Initiation au développement

TD n° 11 – Recherche Dichotomique

Soit le paquetage *Tableau_ Entiers__Triés* spécifié comme suit :

```
package Tableau_Entiers_Triés is

        DEBORDEMENT : exception ;
        TRANCHE_INVALIDE : exception ;

        TAILLE_MAX : constant Integer := 100 ;
        type  Tab_Entiers_Triés is array (1 .. TAILLE_MAX) of Integer ;

        -- lit n entiers et les placent triés par ordre croissant dans le tableau tab
        -- entraine pour tout i variant de 1 à n, tab(i) <= tab(i+1)
        -- entraine n <= TAILLE_MAX
        -- lève l'exception DEBORDEMENT si plus de TAILLE_MAX entiers sont entrés
        procedure lire (tab : out Tab_Entiers_Triés  ; n : out Integer);

        -- affiche les n éléments triés par ordre croissant du tableau tab
        -- nécessite 0 <= n <= TAILLE_MAX
        -- lève l'exception TRANCHE_INVALIDE si n<0 ou n > TAILLE_MAX
        procedure écrire (tab : in Tab_Entiers_Triés ; n : in Integer);

end Tableau_Entiers_Triés ;
```

et le programme principal suivant :

```
with Tableau_Entiers_Triés ; use Tableau_Entiers_Triés ;

procedure main is

tab : Tab_Entiers_Triés ;
n : Integer ; -- nombre d'éléments du tableau

begin
        -- rentre les valeurs triées dans le tableau
        lire (tab, n) ;
        -- affiche les éléments du tableau
        écrire (tab, n) ;

end main ;
```

Dans ce type de tableau *tab*, la suite des v_1, v_2, …, v_n valeurs sont rangées par ordre croissant. On désire rechercher par dichotomie si une valeur donnée appartient à *tab*. La **dichotomie**[1] consiste à réduire successivement le domaine de recherche en deux sous-ensembles, en distinguant les deux sous-ensembles par un élément milieu de l'ensemble.

[1] Division en deux

Questions

1) Spécification

1.1) Donner en langage Ada l'en-tête d'un sous-programme *rechercher_Par_Dichotomie* recherchant par dichotomie la position d'une occurrence d'un élément *x* dans une tranche du tableau *tab* de *n* éléments. Le sous-programme renvoie VRAI si l'élément est trouvé et FAUX sinon. Si l'élément a été trouvé, il retourne également le rang de cet élément dans le tableau.

1.2) Compléter cet en-tête en précisant les pré conditions et l'exception susceptible d'être levée par cette opération.

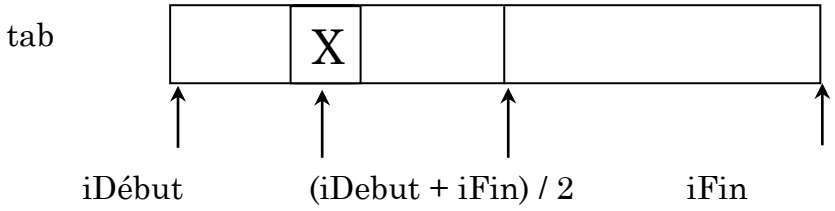
1.3) Compléter la spécification du TAD *Tableau_Entiers_Triés*.

2) Algorithme

2.1) Donner le principe de l'algorithme, en montrant qu'il nécessite une répétition.

2.2) En déduire l'algorithme général de la recherche dichotomique d'un élément *x* dans une tranche de tableau trié *tab (1..n)*.

2.3) Exprimer formellement les deux conditions de sortie de la boucle en considérant l'évolution des 2 indices, *iDébut*, et *iFin* pour la recherche d'un élément *x* comme le montre le schéma ci-dessous :



2.4) Exprimer formellement comment choisir la tranche en fonction de l'emplacement de la valeur de l'élément à rechercher.

3) Programmation

3.1) Ecrire en langage Ada le corps du sous-programme *rechercher_Par_Dichotomie* en levant l'exception.

3.2) Effectuer la trace du sous-programme avec le jeu d'essai *tab(1..11) = (10, 20, 20, 20, 20, 35, 50, 60, 65, 70, 75)* et *x* ayant successivement les valeurs *5*, *10*, *20* et *80*. Vous pourrez comparer votre résultat avec le programme d'affichage de la recherche présenté en cours.

4) Calculer de la complexité des algorithmes de recherche séquentielles et dichotomiques dans les cas les plus favorables, les moins favorables et le cas moyen.

5) Pour aller plus loin : fournir une version récursive du sous-programme *rechercher_Par_Dichotomie* défini itérativement à la question 3.1.

a) x = 5

situation	iDebut	iFin	iMilieu	tab(iMilieu)	trouve	rang
1	1	11	6	35	?	?
2						
4						
2						
4						
2						
4						
6						

b) x = 10

situation	iDebut	iFin	iMilieu	tab(iMilieu)	trouve	rang
1	1	11	6	35	?	?
2						
4						
2						
4						
5						

c) x = 20

situation	iDebut	iFin	iMilieu	tab(iMilieu)	trouve	rang
1	1	11	6	35	?	?
2						
4						
5						

d) x = 80

situation	iDébut	iFin	iMilieu	tab[iMilieu]	trouvé	rang
1	1	11	6	35	?	?
3						
4						
3						
4						
3						
4						
3						
4						
6						