Étude d'algorithmes de tris et de rangements classiques

TD-P#3 – Remise à niveau pour l'informatique

1 Introduction

Présentation des deux algorithmes de tris classiques tri par insersertion et tri sélection. Étude de leur perfomance et complexité. Puis étude de deux algorithmes de rangement : le drapeau hollandais et Noir & Blanc. Etude de leur perfomance et complexité. Présenter les codes, résultats, discussions et conclusions dans un rapport LaTeX, Python note book ou autre.

2 Algorithmes de tri standards

Le principe est de trier dans l'ordre croissant un tableau T de n éléments. Les trois algorithmes trient les tableaux sur place.

2.1 Le tri par insertion

Ce tri est naturellement utilisé pour ranger des cartes à jouer. On parcourt le tableau de droite à gauche et on range chaque élément à sa place parmis les éléments précédents déjà rangés. Plus précisément, à l'étape i : on suppose le sous-tableau T[1:i-1] est déjà trié, on insère l'élément $T[i]$ à la bonne place dans ce tableau en comparant les éléments en partant de la fin (de $i-1$ à 1).

2.2 Le tri sélection

L'idée générale est

- chercher le plus petit élément du tableau et de l'échanger avec l'élément d'indice 1;
- chercher le deuxième plus petit élément et l'échanger avec l'élément d'indice 2;
- itérer le procédé jusqu'à la fin du tri du tableau

2.3 Le tri à bulles

Il consiste à comparer répétitivement les éléments consécutifs d'un tableau, et à les permuter lorsqu'ils sont mal triés. Il doit son nom au fait qu'il déplace rapidement les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface d'un liquide. L'idée générale est de

- Faire remonter les éléments les plus grands en échangeant les éléments consécutifs mal arrangés vers la fin du tableau
- effectuer autant de passes que nécessaire pour que tous les éléments soient à la bonne place
- le tri est fini si aucun élément n'a été échangé lors d'un passage.

Questions 1.

1. Écrire une fonction `randtable(n,a,b)` qui retourne une liste de n entiers tirés aléatoirement entre a et b.
2. Écrire trois fonctions `insertion(T), bubble(T), selection(T)` correspondant aux tris présentés ici. Ces fonctions trient les tableaux sur place et retournent le nombre de comparaisons d'éléments du tableau effectuées.
3. On souhaite comparer le nombre moyen de comparaisons effectuées par chaque algorithme pour des tableaux de tailles $50, 100, 150, \ldots, 500$. Pour chaque taille de tableau n, calculer le nombre moyen de comparaisons pour 100 tableaux aléatoires ayant des valeurs comprises entre 1 et n.

4. Tracer les courbes correspondant au nombre de comparaisons moyen en fonction des différentes tailles considérées. Quelle est l'algorithme de tri standard le plus efficace en moyenne ? Estimer grossièrement son facteur de gain par rapport aux deux autres tris.

5. Ajouter les coubres $n^2/2$ et $n^2/4$ en déduire la complexité moyenne du nombre de comparaisons des trois algorithmes.

6. Décrire le meilleur des cas et le pire des cas pour les trois algorithmes. Tracer les courbes pour ces deux cas pour les mêmes tailles de tableaux $50, 100, 150, \ldots, 500$.

7. Comparer avec les courbes de n^2 et n.

8. En déduire la complexité du nombre de comparaisons des trois algorithmes dans le meilleur et le pire des cas.

2.4 Les algorithmes

```
ALGORITHME TriInsertion(T)
DONNEES
    T un tableau de taille n
Variables
    i,j entiers
DEBUT
  i ← 2
  TQ i ≤ n FAIRE
     x ← T[i]
     j ← i
     TQ j > 1 ET T[j-1]>x FAIRE
        T[j] ← T[j-1]
        j ← j-1
     FTQ
     T[j] ← x
     i ← i+1
  FTQ
FIN
```

```
ALGORITHME TriSelection(T)
DONNEES
    T tableau de taille n
VARIABLES
    imin,i,j: entiers
DEBUT
  i←1
  TQ i ≤ n-1 FAIRE
    imin←i
    j←i+1
    TQ j ≤ n FAIRE
      SI T[j] < T[imin] ALORS
        imin←j
      FSI
      j←j+1
    FTQ
    Echanger(T,i,imin)
    i←i+1
  FTQ
FIN
```

```
ALGORITHME TriBulles(T)
DONNEES
    T tableau de taille n
VARIABLES
    d,i: entiers
    echange: booleen
DEBUT
  d←n
  echange←VRAI
  TQ echange FAIRE
    i←1
    echange←FAUX
    TQ i < d FAIRE
      SI T[i] > T[i+1] ALORS
        Echanger(T,i,i+1)
        echange←VRAI
      FSI
      i←i+1
    FTQ
  d←d-1
  FTQ
FIN
```

3 Meilleurs algorithmes de tris comparatifs

Les algorithmes de tris présentés ici ne sont pas très efficaces du point de vue de leur complexité. En effet, les algorithmes utilisés en pratique pour trier de grandes tailles de données sont de complexité $\Theta(n\log(n))$. On peut même démontrer que cette complexité est optimale pour la classe des tris comparatifs. On citera parmi ceux-ci les deux plus classiques:

- tri fusion,
- tri rapide (quicksort).

3.1 Tri fusion

À partir de deux listes triées, on peut facilement construire une liste triée comportant les éléments issus de ces deux listes (leur fusion). Le principe de

l'algorithme de tri fusion repose sur cette observation : le plus petit élément de la liste à construire est soit le plus petit élément de la première liste, soit le plus petit élément de la deuxième liste. Ainsi, on peut construire la liste élément par élément en retirant tantôt le premier élément de la première liste, tantôt le premier élément de la deuxième liste (en fait, le plus petit des deux, à supposer qu'aucune des deux listes ne soit vide, sinon la réponse est immédiate).

```
ALGORITHME triFusion(T[1,...,n])
DONNEES
    T un tableau de taille n
DEBUT
    SI n ≤ 1 ALORS
        RENVOYER T
    SINON
        RENVOYER fusion(triFusion(T[1,..., n/2]),
                 triFusion(T[n/2 + 1,..., n]))
    FSI
FIN
```

```
ALGORITHME fusion(A[1,..., a], B[1,..., b])
DONNEES
    A un tableau de taille a
    B un tableau de taille b
DEBUT
    SI A est le tableau vide ALORS
        RENVOYER B
    SI B est le tableau vide ALORS
        RENVOYER A
    SI A[1] ≤ B[1] ALORS
        RENVOYER A[1] ⊕ fusion(A[2,..., a], B)
    SINON
        RENVOYER B[1] ⊕ fusion(A, B[2,..., b])
FIN
```

Questions 2.

1. Écrire une fonction itérative `iterative_merge(T1,T2)` qui retourne un tableau trié contenant la fusion des deux tableaux où T1 et T2 sont initialement triés.
2. Écrire une fonction itérative `recursive_merge(T1,T2)` qui retourne un tableau trié contenant la fusion des deux tableaux où T1 et T2 sont initialement triés selon l'algorithme précédent.
3. Compter le nombre de comparaisons entre éléments de tableaux pour les deux fonctions.
4. Estimer la complexité $F(n)$ en nombre de comparaisons d'éléments du tableau de la fusion itérative ou récursive de deux tableaux de taille n et m.
5. Écrire une fonction `iterative_merge_sort (T)` qui fait le tri fusion d'un tableau T de n éléments en utilisant la fusion itérative de deux tableaux.
6. Écrire une fonction `recursive_merge_sort (T)` qui fait le tri fusion d'un tableau T de n éléments en utilisant la fusion récursive de deux tableaux.
7. On note $T(n)$ le nombre de comparaisons effectuées dans l'algorithme `triFusion`. Vérifier que $T(n) = \begin{cases} \Theta(1), & \text{si } n \leq 1 \\ 2T(n/2) + \Theta(n), & \text{sinon} \end{cases}$
   Quelle est la valeur de $T(n)$ après k appels récursifs ? En déduire que $T(n) = \Theta(n \log n)$

4 Le tri comptage

Le tri comptage (counting sort), appelé aussi tri casier (bucket), est un algorithme de tri par dénombrement qui s'applique sur des valeurs entières. On considère un tableau d'entiers T de taille n dont les éléments sont bornés entre 1 et k.

L'idée principale consiste à compter le nombre d'occurences de chaque élément de T compris entre 1 et k dans un tableau auxiliaire dit tableau de comptage dont la taille est k.

Par exemple pour le tableau `T=[1,2,1,5,3,5,8]` on construit le tableau `C` où `C[i]` est le nombre d'occurences de i dans le tableau T, on obtient `C=[2,1,1,0,2,0,0,1]`. Notons que C est de taille 8 la plus grande valeur du tableau.

Questions 3.

1. Écrire un algorithme `TriComptage(T)` qui trie le tableau T dans l'ordre croissant par la méthode de comptage.

2. Déterminer les complexités en temps et en espace de cet algorithme.

3. Discuter de l'efficacité de cet algorithme et de son cadre d'usage.

4. Écrire la fonction correspondante en Python.

5 Algorithmes de rangement

5.1 Rangement Noir & Blanc

Pour un tableau contenant deux valeurs NOIR ou BLANC, il s'agit de ranger les noirs d'un côté et les blancs de l'autre, pour cela on procède de la façon suivante :

- Parcourir le tableau;
- Mettre les éléments `NOIR` à gauche du tableau en partant du début;
- Mettre les éléments `BLANC` à droite du tableau en partant de la fin;
- Garder en mémoire la position du dernier élément `NOIR` et du premier élément `BLANC`.

```
ALGORITHME NoirEtBlanc(T)
DONNEES
  T: tableau de taille n
     de NOIR ou BLANC
VARIABLES
  g,d: entiers
DEBUT
  g←1
  d←n
  TQ g ≤ d FAIRE
    SI T[g] = BLANC ALORS
      Echanger(T,g,d)
      d←d-1
    SINON
      g←g+1
    FSI
  FTQ
FIN
```

- Arrêt: la suite des valeurs prises par $d - g$ est strictement décroissante
- Validité: ($T[1 : g - 1]$ ne contient que des éléments `NOIR` et $T[d+1 : n]$ des éléments `BLANC`) est un invariant

5.2 Le problème du drapeau hollandais

Le problème du drapeau hollandais est un problème de programmation, présenté par Edsger Dijkstra, qui consiste à réorganiser une collection d'éléments identifiés par leur couleur, sachant que seules trois couleurs sont présentes (par exemple, rouge, blanc, bleu, dans le cas du drapeau des Pays-Bas).

Étant donné un nombre quelconque de balles rouges, blanches et bleues alignées dans n'importe quel ordre, le problème consiste à les réarranger dans le bon ordre : les bleues d'abord, puis les blanches, puis les rouges.

La solution naïve consiste à trier le tableau. L'autre solution présentée ici consiste à :

- Parcourir le tableau;
- Mettre les éléments `BLEU` à gauche du tableau en partant du début;
- Mettre les éléments `BLANC` à droite des éléments `BLEU`;
- Mettre les éléments `ROUGE` à droite du tableau en partant de la fin;

- Garder en mémoire la position du dernier élément `BLEU`, du dernier élément `BLANC` et du premier élément `ROUGE`.

```
ALGORITHME BleuBlancRouge(T):
DONNEES
  T: tableau de taille n
     de 3 couleurs
VARIABLES
  b,w,r: entiers
DEBUT
  b,w,r←1,1,n
  TQ w ≤ r FAIRE
    SI T[w] = BLANC ALORS
      w←w+1
    SINON SI T[w] = ROUGE ALORS
      Echanger(T,w,r)
      r←r-1
    SINON
      Echanger(T,w,b)
      b←b+1
      w←w+1
    FSI
  FTQ
FIN
```

- Arrêt: la suite des valeurs prises par $r - w$ est strictement décroissante
- Validité: ($T[1 : b - 1]$ ne contient que des éléments `BLEU`, $T[b : w - 1]$ ne contient que des éléments `BLANC` et $T[r + 1 : n]$ des éléments `ROUGE`) est un invariant

Questions 4.

1. Écrire la fonction `NZP(T)` qui range les entiers du tableau T dans l'ordre négatif, zéro et positif et retourne le nombre de valeurs des 3 catégories sans les compter explicitement.
2. Modifier la fonction pour qu'elle retourne le nombre de comparaisons d'éléments du tableau et le nombre d'échanges.
3. Décrire le pire des cas et le meilleur des cas pour `NZP`. Puis donner un estimation pour chacun de ces cas du nombre d'échanges $E(n)$ et du nombre de comparaisons d'éléments du tableau $C(n)$.
4. Écrire la fonction `PI(T)` qui range les entiers du tableau T dans l'ordre pair puis impair et retourne le nombre de valeurs des pair et impair, sans les compter explicitement.
5. Décrire le pire des cas et le meilleur des cas pour `PI`. Puis donner un estimation pour chacun de ces cas du nombre d'échanges $E(n)$ et du nombre de comparaisons d'éléments du tableau $C(n)$.

6 Rapport

Présenter les codes, résultats, discussions et conclusions dans un rapport LaTeX, Python note book ou autre.