{i,i+1} = \max(e(i,i+1), 0)$$

$$= e(i,i+1)$$

Il se démontre trivialement.

Q.2. $E_{i,j}$ Montrer que la formule est vraie pour $E_{i,j}$ où $j > i+1$.

Faut montrer que :

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

Après la Q2. On a :

$$E_{i,j} = \begin{cases} E_{i+1,j-1} + e(i,j). & (\text{En déduire que } i < j \text{ sont couple} \\ & \text{ou pas, on} \cancel{\text{calcul la suite restant}} \text{ continu la vérification pour la} \\ & \text{suite restant.}) \\ E_{i,j-1} & (\text{En cas de } j \text{ n'est pas couple}) \\ E_{i,k-1} + E_{k,j} & (\text{En cas de } \cancel{(k,j)} \text{ d'il existe un} \\ & \text{couple } (k,j) \text{ tel que } k \in \{i+1, \dots, j-1\}) \end{cases}$$

qui contient tous cas possibles de la valeur ~~E_{i,j}~~. Pour obtenir une valeur maximal, il suffit de calculer les ~~3~~ valeurs de 3 cas, et prends celui qui est la plus grande. Spécialement pour le troisième cas, car k est un variable dans l'intervalle $\{i+1, j-1\}$, il faut calculer tous les valeurs et prends la plus grande. Donc $E_{i,j}$ est bien égale :

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

2. Par Q3.1. On a déjà :

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

On peut considérer que $E_{i,j-1}$ est un cas ~~particulier~~ ^{de} le troisième cas où $k=j$, c.ad. $E_{i,j-1} = E_{i,j-1} + E_{j,j}$ car

$$E_{j,j} = 0. \text{ Donc. } E_{i,j} = \max(E_{i+1,j-1} + e(i,j), \max_{i < k < j} E_{i,k-1} + E_{k,j})$$

Q4. def match (L, i, j):

if ((a[i] == "A" and L[j] == "T") or
((L[i] == "T") and L[j] == "A")) or
((L[i] == "C") and L[j] == "G")) or
((L[i] == "G") and L[j] == "C")) :

return 1

else :

return 0.

def findMaxInRange (a, i, j) :

L = []

for k in range (i+1, j+1)

L.append (tailleMaxRec (a, i, k-1) + tailleMaxRec
(a, k, j))

return max (L)

def tailleMaxRec (a, i, j) :

if (a == "" or i >= j) :

return 0

else :

return max (tailleMaxRec (a, i+1, j-1) + match (a, i, j),
findMaxInRange (a, i, j))

Q.5. $P(\pi)$: la fonction tailleMaxRec se termine et retourne la bonne valeur pour $i \in \{1, \dots, n\}$, $j \in \{1, \dots, n\}$.

$\forall p$ tel que $p = j - i$ et $p \geq 0$,

(B) $p = 0$. donc $i = j$. la fonction se termine avec la valeur 0, qui est bien la valeur de $E_{i,i}$.

I1. Supp $p = j - i$, pour tous $p < p_0$. $P(\pi)$ vraie. Montrons que $P(\pi)$ vraie pour $p = p_0$.
~~La fonction retourne~~ $\max(\text{tailleMaxRec}(a, i+1, j-1) + \text{match}(a, i, j))$, $\text{findMaxInRange}(a, i, j)$.

Par HR. $j-1-i+1 = j-i-2 < p_0$. Il se termine et retourne la bonne valeur. Ensuite, $\text{match}(a, i, j)$ se termine ~~trivialement~~ et retourne la bonne valeur trivialement.

Dans findMaxInRange , il y a $(j-i)$ boucle qui contient

$\text{tailleMaxRec}(a, i, k-1)$, où $k \in [i+1, j]$, se termine ~~par HR~~ et retourne la bonne valeur car $k-1-i < p_0$.

$\text{tailleMaxRec}(a, k, j)$ où $k \in [i+1, j]$ se termine et retourne la bonne valeur par HR car $j-k < p_0$.

Donc findMaxInRange se termine et retourne la bonne valeur.

Par Q3. on a $E_{i,j} = \max(E_{i+1,j-1} + e(i,j), \max_{i \leq k \leq j} E_{i,k-1}, E_{k,j})$

Donc $P(\pi)$ vraie pour p :

Q6-1: $U_0 = 1$
 $U_1 = 3$ (redouble à la main).
 $U_2 = 7$

Q6-2:

def tailleMaxRec(a, i, j):

if (a == " " or i >= j):
 return 0

else:

$1 + U_{p-2}$ (1)

return max (tailleMaxRec(a, i+1, j-1) + C(a, i, j),
 findMaxInRange(a, i, j)) (2)

def findMaxInRange(a, i, j):

$L = []$

[for k in range(i+1, j+1):

i.append (tailleMaxRec(a, i, k-1) +

tailleMaxRec(a, k, j)) (4)

$p(n)$ où $n = j - i$ $\Leftarrow U_p = U_{p-2} + 1 + 2 \sum_{i=0}^{p-1} U_i$ est bien le
 On suppose que $k \in [2, n]$, $p(k)$ vraie. \therefore tailleMaxRec(7)

Base: $n=2$ $U_2 = U_0 + 1 + 2 \sum_{i=0}^1 U_i = U_0 + 1 + 2(U_0 + U_1)$
 $= 7$ (vérifié)

Hérédité: $n = j - i$ donc $j = i + n$.

tailleMaxRec(a, i, j) tombe forcément sur "else" car $n \geq 2$. On

Un appel d'abord (1) une fois tailleMaxRec(a, i+1, j-1). Puis
 HR, il y a (3) U_{p-2} fois d'appels de tailleMaxRec à
 l'intérieur. maintenant, on calcule le nbre d'appel dans findMaxInRange(a,
 i, j). Dans la boucle for, k va de i+1 à j.

Le nbre d'appel en (3) U_{k-i} $\left\{ \begin{array}{l} k=i+1 \xrightarrow{\text{HR}} U_0 \\ k=i+2 \xrightarrow{\text{HR}} U_1 \\ \vdots \\ k=i+p \xrightarrow{\text{HR}} U_{p-1} \end{array} \right. = \sum_{i=0}^{p-1} U_i$

Le nbre d'appel

en (4) U_{i+p-k} $\left\{ \begin{array}{l} k=i+1 \xrightarrow{\text{HR}} U_{p-1} \\ k=i+2 \xrightarrow{\text{HR}} U_{p-2} \\ \vdots \\ k=i+p \xrightarrow{\text{HR}} U_0 \end{array} \right. = \sum_{i=0}^{p-1} U_i$

Puisque (1) (2) (3) (4), on a bien $p(n)$ vraie.

Conclusion: $U_p = U_{p-2} + 1 + 2 \sum_{i=0}^{p-1} U_i$ est bien le nbre d'appel total.

U6-3 : $U_0 \geq 2^n$.

$P(n)$: Pour tout $n \geq 2$, $U_n \geq 2^n$.

On suppose que $k \in [2, n-1]$. Pour tout k , $P(k)$ vérié.

Base : $n=2$, $U_2 = 7$, $2^2 = 4$ On a $U_2 \geq 2^2$ vrai.

Hérédité : D'après U6-2, on a

$$U_n = U_{n-2} + 1 + 2 \sum_{i=0}^{n-1} U_i \\ \geq U_{n-2} + 2 \cdot U_{n-1}$$

$$\text{D'après HR} \geq 2^{n-2} + 2 \cdot 2^{n-1} \\ = 2^{n-2} + 2^n \geq 2^n. P(n) \text{ vérifié}$$

Conclusion : En plus, pour $n=0$ et $n=1$ on a

$$U_0 = 1 \geq 2^0 \text{ et } U_1 = 3 \geq 2^1 = 2.$$

Donc $U_n \geq 2^n$ pour tout $n \in \mathbb{N}$.

U6-4 : D'après U6-3, la complexité de tailleMaxRei est en $\mathcal{O}(2^n)$.

U7 : Sa complexité est en $\Theta(n^3)$. Car dans la première boucle, elle fait $n-1$ itérations, la deuxième boucle coupe de $n-p$ fois itérations.

On se sert de $\max_{k \leq i+k-p} E_k, k-1 + E_k, i+p$, c'est équivalent à une boucle qui effectue $i+p$ itérations. Donc le nombre totale d'itération de l'algorithme est :

$$\sum_{p=1}^{n-1} \sum_{i=1}^{n-p} \sum_{k=i+1}^{i+p} = \sum_{p=1}^{n-1} \sum_{i=1}^{n-p} \frac{(i+1+i+p)p}{2}$$

$$= \sum_{p=1}^{n-1} \sum_{i=1}^{n-p} \frac{(2i+p+1)p}{2} = \sum_{p=1}^{n-1} \frac{((3+p)p + (2(n-p)+p+1)p)}{2} \cdot (n-p)$$

$$\begin{aligned}
 C &= \sum_{p=1}^{n-1} \cdot \sum_{i=1}^{n-p} \cdot \sum_{k=i+1}^{i+p} \\
 &= \sum_{p=1}^{n-1} \cdot \sum_{i=1}^{n-p} \cdot \frac{(2i+p+1)p}{2} \\
 &= \sum_{p=1}^{n-1} \left(\frac{(3+p)p}{2} + \frac{(2(n-p)+p+1)p}{2} \right) \cdot (n-p) \\
 &= \sum_{p=1}^{n-1} \frac{p(n+2)(n-p)}{2} \\
 &= \left(\frac{(n+2)(n-1)}{2} + \frac{(n-1)(n+2)}{2} \right) \cdot (n-1) \\
 &= \frac{(n+2)(n-1)(n-1)}{2} \in \Theta(n^3)
 \end{aligned}$$

Exercice 2 :

(Q1) Voir le code

(Q2) Voir le code

(Q3) Supposons que le temps raisonnable est environ 20s.

tailleMaxIter : chercher dans la séquence de taille 850
en 21.3 secondes

tailleMaxRec : chercher dans la séquence de taille 16
en 21.03 secondes

Voir le code.

(Q4). taille MaxRec :

n	$C_{\text{Rec}}(n)$	$\log_2(C_{\text{Rec}}(n))$
16	$\approx 20s$	$\log_2^{20} \approx 4$
15	$\approx 6s$	$\log_2^6 \approx 2$
14	$\approx 2s$	$\log_2^2 = 1$
:	:	:

Pour on en déduit que $\frac{\log_2(C_{\text{Rec}}(n+1))}{\log_2(C_{\text{Rec}}(n))} \approx 2$

Donc elle est linéaire.

taille MaxIter

n	$C_{\text{Iter}}(n)$	$C_{\text{Iter}}(n)/n^3$
1000	$\approx 36s$	$\approx 3.6 \times 10^{-8}$
900	$\approx 26s$	$\approx 3.57 \times 10^{-8}$
800	$\approx 17s$	$\approx 3.3 \times 10^{-8}$
700	$\approx 11s$	$\approx 3.2 \times 10^{-8}$
600	$\approx 7s$	$\approx 3.2 \times 10^{-8}$
500	$\approx 4s$	$\approx 3.2 \times 10^{-8}$
400	$\approx 2s$	$\approx 3.125 \times 10^{-8}$
:	:	:
:	:	:

Pour $\frac{C_{\text{Iter}}(n)}{n^3}$ est une fonction constante

quand n croît.