Complexité des algorithmes

Compétences

- Prévoir la complexité théorique d'un algorithme.
- Étudier la complexité temporelle expérimentale d'un algorithme

1 Niveaux de couleurs

Un pixel possède une valeur qui peut-être un scalaire (un nombre) et représenter un niveau de gris, ou un vecteur souvent à 3 composantes représentant chacune une couleur ayant des niveaux d'intensité $[0,255]$.

Pour refaire les niveaux d'une image, on peut utiliser la méthode suivante :

— pour chaque couleur, on cherche la valeur minimum v_{min} et la valeur maximum v_{max} apparaissant parmi les pixels de l'image, on obtient la gamme de cette couleur : l'intervalle $[v_{min}, v_{max}]$

— puis, on réaffecte à chaque pixel une nouvelle valeur v_N sur cette couleur, calculée en projetant l'intervalle de départ sur l'ensemble des valeurs disponibles $[0,255]$. On obtient cette valeur à partir de la valeur initiale v_I en calculant :

$$v_N = \frac{255(v_I - v_{min})}{v_{max} - v_{min}}$$

1. Écrivez une fonction `minimumR(img)` qui calcule la valeur minimale des coefficients de rouge sur toute l'image. Écrivez de même une fonction `maximumR(img)`.
2. Modifiez les fonctions pour qu'elles renvoient les minimum et maximum de chaque couleur.
3. Écrivez le programme qui refait les niveaux de couleurs de l'image avec la méthode précédente. Vous pourrez tester votre programme en déchiffrant l'image `hell.png`.

2 ModifiedBinarySearch

2.1 Introduction

L'algorithme modifié de dichotomie est un algorithme présenté par [Chadha et al.(2014)Chadha, Misal, and Mokashi] dans un article de recherche. Cet algorithme optimise le cas le plus défavorable de l'algorithme de dichotomie en comparant l'élément recherché avec le premier et le dernier élément du tableau, ainsi que l'élément central. Il vérifie également que l'élément recherché, un nombre dans notre cas, appartient à la plage de nombres présents dans le tableau à chaque itération, ce qui permet de réduire le temps pris par les pires cas de l'algorithme de recherche dichotomique. Dans ce TP nous allons essayer de suivre les différentes étapes de cet article pour mieux comprendre les améliorations approtées par cet algorithme.

2.2 Concept of Modified Binary Search

Le concept de recherche dichotomique utilise uniquement l'élément du milieu pour vérifier s'il correspond à l'élément recherché noté x. Pour cette raison, si x est présent à la première position, la recherche prend plus de temps, ce qui en fait le pire cas de l'algorithme.

Pour améliorer les performances de la recherche dichotomique, l'algorithme modifié de la recherche dichotomique ne vérifie pas seulement si x est présent pour l'index du milieu mais il vérifie également s'il est présent à la première et dernière position du tableau intermédiaire pour chaque itération. Cet algorithme optimise aussi le cas où x n'est pas dans le tableau et distingue dans cette situation deux cas, celui où la valeur de x est en dehors de la plage des valeurs du tableau et celui où la valeur de x est comprise entre le plus petit et le plus grand élément du tableau.

2.3 Algorithms

A Binary Search

Écrire la fonction de recherche dichotomique `binarySearch(x, tab, n)` qui prend en arguments la valeur recherchée, un tableau d'entiers et le nombre de valeurs présentes dans le tableau. Cette fonction renvoie la position dans le tableau de la valeur recherchée. Si cette valeur ne se trouve pas le tableau la fonction renvoie -1.

B Modified Binary Search

À partir des informations données dans 2.2 écrire un fonction `modifiedBinarySearch(x, tab, n)` de l'algorithme modifié de recherche dichotomique. Cette fonction prend les mêmes arguments et renvoie les mêmes informations que la fonction `binarySearch`

2.4 Performance Analysis

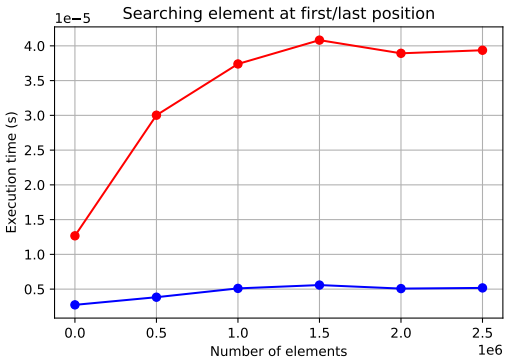
Soient $x \in \mathbb{N}$ la valeur recherchée et `tab[]` un tableau de valeurs entières. Dans cette partie il s'agit d'étudier le nombre de coups nécessaires pour atteindre x avec l'algorithme classique de dichotomie et l'algorithme modifié. Pour notre étude nous travaillerons avec le tableau d'entiers suivant :

`tab[]`

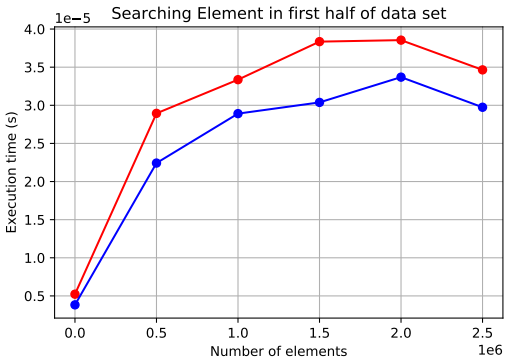
1	4	21	53	64	98	102	171	188	205
0	1	2	3	4	5	6	7	8	9

1. Appliquez les deux algorithmes à une valeur du tableau et vérifiez que l'index renvoyé est le bon.
2. Faire de même avec une valeur entière qui n'est pas dans le tableau.
3. Écrire un script python permettant de comparer la complexité temporelle expérimentale des deux algorithmes suivant la place de l'élément recherché (Fig1 et Fig2). Pour manipuler le temps avec Python disposez du module `time`.

```python
import time
...
start = time.perf_counter()
# Ici le code dont vous voulez mesurer
# le temps d'exécution
interval = time.perf_counter() - start
```

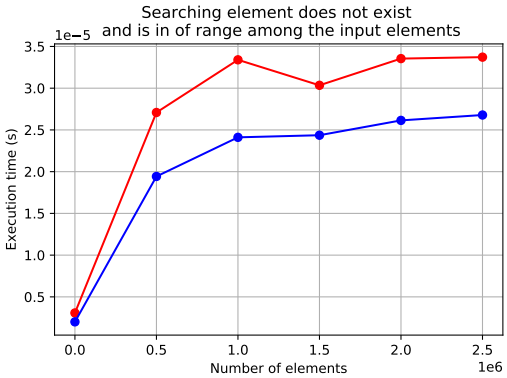


(a) First element

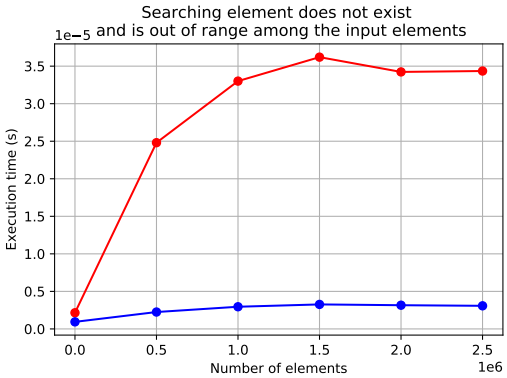


(b) First half

FIGURE 1 – Analyse des performances de l'algorithme de dichotomie modifié quand l'élément x occupe la première place ou une position quelconque dans la première moitié du tableau



(a) Element not in the array but in range of data set



(b) Element not in the array and out of range of data set

FIGURE 2 – Analyse des performances de l'algorithme de dichotomie modifié quand l'élément x n'est pas dans le tableau

Références

[Chadha et al.(2014)Chadha, Misal, and Mokashi] A. R. Chadha, R. Misal, and T. Mokashi, "Modified binary search algorithm," *CoRR*, vol. abs/1406.1677, 2014. [Online]. Available : http://arxiv.org/abs/1406.1677