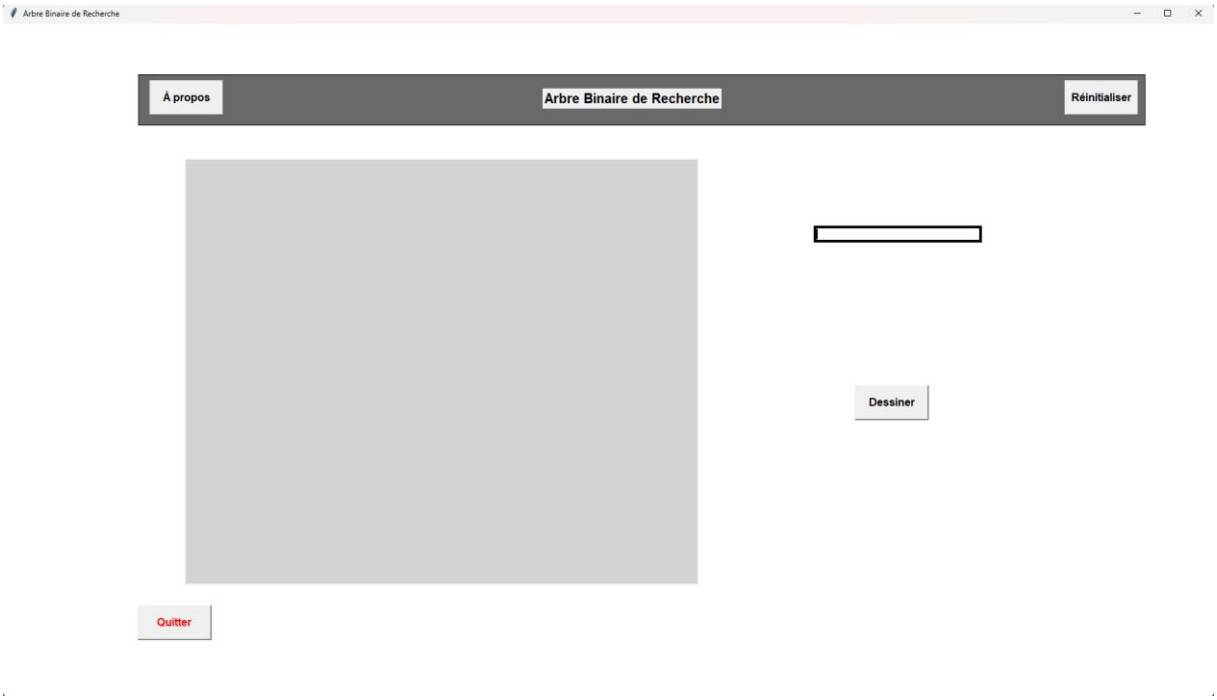
Compte rendu Project arbre binaire de recherche



Merlin Kylian | Marchand Baptiste T-GD

Table des matières

Introduction

Dans le cadre de ce projet, nous avons pour objectif de concevoir une interface graphique permettant de représenter un arbre binaire de recherche en fonction des clés saisies par l'utilisateur. L'interface devra offrir une expérience intuitive, avec des fonctionnalités essentielles comme l'affichage graphique de l'arbre, la possibilité de réinitialiser la saisie, ainsi que des options d'information et de fermeture.

Ce projet s'appuie sur le module tkinter pour la création de l'interface utilisateur et impose certaines contraintes techniques, notamment une limitation de la hauteur de l'arbre à 4 niveaux et une résolution d'affichage adaptée aux standards d'écrans (inférieure à 1920 x 1080 pixels). De plus, l'arbre devra respecter les principes des arbres binaires de recherche, où chaque nœud possède au maximum deux fils, le sous-arbre gauche contenant des valeurs inférieures à la racine et le sous-arbre droit des valeurs supérieures.

L'objectif final est d'obtenir une application fonctionnelle et ergonomique, facilitant la visualisation des arbres binaires de recherche et leur exploration à travers différentes interactions.

1. Répartition des tâches

Ce projet a été réalisé en binôme. Les tâches ont été réparties en fonction de nos préférences et de nos compétences dans chaque domaine demandé.

1.1 : Travaille réaliser par Kylian Merlin

- Code des classes Nœud et ArbreBinaire

- Algorithmes des fonctions dans les classes Nœud et ArbreBinaire

1.2 : Travaille réaliser par Baptiste Marchand

- Code de la classe Application

- Algorithmes des fonctions dans la classe Application

1.3 : Travaille réaliser à deux

- Algorithme du programme principal

- Compte rendu

- Plan de conception

2. Schéma des classes

Cette section présente les différentes classes utilisées dans le projet, en détaillant leurs attributs et méthodes.

2.1 : Classe Nœud

Noeud
Attribut : - valeur - sag - sad - hauteur
Méthode : -__init__() le constructeur - ajouter()

Dans la classe Nœud, tous les attributs et méthodes sont publics.

2.2 : Classe ArbreBinaire

ArbreBinaire
Attribut : - racine
Méthode : - __init__() le constructeur - ajouter() - afficher() - _afficher_noeud()

Dans la classe ArbreBinaire, tous les attributs sont publics. Toutes les méthodes sont publiques, sauf _afficher_noeud() qui est privée.

2.3 : Classe Application

Application
Attribut : - _fenetre - _arbre - _canvas - _entry
Méthode : - __init__() le constructeur - ajouter_valeur() - reinitialiser() - create_rectangle() - FenetreAPropos()

Dans la classe Application, tous les attributs sont privés. Aucune méthode n'est privée.

3. Algorithme

3.1 : Algorithme du programme principal

Nom : ArbreBinairedeRecherche
Entrée :
- Nombres entiers saisis par l'utilisateur (compris entre 1 et 99).

Sortie :
- Affichage d'un arbre binaire de recherche sous forme graphique.

Début

Importer le module tkinter

Définir les fonctions pour gérer les événements :

- Réinitialiser l'affichage
- Ajouter des valeurs et dessiner l'arbre
- Afficher une fenêtre "À propos"
- Quitter l'application

Créer la fenêtre principale de l'application ABR

Déclarer les variables nécessaires :

- Une variable pour stocker l'arbre binaire
- Une variable pour afficher l'arbre binaire

Créer les widgets :

- Une entrée pour la saisie des valeurs
- Un canevas pour afficher l'arbre
- Boutons : "Réinitialiser", "Dessiner", "À propos", "Quitter"

Placer les widgets dans la fenêtre

Tant que l'application est en cours d'exécution :

Attendre une action utilisateur

-Si "Réinitialiser" est cliqué exécuter la fonction de réinitialisation

-Si "Dessiner" est cliqué dessiner l'arbre

-Si "À propos" est cliqué ouvrir la page "A propos"

-Si "Quitter" est cliqué fermer l'application

Fin tant que

Fin

3.2 : Algorithmes des fonctions

3.2.1 : Algorithme class Nœud

```
Début
  Classe Noeud
    Définir les attributs :
      valeur : entier
      sag (sous-arbre gauche) : Noeud
      sad (sous-arbre droit) : Noeud
      hauteur : entier (par défaut 0)

    Méthode ajouter(valeur, hauteur)
      Si hauteur > 4
        Afficher "Impossible d'ajouter, hauteur maximale atteinte"
        Retourner
      Fin Si

      Si valeur < self.valeur
        Si sag est None
          Créer un nouveau Noeud avec valeur et hauteur
          Noeud → sag  // Créer un nouveau Noeud à gauche
          Afficher "Ajout à gauche"
        Sinon
          sag.ajouter(valeur, hauteur + 1)  // Appel récursif pour ajouter à gauche
        Fin Si
      Sinon Si valeur > self.valeur
        Si sad est None
          Créer un nouveau Noeud avec valeur et hauteur
          Noeud → sad  // Créer un nouveau Noeud à droite
          Afficher "Ajout à droite"
        Sinon
          sad.ajouter(valeur, hauteur + 1)  // Appel récursif pour ajouter à droite
        Fin Si
      Sinon
        Afficher "Valeur déjà présente dans l'arbre"
      Fin Si
    Fin Méthode
  Fin Classe
Fin
```

3.2.2 : Algorithme class ArbreBinaire

```
Début
  Classe ArbreBinaire
    Définir les attributs :
      racine : Noeud (par défaut None)

    Méthode ajouter(valeur)
      Si racine est None
        Créer un Noeud avec valeur et hauteur 0
        racine → Noeud  // Créer la racine de l'arbre
        Afficher "Racine ajoutée : valeur"
      Sinon
        racine.ajouter(valeur, 1)  // On commence à la hauteur 1
      Fin Si
    Fin Méthode

    Méthode afficher(canvas, x, y, profondeur=0, espacement=170)
      Appeler _afficher_noeud avec les paramètres du canvas, x, y, profondeur, espacement
    Fin Méthode

    Méthode _afficher_noeud(canvas, noeud, x, y, profondeur, espacement)
      Si noeud est None
        Retourner
      Fin Si

      Définir rayon = 20
      Définir hauteur_ecart = 120

      Si noeud.sag existe
        Calculer x_sag, y_sag
        Dessiner une ligne du noeud actuel au sous-arbre gauche
        (x, y) → (x_sag, y_sag)  // Lien vers le sous-arbre gauche
        Appeler _afficher_noeud sur le sous-arbre gauche avec les nouvelles coordonnées
      Fin Si

      Si noeud.sad existe
        Calculer x_sad, y_sad
        Dessiner une ligne du noeud actuel au sous-arbre droit
        (x, y) → (x_sad, y_sad)  // Lien vers le sous-arbre droit
        Appeler _afficher_noeud sur le sous-arbre droit avec les nouvelles coordonnées
      Fin Si

      Dessiner un cercle pour le noeud actuel avec la valeur du noeud
      Afficher la valeur du noeud à l'intérieur du cercle
    Fin Méthode
  Fin Classe
Fin
```

3.2.3 : Algorithme class Application

Début
  Classe Application:
    Définir les attributs :
      Initialiser la fenêtre principale avec les dimensions et le titre
      Configurer la couleur de fond de la fenêtre
      Initialiser un arbre binaire

      Créer un canevas pour afficher l'arbre
      Ajouter un rectangle sur le canevas
      Ajouter un titre à l'application

      Créer un canevas pour afficher l'arbre binaire
      Créer un champ de texte pour saisir les valeurs

Créer des boutons pour les différentes actions (Dessiner, Réinitialiser, À propos, Quitter)
      Placer les boutons sur la fenêtre

    Méthode ajouter_valeurs(self):
      Essayer:
          Récupérer les valeurs saisies dans le champ de texte et les convertir en une liste d'entiers
          Filtrer les valeurs pour qu'elles soient comprises entre 1 et 99
          Si aucune valeur valide n'est saisie, afficher un message d'erreur et retourner
          Ajouter les valeurs à l'arbre binaire
          Effacer le canevas avant de redessiner l'arbre
          Afficher l'arbre sur le canevas
      En cas d'erreur de conversion:
          Afficher un message d'erreur

    Méthode reinitialiser(self):
      Réinitialiser l'arbre binaire
      Effacer le canevas et le champ de texte

    Méthode create_rectangle(self, canvas, x1, y1, x2, y2, **kwargs):
      Créer un rectangle sur le canevas avec les coordonnées et options spécifiées

    Méthode FenetreAPropos(self):
      Créer une fenêtre "À propos" avec des informations sur l'application
      Ajouter un canevas et un rectangle sur la fenêtre "À propos"
      Ajouter un titre et des textes explicatifs sur la fenêtre "À propos"
      Ajouter un label avec les noms des créateurs
      Ajouter un bouton pour fermer la fenêtre "À propos"

Méthode principale:

    Créer une instance de Tkinter
    Créer une instance de la classe Application avec la fenêtre principale
    Lancer la boucle principale de Tkinter

Fin

4. Script commenté

Voici le script commenté du projet d'arbre binaire de recherche en Python.

```python
import tkinter as tk

class Noeud:
    def __init__(self, valeur, hauteur=0):
        # Initialisation d'un nœud avec une valeur, une hauteur et deux pointeurs vers les sous-arbres gauche et droit
        self.valeur = valeur
        self.sag = None  # Sous-arbre gauche
        self.sad = None  # Sous-arbre droit
        self.hauteur = hauteur  # Stockage de la hauteur du nœud

    def ajouter(self, valeur, hauteur):
        # Ajoute une valeur dans l'arbre en respectant la contrainte de hauteur maximale (4)
        if hauteur > 4:
            print(f"Impossible d'ajouter {valeur}, hauteur maximale atteinte.")
            return

        # Si la valeur est inférieure à la valeur du nœud, elle va dans le sous-arbre gauche
        if valeur < self.valeur:
            if self.sag is None:
                self.sag = Noeud(valeur, hauteur)  # Création d'un nouveau nœud à gauche
                print(f"Ajout de {valeur} à gauche de {self.valeur} (Hauteur {hauteur})")
            else:
                self.sag.ajouter(valeur, hauteur + 1)  # Appel récursif pour ajouter à gauche

        # Si la valeur est supérieure, elle va dans le sous-arbre droit
        elif valeur > self.valeur:
            if self.sad is None:
                self.sad = Noeud(valeur, hauteur)  # Création d'un nouveau nœud à droite
                print(f"Ajout de {valeur} à droite de {self.valeur} (Hauteur {hauteur})")
            else:
                self.sad.ajouter(valeur, hauteur + 1)  # Appel récursif pour ajouter à droite
        else:
            print(f"La valeur {valeur} existe déjà dans l'arbre.")  # La valeur existe déjà dans l'arbre

class ArbreBinaire:
    def __init__(self):
        # Initialisation d'un arbre binaire avec une racine vide
        self.racine = None

    def ajouter(self, valeur):
        # Ajoute une valeur à l'arbre binaire
        if self.racine is None:
            self.racine = Noeud(valeur, hauteur=0)  # Création de la racine si l'arbre est vide
            print(f"Racine ajoutée : {valeur}")
        else:
            self.racine.ajouter(valeur, 1)  # On commence à la hauteur 1 si la racine existe déjà

    def afficher(self, canvas, x, y, profondeur=0, espacement=170):
        # Affiche l'arbre sur une toile graphique
        self._afficher_noeud(canvas, self.racine, x, y, profondeur, espacement)

    def _afficher_noeud(self, canvas, noeud, x, y, profondeur, espacement):
        # Affiche récursivement un nœud et ses sous-arbres sur le canevas
        if noeud is None:
            return

```

```
        rayon = 20  # Rayon du cercle représentant un nœud
        hauteur_ecart = 120  # Écart vertical entre les nœuds

        # Si le nœud a un sous-arbre gauche, on dessine une ligne et on affiche récursivement ce sous-arbre
        if noeud.sag:
            x_sag, y_sag = x - espacement, y + hauteur_ecart
            canvas.create_line(x, y + rayon, x_sag, y_sag - rayon, arrow=tk.LAST)  # Dessiner la ligne de connexion
            self._afficher_noeud(canvas, noeud.sag, x_sag, y_sag, profondeur + 1, espacement // 2)  # Appel récursif pour le sous-arbre gauche

        # Si le nœud a un sous-arbre droit, on dessine une ligne et on affiche récursivement ce sous-arbre
        if noeud.sad:
            x_sad, y_sad = x + espacement, y + hauteur_ecart
            canvas.create_line(x, y + rayon, x_sad, y_sad - rayon, arrow=tk.LAST)  # Dessiner la ligne de connexion
            self._afficher_noeud(canvas, noeud.sad, x_sad, y_sad, profondeur + 1, espacement // 2)  # Appel récursif pour le sous-arbre droit

        # Dessiner le cercle représentant le nœud
        canvas.create_oval(x - rayon, y - rayon, x + rayon, y + rayon, outline="black", width=2, fill="white")
        # Afficher la valeur du nœud dans le cercle
        canvas.create_text(x, y, text=str(noeud.valeur), font=("Arial", 12, "bold"))

class Application:
    def __init__(self, root):
        self._fenetre = root
        self._fenetre.geometry("1800x1000")
        self._fenetre.title("Arbre Binaire de Recherche")
        self._fenetre.config(bg="white")
        self._arbre = ArbreBinaire()

        # Canevas pour afficher l'arbre
        canevas = tk.Canvas(self._fenetre, width=1604, height=130, bg="#FFFFFF", highlightbackground="#FFFFFF", highlightthickness=5)
        canevas.pack(pady=20)
        self.create_rectangle(canevas, 108, 55, 1604, 130, fill="Dim Gray")

        # Titre de l'application
        titre = tk.Label(self._fenetre, text="Arbre Binaire de Recherche", font=("Arial", 15, "bold"))
        titre.place(x=802, y=96)

        # Canevas pour afficher l'arbre binaire
        self._canvas = tk.Canvas(self._fenetre, width=760, height=630, bg="#D3D3D3")
        self._canvas.place(x=270, y=200)

        # Champ de texte pour saisir les valeurs
        self._entry = tk.Entry(self._fenetre, bd=4, width=40, relief="solid")
        self._entry.place(x=1205, y=300)

        # Boutons
        bouton_ajouter = tk.Button(self._fenetre, text="Dessiner", font=("Arial", 12, "bold"), width=10, height=2, command=self.ajouter_valeurs)
        bouton_ajouter.place(x=1266, y=537)

        bouton_reset = tk.Button(self._fenetre, text="Réinitialiser", font=("Arial", 12, "bold"), width=10, height=2, command=self.reinitialiser)
        bouton_reset.place(x=1540, y=90)

        bouton_Apropos = tk.Button(self._fenetre, text="À propos", font=("Arial", 12, "bold"), width=10, height=2, command=self.FenetreAPropos)
        bouton_Apropos.place(x=1400, y=90)

        bouton_quitter = tk.Button(self._fenetre, text="Quitter", font=("Arial", 12, "bold"), width=10, height=2, fg='red', command=self._fenetre.destroy)
        bouton_quitter.place(x=200, y=864)

        canevas.create_window(1540, 90, window=bouton_reset)
        canevas.create_window(180, 90, window=bouton_Apropos)

    def ajouter_valeurs(self):
        try:
            # Récupération et conversion des entrées en une liste d'entiers
            valeurs = list(map(int, self._entry.get().split()))
            # Filtrage des valeurs pour qu'elles soient comprises entre 1 et 99
            valeurs = [val for val in valeurs if 1 <= val <= 99]
            # Vérification si au moins une valeur valide a été saisie
            if not valeurs:
                print("Aucune valeur valide saisie.")
                return
            # Ajout des valeurs à l'arbre binaire
            for valeur in valeurs:
                self._arbre.ajouter(valeur)
            # Effacement du canvas avant de redessiner l'arbre
            self._canvas.delete("all")
            self._arbre.afficher(self._canvas, 380, 85)
        except ValueError:
            # Gestion des erreurs si la conversion en entier échoue
            print("Veuillez entrer des nombres valides.")


    def reinitialiser(self):
        # Réinitialise l'arbre et efface le canevas et le champ de texte
        self._arbre = ArbreBinaire()
        self._canvas.delete("all")
        self._entry.delete(0, tk.END)

    def create_rectangle(self, canvas, x1, y1, x2, y2, **kwargs):
        # Crée un rectangle sur le canevas avec les coordonnées et options spécifiées
        return canvas.create_rectangle(x1, y1, x2, y2, **kwargs)

    def FenetreAPropos(self):
        # Crée une fenêtre "À propos" avec des informations sur l'application
        fenetreAP = tk.Toplevel(self._fenetre)
        fenetreAP.geometry("1100x680")
        fenetreAP.title("À propos")
        fenetreAP.configure(bg="#FFFFFF")

        canevas = tk.Canvas(fenetreAP, width=1200, height=100, bg="#FFFFFF", highlightbackground="#FFFFFF", highlightthickness=5)
        canevas.pack(pady=20)
        self.create_rectangle(canevas, 108, 55, 1000, 100, fill="Dim Gray")

        titre = tk.Label(fenetreAP, text="À propos", font=("Arial", 15, "bold"))
        titre.place(x=500, y=82)

```

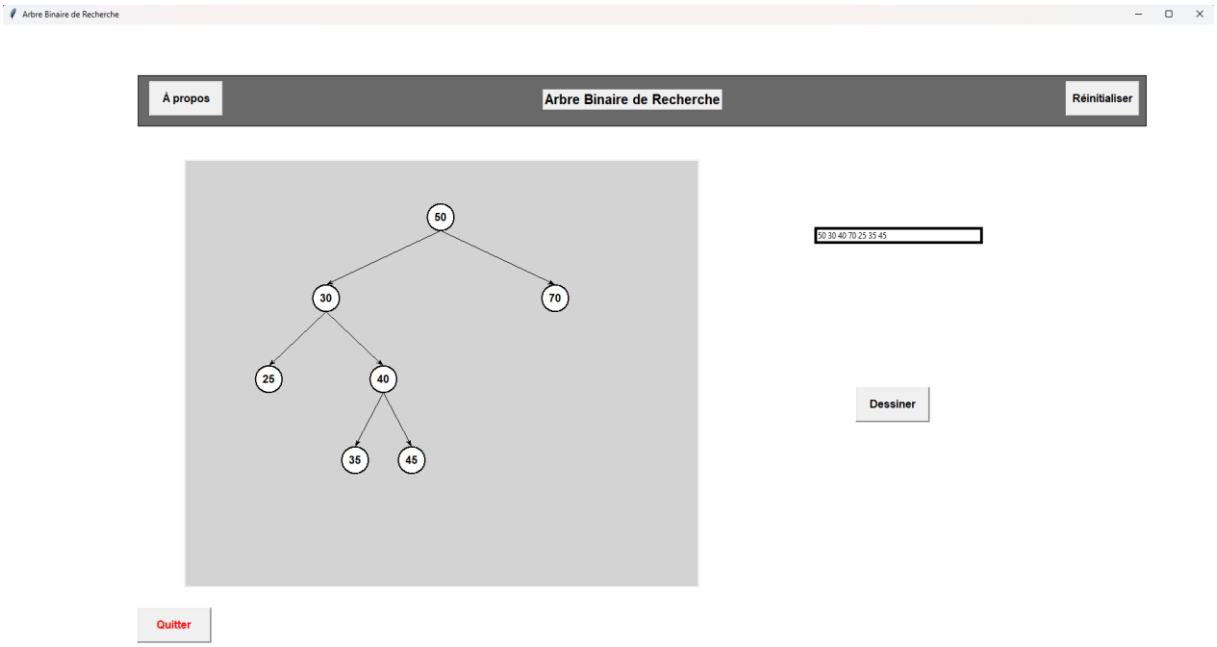
```
        textes = [
            "Comment utiliser notre application ?",
            "- Saisissez des nombres entre 1 et 99.",
            "- Cliquez sur 'Dessiner' pour afficher l'arbre.",
            "- L'arbre est limité à une hauteur de 4.",
            "- Utilisez 'Réinitialiser' pour recommencer.",
            "- Cliquez sur 'Quitter' pour fermer l'application."
        ]

        y_position = 144
        for texte in textes:
            tk.Label(fenetreAP, text=texte, font=("Arial", 12, "bold")).place(x=101, y=y_position)
            y_position += 60

        createur = tk.Label(fenetreAP, text="Merlin Kylian | Marchand Baptiste", font=("Arial", 10))
        createur.place(x=770, y=630)

        boutonQ = tk.Button(fenetreAP, text="Quitter", font=("Arial", 12, "bold"), width=10, height=2, fg='red', command=fenetreAP.destroy)
        boutonQ.place(x=100, y=590)

if __name__ == "__main__":
    root = tk.Tk()
    app = Application(root)
    root.mainloop()
```

5. Tests de validation

Voici quelques tests réalisés pour confirmer le bon fonctionnement du code.

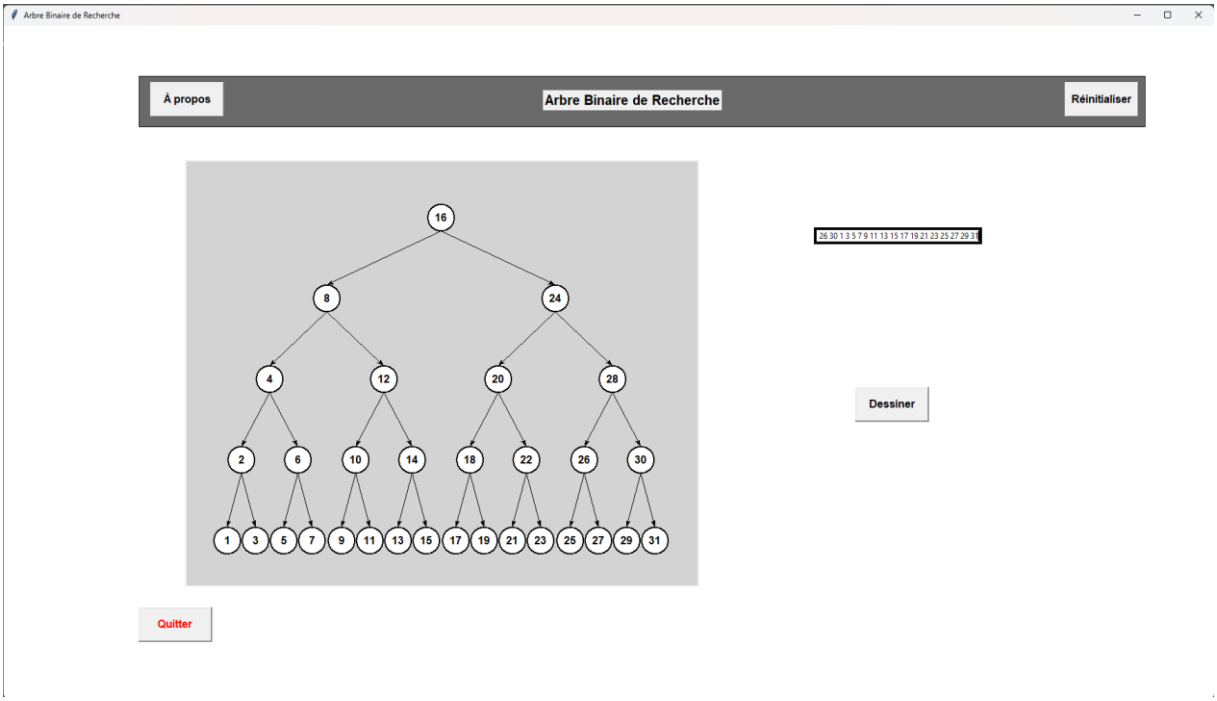
Tout d'abord, reprenons l'exemple donné dans le sujet :

50 30 40 70 25 35 45



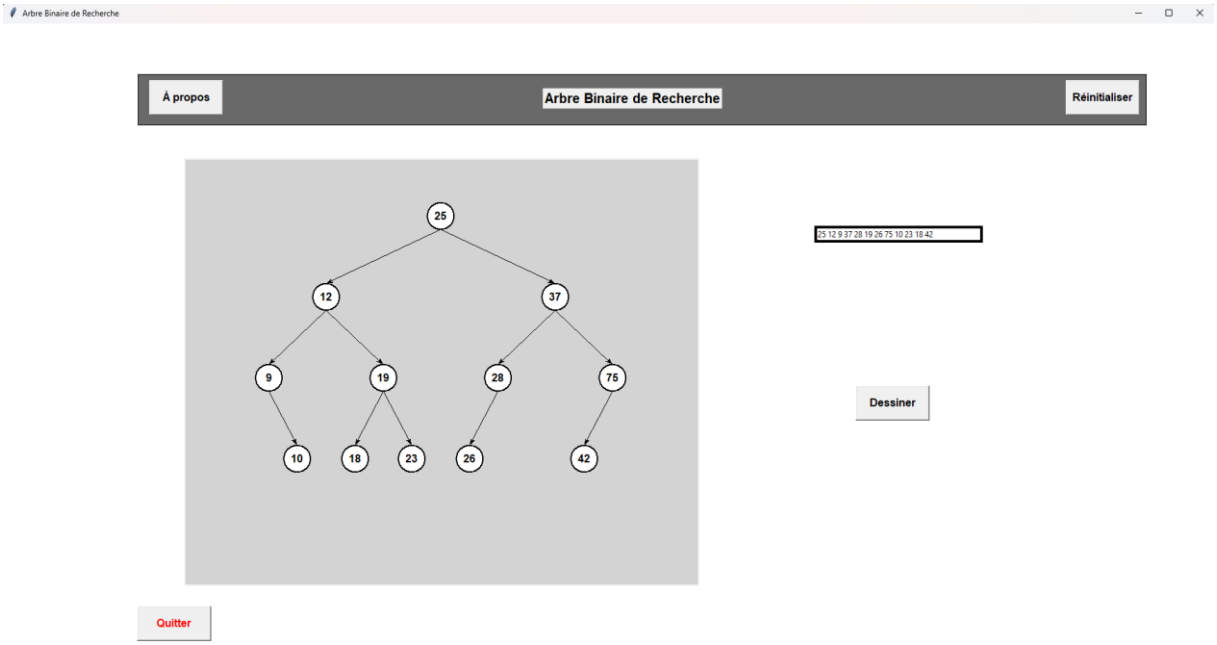
Maintenant avec un arbre binaire de recherche complet :

0 16 8 24 4 12 20 28 2 6 10 14 18 22 26 30 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31



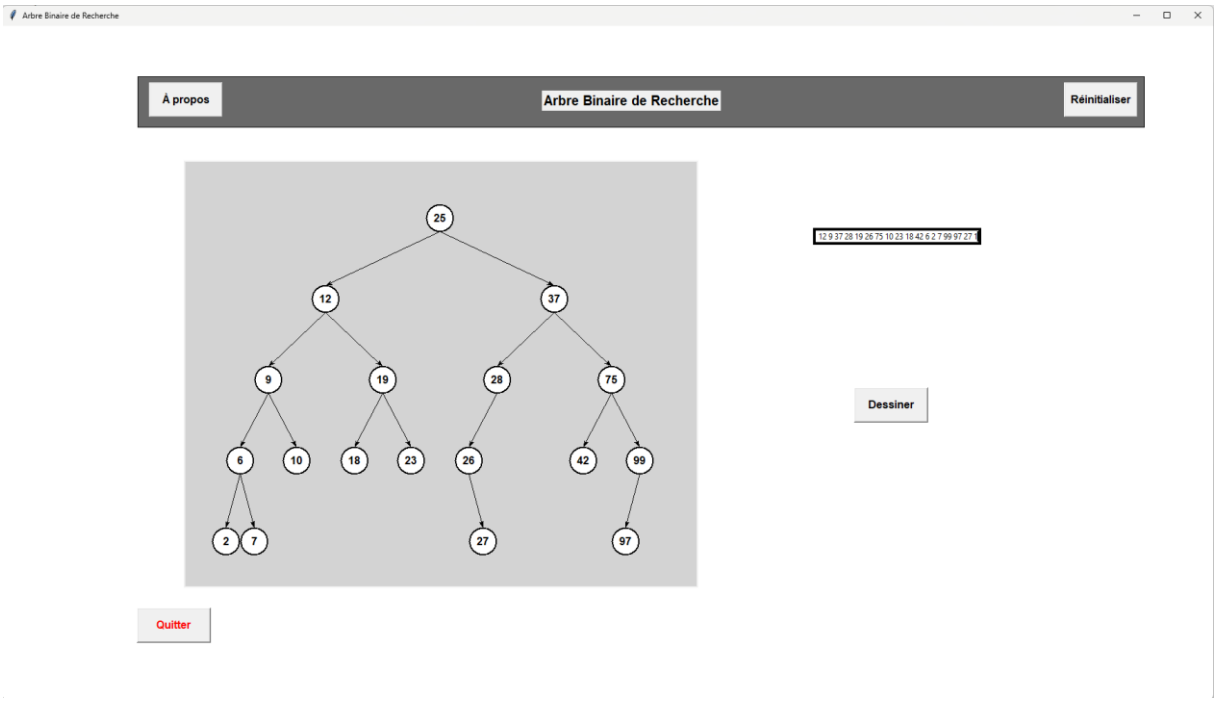
Avec un arbre binaire utilisé en cours :

25 12 9 37 28 19 26 75 10 23 18 42



Avec un arbre binaire ayant une valeur dépassant la hauteur 4 :

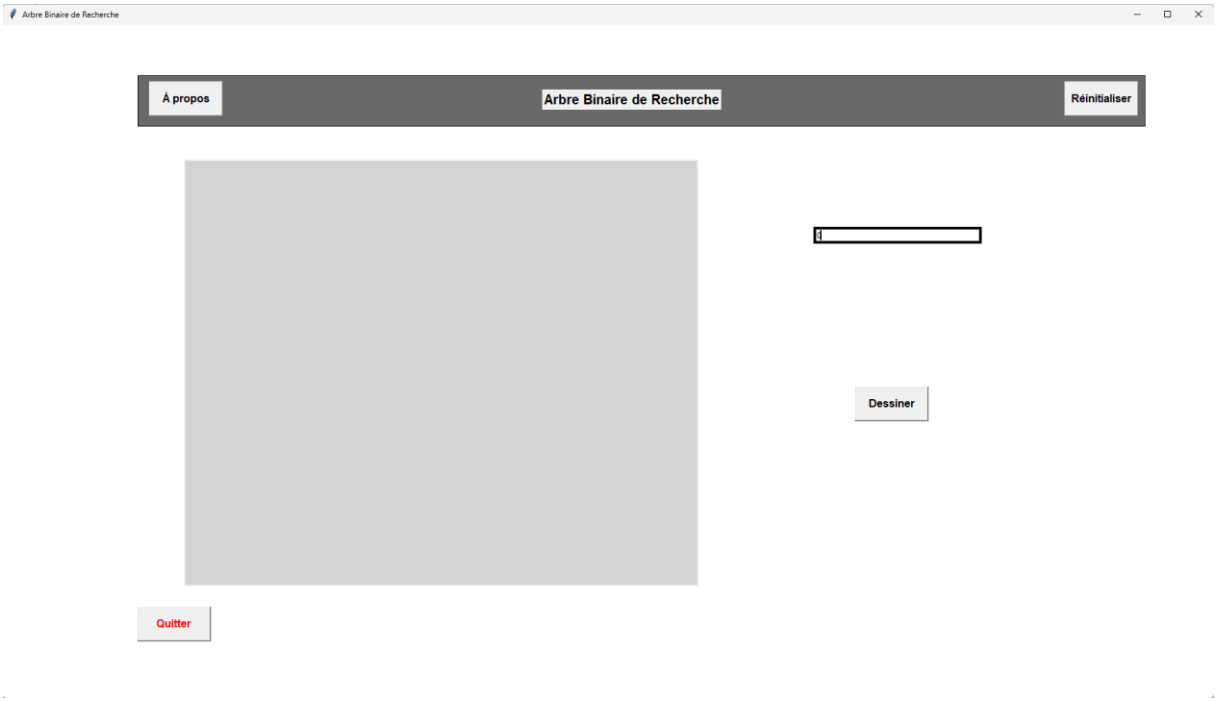
25 12 9 37 28 19 26 75 10 23 18 42 6 2 7 99 97 27 1



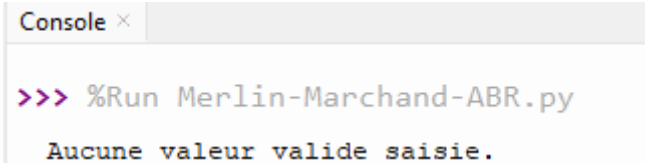
Message pour suivre les étapes dans la console :

```
Console ×

>>> %Run Merlin-Marchand-ABR.py

  Racine ajoutée : 25
  Ajout de 12 à gauche de 25 (Hauteur 1)
  Ajout de 9 à gauche de 12 (Hauteur 2)
  Ajout de 37 à droite de 25 (Hauteur 1)
  Ajout de 28 à gauche de 37 (Hauteur 2)
  Ajout de 19 à droite de 12 (Hauteur 2)
  Ajout de 26 à gauche de 28 (Hauteur 3)
  Ajout de 75 à droite de 37 (Hauteur 2)
  Ajout de 10 à droite de 9 (Hauteur 3)
  Ajout de 23 à droite de 19 (Hauteur 3)
  Ajout de 18 à gauche de 19 (Hauteur 3)
  Ajout de 42 à gauche de 75 (Hauteur 3)
  Ajout de 6 à gauche de 9 (Hauteur 3)
  Ajout de 2 à gauche de 6 (Hauteur 4)
  Ajout de 7 à droite de 6 (Hauteur 4)
  Ajout de 99 à droite de 75 (Hauteur 3)
  Ajout de 97 à gauche de 99 (Hauteur 4)
  Ajout de 27 à droite de 26 (Hauteur 4)
  Impossible d'ajouter 1, hauteur maximale atteinte.
```

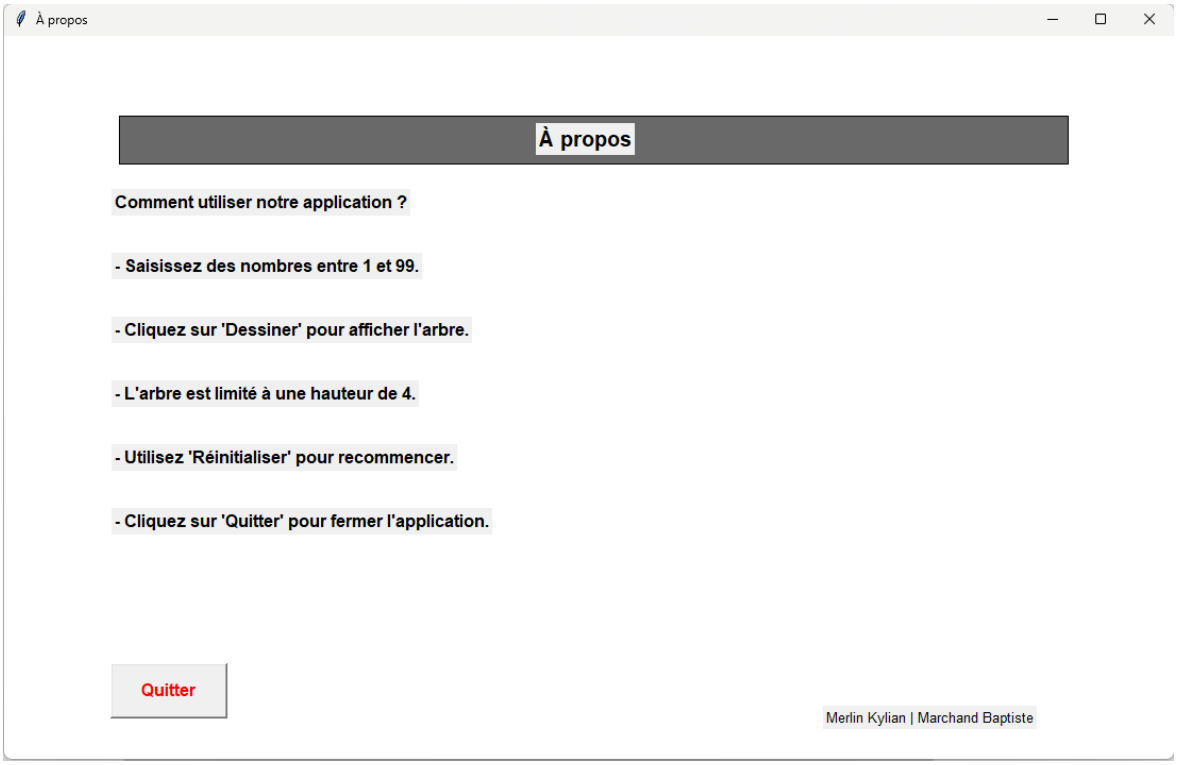
Lors de la saisie d'une valeur non traitable comme -1, 0 ou 100 :



Rien ne s'affiche dans le canevas et un message est renvoyé dans la console.



La fenêtre à propos :



Conclusion

Ce projet a permis d'approfondir nos connaissances sur les arbres binaires de recherche ainsi que sur leur implémentation en Python avec une interface graphique. Nous avons développé une application fonctionnelle qui respecte les contraintes imposées, notamment la limitation de la hauteur de l'arbre.