Deux projets : tri de tableaux et tracé de fractales

Consignes

— Le projet est à faire de manière individuelle.
— Il est demandé de rendre une archive `nom_prenom.zip` qui contient les fichier `.c` et `.py`.
— L'archive doit être rendue sur Moodle à la fin des 6h de projet.
— Il est conseillé d'utiliser un éditeur directement sur le PC (e.g. Visual Studio Code).

Les deux projets peuvent être faits dans l'ordre que vous voulez.

1 Tri de tableaux (langage C uniquement)

Pour ce projet, l'objectif est de trier un grand nombre de valeurs efficacement. Mais si l'on a plusieurs algorithmes possibles pour trier, lequel est-ce que l'on choisit ? Pour faire ce choix, il faut comparer les algorithmes pour déterminer lequel a les meilleures performances.

1.1 Fonctionnement du programme

L'objectif est de comparer les algorithmes suivants : le tri à bulles, le tri par sélection et le tri à peigne. Ils seront introduit un peu plus bas dans ce document.

Le programme contient une procédure `comparaison` :

1. elle reçoit en paramètre un entier $N \in \mathbb{N}$, un entier $asc \in \mathbb{N}$;
2. elle initialise un tableau de N entiers aléatoires ;
3. si $asc \geq 0$, il faut trier par ordre croissant, sinon par ordre décroissant ;
4. les algorithmes de tri sont utilisés sur une copie du tableau non-trié ;
5. elle affiche le temps d'exécution de chaque algorithme de tri.

1.2 Utilitaires

Générer des valeurs aléatoires La génération de nombre aléatoires requiert `time.h` et peut être réaliser via les instruction

```
srand(time(NULL));
rand() % 100; // nombre entre 0 et 99
```

Mesurer le temps d'exécution Supposons que l'on veuille mesurer le temps d'exécution de `mon_algorithme`, alors via `time.h` on peut écrire

```
clock_t debut = clock();
mon_algorithme();
clock_t fin = clock();
double duree = difftime(fin, debut);
```

et la variable `duree` contient le temps d'exécution en millisecondes.

1.3 Recommandations

Pour faciliter l'implémentation des algorithmes, il est recommandé d'écrire les fonctions suivantes
- void echanger(int* a, int* b) pour échanger deux entiers ;
- void valid_tri(int T[], int n) pour vérifier si le tableau `T` est trié et affiche un message d'erreur si ce n'est pas le cas ;
- void afficher_tab(int T[], int n) pour afficher un tableau de nombres entiers ;
- void remplir_tab(int T[], int n) pour affecter une valeur aléatoire à chaque case du tableau ;
- void copie_tab(int dest [], int T[], int n) qui copie les valeurs du tableau `T` dans le tableau `dest`.

1.4 Algorithmes de tri

Tri par sélection Nous recherchons le maximum de tous les éléments (N), et nous le plaçons en dernière position en échangeant sa position avec l'élément placé en dernier. Il ne reste plus qu'à trier les $N-1$ premiers éléments, pour lesquels nous répétons le même procédé.

Tri à bulles L'algorithme parcourt le tableau, et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que le tableau est trié. Nous arrêtons alors l'algorithme.

```
Procedure tri_bulles(T)
    ech ∈ Bool
    i, j ∈ ℕ
    ech ← True
    i ← longueur(T)
    while i > 0 and ech == True do
        ech ← False
        j ← 0
        while j < i − 1 do
            if T[j] > T[j + 1] then
                echanger(T[j], T[j + 1])
                ech ← True
            end
            j ← j + 1
        end
        i ← i − 1
    end
end
```

Tri à peigne C'est une variante du tri à bulles. On ne compare plus des éléments successifs pour les échanges, mais à la place, on compare des éléments qui sont séparés d'un distance h au niveau de l'indice. Cet indice h est successivement divisé par $1,3$.

```
Procedure tri_peigne(T)
    ech ∈ Bool
    h, j ∈ ℕ
    ech ← True
    h ← longueur(T)
    while h > 1 or ech == True do
        h ← int(h/1.3)
        if h < 1 then
            h ← 1
        end
        ech ← False
        j ← 0
        while j < longueur(T) − h do
            if T[j] > T[j + 1] then
                echanger(T[j], T[j + h])
                ech ← True
            end
            j ← j + 1
        end
    end
end
```

2 Tracé de fractales (langage Python uniquement)

2.1 Première partie, tracé de la fractale Mandelbrot

Dans ce projet, nous avons généré une image qui correspond à ce que l'on appel une fractale. Pour cela, il faudra compléter un les fonctions tel que décrit ci-dessous.

Générations des axes : La fonction `generer_axe` prend en paramètre
- la taille de l'axe, que nous notons ici t
- son centre, que nous notons c
- et le nombre de graduations de l'axe, que nous notons n.

La fonction renvoie en sortie un tableau contenant :

$$c - t/2, c - t/2 + t/n, c - t/2 + 2t/n, c - t/2 + 3t/n, \ldots, c - t/2 + (n-1)3t/n.$$

Calcul des itération de Mandebrot : La fonction `mondelbrot_iter` prend en paramètre
- une valeur x (élément de l'axe des x)
- une valeur y (élément de l'axe des y)

Cette fonction doit calculer les termes de la suite définie via $z_0 = 0$ et

$$z_{k+1} = z_k^2 + c$$

où $c = x + iy$. Lorsque le module de z_k est plus grand que deux, l'algorithme doit s'arrêter et renvoyer k. De plus, pour ne pas s'exécuter indéfiniment, l'algorithme ne dépassera pas $k = 200$ et renverra donc cette valeur si elle est atteinte.

Calcul de l'image : La fonction `tracer_domaine` prend en paramètre
- la taille des axes, que nous notons ici t
- centre, un tableau de deux valeurs qui contient le centre pour l'axe des x et l'axe y
- et le nombre de graduations de chaque axe, que nous notons n.

L'algorithme a pour rôle d'affecter les valeurs à la matrice J qui est de taille $n \times n$. Il parcours les valeurs des axes x et y, et fourni ces valeurs à la fonction `mandelbrot_iter` et stock le résultat dans la matrice J aux indices associés aux axes (indice de l'axe des x pour la colonne et indice de l'axe des y pour les lignes).

2.2 Deuxième partie, tracé d'une fractale Julia

Pour cette partie, il faut implémenter une fonction `iter_julia`. Celle-ci prend en paramètre
- une valeur x (élément de l'axe des x)
- une valeur y (élément de l'axe des y)
- $c \in \mathbb{C}$, une valeur complexe

et calcule les termes de la suite définie via $z_0 = x + iy$ et

$$z_{k+1} = z_k^2 + c.$$

Comme pour le cas de Mandelbrot, lorsque le module de z_k est plus grand que deux, l'algorithme doit s'arrêter et renvoyer k, et ne dépassera pas $k = 200$.

Adapter le code : Comme la fonction précédente dépend du paramètre $c \in \mathbb{C}$, il faut adapter la fonction `tracer_domaine`, afin de pouvoir sélectionner la fractale que l'on souhaite tracer et prendre en compte c si nécessaire.

Tracés de julia : Vous pouvez essayer de tracer la fractale avec les paramètres suivants
- centre $= [0, 0]$
- taille $= 4$
- $n = 1024$
- cmap = "RdGy_r"
- $c = 0 + 0.65j$ ou $c = -1.25 - 0.02j$ ou $c = -0.5 - 0.6j$.