Chapitre 6 : Les algorithmes de tri et de recherche

Juan Carlos QUEZADA

INSA Strasbourg

Complexité en temps d'un algorithme

Définition

La rapidité d'un algorithme est exprimée sous la forme du nombre d'opérations effectuées par celui-ci. En général, ce nombre dépend des données d'entrée. Il existe différents nombres d'opérations :

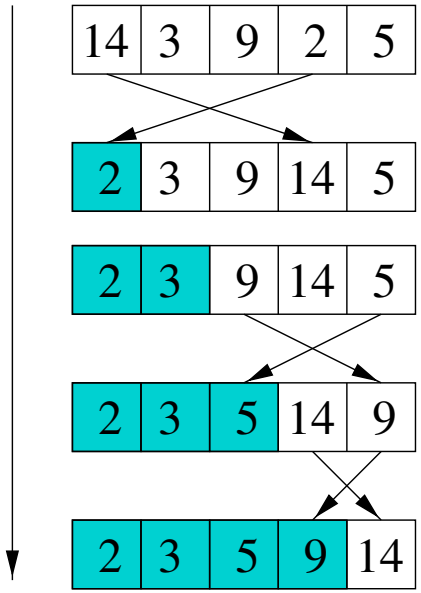
- le nombre d'opérations minimal, qui se produit dans le meilleur des cas,
- le nombre d'opérations maximal, qui se produit dans le pire des cas,
- le nombre moyen d'opérations, qui est une moyenne de ce que l'on peut espérer d'un algorithme dans la plupart des cas.

Un algorithme de tri d'un tableau de n éléments effectuera un nombre moyen d'opérations dépendant de n (la taille du tableau)

Tri par sélection

Définition

- on cherche l'élément de plus petite valeur dans le tableau
- le placer en tête du tableau
- recommencer avec le tableau moins la première case
- complexité en temps $O(n^2)$



Tri par sélection

Pseudo-langage

```
Pour i de 0 à N-1 faire
    min=i
    Pour j de i+1 à N faire
        Si tab[j]<tab[min] alors
            min=j
        fin si
        échanger(tab[i],tab[max])
    fin pour
fin pour
```

Tri par insertion

Principe

- Méthode utilisée pour trier une main de jeu de carte.
- On prend les cartes une par une de la gauche vers la droite.
- Pour chaque carte on la place au bon ordre parmi les précédentes.
- complexité en temps $O(n^2)$

Pseudo-langage

```
Pour i de 1 à N-1 faire
    actuelle=tab[i]
    j=i
    Tant que (j>0 et tab[j-1]>actuelle) faire
        tab[j]=tab[j-1]
        j=j-1
    fin tant que
    tab[j]=actuelle
```

Tri de bulles

Principe

- Les éléments les plus petits se « déplacent » vers le début du tableau.
- Les éléments contiguës sont échangés 2 à 2.
- complexité en temps $O(n^2)$

Pseudo-langage

```
Pour i de 0 à N-1 faire
    Pour j de 0 à N-1 faire
        Si T[j]>T[j+1] alors
            échanger(T[j],T[j+1])
        fin si
    fin pour
fin pour
```

Algorithme de recherche

Recherche dichotomique

Algorithme de recherche pour trouver la position d'un élément dans un tableau trié. Le principe est le suivant : comparer l'élément avec la valeur de la case au milieu du tableau ; si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinent. Complexité de $O(\log(n))$.

Algorithme de recherche

Recherche dichotomique : Pseudo-langage

```
début = 0
fin = N
trouvé = faux
Tant que (trouvé != vrai et début <= fin) faire
    milieu = (début + fin)/2
    si (t[milieu] == valeur_cherchée) alors
        trouvé = vrai
    sinon
        si (valeur_cherchée > t[milieu]) alors
            début = milieu+1
        sinon
            fin = milieu-1
        fin si
    fin si
fin tant que
```

Exercices

Exercice 1 : Tri par insertion

- Écrire un algorithme pour trier un tableau de 100 entiers par la méthode du tri par insertion.
- Le tableau sera initialisé avec des valeurs aléatoires.

Exercice 2 : Tri par sélection décroissant

- Soit un tableau T de N nombres entiers.
- Écrire un algorithme pour trier ce tableau par ordre décroissant en utilisant le tri par sélection. La taille N du tableau est saisie par l'utilisateur.
- Les boucles de contrôle de saisie nécessaires seront mises en place. Les valeurs du tableau sont elles aussi saisies par l'utilisateur.

Exercices

Exercice 3

Écrire un algorithme qui :

- remplit un tableau T d'entiers avec les valeurs suivantes : 1, 5, 9, 12, 15, 21, 29.
- recherche dans ce tableau au moyen d'une dichotomie les valeurs suivantes : 6, 1, 12, 30, 0,21, 29
- affiche le résultat de ces recherches sous forme d'un message approprié.

Exécuter cet algorithme à la main en remplissant un tableau qui comporte en colonnes les noms des différentes variables de l'algorithme et en ligne les valeurs prises par ces variables au fur et à mesure de l'exécution de l'algorithme. Par exemple, si *val* est la variable utilisée pour stocker la valeur à rechercher dans T on aura au début de l'algorithme le tableau suivant :

T[0]	T[1]	T[2]	T[3]	T[4]	T[5]	T[6]	val	...	...
1	5	9	12	15	21	29			

Exercices

Exercice 4 : Tri par sélection et sous-programmes

Soit un tableau T de N nombres entiers. Écrire un algorithme pour trier ce tableau par ordre **décroissant** en utilisant le tri par sélection. Pour ce faire, écrire les sous-programmes suivants :

- Sous-programme *saisir_tableau* qui saisit les N nombres d'un tableau unidimensionnel
- Sous-programme *plus_grand* qui appliqué à un entier i ($0 \leq$ i $\leq$ N-1) et à un tableau recherche dans ce tableau, à partir du rang i, le plus grand nombre et retourne son rang.
- Sous-programme *echanger* qui appliqué à deux entiers i et j ($0 \leq$ i $\leq$ N-1 et $0 \leq$ N-1) et à un tableau échange les nombres contenus dans les cases i et j.
- Sous-programme *trier* qui appliqué à un tableau le trie par la méthode du tri par sélection.
- Sous-programme *afficher_tableau* qui affiche le contenu d'un tableau unidimensionnel sur une seule ligne et sous la forme $[n_1, ..., n_k]$.

Écrire l'algorithme principal qui permet qui permet la saisie du tableau T, le trie puis l'affichage à l'aide des sous-programmes précédents.

Exercice 5 : Le jeu du devin

Attention

L'exercice suivant est à rendre sur Moodle pour la semaine prochaine

Introduction

Les trois parties suivantes ont pour but de concevoir un jeu dans lequel l'utilisateur joue contre l'ordinateur. La règle du jeu peut s'énoncer ainsi :

- Le jeu met en opposition deux joueurs, le sage et le devin. Le sage choisi un nombre que le devin doit découvrir. Le devin propose des nombres jusqu'à ce qu'il découvre le nombre choisi par le sage. A chaque proposition, le sage ne peut répondre que par l'une des trois propositions suivantes :
  - le nombre à découvrir est plus grand que le nombre proposé
  - le nombre à découvrir est plus petit que le nombre proposé
  - le nombre est découvert

Exercice 5 : Le jeu du devin

Introduction

- Lorsque le nombre est découvert on totalise le nombre de coups qui ont été nécessaires et on les ajoute au score du devin.
- Une partie est constituée d'un nombre pair de manches, dans chaque manche le devin doit découvrir le nombre caché pour établir son score sur la manche.
- Les joueurs échangent leur rôle pour la manche suivante.
- Le score sur la partie est constitué de la somme des scores du joueur sur les différentes manches.
- Le gagnant de la partie est celui qui a totalisé le plus petit nombre de coups sur la partie.

Exercice 5 : Le jeu du devin

L'utilisateur devine un nombre

Dans cette première partie on considère que l'ordinateur est le sage et que vous êtes le devin.

- Écrire l'algorithme correspondant à une manche dans cette configuration.
- On suppose que l'ordinateur effectue un choix aléatoire du nombre à découvrir.

L'ordinateur devine le nombre du joueur

Dans cette deuxième partie, vous êtes le sage et l'ordinateur est le devin.

- Ecrire l'algorithme correspondant à une manche dans cette configuration.
- On signalera à l'ordinateur que sa proposition :
  - est inférieure au nombre choisi en tapant le caractère '<'
  - est supérieure au nombre choisi en tapant le caractère '>'
  - est égale au nombre 'à découvrir en tapant le caractère '='

On réfléchira bien à l'efficacité de la technique de recherche choisie.

Exercice 5 : Le jeu du devin

Le jeu complet

Proposez l'algorithme du jeu complet. Cet algorithme devra permettre de définir le nombre de manches dans la partie et de choisir qui débute la partie au moyen d'un menu :

1. ordinateur / humain
2. humain / ordinateur

Après la partie, l'algorithme devra désigner le vainqueur et les scores obtenus.