Série N°01 – Rappel et complexité

Exercice 01 : Recherche dans un tableau circulaire

Un tableau trié circulaire est un tableau trié dans lequel les éléments sont circulairement déplacés. Par exemple, le tableau [12, 14, 18, 21, 3, 6, 8, 9] est un tableau trié (dans un ordre croissant) circulaire. On appel l'élément 3 un point de rotation (ou le pivot) du tableau. L'objectif de cet exercice est d'écrire une fonction qui prend en paramètre un tableau trié circulaire et un élément et qui retourne l'indice de l'élément dans le tableau. Si l'élément n'est pas dans le tableau, la fonction retourne -1.

1. Commencer par la solution la plus naïve qui consiste à parcourir le tableau linéairement jusqu'à trouver l'élément, ou atteindre la fin du tableau. La complexité de cette solution est linéaire ***(O(n))***.

```
int recherche_lineaire(int T[], int n, int v) {
    for (int i = 0; i < n; ++i) {
        if (T[i] == v) return i;
    }
    return -1;
}
```

2. Il est également possible d'utiliser la recherche dichotomique, vu que les éléments du tableau sont triés. Néanmoins, il faut l'adapter pour qu'elle fonctionne avec un tableau trié circulaire. Il faudra donc :
    a. Chercher le point de rotation.

```
// La solution est légèrement différente de celle que nous avons faite en TD
int trouver_pivot(int T[], int n) {
    int debut = 0;
    int fin = n - 1;
    while (debut <= fin) {
        int m = (debut + fin) / 2;
        // Nous utilisons le « % n » pour nous assurer de ne pas
        // dépasser la taille du tableau. Dans le cas ou milieu = n - 1
        // sa valeur est donc comparée avec le premier élément.
        if (T[m] > T[(m + 1) % n]) return m + 1;
        else if (T[debut] <= T[m]) debut = m + 1;
        else fin = m - 1;
    }
    return 0;
}
```

    b. Appliquer la recherche dichotomique sur un tableau qui commence à partir du point de rotation, jusqu'à l'élément qui se trouve juste avant.

```
// La solution est légèrement différente de celle que nous avons faite en TD

// La solution est similaire à une recherche dichotomique classique
// sauf que le début et la fin de l'interval de recherche sont initialisés
// par rapport au pivot et la valeur recherchée.
int recherche_dichotomique_circulaire(int T[], int n, int v) {
    int pivot = trouver_pivot(T, n);
    int debut, fin;
    // Déterminer dans quelle moitié du tableau l'élément peut se trouver
    if (v >= T[0] && v <= T[pivot - 1]) {
        debut = 0;
        fin = pivot - 1;
    } else {
        debut = pivot;
        fin = n - 1;
    }
    // Appliquer la recherche dichotomique dans la moitié appropriée
    while (debut <= fin) {
        int m = (debut + fin) / 2;
        if (T[m] == v) return m;
        else if (T[m] < v) debut = m + 1;
        else fin = m - 1;
    }
    return -1;
}
```

Noter qu'il est aussi nécessaire d'utiliser la recherche dichotomique afin de trouver le point de rotation pour que cette solution soit utile.

Exercice 02 : Sous tableau avec la somme max (compétition – solve it 2023)

Écrire une fonction qui prend en paramètre un tableau d'entiers et qui retourne la somme maximale d'un sous-tableau de ce tableau. Par exemple, si le tableau d'entrée est le suivant : ***A = [1, 2, -4, 3, 2, -1, 3, -1]***, la fonction doit retourner ***7***, qui est la somme maximale du sous-tableau ***[3, 2, -1, 3]***.

1. La solution naïve consiste à calculer la somme de tous les sous-tableaux possibles, les comparer et calculer leur max. Donner cette solution. Quelle est sa complexité ?

```
// La pire solution possible ☺ ☺ (complexité O(n³))
int max_subarray(int T[], int n) {
    int max = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            int current = 0;
            for (int k = i; k <= j; k++) {
                current += T[k];
            }
            if (current > max) max = current;
        }
    }
    return max;
}
```

2. Donner une solution plus efficace avec une complexité linéaire ***(O(n))***.

```
int max_subarray_linear(int T[], int n) {
    int max = 0;
    int current = 0;
    for (int i = 0; i < n; i++) {
```

```
        current = current + T[i];
        if (current > max) max = current;
        if (current < 0) current = 0;
    }
    return max;
}
```

Exercice 03 : Recherche dans une matrice triée

Soit une matrice d'entiers ***n*** lignes et ***m*** colonnes triée tel que :

- Les lignes sont triées par ordre croissant.
- Le premier élément de chaque ligne est supérieur au dernier élément de la ligne précédente.

Ecrire une fonction « ***recherche2D*** » qui prend en paramètres une matrice triée et un entier et qui renvoie vrai si l'entier est présent dans la matrice et faux sinon.

1. Commencer initialement par l'implémentation de la version naïve.

```
int recherche2D_naive(int M[][], int n, int m, int v) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (M[i][j] == v) return 1;
        }
    }
    return 0;
}
```

2. Il est possible d'optimiser votre solution en utilisant la recherche dichotomique une première fois pour trouver la ligne (possible) du la valeur recherchée. Puis utiliser la recherche dichotomique une deuxième fois pour dire si la valeur recherchée existe ou pas.

```
int recherche_dichotomique_2D(int M[][], int n, int m, int v) {
    // Nous allons appliquer le même algorithme de recherche dichotomique afin
    // de localiser la ligne sur laquelle la valeur recherchée peut se trouver.
    int debut = 0;
    int fin = n - 1;
    while (debut <= fin) {
        int m = (debut + fin) / 2;
        // Si la valeur recherchée est supérieure au premier élément
        // de la ligne, et inférieure à son dernier élément.
        // Appliquer une recherche dichotomique classique sur la ligne en question
        if (M[m][0] <= v && v <= M[m][m-1])
            return recherche_dichotomique(M[m], m, v);
        else if (M[m][0] > v) fin = m - 1;
        else debut = m + 1;
    }
    return 0;
}
```

Exercice 04 : Médian de deux tableaux

Le médian d'un tableau est l'élément qui se trouve au milieu lorsque celui-ci est trié. Si le tableau contient un nombre impair d'éléments, le médian sera l'élément central. Si le tableau contient un nombre pair d'éléments, le médian est défini comme la moyenne des deux éléments centraux.

1. Ecrire une fonction ***« median »*** qui prend comme paramètre un tableau trié ***A*** avec sa taille, et qui retourne le médian de ce tableau. La fonction suppose que le tableau est déjà trié.

```
double median(int T[], int n) {
    if (size % 2 == 0) return (double) (T[n / 2 - 1] + T[n / 2]) / 2;
    else return T[n / 2];
}
```

2. Ecrire une fonction ***« median2 »*** qui prend comme paramètre deux tableaux triés ***A*** et ***B*** de tailles ***n*** et ***m*** respectivement, et qui retourne le médian de ces deux tableaux.
    a. La solution la plus naïve serait de fusionner les deux tableaux, puis chercher le médian du résultat.

```
double median(int A[], int n, int B[], int m) {
    // Fusionner les deux tableaux dans un autre.
    int T[n + m];
    int i = 0, j = 0, k = 0;
    while (i < n && j < m) {
        if (A[i] < B[j]) T[k++] = A[i++];
        else T[k++] = B[j++];
    }
    while (i < n) T[k++] = A[i++];
    while (j < m) T[k++] = B[j++];
    // Calculer le médian du tableau T.
    return median(T, k);
}
```

    b. Il est également possible de chercher le médian sans fusionner les tableaux, il suffit de parcourir leurs éléments dans un ordre croissant, jusqu'à le position ***(n + m) / 2***, puis calculer la valeur du médian.

```
// Cette solution a été légèrement expliqué en TD, elle consiste à compter
// les indices supposés dans le tableau fusionné qui n'est pas réellement créé.
double median2(int A[], int n, int B[], int m) {
    int i = 0, j = 0, m1 = -1, m2 = -1;
    for (int count = 0; count <= (n + m) / 2; count++) {
        m1 = m2;
        if (i != n && j != m) {
            if (A[i] < B[j]) m2 = A[i++];
            else m2 = B[j++];
        }
        else if (i < n) m2 = A[i++];
        else m2 = B[j++];
    }
    if ((n + m) % 2 == 0) return (double) (m1 + m2) / 2;
    else return m2;
}
```

c. Il est finalement possible d'adapter la recherche binaire pour réduire la complexité de votre algorithme à une complexité sous linéaire.

```
// Cette solution n'a pas été faite en TD, je vous ferai très probablement
// une vidéo consacré à  cette solution sur la plateforme de l'université.

double median2(int A[], int n, int B[], int m) {

    if (n > m)
        return median2(B, m, A, n);

    int imin = 0, imax = n, half_len = (n + m + 1) / 2;

    while (imin <= imax) {


        int i = (imin + imax) / 2;
        int j = half_len - i;

        if (i < n && B[j-1] > A[i]) imin = i + 1;
        else if (i > 0 && A[i-1] > B[j]) imax = i - 1;
        else {

            int max_of_left, min_of_right;

            if (i == 0) max_of_left = B[j-1];
            else if (j == 0) max_of_left = A[i-1];
            else max_of_left = A[i-1] > B[j-1] ? A[i-1] : B[j-1];

            if ((n + m) % 2 == 1)
                return max_of_left;

            if (i == n) min_of_right = B[j];
            else if (j == m) min_of_right = A[i];
            else min_of_right = A[i] < B[j] ? A[i] : B[j];

            return (max_of_left + min_of_right) / 2.0;

        }

    }

    return 0;

}
```