Lille iut Université de Lille

TD 1 : Étude de quelques tris

Structures de données
DUT info 2ème année
2020–2021

Dans ce TD, vous étudierez et écrirez plusieurs algorithmes de tri classiques. Vous utiliserez la syntaxe du langage Java.

1 Tri à bulles

Le tri à bulles consiste à successivement comparer les cases adjacentes d'un tableaux de valeurs et à les échanger de manière à faire "remonter" les valeurs les plus grandes vers le bout du tableau.

Question 1 : Écrivez la méthode `static void swap(int [] tab, int idx, int idx2)` qui permet d'échanger deux valeurs d'un tableau d'entiers. Utilisez ensuite cette dernière pour écrire la méthode `bubbleSort` qui permet de réaliser le tri à bulles. Dans cette première version, vous utiliserez deux boucles for.

```
static void swap(int [] tab, int idx, int idx2){
      int tmp  = tab[idx];
      tab[idx] = tab[idx2];
      tab[idx2] = tmp;
  }

static void bubbleSort(int [] tab){
      for(int i=tab.length-1; i>0; i--){
        for(int j=0; j<i; j++){
          if(tab[j]>tab[j+1]){
           swap(tab,j,j+1);
          }
        }
      }
}
```

Question 2 : Appliquez le tri à bulle au tableau de la figure 1. Que remarquez-vous ?

On constate qu'il y a peu de valeurs à trier et que l'on continue à comparer des valeurs alors que le tableau est déjà trié.

Question 3 : Pour pallier à cela, modifiez le code de la question 1 afin d'éviter tout parcours supplémentaire inutile.

```
  static void bubbleSortOpt(int [] tab){
      int end = tab.length - 1;
      while(end > 0){
          int perm = -1; // or if we need all positive numbers
          for(int i=0; i<end; i++){
              if(tab[i]>tab[i+1]){
                  swap(tab,i,i+1);
                  perm = i;
              }
          }
          end = perm;
      }
  }
```

22	11	8	17	25	34	47	52	69	98
0	1	2	3	4	5	6	7	8	9

FIGURE 1 – Tableau d'entiers

Question 4 : Combien y a-t-il comparaisons entre les éléments du tableau, effectuées lors de l'exécution de cette fonction dans le pire des cas ? Et combien de permutations ?

À chaque itération, il y a n-1 comparaisons et au plus n-1 permutations. Il y a donc exactement $\frac{n\times(n-1)}{2}$ comparaisons et dans le pire cas autant de permutations.

2 Tri par sélection

Le tri par sélection consiste à chaque étape à sélectionner le plus petit élément de la partie du tableau qui n'est pas encore triée et à l'échanger avec le premier élément de cette partie non triée. Ainsi, au départ la partie non triée du tableau correspond au tableau entier puis elle se rétrécie jusqu'à ne contenir que la dernière case du tableau.

Question 5 : Écrivez les quelques instructions qui permettent de trouver l'indice du plus petit élément d'un tableau `tab` entre les index `iBeg` (inclu) et `iEnd` (exclu). Combien de comparaisons entre les éléments du tableau faut-il faire ?

```
int minIdx = iBeg;

for(int i = iBeg+1; i<iEnd; i++){
    if(tab[i] < tab[minIdx]){
        minIdx = i;
    }
}
```

Il y aura iEnd - iBeg - 1 comparaisons effectuées par la fonction `minIdx`.

Question 6 : Écrivez une fonction qui fait un tri par sélection sur un tableau d'`int`, dont le prototype serait le suivant : `static void selectSort(int[] tab)`. Vous pourrez réutiliser la fonction `swap` définie plus tôt.

```
static void selectSort(int [] tab){

    int minIdx;
    for(int i=0; i<tab.length; i++){
        minIdx = i;
        for(int j=i+1; j<tab.length; j++){
            if(tab[j] < tab[minIdx]){
                minIdx=j;
            }
        }
        if(minIdx!=i){
            swap(tab,i,minIdx);
        }
    }
}
```

Question 7 : Combien y a-t-il comparaisons entre les éléments du tableau, effectuées lors de l'exécution de cette fonction ? Et combien de permutations d'éléments du tableau dans le pire des cas ?

Le nombre de comparaisons est similaire à celui du tri à bulles. Il y a donc exactement $\frac{n\times(n-1)}{2}$ comparaisons. Dans le pire cas, il y a $n-1$ permutations.

On note que la complexité pire cas du tri par sélection est la même que celle du tri à bulles. Cependant le nombre de permutations est bien moindre.

3 Tri par insertion

Le tri par insertion consiste à parcourir le tableau et à chaque étape à insérer l'élément courant à sa place dans le début du tableau déjà trié.

Question 8 : Écrivez les quelques instructions qui permettent d'insérer un élément dans un tableau `tab` trié. Le tableau est trié entre les bornes `iBeg` (inclu) et `iEnd` (exclu), l'élément à insérer est situé à l'index `iEnd`. Combien de comparaisons faut-il faire ? Combien de copies d'éléments ?

```
int tmp = tab[iEnd];
int i = iEnd;
while(i > iBeg && tab[i-1] >= tmp){
    tab[i] = tab[i-1];
    i--;
}
tab[i] = tmp;
```

Il faut faire iEnd-iBeg comparaisons et autant de copies sont nécessaires dans le pire cas.

À faire : recherche dichotomique de l'index d'insertion !

Question 9 : Écrivez une fonction qui fait un tri par insertion sur un tableau d'entiers, dont le prototype serait le suivant : `static void insertSort(int[] tab)`

Première version avec recherche séquentielle de l'index d'insertion :

```
static void insertSort(int [] tab){
    int tmp,j;
    for(int i = 1; i<tab.length; i++){
        tmp = tab[i];
        j = i;
        while(j > 0 && tab[j-1] >= tmp){
            tab[j] = tab[j-1];
            j--;
        }
        tab[j] = tmp;
    }
}
```

Version avec recherche dichotomique de l'index d'insertion :

```
public static void insertSort(int[] tab) {
    for(int i=1; i<tab.length; ++i) {
        int index = indiceInsertion(tab, tab[i], i);
        inserer(index, i, tab);
    }
}

private static int indiceInsertion(int[] tab, int val, int idxMax) {
    int indexMin = 0;
    int indexMax = idxMax;
    while(indexMax - indexMin > 1) {
        int mid = (indexMax + indexMin) / 2;
        if(tab[mid] > val) {
            indexMax = mid;
        }
        else {
            indexMin = mid;
        }
    }
    return tab[indexMin]<=val ? indexMin+1 : indexMin;
}

private static void inserer(int iInsert, int iVal, int[] tab) {
    int tmp = tab[iVal];
    System.arraycopy(tab, iInsert, tab, iInsert+1, iVal-iInsert);
    tab[iInsert] = tmp;
}
```

Question 10 : Combien y a-t-il comparaisons entre les éléments du tableau, effectuées lors de l'exécution de cette fonction ? Et combien de copies d'éléments du tableau ?

$\frac{n\times(n-1)}{2}$ comparaisons et copies dans le pire cas.

Dans le cas de l'algorithme utilisant la dichotomie, nous avons alors $n \times Logn$ comparaisons et $\frac{n\times(n-1)}{2}$ copies dans le pire cas.

4 Fusion de deux tableaux triés

Question 11 : Écrivez une fonction qui fait la fusion de deux tableaux d'entiers triés, dont le prototype serait le suivant : `void fuseTab(int[] tabInA, int[] tabInB, int[] tabOut)`
Le tableau `tabOut` (préalablement alloué avec une taille suffisante avant l'appel de la fonction `fuseTab`) doit être trié également et la fonction doit faire un seul "balayage" des tableaux.

```
void fuseTab(int[] tabInA, int[] tabInB, int[] tabOut){

    int aIdx = 0, bIdx = 0, outIdx = 0;
    while(aIdx < tabInA.length && bIdx < tabInB.length){
        if(tabInA[aIdx] < tabInB[bIdx]){
            tabOut[outIdx++] = tabInA[aIdx++];
```

```
            }else{
                tabOut[outIdx++] = tabInB[bIdx++];
            }
        }
        while(aIdx < tabInA.length){
            tabOut[outIdx++] = tabInA[aIdx++];
        }
        while(bIdx < tabInB.length){
            tabOut[outIdx++] = tabInA[bIdx++];
        }
    }
```

5 Tri de chaînes de caractères

Question 12 : En considérant l'ordre lexicographique, les propositions suivantes sont-elles vraies ?

"glas" < "glasse"
Vrai
"voiture" > "moto"
Vrai
"truc" == "curt"
Faux
"chapeau" > "char"
Faux

En Java, la classe `String` implémente une méthode `public int compareTo(String anotherString)` de l'interface `Comparable` qui permet dans le cas des chaînes de comparer deux chaînes de caractères suivant l'ordre lexicographique. La méthode renvoie un entier négatif si l'objet en question précède l'argument dans l'ordre lexicographique, un entier positif si l'objet succède à l'argument et 0 si les deux chaînes sont égales.

Question 13 : L'expression booléenne `"Abc".compareTo("abc") > 0` est-elle vraie ?
Non car la méthode compareTo se base sur la valeur des caractères dans la table unicocde où la valeur d'une lettre majuscule est inférieure à celle de la même lettre en minuscule. Donc la valeur de retour est un entier négatif. Remarque : la table unicode est une extension de la table ascii.
Question 14 : Modifiez l'algorithme de tri de façon à trier un tableau de chaînes de caractères, la méthode aura donc le prototype suivant : `void insertSort(String [] tab)`.

```
void insertSort(String [] tab){
        String tmp;
        int j;
        for(int i = 1; i<tab.length; i++){
            tmp = tab[i];
            j = i;
            while(j > 0 && tab[j-1].compareTo(tmp)>=0){
                tab[j] = tab[j-1];
                j--;
            }
            tab[j] = tmp;
        }
    }
```