Rapport du Projet :

Application de visualisation des algorithmes de parcours, D'un arbre binaire.

Réalisé par:

AMASTIK Oussama

Table des matières

Table of Contents

Introduction Générale :

Dans le domaine de l'informatique, la représentation et la manipulation de données structurées sous forme d'arbres binaires sont primordiales. Elles trouvent des applications variées et essentielles, allant de la gestion des bases de données à l'optimisation des algorithmes de recherche.

Le présent rapport détaille le projet de développement d'une application dédiée à la visualisation interactive des algorithmes de parcours d'un arbre binaire.

L'objectif principal de cette application est double : offrir un outil pédagogique pour les étudiants et professionnels désireux de comprendre en profondeur le fonctionnement des parcours d'arbres binaires, et fournir une plate-forme de test pour l'optimisation de ces algorithmes dans un environnement contrôlé.

Pour atteindre ces fins, le projet a été conçu en utilisant le langage de programmation Python, reconnu pour sa polyvalence et son efficacité, et la bibliothèque graphique Tkinter, qui permet de construire des interfaces utilisateur graphiques intuitives et réactives. Ce rapport commence par exposer les fondements théoriques des arbres binaires et de leurs parcours, puis il s'attarde sur les choix techniques et conceptuels qui ont mené à la réalisation de l'application.

Chapitre 1: Introduction du projet:

Dans ce premier chapitre, nous introduirons le projet en abordant une thématique qui englobe le contexte du projet, exposant ainsi la problématique à laquelle il cherche à répondre. Nous présenterons également les objectifs poursuivis par ce projet, établissant ainsi les bases nécessaires pour une compréhension approfondie de l'ensemble du travail à venir.

1. Thématique :

1.1 Contexte du Projet :

L'évolution rapide du domaine de l'informatique a donné naissance à des structures de données complexes et à des algorithmes sophistiqués. Parmi ces structures, les arbres binaires jouent un rôle central en tant que composants fondamentaux. Ils sont utilisés pour organiser et gérer efficacement des ensembles de données dans divers domaines tels que la recherche, la gestion de bases de données, la conception d'algorithmes, et bien d'autres.

Cependant, la compréhension des arbres binaires et de leurs algorithmes de parcours peut s'avérer difficile pour de nombreux apprenants en informatique, en particulier pour ceux qui sont nouveaux dans le domaine. La nature abstraite de ces concepts, ainsi que l'absence de visualisation interactive, peuvent représenter un obstacle significatif à leur compréhension.

Dans ce contexte, le projet vise à développer une application dédiée à la visualisation des algorithmes de parcours d'un arbre binaire. L'objectif principal de cette application est de permettre aux utilisateurs de visualiser les différents algorithmes de parcours (préfixe, infixe et postfixe) d'un arbre binaire de manière interactive. L'application sera dotée d'une interface graphique conviviale, permettant aux utilisateurs de créer les arbres binaires et de suivre pas à pas ces parcours et d'observer les résultats à chaque étape.

1.2 Problématique :

La compréhension des arbres binaires et de leurs algorithmes de parcours représente un défi pour de nombreux apprenants en informatique.

La nature abstraite de ces concepts, combinée à l'absence de visualisation interactive, peut rendre l'apprentissage difficile.

Comment rendre l'apprentissage des parcours d'arbres binaires plus accessible et engageant pour les étudiants et les professionnels de l'informatique ?

1.3 Objectifs :

Les objectifs du projet de développement d'une application dédiée à la visualisation des algorithmes de parcours d'un arbre binaire peuvent inclure :

1. L'affichage graphique de l'arbre : L'application doit permettre à l'utilisateur de créer un arbre binaire en utilisant une interface conviviale et interactive.

2. La visualisation des parcours : L'utilisateur doit être en mesure de choisir l'algorithme de parcours (préfixe, infixe ou postfixe) qu'il souhaite visualiser et de lancer la visualisation.

3. L'animation des parcours : L'application devra animer les différents parcours de façon à montrer étape par étape le processus, en mettant en évidence les nœuds visités.

4. Les contrôles d'animation : L'utilisateur devra pouvoir démarrer, mettre en pause, reprendre ou arrêter l'animation à tout moment.

Chapitre 2: Modelisation et Outils:

Ce chapitre explore la phase cruciale de modélisation du projet à travers l'utilisation de diagrammes, tels que les diagrammes de cas d'utilisation, mettant en évidence les interactions entre les acteurs et le système. Les diagrammes d'activité sont également employés pour détailler les étapes séquentielles des processus clés.

Parallèlement, nous introduisons les outils technologiques qui ont été essentiels à la réalisation du projet. Python, langage de programmation polyvalent, et Spyder, environnement de développement intégré, occupent une place centrale dans la concrétisation de nos idées. Ces choix résultent de leur adaptabilité, efficacité et compatibilité avec les objectifs du projet.

Ce chapitre établit ainsi un socle solide pour la compréhension de notre approche méthodologique, jetant les bases nécessaires à la mise en œuvre pratique du projet.

2. Modelisation:

UML (Unified Modeling Language) que l'on peut traduire par « langage de modélisation unifié » est un langage formel et normalisé en termes de modélisation objet. Son indépendance par rapport aux langages de programmation, aux domaines de l'application et aux processus, son caractère polyvalent et sa souplesse ont fait lui un langage universel.

En plus UML est essentiellement un support de communication, qui facilite la représentation et la compréhension de solution objet. Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation des solutions. L'aspect de sa notation, limite l'ambigüité et les incompréhensions.

UML fournit un moyen astucieux permettant de représenter diverses projections d'une même représentation grâce aux vues.

Une vue est constituée d'un ou plusieurs diagrammes. Dans notre cas, on a élaboré 2 diagrammes de vue statique : Diagramme de cas d'utilisation et diagramme de classe, et aussi un diagramme de vue dynamique : Diagramme de séquence.

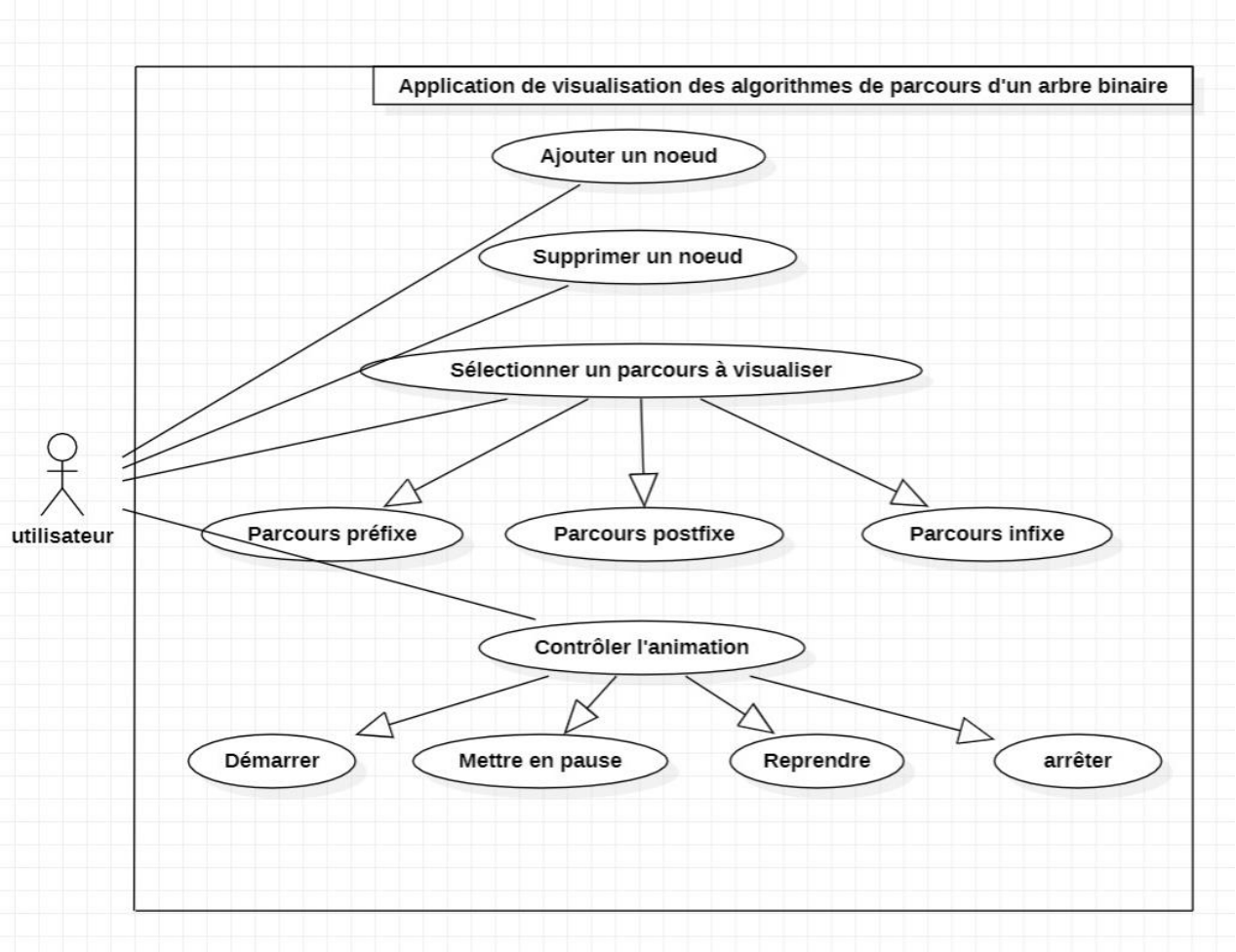
2.1 Diagramme de cas d'utilisation:

Le diagramme de cas d'utilisation est un outil de modélisation qui permet de représenter les interactions entre les acteurs, tels que les utilisateurs et les systèmes externes, ainsi que les différentes fonctionnalités offertes par un système.

Son objectif principal est de mettre en avant les objectifs fonctionnels du système, plutôt que de se concentrer sur les détails techniques de sa réalisation.

En d'autres termes, il sert à visualiser les scénarios d'utilisation et les relations entre les acteurs et les cas d'utilisation, afin de mieux comprendre les besoins et les fonctionnalités essentielles du système.

Diagramme de cas d'utilisation du projet :



Description des cas d'utilisation de l'utilisateur :

- **Créer un Arbre Binaire :**
  - L'utilisateur a la possibilité de créer un nouvel arbre binaire en spécifiant les valeurs des nœuds.
  - L'utilisateur peut sélectionner l'algorithme de parcours qu'il souhaite visualiser parmi les options : préfixe, infixe, postfixe ou en largeur.
- **Lancer la Visualisation des Parcours :**

- Une fois l'algorithme de parcours choisi, l'utilisateur peut lancer la visualisation pour observer étape par étape le processus de parcours de l'arbre.

- **Supprimer un Nœud de l'Arbre Binaire** :
  - L'utilisateur a la possibilité de supprimer un nœud spécifique de l'arbre en indiquant sa valeur.
  - La suppression du nœud peut entraîner des ajustements dans la structure de l'arbre.
- **Démarrer l'Animation des Parcours :**

Après avoir choisi l'algorithme et lancé la visualisation, l'utilisateur peut démarrer l'animation pour observer la séquence de parcours.

- **Mettre en Pause, Reprendre, Arrêter l'Animation :**
  - Pendant l'animation, l'utilisateur a la possibilité de mettre en pause, reprendre ou arrêter l'animation selon ses besoins.

- **Description des cas d'utilisation du système :**

- **Créer un Arbre Binaire :**
  - Le système prend en charge la création d'un nouvel arbre binaire en instanciant la classe ArbreBinaire.
  - Il crée la racine de l'arbre et permet l'ajout de nœuds.
- **Choisir un Algorithme de Parcours :**
  - Le système offre à l'utilisateur la possibilité de choisir parmi les algorithmes de parcours disponibles (préfixe, infixe, postfixe, en largeur).
- **Lancer la Visualisation des Parcours :**
  - Le système prépare l'arbre pour le parcours sélectionné et initialise les paramètres nécessaires à la visualisation.
- **Supprimer un Nœud de l'Arbre Binaire :**
  - Le système gère la suppression d'un nœud spécifié par l'utilisateur, en ajustant la structure de l'arbre en conséquence.
- **Démarrer l'Animation des Parcours :**
  - Le système orchestre l'animation en itérant à travers les nœuds de l'arbre selon l'algorithme de parcours choisi.
- **Contrôler l'Animation :**

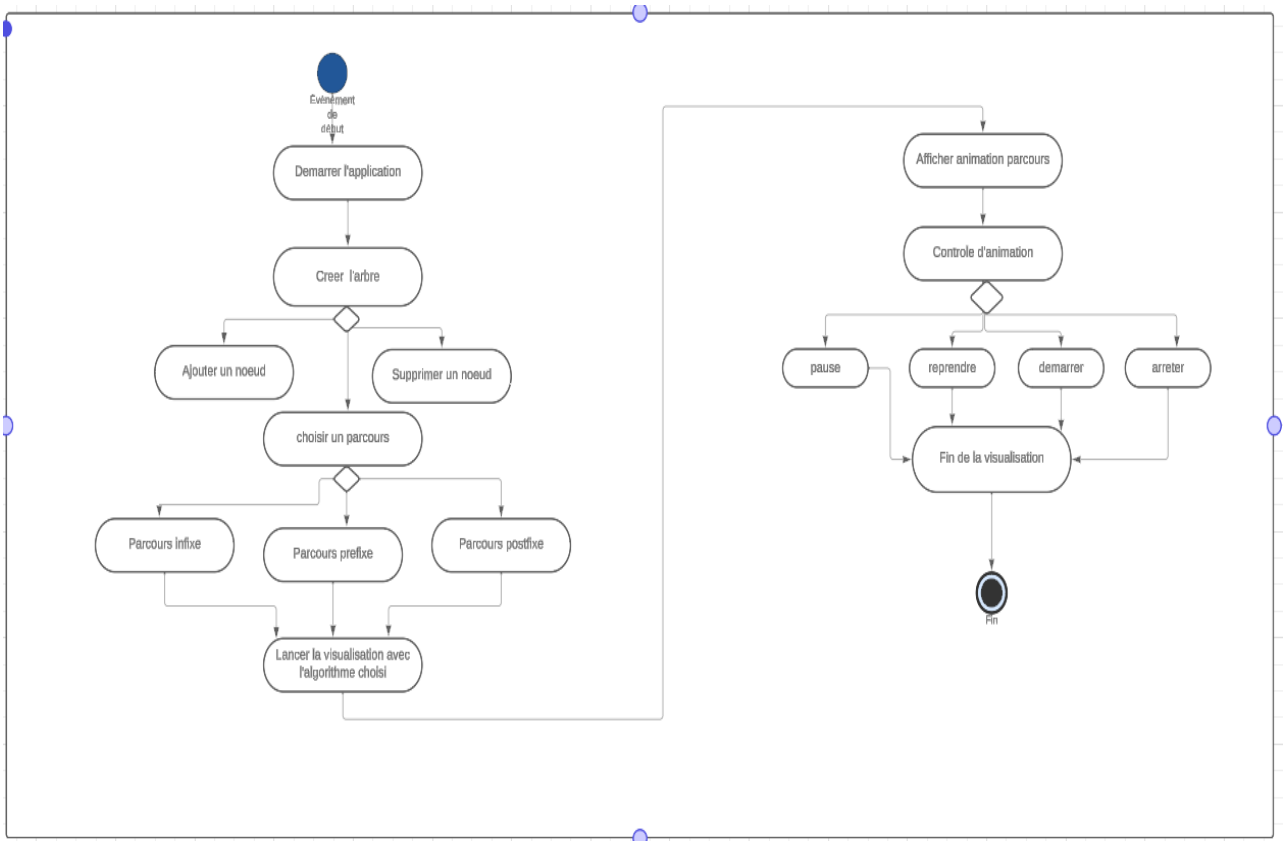
- Le système répond aux commandes de l'utilisateur pour mettre en pause, reprendre ou arrêter l'animation en cours.

2.2 Diagramme d'activites:

Le diagramme d'activités est un type de diagramme UML (Unified Modeling Language) utilisé pour modéliser le flux de contrôle ou le flux d'activités au sein d'un système, d'un processus ou d'un cas d'utilisation.

Il représente graphiquement les étapes et les décisions d'un processus ou d'une activité, montrant comment les différents éléments interagissent entre eux.

Diagramme d'activités pour le projet :



3. Outils:

3.1 Langages, Outils Et Bibliothèques Utilisées :

1. PYTHON :

Python est un langage de programmation interprété, de haut niveau et polyvalent. Il a été créé par Guido van Rossum et publié pour la première fois en 1991. Python se distingue par sa syntaxe claire et lisible, ce qui en fait un langage très apprécié des développeurs.

2. Star Uml :

Star UML est un outil de modélisation UML utilisé pour concevoir des diagrammes de modèles logiciels. Il permet aux développeurs de créer des représentations visuelles des structures et des comportements d'un système logiciel, ce qui facilite la communication et la compréhension entre les membres d'une équipe de développement.

3. Tkinter:

Tkinter constitue une bibliothèque graphique native du langage de programmation Python. Son appellation provient de "Tk interface," faisant référence à la boîte à outils graphique "Tk," une solution largement utilisée pour concevoir des interfaces utilisateur graphiques (GUI). Tkinter met à disposition des développeurs des modules Python dédiés à la création d'éléments essentiels tels que fenêtres, boutons, champs de texte, listes, etc. Cette bibliothèque simplifie ainsi le processus de développement d'applications en offrant une interface utilisateur visuelle, favorisant une expérience utilisateur intuitive et interactive.

4. Spyder:

Spyder se présente comme un environnement de développement intégré (IDE) open source, élaboré expressément pour la programmation en langage Python. Il propose un ensemble exhaustif d'outils dédiés aux développeurs et aux scientifiques de données œuvrant dans le domaine spécifique de l'analyse de données, du traitement numérique, et de la science des données. Spyder s'affirme ainsi comme une plateforme de choix pour les professionnels engagés dans ces secteurs, mettant à leur disposition des fonctionnalités adaptées à leurs besoins spécifiques en matière de programmation Python.

Chapitre 3: Implémentation et Realisation:

Ce chapitre comportera une explication détaillée des fonctions utilisés dans le code source, accompagnée d'une démonstration illustrative des résultats obtenus.

3. Implémentation

Pour l'implémentation, nous allons diviser le code en trois parties distinctes. La première partie sera dédiée à la logique de l'arbre, la deuxième partie abordera la logique derrière chaque bouton, et enfin, la troisième partie se concentrera sur l'implémentation de l'interface graphique.

3.1 Partie 1(logique de l'arbre) :

3.1.1 Commençant avec la class Nœud :

La classe **Nœud** représente un nœud dans un arbre binaire, avec une valeur associée et des références vers les nœuds fils gauche et droit :

code_pour_rapport.py > ArbreBinaire > inserer

```
1
2   # Classe représentant un nœud dans un arbre binaire
3   class Noeud:
4       def __init__(self, valeur):
5           self.valeur = valeur
6           self.gauche = None
7           self.droite = None
8
```

3.1.2 La class arbre :

La classe **ArbreBinaire** représente une structure d'arbre binaire avec des opérations telles que l'insertion, la suppression et le parcours des nœuds :

```
9   # Classe représentant un arbre binaire
10  class ArbreBinaire:
11      def __init__(self):
12          self.racine = None  # Initialise la racine de l'arbre à None
13          self.ligne_ids = {}  # Dictionnaire pour stocker les identifiants des lignes utilisées pour dessiner l'arbre
14
15      # Méthode pour insérer une valeur dans l'arbre
16      def inserer(self, valeur):
17          if self.racine is None:
18              self.racine = Noeud(valeur)
19          else:
20              self._inserer(valeur, self.racine)
21
```

3.1.3 La fonction insérer :

La fonction **insérer** de la classe **ArbreBinaire** insère un nouveau nœud avec une valeur donnée dans l'arbre binaire, en respectant les propriétés d'un arbre binaire, où les valeurs inférieures sont situées du côté gauche et les valeurs supérieures du côté droit.

```
22      # Méthode auxiliaire récursive pour l'insertion d'une valeur dans l'arbre
23      def _inserer(self, valeur, noeud):
24          if valeur < noeud.valeur:
25              if noeud.gauche is None:
26                  noeud.gauche = Noeud(valeur)
27              else:
28                  self._inserer(valeur, noeud.gauche)
29          elif valeur > noeud.valeur:
30              if noeud.droite is None:
31                  noeud.droite = Noeud(valeur)
32              else:
33                  self._inserer(valeur, noeud.droite)
34
```

3.1.4 La fonction supprimer :

La fonction **_supprimer** de la classe **ArbreBinaire** permet de supprimer un nœud avec une valeur donnée de l'arbre binaire, en respectant les propriétés d'un arbre binaire. Elle gère différents cas, tels que la suppression d'un nœud sans enfant, avec un seul enfant, ou avec deux enfants, en réorganisant l'arbre de manière appropriée.

```
35      # Méthode pour supprimer une valeur de l'arbre
36      def supprimer(self, valeur):
37          self.racine = self._supprimer(self.racine, valeur)
38
39      # Méthode auxiliaire récursive pour la suppression d'une valeur dans l'arbre
40      def _supprimer(self, noeud, valeur):
41          if noeud is None:
42              return noeud
43          if valeur < noeud.valeur:
44              noeud.gauche = self._supprimer(noeud.gauche, valeur)
45          elif valeur > noeud.valeur:
46              noeud.droite = self._supprimer(noeud.droite, valeur)
47          else:
48              if noeud.gauche is None:
49                  return noeud.droite
50              elif noeud.droite is None:
51                  return noeud.gauche
52              temp_noeud = self._min_value_node(noeud.droite)
53              noeud.valeur = temp_noeud.valeur
54              noeud.droite = self._supprimer(noeud.droite, temp_noeud.valeur)
55          return noeud
56
```

3.1.5 La fonction min value :

La fonction **_min_value_node** de la classe **ArbreBinaire** retourne le nœud contenant la valeur minimale dans un sous-arbre donné. Elle traverse le sous-arbre vers la gauche jusqu'à atteindre le nœud le plus à gauche, qui contient la valeur minimale dans cet arbre binaire.

```
57      # Méthode auxiliaire pour trouver le nœud avec la valeur minimale dans un sous-arbre
58      def _min_value_node(self, noeud):
59          current = noeud
60          while current.gauche is not None:
61              current = current.gauche
62          return current
```

3.1.6 Le parcours préfixe :

La fonction **parcours_prefixe** dans la classe **ArbreBinaire** effectue un parcours en profondeur préfixe de l'arbre binaire. Elle visite chaque nœud en suivant l'ordre : nœud actuel, nœud de gauche, nœud de droite. Pendant le parcours, elle met en évidence chaque nœud sur le canevas graphique en utilisant la fonction **mettre_en_evidence**. Les valeurs des nœuds visités sont ajoutées à la liste résultat, qui est utilisée pour afficher le résultat du parcours. La fonction prend également en charge la pause et l'arrêt de l'animation pour permettre une interaction utilisateur pendant l'exécution de l'algorithme.

```
91      # Méthode pour effectuer un parcours préfixe de l'arbre avec mise en évidence des nœuds
92      def parcours_prefixe(self, noeud, resultat, canvas, x, y, espace, parent_pos=None):
93          if noeud:
94              self.mettre_en_evidence(canvas, x, y, espace, noeud, parent_pos)
95              resultat.append(noeud.valeur)
96              if noeud.gauche:
97                  self.parcours_prefixe(noeud.gauche, resultat, canvas, x - espace, y + 70, espace / 2, parent_pos=(x, y))
98              if noeud.droite:
99                  self.parcours_prefixe(noeud.droite, resultat, canvas, x + espace, y + 70, espace / 2, parent_pos=(x, y))
```

3.1.7 Le parcours infixe :

La fonction **parcours_infixe** dans la classe **ArbreBinaire** effectue un parcours en profondeur infixe de l'arbre binaire. Elle visite chaque nœud en suivant l'ordre : nœud de gauche, nœud actuel, nœud de droite. Pendant le parcours, elle met en évidence chaque nœud sur le canevas graphique en utilisant la fonction **mettre_en_evidence**. Les valeurs des nœuds visités sont ajoutées à la liste **resultat**, qui est utilisée pour afficher le résultat du parcours. La fonction prend également en charge la pause et l'arrêt de l'animation pour permettre une interaction utilisateur pendant l'exécution de l'algorithme.

```
101     # Méthode pour effectuer un parcours infixe de l'arbre avec mise en évidence des nœuds
102     def parcours_infixe(self, noeud, resultat, canvas, x, y, espace, parent_pos=None):
103         if noeud:
104             if noeud.gauche:
105                 self.parcours_infixe(noeud.gauche, resultat, canvas, x - espace, y + 70, espace / 2, parent_pos=(x, y))
106             self.mettre_en_evidence(canvas, x, y, espace, noeud, parent_pos)
107             resultat.append(noeud.valeur)
108             if noeud.droite:
109                 self.parcours_infixe(noeud.droite, resultat, canvas, x + espace, y + 70, espace / 2, parent_pos=(x, y))
110
```

3.1.8 Le parcours postfix :

La fonction **parcours_postfixe** dans la classe **ArbreBinaire** effectue un parcours en profondeur postfixe de l'arbre binaire. Elle visite chaque nœud en suivant l'ordre : nœud de gauche, nœud de droite, nœud actuel. Pendant le parcours, elle met en évidence chaque nœud sur le canevas graphique en utilisant la fonction **mettre_en_evidence**. Les valeurs des nœuds visités sont ajoutées à la liste **resultat**, qui est utilisée pour afficher le résultat du parcours. La fonction prend également en charge la pause et l'arrêt de l'animation pour permettre une interaction utilisateur pendant l'exécution de l'algorithme.

```
111     # Méthode pour effectuer un parcours postfixe de l'arbre avec mise en évidence des nœuds
112     def parcours_postfixe(self, noeud, resultat, canvas, x, y, espace, parent_pos=None):
113         if noeud:
114             if noeud.gauche:
115                 self.parcours_postfixe(noeud.gauche, resultat, canvas, x - espace, y + 70, espace / 2, parent_pos=(x, y))
116             if noeud.droite:
117                 self.parcours_postfixe(noeud.droite, resultat, canvas, x + espace, y + 70, espace / 2, parent_pos=(x, y))
118             self.mettre_en_evidence(canvas, x, y, espace, noeud, parent_pos)
119             resultat.append(noeud.valeur)
```

3.1.9 La fonction position du nœud :

La fonction **position_noeud** dans la classe **ArbreBinaire** retourne la position (coordonnées x, y) d'un nœud donné dans l'arbre binaire, en fonction de sa valeur. Elle effectue une recherche en largeur **(BFS)** pour parcourir l'arbre jusqu'à trouver le nœud avec la valeur spécifiée. Les coordonnées x, y sont mises à jour à mesure que l'algorithme parcourt l'arbre.

Cette fonction est utilisée pour obtenir la position d'un nœud spécifique afin de l'illustrer ou de le mettre en évidence sur le canevas graphique.

```
121     # Méthode pour obtenir la position d'un nœud dans le canvas
122     def position_noeud(self, noeud, x, y, espace):
123         if noeud is None:
124             return None
125         queue = [(self.racine, x, y)]
126         while queue:
127             current, cur_x, cur_y = queue.pop(0)
128             if current == noeud:
129                 return cur_x, cur_y
130             if current.gauche:
131                 queue.append((current.gauche, cur_x - espace, cur_y + 70))
132             if current.droite:
133                 queue.append((current.droite, cur_x + espace, cur_y + 70))
134         return None
```

3.1.10 La fonction trouver parent :

La fonction **_trouver_parent** dans la classe **ArbreBinaire** permet de trouver le parent d'un nœud donné dans l'arbre binaire. Elle effectue une recherche en largeur **(BFS)** jusqu'à ce qu'elle trouve le nœud spécifié, et elle retourne le parent de ce nœud. Cette fonction est utilisée dans le contexte de l'animation graphique pour mettre en évidence la connexion entre un nœud et son parent.

```
136     # Méthode auxiliaire pour trouver le parent d'un nœud
137     def _trouver_parent(self, child):
138         parent = None
139         queue = [(self.racine, None)]
140         while queue:
141             current, current_parent = queue.pop(0)
142             if current == child:
143                 return current_parent
144             if current.gauche:
145                 queue.append((current.gauche, current))
146             if current.droite:
147                 queue.append((current.droite, current))
148         return parent
149
```

3.2 Partie 2 (Fonction liée aux buttons) :

3.2.1 Le button insérer nœud :

Cette fonction est appelée lorsque l'utilisateur clique sur le bouton "Insérer un nœud". Elle récupère la valeur entrée par l'utilisateur, l'insère dans l'arbre binaire à l'aide de la méthode **inserer** de la classe **ArbreBinaire**, puis met à jour le canevas en redessinant l'arbre.

```
def inserer_noeud():
    valeur = valeur_insertion.get()  # Récupère la valeur depuis la zone de texte
    if valeur:
        valeur = int(valeur)  # Convertit la valeur en entier
        arbre.inserer(valeur)  # Insère la valeur dans l'arbre binaire
        canvas.delete("all")  # Efface le canevas
        arbre.dessiner(canvas, 400, 30, 200)  # Redessine l'arbre avec la nouvelle configuration
    entry_inserer.pack_forget()  # Cache la zone de texte d'insertion
    btn_inserer.pack_forget()  # Cache le bouton insérer
    valeur_insertion.set('')  # Efface la zone de texte après l'insertion
    hide_widgets()  # Cache les widgets après l'insertion
```

3.2.2 Le button supprimer un nœud :

La fonction **supprimer_noeud** supprime un nœud de l'arbre binaire en fonction de la valeur fournie par l'utilisateur. Elle efface ensuite le canevas et redessine l'arbre mis à jour. Les éléments d'interface, tels que la zone de texte de suppression, sont également masqués après l'opération.

```
def supprimer_noeud():
    valeur = valeur_suppression.get()  # Récupère la valeur depuis la zone de texte
    if valeur:
        valeur = int(valeur)  # Convertit la valeur en entier
        arbre.supprimer(valeur)  # Supprime la valeur de l'arbre binaire
        canvas.delete("all")  # Efface le canevas
        arbre.dessiner(canvas, 400, 30, 200)  # Redessine l'arbre avec la nouvelle configuration
entry_supprimer.pack_forget()  # Cache la zone de texte de suppression
btn_supprimer.pack_forget()  # Cache le bouton supprimer
valeur_suppression.set('')  # Efface la zone de texte après la suppression
hide_widgets()  # Cache les widgets après la suppression
```

3.2.3 Le button du parcours prefixe :

La fonction **afficher_parcours** est liée au bouton "Parcours Préfixe". Elle effectue un parcours préfixe de l'arbre binaire, mettant en évidence chaque nœud visité visuellement sur le canevas. Le résultat du parcours est affiché dans un label sur l'interface graphique. La fonction prend en charge la gestion de l'animation, de la pause et de l'arrêt pour une expérience utilisateur interactive.

3.2.4 Le button de parcours suffixe :

La fonction associée au bouton Parcours Postfixe réalise le parcours **postfixe** de l'arbre binaire. Elle met en évidence visuellement chaque nœud visité sur le canevas, tout en mettant à jour le résultat du parcours qui est affiché dans un label sur l'interface graphique. La fonction prend en charge la gestion de l'animation, de la pause et de l'arrêt pour offrir une expérience interactive à l'utilisateur.

3.2.5 Le button de parcours infixe :

La fonction associée au bouton "Parcours Infixe" effectue un parcours infixe de l'arbre binaire. Elle met en évidence visuellement chaque nœud visité sur le canevas, tout en mettant à jour le résultat du parcours qui est affiché dans un label sur l'interface graphique. La fonction prend en charge la gestion de l'animation, de la pause et de l'arrêt pour offrir une expérience interactive à l'utilisateur.

- Voici la fonction qui contrôle les trois parcours :

```
def afficher_parcours(mode):
    global is_paused, is_stopped
    is_paused = False
    is_stopped = False
    hide_widgets()
    resultat = []
    espace_initial = 200
    x_initial = 400
    y_initial = 30
    canvas.delete("all")
    arbre.dessiner(canvas, x_initial, y_initial, espace_initial)

    # Réinitialiser le label_resultat_parcours avant le parcours
    label_resultat_parcours.pack_forget()
    resultat_parcours.set("")

    if mode == "prefixe":
        arbre.parcours_prefixe(arbre.racine, resultat, canvas, x_initial, y_initial, espace_initial)
    elif mode == "infixe":
        arbre.parcours_infixe(arbre.racine, resultat, canvas, x_initial, y_initial, espace_initial)
    elif mode == "postfixe":
        arbre.parcours_postfixe(arbre.racine, resultat, canvas, x_initial, y_initial, espace_initial)

    label_resultat_parcours.pack(pady=10, padx=5, fill='x')  # Affiche le label de résultat

    # Mettre à jour le StringVar avec le résultat du parcours
    resultat_parcours.set(', '.join(str(x) for x in resultat))
    # Affichez le label
    label_resultat_parcours.pack(pady=10, padx=5, fill='x')
```

3.2.6 Les buttons d'animation (Pause, Play et Stop) :

Ces fonctions permettent de contrôler l'animation du parcours, offrant à l'utilisateur une expérience interactive pendant la visualisation de l'arbre binaire :

- La fonction associée au bouton **"Play"** (▶) reprend l'animation du parcours en cours, permettant à l'utilisateur de visualiser la progression du parcours sur le canevas.

- La fonction liée au bouton **"Pause"** (■) met en pause l'animation en cours, offrant à l'utilisateur la possibilité de suspendre temporairement le parcours.

- La fonction liée au bouton **"Stop"** (⏸) arrête complètement l'animation en cours et réinitialise l'état du parcours, offrant à l'utilisateur la possibilité de mettre fin au parcours en cours.

```
def main():
    root = tk.Tk()
    root.title("Manipulation d'un arbre binaire")
    arbre = ArbreBinaire()

    # Sidebar styling
    sidebar = tk.Frame(root, bg='#008888', width=600, height=600)
    sidebar.pack(expand=False, fill='y', side='left', anchor='nw',)


    # Variables d'état pour l'animation
    is_paused = False
    is_stopped = False

    def pause_animation():
        global is_paused
        is_paused = True

    def resume_animation():
        global is_paused
        is_paused = False

    def stop_animation():
        global is_stopped, is_paused
        is_stopped = True
        is_paused = False

    def start_animation():
            global is_stopped, is_paused
            is_stopped = False
```

Partie 3 : (l'interface graphique) :

L'interface graphique (GUI) que nous avons utilisé est Tkinter, une bibliothèque standard de Python pour créer des interfaces graphiques.

Voici une explication détaillée des composants de l'interface graphique et de leurs fonctionnalités :

1. Fenêtre Principale (root) :

- La fenêtre principale est créée avec `tk.Tk()` et porte le nom "Manipulation d'un arbre binaire".
- C'est la fenêtre principale qui contient tous les autres éléments de l'interface graphique.

2. Barre Latérale sidebar) :

- La barre latérale est un cadre (`Frame`) qui occupe la partie gauche de la fenêtre principale.
- Elle contient des boutons et des zones de texte pour l'interaction avec l'utilisateur.

3. Boutons pour l'Insertion et la Suppression :

- Deux boutons "Insérer un nœud" et "Supprimer un nœud" sont créés avec `ttk.Button`.
- Lorsqu'un de ces boutons est cliqué, cela déclenche l'affichage de la zone de texte correspondante (`entry_inserer` ou `entry_supprimer`) et du bouton associé (`btn_inserer` ou `btn_supprimer`).

4. Zone de Texte pour l'Insertion (entry_inserer) :

- La zone de texte est créée avec `ttk.Entry` et permet à l'utilisateur de saisir une valeur pour l'insertion d'un nœud.
- La zone de texte est affichée lorsque le bouton "Insérer un nœud" est cliqué.

5.Zone de Texte pour la Suppression (entry_supprimer) :

- La zone de texte est créée avec `ttk.Entry` et permet à l'utilisateur de saisir une valeur pour la suppression d'un nœud.
- La zone de texte est affichée lorsque le bouton "Supprimer un nœud" est cliqué.

6. Boutons pour les Parcours :

- Trois boutons pour les parcours préfixe, infixe et postfixe sont créés avec `ttk.Button`.
- Chaque bouton déclenche l'affichage du parcours correspondant dans l'arbre binaire.

7. Canvas (canvas) :

- Le canevas est créé avec `tk.Canvas` et occupe la majeure partie de la fenêtre principale.
- Il est utilisé pour dessiner et afficher l'arbre binaire ainsi que les animations de parcours.

8. Résultat du Parcours (label_resultat_parcours) :

- Le résultat du parcours est affiché dans une étiquette créée avec `tk.Label`.
- L'étiquette est mise à jour dynamiquement lorsqu'un parcours est effectué.

9. Boutons de Contrôle de l'Animation :

- Trois boutons avec des symboles Unicode pour jouer, mettre en pause et arrêter l'animation sont créés avec `ttk.Button`.
- Ces boutons permettent à l'utilisateur de contrôler le déroulement des animations de parcours.

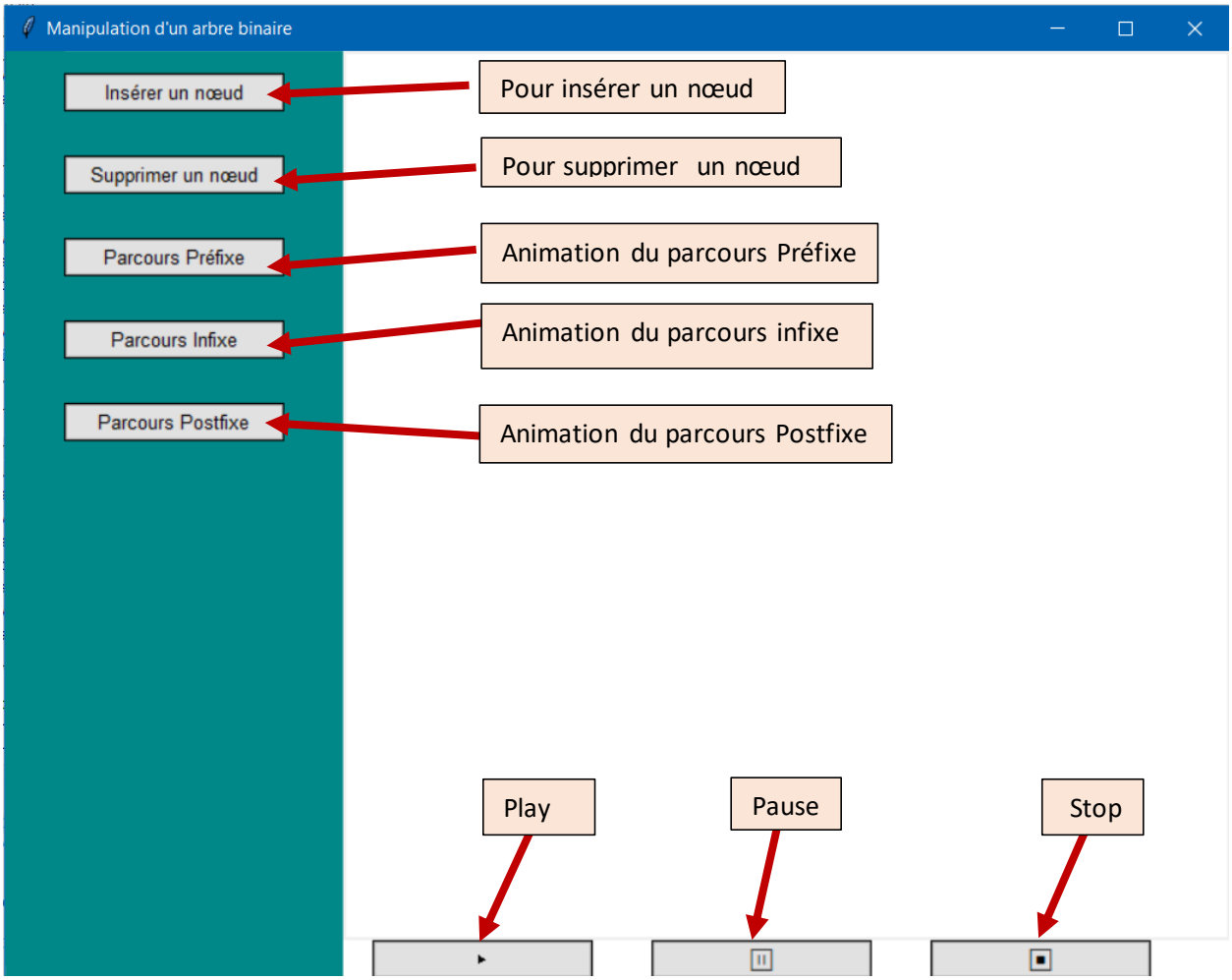
10. Boucle Principale (root.mainloop())

- La boucle principale est lancée à la fin du script avec `root.mainloop()`.
- C'est elle qui maintient l'interface graphique en attente des interactions de l'utilisateur.

4. La réalisation :

Pour l'implémentation, nous allons vous montrer le résultat du code mentionné précédemment et vous présenter la version finale de notre application, accompagnée d'une démonstration :

Voici la page principale du notre application :

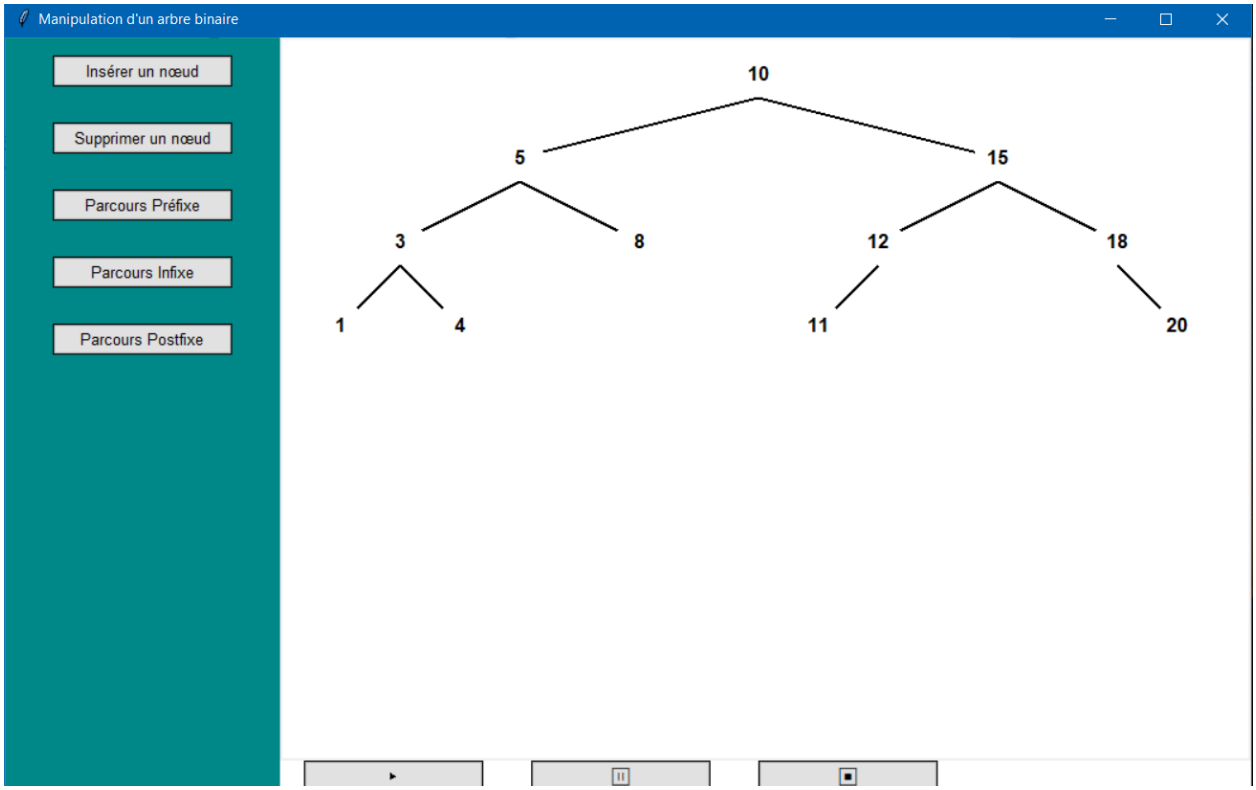


Ce screen démontre le rôle de chaque button dans l'App .

Maintenant prenant un exemple d'un arbre sous forme d'une liste nommée L et on va essayiez d'initialiser la même liste dans l'App :

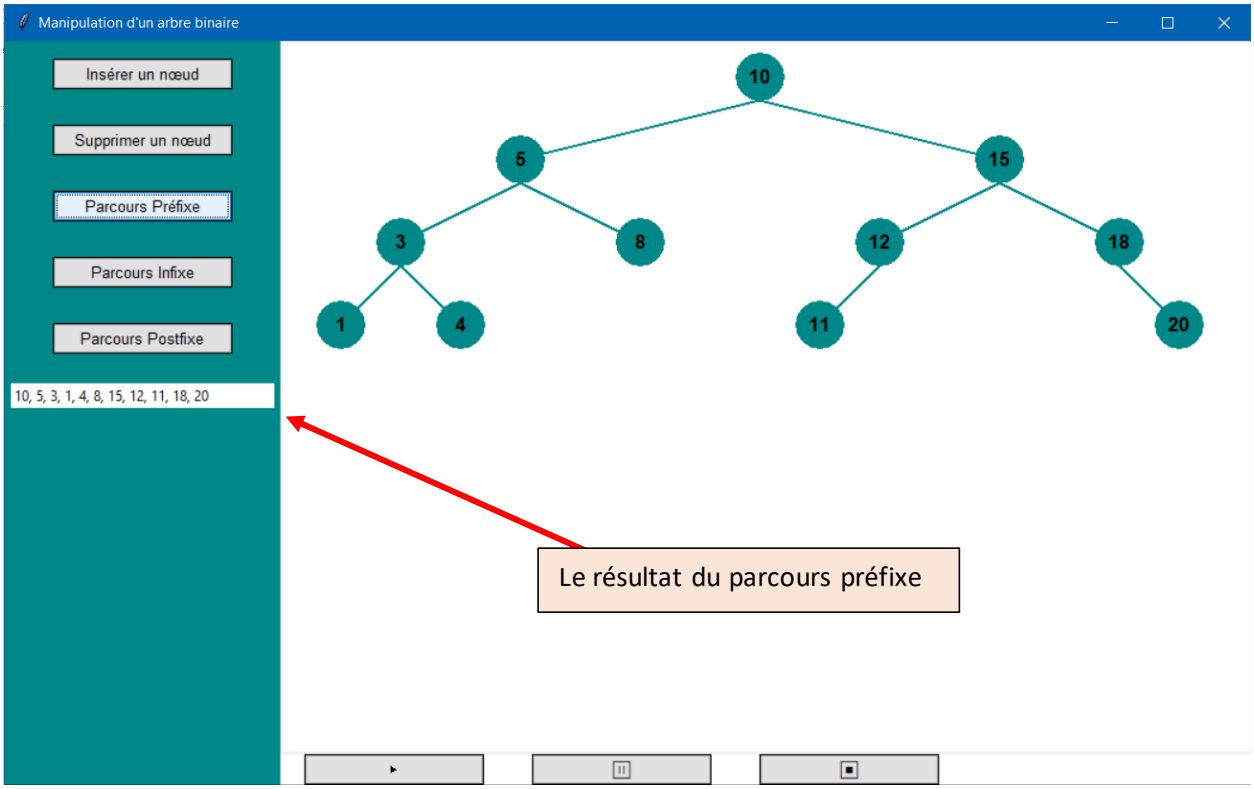
L = [10, [5, [3, [1, None, None], [4, None, None]], [8, None, None]], [15, [12, [11, None, None], None], [18, None, [20, None, None]]]]

Et voici comment va apparaitre dans l'application :



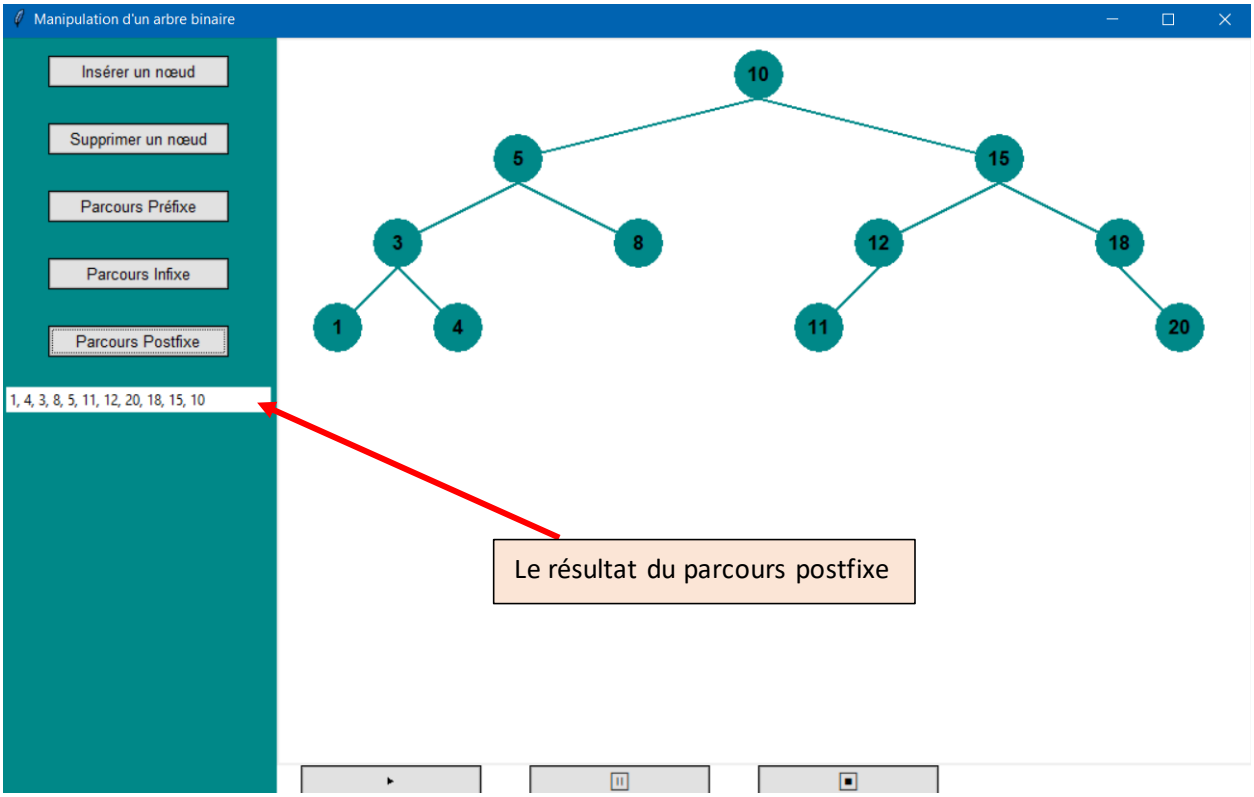
Passant à l'animation on va essayiez de parcourir cette liste avec les trois type Prefixe, Infixe et Postfixe :

- **Parcours Préfixe :**



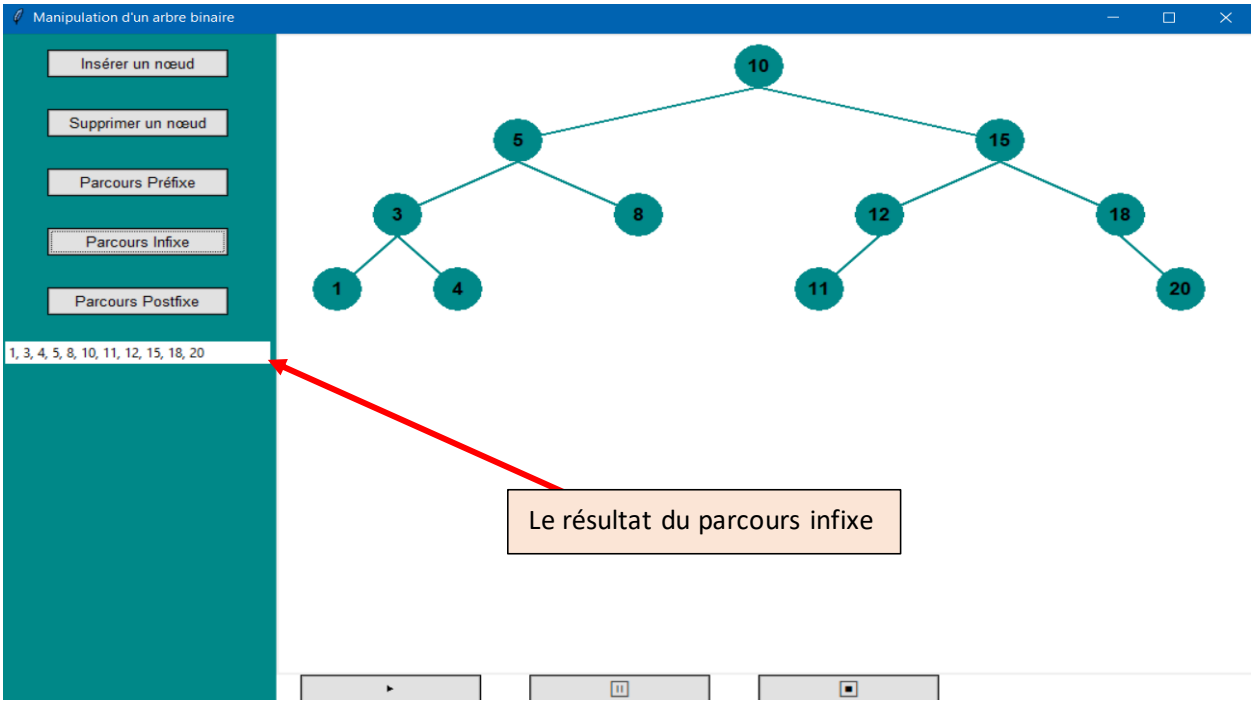
Le résultat de parcours préfixe du l'arbre **L** est [10,5,3,1,4,8,15,12,11,18,20].

- Le parcours **Postfixe** :



Le résultat de parcours postfixe du l'arbre **L** est [1 ,4,3,8,5,11,12,20,18,15,10].

- Le parcours Infixe :



Conclusion générale

Ce projet de création d'une application pour la manipulation d'un arbre binaire a été réalisé en utilisant la bibliothèque Tkinter pour l'interface graphique en Python. L'application permet d'insérer et de supprimer des nœuds dans un arbre binaire, d'effectuer des parcours préfixe, infixe et postfixe animés, tout en offrant un contrôle interactif sur le déroulement des animations.

L'utilisation de structures de données telles que les listes pour représenter l'arbre et l'implémentation de différentes méthodes pour les opérations sur l'arbre montrent une approche orientée objet.

L'interface graphique offre une expérience conviviale à l'utilisateur, permettant une interaction aisée avec l'arbre binaire. Les animations ajoutent une dimension visuelle qui facilite la compréhension des opérations effectuées sur l'arbre.

Ce projet illustre la combinaison de concepts de programmation objet, d'algorithmes sur les arbres binaires, et d'éléments d'interface utilisateur pour créer une application interactive et éducative.

En résumé, ce projet démontre la création réussie d'une application interactive pour la manipulation d'arbres binaires, intégrant des fonctionnalités d'insertion, de suppression et de parcours, le tout présenté de manière visuellement attrayante.