Lab complexité : le parrallélisme

« “Les parallèles, c’est comme les rails d’un train, elles tournent en même temps !” » « Denis Guedj / Le théorème du perroquet »

I. Les arbre binaires

a. La classe Arbre

Construire une classe Arbre qui contient les méthodes suivantes :

- Generer_Noeud(self) : cette méthode retourne un nouveau noeud sous le format que nous spécifierons plus tard.
- Ajouter_Noeud_Aléa(self, nouveau_noeud, Noeud=None) : cette méthode prend un Noeud en paramètre et ajoute un nouveau noeud directement sous ce Noeud, en choisissant aléatoirement sa position (gauche ou droite). Si le Noeud possède déjà 2 feuilles, droite et gauche, alors le nouveau noeud sera ajouté sous l’une de ses feuilles, choisie aléatoirement.
- Rechercher_Cle(self, cle , Noeud=None) : cette méthode recherche une clé parmis les sous-noeud de ‘Noeud’ de Arbre en utilisant un algorithme récursif de parcours postfixe.

La structure du noeud que nous utiliserons tout au long de ce Lab est la suivante :

```
Structure Noeud

    Cle : entier,
    gauche : * Noeud,
    droite : * Noeud,

Fin Structure
```

b. Parcours de l'Arbre Binaire:

Construire les deux méthodes suivantes :

- La méthode to_list_profondeur(self, Arbre=None) qui parcours l'arbre en profondeur et retourne une liste de ses clés.
- La méthode to_list_largeur(self, Arbre=None) qui parcours l'arbre en largeur et retourne une liste de ses clés.

c. Optimisation de l'algorithme de recherche :

Ajouter à la classe les 3 méthodes suivantes :

- Recherche_Cle_Non_Opt(self,cle) : Cette recherche et retourne une cle spécifique dans l'arbre.
- Recherche_Cle_Concurente(self,cle) : Cette recherche et retourne une cle spécifique dans l'arbre. Cette méthode utilise les threads pour parralléliser son fonctionnement.
- Recherche_Cle_Parralèle(self,cle) : Cette recherche et retourne une cle spécifique dans l'arbre. Cette méthode utilise les process pour parralléliser son fonctionnement.

II. Tests du code

a. Temps d'exécution

Ecrire le code qui réalise l'algorithme suivant pour tester les temps d'exécution des algorithmes de recherche parralléle:

- **DataFrame <= Charger CSV**
- **List <= Lire_colonne_entiers(DataFrame)**
- **Arbre <= arbre**
- **Arbre .Remplir_Arbre(List)**
- **List_cle <= Aleatoire(List,5)**
- **Pour cle dans list_cle faire**
    - **Recherche_Cle_Non_Opt(cle)**
    - **Recherche_Cle_Concurente(cle)**
    - **Recherche_Cle_Parralèle(cle)**
- **Fin pour**

b. Sauvegarde dans un fichier csv

Ecrire le code qui réalise l'algorithme suivant :

- **DataFrame <= Charger_csv ()**
- **List <= Lire_colonne_entiers(DataFrame)**
- **Arbre <= arbre**
- **Arbre .Remplir_Arbre(List)**
- **List_pro = Arbre .to_list_profendeur()**
- **List_larg = Arbre . to_list_largeur ()**
- **Ajouter List_Pro à la DF**
- **Ajouter List_Larg à la DF**
- **Sauvegarder DF dans un fichier « .csv »**