eBook Gratuit

APPRENEZ big-o

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#big-o

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: big-o

It is an unofficial and free big-o ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official big-o.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec big-o

Remarques

Cette section fournit une vue d'ensemble de ce qu'est big-o et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans big-o, et établir un lien avec les sujets connexes. La documentation de big-o étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Qu'est-ce que la notation Big-O?

La notation Big-O est une notation utilisée pour parler des taux de croissance à long terme des fonctions. Il est souvent utilisé dans l'analyse d'algorithmes pour parler de l'exécution d'un algorithme ou de concepts connexes tels que la complexité de l'espace.

En usage courant, la notation big-O est utilisée pour parler de la façon dont le runtime d'un algorithme évolue en fonction de la taille de l'entrée. Par exemple, nous dirions que le tri par sélection a un temps d'exécution de $O(n^2)$ car le temps d'exécution augmente de façon quadratique en fonction de la taille du tableau à trier. C'est-à-dire que si vous doublez la taille de l'entrée, le temps d'exécution du tri de la sélection devrait à peu près doubler. Lors de l'utilisation de la notation big-O, la convention consiste à supprimer les coefficients et à ignorer les termes d'ordre inférieur. Par exemple, bien que techniquement, il ne soit pas faux de dire que la recherche binaire s'exécute dans le temps $O(2 \log_2 n + 17)$, elle est considérée comme de mauvaise qualité

Formellement, la notation big-O est utilisée pour quantifier le comportement à long terme d'une fonction. On dit que $f(n) = O(g)$ (parfois noté $f(n) \in O(g(n))$ dans certaines sources) s'il y a des constantes fixes c et n_0 telles que $f(n) \leq c \cdot g(n)$ pour tout $n \geq n_0$. Cette définition formelle explique pourquoi nous ne nous soucions pas des termes d'ordre inférieur (ils peuvent être englobés en augmentant c et en augmentant n_0) et des facteurs constants (le terme c les absorbe). Cette définition formelle est souvent utilisée dans l'analyse rigoureuse des algorithmes, mais elle est rarement utilisée familièrement.

Calculer Big-O pour votre code

Une façon de calculer la valeur Big-O d'une procédure que vous avez écrite consiste à déterminer quelle ligne de code s'exécute le plus souvent dans votre fonction, compte tenu de votre taille d'entrée *n* . Une fois que vous avez ce nombre, éliminez tous les termes, sauf les plus rapides, et éliminez les coefficients - c'est la notation Big-O de votre fonction.

Par exemple, dans cette fonction, chaque ligne s'exécute exactement une fois et pour la même

durée, quelle que soit la taille de `a` :

```
int first(int[] a){
   printf("Returning the first element of a");
   return a[0];
}
```

La fonction elle-même peut prendre 1 milliseconde ((1 ms) * n^0) ou 100 millisecondes ((100 ms) * n^0) - la valeur exacte dépendra de la puissance de l'ordinateur concerné et de ce que `printf()` imprime. Mais comme ces facteurs ne changent pas avec la taille de `a` , ils importent peu pour les calculs de Big-O - ce sont des coefficients constants, que nous supprimons. Par conséquent, cette fonction a une valeur Big-O de *O (1)* .

Dans cette fonction, la ligne 3 (`sum += a[i];`) s'exécute une fois pour chaque élément dans `a` , pour un total de `a.length` (ou *n*) fois:

```
int sum(int[] a){
   int sum = 0;
   for (int i = 0; i < a.length; i++){
        sum += a[i];
   }
   return sum;
}
```

Les instructions `i++` et `i < a.length` s'exécutent également *n* fois - nous aurions pu choisir ces lignes, mais nous n'avons pas à le faire. En outre, `int sum = 0;` , `int i = 0` et `return sum;` chaque exécution une fois, ce qui est moins de `n` fois - nous ignorons ces lignes. Peu importe combien de temps `sum += a[i]` prend pour fonctionner - c'est un coefficient qui dépend de la puissance de l'ordinateur - nous supprimons donc ce coefficient. Par conséquent, cette fonction est *O (n)* .

S'il existe plusieurs chemins de code, big-O est généralement calculé à partir du pire des cas. Par exemple, même si cette fonction peut quitter immédiatement peu importe la taille de `a` (si `a[0]` vaut `0`), il existe toujours un cas qui fait que la ligne 6 exécute `a.length` fois.

```
int product(int[] a){
    int product = 0;
    for (int i = 0; i < a.length; i++){
        if (a[i] == 0)
            return 0;
        else
            product *= a[i];
    }
    return product;
}
```

Lire Commencer avec big-o en ligne: https://riptutorial.com/fr/big-o/topic/1340/commencer-avec-big-o

Chapitre 2: Calcul de Big-O

Examples

O (n) fonctions.

Les fonctions qui ont la valeur `O(n)` augmentent linéairement le nombre d'opérations, à mesure que l'entrée devient très importante. Un exemple simple d'une fonction qui est O (n) serait l'algorithme de recherche linéaire, qui s'exécute une fois pour la taille de l'entrée.

Le pseudo-code suivant serait `O(n)` , car il sera toujours limité par la taille d'entrée, car l'algorithme ne fonctionnera jamais plus de temps que la taille d'entrée.

```
function LinearSearch (SearchArray, SearchFor)
    for each element in SearchArray
        if the element is SearchFor
            return the index of element
```

Lire Calcul de Big-O en ligne: https://riptutorial.com/fr/big-o/topic/4307/calcul-de-big-o

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec big-o	Community, intboolstring, templatetypedef, TheHansinator
2	Calcul de Big-O	intboolstring