Sylvain, Raphaël
(111 124 564)

Conception et analyse d'algorithmes
IFT-3001

Travail 2
Question 2

Travail présenté à
Yanick Ouellet

Département d'informatique et de génie logiciel
Univesité Laval
Hiver 2019

Sylvain, Raphaël
111 124 564

Question 2

Description

Soit un menu R où pour un item x, il y a un nombre a_x d'ailes et b_x de pintes de bière pour un coût c_x d'associé.

Définition du tableau

Le tableau M contient le prix minimum.

Définition des dimensions du tableau

La première dimension va de 0 jusqu'au nombre d'items dans le menu.
La deuxième dimension va de 0 jusqu'au nombre d'ailes commandées.
La troisième dimension va de 0 jusqu'au nombre de bières commandées.

Définition d'une cellule

La cellule $M[i, j, k]$ contient le prix minimum pour une commande de j ailes et k pintes de bière dans un menu contenant les i premiers items du meen.

Elle contient l'infini si cette combinaison de j ailes et k pinte de bière est impossible.

Conditions initiales

La cellule $M[0, 0, 0] = 0$
La cellule $M[0, j, k] = \infty$ $(\forall j, k \in \mathbb{N} | j + k > 0)$

Récurrence

$$M[i, j, k] = \begin{cases} M[i-1, j, k] & \text{si } k - b_i < 0 \\ M[i-1, j, k] & \text{si } j - a_i < 0 \\ \min\left(M[i, j-a_i, k-b_i] + c_i, M[i-1, j, k]\right) & \text{sinon} \end{cases}$$

Analyse de la fonction `commande`

Le temps d'exécution de l'algorithme dépend du nombre d'items n dans le menu, le nombre d'ailes a à commander et le nombre de pintes de bière b commandées.

Cette méthode est composée de deux appels à des fonctions. Nous analyserons donc chacune des fonctions et nous pourrons donner notre réponse selon le maximum des deux.

Analyse de la fonction `genererTableau`

Le temps d'exécution de l'algorithme dépend du nombre d'items n dans le menu, le nombre d'ailes a à commander et le nombre de pintes de bière b commandées.

Nous devons séparer l'analyse en plusieurs blocs, puisqu'il y a des appels de fonction.

Bloc A

Le bloc A est tout ce qui se trouve au dessus des boucles `for`.

Nous avons deux appels de fonction ici. Un appel à `vector::size` et au constructeur de `Tableau::Tableau`

Ils sont tous les deux exécutés une seule fois.

L'appel à `vector::size` ce fait en tant constant $\Theta(1)$.

L'appel à `Tableau::Tableau` ce fait en temps linéaire sur le nombre de case du tableau.

Il n'y a pas de pire cas.

Dans notre cas, il se fait donc en tout temps à une complexité de

$$
\begin{aligned}
& C_{Tableau}(n, a, b) \\
\in\ & \Theta((n+1) * (a+1) * (b+1)) \\
=\ & \qquad \langle \text{ Étendre polynôme } \rangle \\
& \Theta(nab + na + nb + ab + n + a + b + 1) \\
=\ & \qquad \langle \text{ Règle du maximum } \rangle \\
& \Theta(nab)
\end{aligned}
$$

Donc, le bloc A a une complexité de $\Theta(\max(1 + nab)) = \Theta(nab)$

Bloc B

Le bloc B est constitué de la boucle `for` et de ces sous-boucles.

L'opération de base est la comparaison `i == 0`, car c'est l'opération exécuté le plus souvent et que tous les appels de fonction se font en temps constants, incluant `Tableau::at`.

Il n'y a pas de pire cas.

Le nombre de fois que cette opération peut être exécutée nous est donné par la sommation suivante :

$$
\begin{aligned}
& C^{B}_{genererTableau}(n, a, b) \\
= \quad & \langle \text{ Définition de la sommation selon l'algorithme } \rangle \\
& \sum_{i=0}^{n} \sum_{j=0}^{a} \sum_{k=0}^{b} 1 \\
= \quad & \langle \text{ Règle de sommation } \rangle \\
& \sum_{i=0}^{n} \sum_{j=0}^{a} ((b - 0 + 1) \cdot 1) \\
= \quad & \langle \text{ Simplification } \rangle \\
& \sum_{i=0}^{n} \sum_{j=0}^{a} (b + 1) \\
= \quad & \langle \text{ Règle de sommation } \rangle \\
& \sum_{i=0}^{n} ((a - 0 + 1) \cdot (b + 1)) \\
= \quad & \langle \text{ Simplification } \rangle \\
& \sum_{i=0}^{n} ((a + 1) \cdot (b + 1)) \\
= \quad & \langle \text{ Règle de sommation } \rangle \\
& (n - 0 + 1) \cdot ((a + 1) \cdot (b + 1)) \\
= \quad & \langle \text{ Simplification } \rangle \\
& (n + 1) \cdot (a + 1) \cdot (b + 1) \\
= \quad & \langle \text{ Simplification } \rangle \\
& nab + na + nb + ab + n + a + b + 1 \\
\in \quad & \langle \text{ Notation aymptotique } \rangle \\
& \Theta\left(nab + na + nb + ab + n + a + b + 1\right)
\end{aligned}
$$

$=$ $\langle$ Règle du maximum $\rangle$

$\Theta\left(nab\right)$

Conclusion

Puisque le bloc A a une complexité de $\Theta\left(nab\right)$ et le bloc B une complexité de $\Theta\left(nab\right)$, la fonction genererTableau a une complexité de $\Theta\left(nab\right)$, selon la règle du maximum.

$C_{genererTableau}\left(n,a,b\right) \in \Theta\left(nab\right)$

Analyse de la fonction `solutionnerTableau`

Le temps d'exécution de l'algorithme dépend du nombre d'items n dans le menu, le nombre d'ailes a à commander et le nombre de pintes de bière b commandées et du tableau M.

L'opération de base est la comparaison `i > 0` puisque c'est l'opération la plus exécuter et puisque les fonctions appelées sont tous de complexité constante ou constante amortie.

Il y a un meilleur et un pire cas, dépendamment du contenu du tableau M.

Meilleur cas

Le meilleur cas est quand l'élément à la position $M[n,a,b]$ a une valeur infinie.

Dans ce cas, l'opération est exécutée 0 fois et la complexité est définie par : $C_{best}^{solutionnerTableau}\left(n,a,b\right) = 0 \in \Theta\left(1\right)$

Pire cas

Le pire cas est quand un élément du menu donne des ailes gratuites, sans pintes de bières et qu'un autre donne des bières gratuites sans ailes de poulet.

Ceci fait en sorte de construire un tableau tel que l'on ne pourra bouger que d'une case dans une dimension à la fois.

La valeur de la complexité peut être donnée par la récurrence suivante :

$$M\left(n,a,b\right)=\begin{cases}1 & \text{si } n==0\\ 1+M\left(n,a,b-1\right) & \text{si } b>0\\ 1+M\left(n,a-1,0\right) & \text{si } b==0\wedge a>1\\ 1+M\left(n-1,0,0\right) & \text{si } b==0\wedge a==0\wedge n>1\end{cases}$$

On peut la résoudre ainsi :

$$\begin{aligned}
& M\left(n,a,b\right)\\
= & \qquad \langle \text{ Récurrence pour réduire } b \text{ (cas 2) } \rangle\\
& 1+M\left(n,a,b-1\right)\\
= & \ 1+1+M\left(n,a,b-1-1\right)\\
= & \ 1+1+1+M\left(n,a,b-1-1-1\right)\\
= & \ k+M\left(n,a,b-k\right)\\
= & \ b+M\left(n,a,b-b\right)\\
= & \ b+M\left(n,a,0\right)\\
= & \qquad \langle \text{ Récurrence pour réduire } a \text{ (cas 3) } \rangle\\
& b+1+M\left(n,a-1,0\right)\\
= & \ b+1+1+M\left(n,a-1-1,0\right)\\
= & \ b+1+1+1+M\left(n,a-1-1-1,0\right)\\
= & \ b+j+M\left(n,a-j,0\right)\\
= & \ b+a+M\left(n,a-a,0\right)\\
= & \ b+a+M\left(n,0,0\right)\\
= & \qquad \langle \text{ Récurrence pour réduire } n \text{ (cas 4) } \rangle\\
& b+a+1+M\left(n-1,0,0\right)\\
= & \ b+a+1+1+M\left(n-1-1,0,0\right)\\
= & \ b+a+1+1+1+M\left(n-1-1-1,0,0\right)\\
= & \ b+a+i+M\left(n-i,0,0\right)\\
= & \ b+a+n+M\left(n-n,0,0\right)\\
= & \ b+a+n+M\left(0,0,0\right)\\
= & \qquad \langle \text{ Cas de base (cas 1) } \rangle
\end{aligned}$$

$$
\begin{aligned}
& b+a+n+1 \\
\in\ & \Theta\left(b+a+n+1\right) \\
=\ & \Theta\left(b+a+n\right)
\end{aligned}
$$

Donc, en pire cas, l'algorithme de solution s'exécute en $\Theta\left(b+a+n\right)$

Retour sur la fonction `commande`

Puisque la complexité fonction `solutionnerTableau` est au plus de $\Theta\left(n+a+b\right)$, alors que la complexité de la fonction `genererTableau` est en $\Theta\left(nab\right)$,

Puisque $\Theta\left(nab\right) > \Theta\left(n+a+b\right) \forall n, a, b | n > 1 \wedge a > 1 \wedge b > 1$,

Alors, la complexité de `commande` est en $\Theta\left(nab\right)$