

Sylvain, Raphaël
(111 124 564)

Conception et analyse d'algorithmes
IFT-3001

Travail 1

Travail présenté à
Yanick Ouellet

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

Question 1

Description textuelle

L'algorithme se base sur ces observations suivantes :

1. Zéro est l'élément absorbant de la multiplication ;
2. Le résultat est un produit d'éléments de vecteur (une multiplication) ;
3. Lorsque deux zéro sont dans le vecteur, le produit sera toujours zéro ;
4. Nous pouvons émuler la non-production d'un élément, en divisant le produit par celui-ci, pourvu qu'il ne soit pas zéro.

L'algorithme débute donc en initialisant une variable pour conserver le produit total (initialiser à 1), ainsi que deux variables "indicatives" (flags). Une pour conserver l'indice d'un élément absorbant et l'autre pour indiquer s'il y a plus d'un élément absorbant.

Puis, l'algorithme parcourt tous les éléments de notre vecteur entrant. S'il trouve pour la première fois un zéro, il affecte l'indice où cet élément se trouve dans le vecteur. S'il en trouve un deuxième il affecte la valeur booléenne indiquant plusieurs zéros à vrai.

Cette boucle terminée, il entre dans une des trois embranchements dépendants de la valeur des variables indicatrices.

Premièrement, si la variable booléenne indiquant que plusieurs zéros ont été trouvés dans le vecteur entrant, alors nous initialisons le vecteur résultat pour qu'il ne contienne que des zéros, puisque nous sommes garantis qu'il y aura une multiplication par zéro pour tous ces éléments.

Sinon, si la variable indiquant l'indice d'un zéro a été définie, alors tous les éléments du vecteur résultat seront à zéro, à l'exception de l'élément à cet indice qui aura le résultat total du produit.

Finalement, si aucune des deux variables indicatrices n'a été définie, alors chaque élément du vecteur résultat sera égale au produit, divisé par l'élément à l'indice correspondant du vecteur entrant.

Analyse de l'algorithme

Puisque l'algorithme est trop complexe pour être analysé à l'aide d'une seule opératoire de base, nous allons séparer l'algorithme en quatre parties

et analyser chacune de ces parties individuellement, puis conclure sur l'entièreté de l'algorithme à partir de ces sous-analyses.

La taille de l'instance est n , soit la cardinalité du vecteur entrant A .

Bloc A

Le bloc A est composé de la première boucle **for** et des déclarations lui précédents.

Nous pouvons choisir comme opération de base le `!productIsAlwaysZero`, car c'est la comparaison qui est effectuée le plus souvent, c'est-à-dire à chaque itération. De plus, la fonction `A.at(i)` à l'intérieur de la boucle s'effectue en temps constant.

Le nombre de fois que cette opération de base peut s'exécuter dépendant du contenu du vecteur entrant ainsi que de sa cardinalité.

Meilleur cas

En meilleur cas, les deux premiers éléments du vecteur sont 0. Si la boucle découvre deux 0, elle se courtcircuite. Nous avons donc que :

$$\begin{aligned} & C_{best}^A(n) = 3 \\ \Rightarrow & \quad \langle \text{Donc} \rangle \\ & 1 \leq C_{best}^A(n) = 3 \leq 5 \\ \in & \quad \langle \text{Définition de } \Theta \text{ avec } c_1 = 1, c_2 = 5 \text{ et } n_0 = 0 \rangle \\ & \Theta(1) \end{aligned}$$

Pire cas

En pire cas, nous devons parcourir l'ensemble des éléments du vecteur entrant. À chacune des boucles, nous devons effectuer une seule comparaison, puis nous effectuons une dernière comparaison avant de sortir. Nous avons donc

$$\begin{aligned} & C_{worst}^A(n) \\ = & \quad \langle \text{Définition mathématique} \rangle \\ & \sum_0^{n-1} 1 + 1 \end{aligned}$$

$$\begin{aligned}
&= \langle \text{Première règle des sommations} \rangle \\
&\quad (((n-1) - 0 + 1) \times 1) + 1 \\
&= \langle \text{Simplification} \rangle \\
&\quad n + 1 \\
&\Rightarrow \langle \text{Donc} \rangle \\
&\quad \frac{1}{2}n \leq C_{worst}^A(n) = n + 1 \leq 2n \quad \forall n \geq 1 \\
&\in \langle \text{Définition de } \Theta \text{ avec } c_1 = \frac{1}{2}, c_2 = 2 \text{ et } n_0 = 1 \rangle \\
&\quad \Theta(n)
\end{aligned}$$

Bloc B

Le bloc B est composé uniquement du premier bloc du if, soit :

```
B.assign(VECTOR_SIZE, 0);
```

Ce bloc est exécuté uniquement si deux zéros sont trouvés dans le vecteur entrant.

L'opération à utiliser est donc l'appel à la fonction `std::vector::assign`.

La complexité de la fonction `std::vector::assign` est linéaire sur la valeur du premier paramètre. Dans notre cas, ce paramètre est équivalent à la cardinalité du vecteur entrant. Donc

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

Bloc C

Le bloc C est composé uniquement du deuxième bloc du if, soit :

```
B.assign(VECTOR_SIZE, 0);
B.at(indexOfAbsorbingElement) = totalProductExceptZeroes;
```

Ce bloc est exécuté uniquement si un seul zéro a été trouvé dans le vecteur entrant.

L'opération à utiliser est donc l'appel à la fonction `std::vector::assign`, puisque celle-ci a une complexité linéaire sur la valeur du premier paramètre, alors que la fonction `std::vector::at` est constante. Donc

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