Rapport de projet

2019-03-08

Licence informatique

Comparaison de tri en complexité linéaire

Par :
MAJDOUL Kaoutar
CHAMROUK Laila
MESSAOUDI Soukaina
LAAFOU Ayoub

Sous la direction de :
M. GIROUDEAU

Remerciement

Introduction

Dans le cadre du projet se déroulant [à completer]

Table des matières

1 Tri dénombrement

1.1 Description

Le tri dénombrement est un algorithme de tri reposant sur le principe de construction d'un histogramme des données puis le balayage de celui-ci de façon croissante afin d'obtenir les données triées. De ce fait, ici, plusieurs éléments identiques seront représentés par un unique élément quantifié. Ce tri ne peut donc pas être appliqué sur des structures complexes, et il convient exclusivement aux données constituées de nombres entiers compris entre une borne minimale et une borne maximale connues. Dans un souci d'efficacité, celles-ci doivent être relativement proches l'une de l'autre, ainsi que le nombre d'éléments doit être relativement grand.

Tout d'abord, on parcourt le tableau et on compte le nombre de fois que chaque élément apparaît. Une fois qu'on a le tableau des effectifs E, avec E[i] le nombre de fois où i apparaît dans le tableau, on peut le parcourir dans le sens croissant et placer dans le tableau trié E[i] fois l'élément i avec i allant de l'élément minimum du tableau jusqu'à l'élément maximum.

Ainsi, l'algorithme donne le psuedo-code suivant :

```
Chercher l'élément maximum du tableau
Créer un tableau E de taille Max + 1, initialisé à 0

Pour chaque élément du tableau
    Incrémenter E[Tableau[i]]

Pour chaque élément de E
    Recopier E[i] fois le nombre i dans le tableau trié
```

1.2 Exemple

Afin d'illustrer cet algorithme nous allons dérouler un exemple de son utilisation. Voici un tableau d'entier que l'on souhaite trier dans l'ordre croissant en utilisant le tri par dénombrement : 8, 6, 1, 3, 8, 1, 1. La première étape est de créer notre tableau des effectifs H, la deuxième est simplement de le parcourir et de recopier dans le tableau trié les valeurs :

i	H[i]	Action	Tableau trié
0	0	on ne fait rien	
1	3	on ajoute trois fois 1	1 1 1
2	0	on ne fait rien	1 1 1
3	1	on ajoute une fois 3	1 1 1 3
4	0	on ne fait rien	1 1 1 3
5	0	on ne fait rien	1 1 1 3
6	1	on ajoute une fois 6	1 1 1 3 6
7	0	on ne fait rien	1 1 1 3 6
8	2	on ajoute deux fois 8	1 1 1 3 6 8 8

On a atteint le maximum de notre tableau des effectifs H, notre tableau est donc trié : 1, 1, 1, 3, 6, 8, 8.

1.3 Complexité

Notons n la taille de la liste considérée. Le calcul du minimum et du maximum coûte un parcourt de la liste, soit (n) et l'allocation de l'histogramme (m) où m désigne l'étendue de la liste L. Notons H l'histogramme, la construction de celui-ci demande un nouveau parcourt de la liste et à donc un coût de (n). La reconstruction de la liste dans la boucle est facile à analyser globalement, on vide chacun des H[i] et on sait que :

$$\sum_{i=1}^{maxL-minL+1} H[i] = n$$

donc la reconstruction de la liste coûte elle aussi (n).

Les seules opérations que l'on effectue dans notre fonction se font en temps linéaire. L'initialisation du tableau des effectifs se fait en O(n) (avec n la taille de la liste en entrée), et la copie des éléments dans notre tableau trié en O(m) (avec m correspondant à max).

La complexité de l'algorithme de tri par dénombrement T pour une liste L de taille n et d'étendue m est :

$$\mathrm{T(n,m)=(n+m)}$$

Cet algorithme n'est donc linéaire que si la taille m de l'histogramme reste raisonnable O(n) au regard de la taille de la liste, autrement dit quand la dispersion est faible et dans ce cas T(n,m)=(n). C'est la dispersion qui conditionnera l'utilisation ou non de ce tri.

La complexité finale de notre algorithme est donc O(n+m), soit une complexité en temps linéaire.

1.4 Avantages et inconvénients

Points positifs

- Très efficace si les n nombres à trier son petits (¡= n).

- Ne fait aucune comparaison.

- Peut optimiser d'autre algorithme de tri, tel que le tri par base.

- Stable puisqu'il préserve l'ordre initial dans le tableau height

Points négatifs

- S'exécute uniquement sur des nombres entiers.

- La complexité en mémoire est mauvaise car l'algorithme peut prendre très vite beaucoup de place. le tableau des effectifs E a pour taille
la valeur maximale du tableau, or si cette valeur est très grande,
le tableau H prendra énormément de place en mémoire.

- Si les valeurs du tableau sont éloignées entre elles, alors beaucoup d'espace mémoire restera inutilisé.

Cet algorithme est donc très efficace mais il faut savoir faire un choix entre rapidité et stockage, en plus de ne pouvoir l'utiliser que sur des nombre entiers.

2 Conception

3 Résultats

4 Organisation

5 Conclusion