TRIS ET RECHERCHES

- ## Les tris

Les algorithmes de tri servent principalement à ordonner les données, mais ils participent aussi à la conception d'autres algorithmes, comme les recherches, en organisant préalablement les données.

Les algorithmes de tri sont regroupés en deux catégories : les tris élémentaires et les tris avancés. Les tris élémentaires se basent sur les méthodes simples, que l'on retrouve parfois dans la vie active. Les tris avancés s'appuient sur des méthodes plus sophistiquées.

Dans ce fascicule, on va étudier quelques tris élémentaires et nommer quelques tris avancés sans rentrer dans les détails.

➢ Tris élémentaires

A. Le tri par sélection

C'est l'un des tris les plus simples à mettre en œuvre. Cela consiste à rechercher la plus petite valeur (ou la plus grande valeur) et l'inverser avec la première position, puis à rechercher la deuxième plus petite valeur (ou la deuxième plus grande valeur) et à l'inverser avec la deuxième position, et ainsi de suite.

L'inversion de deux valeurs est souvent employée dans les algorithmes de tri.

Le tri par sélection est basé sur deux boucles imbriquées : La première boucle parcourt la liste des valeurs du début à la fin et la seconde boucle recherche la plus petite valeur (ou la plus grande valeur).

Ce tri est appelé aussi tri par la recherche du minimum (ou du maximum).

Voici une procédure qui fait ce travail.

```
Procédure Tri_selection(Tab* : tableau, taille :entier)
Déclaration
Variable
        Imin, i, j : entier ;
        Mem, min : item ; /*type des éléments du tableau*/
Début
        Pour i allant de 1 à taille  faire
            Début
                Imin ← i ;
                Min ← tab( i ) ;
                Pour  j allant de i+1 à taille  faire
                    Si tab( j ) < min alors
                        Début
                            Imin ← j ;
                            Min ← tab( j ) ;
                        Fin
                    Fin si
                Fin pour

                Mem ← tab( i ) ;

                Tab( i ) ← tab( imin ) ;

                Tab( imin ) ← Mem ;

            Fin

        Fin pour
Fin
```

Voici une autre version qui n'utilise pas « min ».

```
Procédure Tri_selection(Tab* : tableau, taille :entier)
Déclaration
Variable
        Imin, i, j : entier ;
        Mem : item ; /*type des éléments du tableau*/
Début
        Pour i allant de 1 à taille  faire
                Début
                    Imin ← i ;
                    Pour  j allant de i+1 à taille  faire
                        Si tab( j ) < tab( imin )  alors
                            Imin ← j ;
                        Fin si
                    Fin pour

                    Mem ← tab( i ) ;

                    Tab( i ) ← tab( imin ) ;

                    Tab( imin ) ← Mem ;

                Fin

        Fin pour

Fin
```

B. Le tri par insertion

Le tri par insertion s'inspire d'un processus naturel utilisé dans le tri des cartes qu'un joueur possède dans sa main. Pour organiser son jeu, le joueur prend chaque carte et l'insère à sa place en décalant les autres cartes.

Le principe est de décaler vers la droite les données situées à droite de la position d'insertion en partant de la fin, pour libérer la case d'insertion. La dernière case est copiée dans la case suivante, puis c'est au tour de l'avant-dernière case et de proche en proche, on remonte jusqu'à la case à libérer. Une fois cette case libre, on y copie la donnée à insérer.

Pour cela, on aura besoin de deux boucles imbriquées : une pour parcourir la liste des valeurs et l'autre pour l'insertion.

Voici une procédure qui réalise ce travail.

```
Procédure  Tri_insertion( Tab* : tableau, taille : entier)
Déclaration
Variable
        I, j : entier ;
        Val : item ;
Début
        Pour i allant de 1 à taille  faire
                Début
                        Val ← tab( i ) ;
                        J ← i ;
                        Tant que (j > 1) et (val < tab( j-1 )) faire
                                Début
                                        Tab( j ) ← tab( j-1 ) ;
                                        J ← j-1 ;
                                Fin
                        Fin tant que
                        Tab( j ) ← Val ;
                Fin
        Fin pour
Fin
```

C. Tri à bulles

Le tri à bulles propose une approche assez déroutante. On peut même se demander comment cette méthode proposée aboutit au tri des données.

Ce tri fait partie des grands classiques des cours d'algorithmique. Ce tri est construit à partir de deux boucles imbriquées qui évoluent en sens inverse l'une de l'autre, et d'un processus d'inversion des éléments successifs qui ne sont pas ordonnés.

Le nom de cet algorithme provient de l'analogie avec les bulles d'une boisson gazeuse qui remontent de proche en proche à la surface !

Voici une procédure qui réalise ce travail.

```
Procédure Tri_bulles(Tab* : tableau, taille : entier)
Déclaration
Variable
        i, j : entier ;
        Mem : item ;
Début
        Pour i allant de taille  à  2 pas -1 faire
            Pour j  allant de 2 à i faire
                Si  tab( j-1 ) > tab( J ) alors
                    Début
                        Mem ← tab( j ) ;
                        tab( j ) ← tab( j-1 ) ;
                        tab( j-1 ) ← Mem ;
                    Fin
                Fin si
            Fin pour
        Fin pour
Fin
```

➢ Tris avancés

On va citer quelques tris avancés sans les étudier, cela se fera peut-être dans les années supérieures.

a. Tri SHELL du nom de son inventeur.
b. Tri indirect par tableau des indices.
c. Tri indirect par tableau de pointeurs
d. Tri rapide ou Quicksort.
e. Tri par tas ou Heapsort.
f. Etc …

Des exercices afférents aux tris est d'écrire en langage C des programmes qui implémentent les algorithmes étudiés ci-dessus, avec Tab qui contient une dizaine de nombres réels.

Puis faire tourner à la main chaque procédure sur un exemple pour voir l'évolution du tri pour chaque méthode.

- **Les recherches.**

La recherche est une opération fondamentale dans un programme qui gère un ensemble de données.

Elle participe souvent à l'élaboration des autres traitements, par exemple en localisant la donnée à traiter.

I. La recherche séquentielle.

La recherche séquentielle est une méthode élémentaire qui consiste à comparer la donnée recherchée à toutes les autres. La recherche est positive c'est-à-dire on a trouvé ce qu'on recherche ou bien elle est négative c'est-à-dire ce qu'on recherche n'y est pas.

Pour cela, on aura besoin d'une seule boucle avec deux critères d'arrêt.

Le résultat est le numéro de la case où se trouve la donnée recherchée ou bien la valeur -1 pour dire que la donnée recherchée n'est pas trouvée !

Voici une fonction qui réalise ce travail.

```
Fonction Reche_seque(Tab : tableau, taille : entier, valeur : item) : entier ;
Déclaration
Variable
        i : entier ;
        Trouve : booleen ;
Début
        i ← 1 ;
        Trouve ← FAUX ;
        Tant que (Trouve = FAUX) et ( i ≤ taille) faire
                Si (tab( i ) = valeur) alors
                   Trouve ← VRAI ;
                Si non
                        i ← i + 1 ;
                Fin si
        Fin tant que
        Si  (Trouve = VRAI) alors
           Retourner i ; /*Le rang de la valeur trouvée dans le vecteur*/
        Si non
              Retourner -1 ; /* pour dire que la valeur cherchée n'existe pas*/
        Fin si
Fin
```

II. La recherche dichotomique.

Dès que le nombre de données devient important, la recherche séquentielle devient obsolète; donc il faut envisager une autre recherche qui diminue le temps de traitement par l'emploi d'algorithmes plus performants comme la recherche dichotomique. Cet algorithme est basé sur le principe « diviser pour résoudre ».

Voici le principe : On divise l'ensemble de recherche en deux sous-ensembles égaux. On détermine ensuite dans quel sous-ensemble doit se trouver la donnée recherchée, puis on poursuit la recherche dans ce sous-ensemble.

Le préalable à cette méthode de recherche est de disposer d'un ensemble trié de données.

L'algorithme de recherche dichotomique peut s'écrire sous deux versions : itérative et récursive mais comme on n'a pas étudié la récursivité l'algorithme récursif ne sera pas exposé.

Voici une fonction itérative :

Pour cela on aura besoin de deux bornes « **Binf** » comme borne inférieure, « **Bsup** » comme borne supérieure, « **Mil** » comme milieu et la booléenne **trouve** en tant que variables locales en plus du tableau **Tab**, **taille** et de la **Valeur** à rechercher en tant que paramètres.

```
Fonction Rech_dicho_ite(Tab : tableau, taille : entier ; valeur : item) : entier;
Déclaration
Variable
        Binf, Bsup, Mil : entier ;
        Trouve : booleen ;
Début
        Binf ← 1 ;
        Bsup ← taille ;
        Trouve ← FAUX ;
        Tant que (Binf < Bsup) et (trouve = FAUX) faire
                Début
                        Mil ← (Binf + Bsup)/2 ;
                        Si Tab(Mil) = valeur alors
                           Trouve ← VRAI ;
                        Si non
                              Si Tab(Mil) > valeur alors
                                 Binf ← Mil-1 ;
                              Si non
                                    Bsup ← Mil+1 ;
                              Fin si
                        Fin si
                Fin
        Fin tant que
        Si Trouve = VRAI alors
           Retourner  Mil ; /* Le rang de la valeur cherchée dans le vecteur*/
        Si non
              Retourner -1 ; /* Pour dire que la valeur cherchée n'existe pas*/
        Fin si
Fin
```

Des exercices afférents aux méthodes de recherche est d'implémenter ces fonctions en langage C puis faire le parallèle entre la fonction de recherche séquentielle et celle de recherche dichotomique.