Méthodes de Conception d'Algorithmes

C. TRABELSI & M. FRANÇOIS

TD6 - - Diviser pour Régner (recherche dichotomique)

Avant propos

La méthode « Diviser pour Régner » est une stratégie aussi répandue qu'efficace qui consiste à séparer un problème en deux sous-problèmes similaires mais indépendants. Les deux sous-problèmes possèdent deux fois moins de données et sont donc plus rapides à traiter. Nous allons voir une de ces implémentations la plus simple : la ***dichotomie****.*

A. Recherche par dichotomie de la racine carrée d'un nombre

L'objectif ici est de rechercher une valeur approchée de la racine carrée d'un nombre réel positif x ($x \geq 1$) à ϵ près, à l'aide d'un algorithme dichotomique.

Pour rappel :
La ***dichotomie*** *(« couper en deux » en grec) est une stratégie algorithmique de la catégorie « Diviser pour Régner ». C'est un processus itératif ou récursif où, à chaque étape, on découpe l'espace de recherche en deux parties (pas forcement égale) puis ayant déterminé dans laquelle se trouve la solution, on restreint l'espace à cette partie.*
On suppose bien sur qu'il existe un test relativement simple permettant à chaque étape de déterminer l'une des deux parties dans laquelle se trouve une solution.
Pour optimiser le nombre d'itérations nécessaires, on s'arrangera pour choisir à chaque étape deux parties sensiblement de même "taille". Le nombre total d'itérations nécessaires à la complétion de l'algorithme étant alors logarithmique en la taille totale du problème initial [WIKI].

Répondre aux questions suivantes :

- ► **1.** Définir « l'espace de recherche » pour le problème de la recherche de la racine carrée d'un nombre noté n.
- ► **2.** On désigne par $[a, b]$ l'intervalle de recherche et ϵ la marge d'erreur. Quelle condition booléenne permet de savoir s'il doit y avoir une nouvelle itération ?
- ► **3.** Quel test va vous permettre de savoir dans laquelle des deux parties se trouve la solution ?
- ► **4.** Écrire l'algorithme correspondant à la fonction suivante (on suppose que $n \geq 1$ et $\epsilon > 0$) :

  ```
  fonction Racine_Carree(n, ϵ);
  ```

- ► **5.** Écrire la version C de cet algorithme. Pour cela vous pouvez utiliser le type `double`. (À faire en TP)

B. Les tableaux

a) Plus petit élément d'un tableau

► **1.** Écrire une fonction `min_TAB`, qui à partir d'un tableau d'entiers TAB non trié de n éléments retourne le plus petit élément du tableau.

```
fonction min_TAB(TAB, n);
```

► **2.** Écrire la version C de cet algorithme. Pour cela vous pouvez utiliser le type `int`. (À faire en TP)

b) Sous-séquences croissantes

Considérons un tableau de 15 éléments donné par :

```
TAB = [1, 2, 5, 3, 12, 25, 13, 8, 4, 7, 24, 28, 32, 11, 14]
```

► **1.** Quelles sont les sous-séquences strictement croissantes dans le tableau TAB ?

► **2.** Écrire une fonction `Determ_Nbre_Seq`, qui à partir d'un tableau d'entiers TAB de n éléments, fournit le nombre de sous-séquences strictement croissantes de ce tableau, ainsi que les indices de début et de fin de la plus grande sous-sequence.

```
fonction Determ_Nbre_Seq(TAB, n);
```

► **3.** Écrire la version C de cet algorithme. Pour cela vous pouvez utiliser le type `int`. (À faire en TP)

c) Recherche par dichotomie

► **1.** Écrire une fonction `Recherche_Dicho`, qui détermine par dichotomie le plus petit indice d'un élément (dont on est sûr de l'existence) dans un tableau d'entiers TAB de n éléments triés dans l'ordre croissant. Il peut y avoir des éléments qui apparaissent plusieurs fois.

```
fonction Recherche_Dicho(TAB, n, ELT);
```

► **2.** Écrire la version C de cet algorithme. Pour cela vous pouvez utiliser le type `int`. (À faire en TP)