Sylvain, Raphaël
(111 124 564)

Conception et analyse d'algorithmes
IFT-3001

Travail 2
Question 3

Travail présenté à
Yanick Ouellet

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

Sylvain, Raphaël
111 124 564

Question 3

Description

L'algorithme commence par utiliser les vecteurs entrants pour former un vecteur d'objet. Il permettra de manipuler plus facilement les éléments.

Il tri ensuite ces objets selon leur jour d'annonce.

Une boucle s'effectue par jour. À chaque tour de boucle, on ajoute les éléments de cette journée dans un tas. Ce tas est trié par date d'échéance.

Avant de terminer la boucle, on prend l'élément avec l'échéance la plus courte du tas. On définit cet objet comme étant le travail à effectuer cette journée.

Analyse

Analyse de la fonction `Travail::GetJourAnnonce()`

Cette fonction n'a qu'une seule opération, qui est l'accès à la valeur de variable. Il s'effectue donc en $\Theta(1)$

Analyse de la fonction `ordonnancement::f2`

Cette fonction n'a qu'une seule opération, qui est l'appel à la fonction `Travail::GetJourAnnonce()`. Il s'effectue donc en $\Theta(1)$ comme cette fonction.

Analyse de la fonction `triParDenombrement`

Cette fonction prend un vecteur A, deux entiers délimiteurs (*lbound* et *ubound*) ainsi qu'une fonction de traduction f. La taille de l'instance est définie par $|A|$ la cardinalité du vecteur entrant et $n = ubound - lbound$ la différence entre les deux entiers délimiteurs.

Par assertion, nous avons $|A| = n$.

Nous devrons effectuer une analyse en plusieurs blocs.

Bloc A

Le bloc A est composé de l'appel à `resize` sur le nouveau vecteur D.
Selon la documentation, cet appel a une complexité de $\Theta(n)$.

Bloc B

Le bloc B est composé de la boucle d'initialisation.
La complexité de cette boucle peut être donnée, en tout temps, par la sommation suivante :
$C^B(n) = \sum_{i=0}^{n-1} 1 = n \in \Theta(n)$

Bloc C

Le bloc C est composé de la boucle de fréquence.
La complexité de cet algorithme dépend de la complexité de la fonction f.
La complexité de cette boucle peut être donnée, en tout temps, par la sommation suivante :

$$\begin{aligned} & C^C(n) \\ = & \sum_{i=0}^{n-1} C_f(n) \\ = & C_f(n) \sum_{i=0}^{n-1} 1 \\ = & C_f(n)\, n \\ \in & \Theta(C_f(n)\, n) \end{aligned}$$

Bloc D

Le bloc D est composé de la boucle de distribution.
La complexité de cette boucle peut être donnée, en tout temps, par la sommation suivante :
$C^D(n) = \sum_{i=0}^{n-1} 1 = n \in \Theta(n)$

Bloc E

Le bloc E est composé de l'appel à `resize` sur le vecteur de solution S.

Selon la documentation, cet appel a une complexité de $\Theta(n)$.

Bloc F

Le bloc F est composé de la boucle de solution.

La complexité de cet algorithme dépend de la complexité de la fonction f.

La complexité de cette boucle peut être donnée, en tout temps, par la sommation suivante :

$$\begin{aligned} & C^F(n) \\ = & \sum_{i=0}^{n-1} C_f(n) \\ = & C_f(n) \sum_{i=0}^{n-1} 1 \\ = & C_f(n)\, n \\ \in & \Theta(C_f(n)\, n) \end{aligned}$$

Conclusion de l'analyse de la fonction `triParDenombrement`

La complexité de la fonction `triParDenombrement` peut être donnée par le maximum des complexités des blocs précédents.

$$\begin{aligned} & C(n) \\ \in & \max\left(C^A, C^B, C^C, C^D, C^E, C^F\right) \\ = & \max\left(\Theta(n), \Theta(n), \Theta(C_f(n)\, n), \Theta(n), \Theta(n), \Theta(C_f(n)\, n)\right) \\ = & \Theta(C_f(n)\, n) \end{aligned}$$

En particulier, considérons le cas où $C_f(n) \in \Theta(1)$, la fonction `triParDenombrement` s'exécutera donc en $\Theta(n)$.

Analyse de la fonction `ordonnancement`

Cette fonction prendre deux vecteurs entrants, A et B, et produit un vecteur sortant C. Par assertion, nous avons que $|A| = |B| = |C|$.

La taille de l'instance est donc $n = |A|$.

Analysons en pire cas cette fonction.

Il faudra séparer l'analyse en plusieurs blocs.

Bloc A

Le bloc A est composé de la boucle d'initialisation du vecteur d'objet de type `Travail`.

Nous pouvons prendre comme opération baromètre l'appel à la fonction `std::vector::emplace_back`.

Cette opération est appelée la plus souvent à une constante près. Toutes les autres opérations s'exécutent en temps constant.

Selon la documentation, un appel à cette méthode a une complexité constante amortie.

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

$C^A(n) = \sum_{i=0}^{n-1} 1 = n \in \Theta(n)$

Bloc B

Le bloc B est composé de l'appel à `triParDenombrement`.

Selon l'analyse précédente, cette fonction s'exécute en $\Theta(n)$ lors que la complexité de la fonction en paramètre est constante. Ce qui est le cas, puis que nous lui envoyons `ordonnancement::f2` qui est en $\Theta(1)$ selon$<$ l'analyse précédente.

Donc $C^B \in \Theta(n)$

Bloc C

Le bloc C est composé de la boucle `for` contant une boucle `while`.

Il est important de noter que dans l'exécution de la méthode `ordonnancement`, le corps du `while` se fera au maximum un nombre n de fois. On peut conclure ceci, puis que nous avons une condition dans le `while` que `positionJourAnnonce < nb_travaux` et dans le corps nous avons un `positionJourAnnonce++`, alors que l'initialisation de cette variable se fait à l'extérieur de toutes les boucles.

Bloc C.while

Dans la boucle `while`, il y a plusieurs appels de fonctions :

1. `std::vector::push_back`
   $\Rightarrow \Theta(1)$

2. `std::make_tuple`
   $\Rightarrow \Theta(1)$
3. `std::vector::at`
   $\Rightarrow \Theta(1)$
4. `Travail::GetJourDu`
   $\Rightarrow \Theta(1)$
5. `std::push_heap`
   $\Rightarrow \log(n)$
6. `std::vector::begin`
   $\Rightarrow \Theta(1)$
7. `std::vector::end`
   $\Rightarrow \Theta(1)$

La complexité du bloc C.while peut donc être donnée par la sommation suivante :

$$\begin{aligned} & C^{C.while}(n) \\ = & \sum_{i=0}^{n-1} (\log(n)) \\ = & \log(n) \sum_{i=0}^{n-1} 1 \\ = & n \cdot \log(n) \\ \in & \Theta(n \log(n)) \end{aligned}$$

Bloc C.for

Pour le reste de la boucle `for`, il y a plusieurs appels de fonctions :

1. `std::pop_head`
   $\Rightarrow 2\log(n)$
2. `std::vector::begin`
   $\Rightarrow \Theta(1)$
3. `std::vector::end`
   $\Rightarrow \Theta(1)$
4. `std::vector::at`
   $\Rightarrow \Theta(1)$
5. `std::get<1>`
   $\Rightarrow \Theta(1)$

6. `std::vector::back`
   $\Rightarrow \Theta(1)$
7. `std::vector::pop_back`
   $\Rightarrow \Theta(1)$
8. `Travail::GetNumero`
   $\Rightarrow \Theta(1)$

La complexité du bloc C.for peut donc être donnée par la sommation suivante :

$$\begin{aligned} & C^{C.for}(n) \\ = & \sum_{i=0}^{n-1} (2\log(n)) \\ = & \ 2\log(n) \sum_{i=0}^{n-1} 1 \\ = & \ 2n\log(n) \\ \in & \ \Theta(n\log(n)) \end{aligned}$$

Conclusion Bloc C

Le bloc C peut donc s'exécuter en $\Theta(\max(n\log(n), n\log(n))) = \Theta(n\log(n))$

Conclusion de l'analyse de `ordonnancement`

En prenant le maximum de chaque analyse, nous avons que
$C(n) \in \Theta(\max(n, n, n\log n)) = \Theta(n\log n)$