Gottfried Wilhelm Leibniz Universität Hannover
Institut für Theoretische Informatik

Eine GUI zur Visualisierung von binären Suchbäumen

Bachelorarbeit

Cord Magerstedt
Matrikelnr. 10019658

Hannover, den 6. Juni 2024

Erstprüfer: PD Dr. rer. nat. habil. Arne Meier
Zweitprüfer: Prof. Dr. rer. nat. Heribert Vollmer
Betreuerin: Vivian Holzapfel

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 6. Juni 2024

_____

Cord Magerstedt

Inhaltsverzeichnis

1 Einleitung

In der heutigen digitalen Welt spielen Datenstrukturen in der effizienten Verarbeitung und Organisation von Informationen eine entscheidende Rolle. Binäre Suchbäume sind eine der grundlegendsten und weit verbreitetsten Datenstrukturen, die in vielen Anwendungen der Informatik verwendet werden, von Datenbanken über Suchalgorithmen bis hin zu Spielentwicklung und künstlicher Intelligenz. Die Fähigkeit, binäre Suchbäume verstehen und effektiv nutzen zu können, ist daher von entscheidender Bedeutung für Studierende und Fachpersonen in der Informatik.

Diese Bachelorarbeit widmet sich der Entwicklung einer grafischen Benutzeroberfläche (GUI), die die Schritte des Einfügens, Löschens und Suchens in binären Suchbäumen animiert darstellt. Das Ziel dieser GUI ist es, eine interaktive Lernumgebung zu schaffen, die Studierenden eine intuitive und visuelle Möglichkeit bietet, die Struktur und Funktionsweise von binären Suchbäumen zu verstehen. Darüber hinaus kann die entwickelte GUI als leistungsstarkes Werkzeug dienen, um das Verständnis und die Anwendung von binären Suchbäumen in der Praxis zu vertiefen und zu festigen.

2 Grundlagen zu binären Suchbäumen

Ein *binärer Suchbaum*, von jetzt an auch mit BST abgekürzt, ist ein ungerichteter, zusammenhängender und zyklenfreier Graph [3, S. 133], bestehend aus einem oder mehreren Knoten, wobei jeder Knoten einen Wert des gleichen Werteraumes enthält und höchstens zwei Kindknoten hat, einen linken und einen rechten. Diese haben die Eigenschaft, dass der Wert des linken Kindknotens kleiner und der Wert des rechten Kindknotens größer ist als der des Elternknotens [3, S. 140]. Der erste Knoten, welcher keinen Elternknoten hat, wird *Wurzel* genannt, wohingegen ein Knoten ohne Kindknoten *Blatt* genannt wird. Die *Höhe* eines Baumes ist gegeben durch den längsten Weg von der Wurzel zu einem Blatt [3, S. 133]. Die *Größe* eines Baumes ist durch die Anzahl der Knoten definiert. Abbildung 2.1 stellt einen zufälligen binären Suchbaum mit der Höhe $h = 2$ und der Größe $n = 6$ dar.

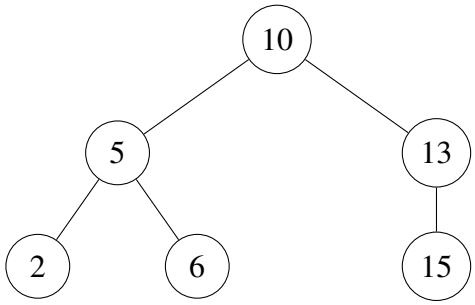

Abbildung 2.1: Binärer Suchbaum mit Höhe $h = 2$ und Größe $n = 6$

Diese Datenstruktur, welche häufig in der Informatik verwendet wird, ermöglicht es, Daten in einer geordneten Art und Weise zu speichern und diese effizient nach Elementen zu durchsuchen.

Die Operationen, die auf einem binären Suchbaum ausgeführt werden können, werden in Kapitel 2.2 beschrieben.

2.1 Motivation

Binäre Suchbäume bieten eine Reihe von Vorteilen, die ihre Verwendung in vielen Anwendungen der Informatik motivieren. Ein wichtiger Aspekt eines binären Suchbaums ist seine Effizienz. Die Zeitkomplexität für Operationen wie das Einfügen, Löschen und Suchen hängt dabei von der Höhe h des Baums ab. Ein balancierter binärer Suchbaum, siehe Abbildung 2.2,

hat eine Höhe von $O(log(n))$ [3, S. 146], wobei sich n auf die Größe des Baumes bezieht. Folglich können die zuvor erwähnten Operationen in logarithmischer Zeit ausgeführt werden, sodass sie besonders effizient sind.

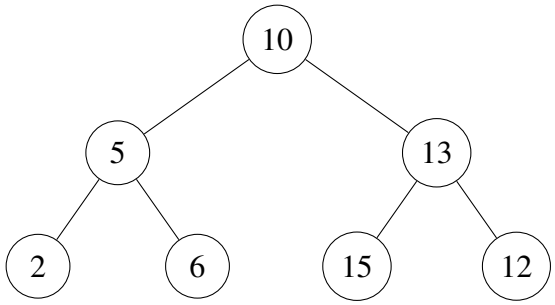

Abbildung 2.2: Balancierter binärer Suchbaum mit Höhe $h = 2$ und Größe $n = 7$

Allerdings kann ein unbalancierter Baum, siehe Abbildung 2.3, der sich wie eine verkettete Liste verhält, eine Höhe von $O(n)$ haben, was zu zeitlich ineffizienten Operationen führt [3, S. 146].

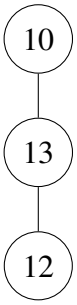

Abbildung 2.3: Unbalancierter binärer Suchbaum mit Höhe $h = 2$ und Größe $n = 3$

Im Vergleich zu anderen Datenstrukturen wie Arrays oder Listen bieten binäre Suchbäume eine effiziente Nutzung des Speichers. Durch die Organisation der Werte im Baum benötigen binäre Suchbäume nur den Speicherplatz, der für die gespeicherten Elemente und die Struktur des Baums selbst erforderlich ist. Dies macht sie ideal für Anwendungen, bei denen der Speicherplatz begrenzt ist oder effizient genutzt werden muss.

Die Konzepte hinter binären Suchbäumen sind leicht verständlich und einfach zu implementieren. Dadurch wird die Entwicklung von Algorithmen und Datenstrukturen, die binäre Suchbäume verwenden, erleichtert, sodass sie eine geeignete Wahl für Programmierer*innen und Entwickler*innen, die robuste und unkomplizierte Lösungen benötigen, darstellen.

2.2 Operationen auf BSTs

In den folgenden Kapiteln werden die möglichen Operationen auf BSTs erläutert. Der verwendete Pseudocode dieser sowie der Ablauf der Operationen folgen hierbei verändert dem des Buches *Algorithmen und Datenstrukturen* [4, S. 266-275]

2.2.1 Suchen

Das Suchen nach einem Element in einem binären Suchbaum ist ein grundlegender Algorithmus, der darauf abzielt, das gesuchte Element schnell zu lokalisieren. Der Prozess erfolgt rekursiv durch den Baum und basiert auf der Ordnungseigenschaft des binären Suchbaums.

Algorithm 1 Search on BST

```
1: procedure TreeSearch(Key k, Knoten v)
2:     if isLeave(node) then return node          ▷ Position, wo k hingehören würde
3:     else if k = key(v) then return v
4:     else if k < key(v) then return TreeSearch(k, leftChild(v))
5:     else
6:         return TreeSearch(k, rightChild(v))
```

Die Suche beginnt am Wurzelknoten des Baumes. Zunächst erfolgt ein Vergleich von dem gesuchten Element mit dem Wert des Wurzelknotens. Entspricht der Wert dem des Elements, ist die Suche erfolgreich abgeschlossen. Andernfalls wird das gesuchte Element mit dem Wert des aktuellen Knotens verglichen, um zu bestimmen, ob die Suche im linken oder rechten Teilbaum fortgesetzt werden soll. Wenn das gesuchte Element kleiner als der Wert des aktuellen Knotens ist, wird die Suche im linken Teilbaum des aktuellen Knotens fortgesetzt, indem der Suchalgorithmus rekursiv aufgerufen wird. Sollte das gesuchte Element größer sein als der Wert des aktuellen Knotens, wird die Suche im rechten Teilbaum fortgesetzt. Dieser Prozess wird so lange wiederholt, bis das gesuchte Element gefunden wird oder festgestellt wird, dass es nicht im Baum vorhanden ist. Sollte das gesuchte Element gefunden wird, ist die Suche erfolgreich abgeschlossen, und der Knoten, der das Element enthält, wird zurückgegeben. Kann das Element nicht lokalisiert werden, wird in der Blattknoten, der das Element enthalten würde, zurückgegeben.

2.2.2 Einfügen

Das Einfügen eines neuen Elements in einen binären Suchbaum ist ein grundlegender Schritt, der die Ordnung des Baumes beibehalten muss. Der Prozess beginnt an der Wurzel des Baumes und erfolgt rekursiv durch den Baum, bis der richtige Ort für das neue Element gefunden wird.

Algorithm 2 Insertion in BST

```
1: procedure INSERTITEM(Element e)
2:     w ← TreeSearch(key(e),root())
3:     if isLeaf(w) then
4:         expandLeaf(w)
5:         replaceAtNode(w,e)
6:     else
7:         return „Schlüssel existiert“
```

Für diese Operation wird zunächst die bereits implementierte Suche verwendet. Das Resultat wird in einer Variable abgespeichert und es wird überprüft, ob das Ergebnis der Suche ein Blatt ist. Ist dies der Fall, wurde der Platz für das einzufügende Element gefunden. Das Blatt wird erweitert und der Wert in den neuen Knoten reingeschrieben. Dieser neue Knoten hat keine Kinder. Sollte der Rückgabewert der Suche ein Knoten sein, bedeutet dies, dass der Wert bereits im Baum existiert und folglich nicht eingefügt werden muss.

2.2.3 Löschen

Das Löschen eines Elements aus einem binären Suchbaum erfordert besondere Sorgfalt, um sicherzustellen, dass die Ordnung des Baumes erhalten bleibt und keine Verletzungen der Baumeigenschaften auftreten.

Algorithm 3 Deletion from BST

```
1: procedure REMOVEITEM(Key k)
2:     w ← TreeSearch(key(k, root())
3:     if w besitzt zwei Blatt-Kinder then lösche w und die Kinder
4:     else if w besitzt ein Blatt-Kind z then removeAboveLeaf(z)
5:     else                                              ▷ w hat zwei Kinder
6:         finde Inorder-Nachfolger y von w, mit linkem Blattkind x
7:         swap(y, w)
8:         removeAboveLeaf(x)
```

Der Prozess erfolgt in zwei Schritten. Zunächst muss das zu löschende Element im Baum gefunden werden. Dies geschieht durch eine Suche im Baum durch die zuvor implementierte Suchfunktion. Nachdem das zu löschende Element gefunden wurde, gibt es drei Fälle zu berücksichtigen:

- Das zu löschende Element hat zwei Blattknoten: Hier werden der Knoten und seine Blattknoten entfernt.

- Das zu löschende Element hat einen Kindknoten: In diesem Fall wird der zu löschende Knoten durch seinen einzigen Kindknoten ersetzt. Dafür wird der Elternknoten des zu löschenden Knotens auf das Kind des zu löschenden Knotens umgebogen. Und anschließend der zu löschende Knoten entfernt.

- Das zu löschende Element hat zwei Kindknoten: In diesem Fall muss der zu löschende Knoten durch seinen kleinsten Nachfolger im rechten Teilbaum ersetzt werden.

Während des Löschvorgangs darauf geachtet werden muss, dass die Ordnungseigenschaften des binären Suchbaums stets erhalten bleiben. Durch die sorgfältige Handhabung der Fälle kann sichergestellt werden, dass der binäre Suchbaum korrekt bleibt und weiterhin effizient für Suchoperationen verwendet werden kann.

2.2.4 Traversierung

Traversierungstechniken sind von entscheidender Bedeutung für die Untersuchung und Manipulation von Bäumen in verschiedenen Anwendungen, einschließlich Datenstrukturen, Algorithmen, Computernetzwerken und künstlicher Intelligenz. Sie ermöglichen eine systematische Analyse und Verarbeitung von Baumstrukturen, unabhängig von ihrer Größe oder Komplexität. Die Traversierungsmethoden sind rekursive Methoden, die beschreiben, in welcher Reihenfolge die Knoten des Baumes besucht werden. Dabei wird jeder Knoten genau einmal besucht. Es gibt verschiedene Arten von Traversierungsstrategien für Bäume, drei von ihnen werden in den nächsten Kapiteln beschrieben.

2.2.4.1 Preorder

Bei der Preorder-Traversierung wird zuerst der Wurzelknoten besucht, dann der linke Teilbaum und zum Schluss der rechte Teilbaum, visualisiert mithilfe der Abbildung 2.4. Der Prozess der Preorder-Traversierung ist anhand folgendem rekursiven Algorithmus beschrieben:

Algorithm 4 PreOrderTraversion

```
1: procedure PreOrder(Tree T, Knoten v)
2:     besuche Knoten v
3:     if not T.isLeaf(v) then PreOrder(T, T.leftChild(v))
4:     if not T.isLeaf(v) then PreOrder(T, T.rightChild(v))
```

Zuerst wird der Wert des Wurzelknotens besucht und verarbeitet. Als nächstes wird der linke Teilbaum des aktuellen Knotens traversiert, indem der Preorder-Traversierungsprozess rekursiv aufgerufen wird. Das heißt, der Algorithmus wird mit dem linken Kindknoten des aktuellen Knotens als Wurzelknoten wiederholt. Nachdem der linke Teilbaum vollständig traversiert wurde, wird der rechte Teilbaum des aktuellen Knotens traversiert, indem der

Preorder-Traversierungsprozess erneut rekursiv aufgerufen wird. Dies geschieht, nachdem der linke Teilbaum vollständig besucht wurde.

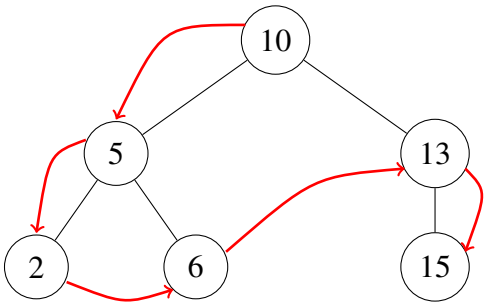

Abbildung 2.4: Preorder on BST, Ausgabe: 10, 5, 2, 6, 13, 15

2.2.4.2 Postorder

Bei der Postorder-Traversierung wird als erstes der linke Teilbaum besucht, dann der rechte Teilbaum und zum Schluss der Wurzelknoten, durch Abbildung 2.5 dargestellt. Der Prozess der Postorder-Traversierung ist anhand folgendem rekursiven Algorithmus beschrieben:

Algorithm 5 PostOrderTraversion

```
1: procedure PostOrder(Tree T, Knoten v)
2:     if not T.isLeaf(v) then
3:         PostOrder(T, T.leftChild(v))
4:     if not T.isLeaf(v) then
5:         PostOrder(T, T.rightChild(v))
6:     besuche Knoten v
```

Der Prozess beginnt damit, den linken Teilbaum des aktuellen Knotens zu traversieren, indem der Postorder-Traversierungsprozess rekursiv aufgerufen wird. Das heißt, der Algorithmus wird mit dem linken Kindknoten des aktuellen Knotens als Wurzelknoten wiederholt. Als nächstes wird der rechte Teilbaum des aktuellen Knotens traversiert, indem der Postorder-Traversierungsprozess erneut rekursiv aufgerufen wird. Dies geschieht, nachdem der linke Teilbaum vollständig traversiert wurde. Nachdem sowohl der linke als auch der rechte Teilbaum des aktuellen Knotens traversiert wurden, wird der Wert des aktuellen Knotens besucht und verarbeitet.

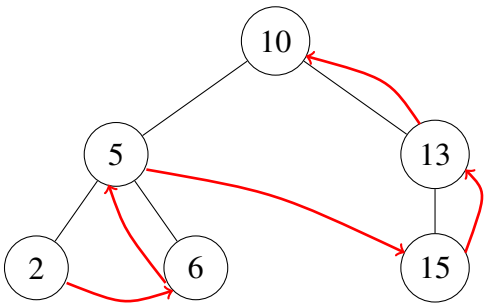

Abbildung 2.5: Postorder on BST, Ausgabe: 2, 6, 5, 15, 13, 10

2.2.4.3 Inorder

Bei der Inorder-Traversierung wird zuerst der linke Teilbaum besucht, dann der Wurzelknoten und zum Schluss der rechte Teilbaum. Dieser Ablauf ist in Abbildung 2.6 zu sehen. Der Prozess der Inorder-Traversierung ist anhand folgendem rekursiven Algorithmus beschrieben:

Algorithm 6 InorderTraversion

```
1: procedure INORDER(Tree t, Knoten v)
2:     if not T.isLeaf(v) then
3:         INORDER(T, T, T.leftChild(v))
4:     besuche Knoten v
5:     if not T.isLeaf(v) then
6:         INORDER(T, T, T.rightChild(v))
```

Der Prozess beginnt damit, den linken Teilbaum des aktuellen Knotens zu traversieren, indem der Inorder-Traversierungsprozess rekursiv aufgerufen wird. Das heißt, der Algorithmus wird mit dem linken Kindknoten des aktuellen Knotens als Wurzelknoten wiederholt. Nachdem der linke Teilbaum vollständig traversiert wurde, wird der Wert des aktuellen Knotens besucht und verarbeitet. Als nächstes wird der rechte Teilbaum des aktuellen Knotens traversiert, indem der Inorder-Traversierungsprozess erneut rekursiv aufgerufen wird. Dies geschieht, nachdem der aktuelle Knoten besucht wurde. Der Inorder-Traversierungsprozess besucht jeden Knoten genau einmal und verarbeitet ihn in der festgelegten Reihenfolge: linker Teilbaum, Wurzel, rechter Teilbaum.

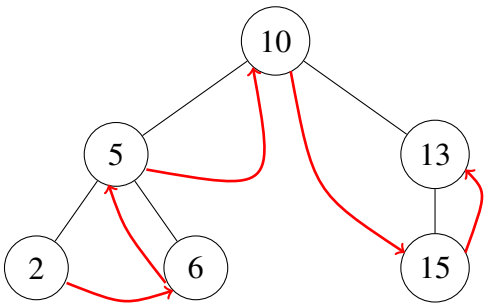

Abbildung 2.6: Inorder on BST, Ausgabe: 2, 6, 5, 10, 15, 13

3 Bewertung von Frameworks und Libraries

Für die Implementierung werden zunächst die zur Verfügung stehenden Libraries für Python betrachtet. Dafür wurde die Webseite *GUI Programming Website in Python* [2] verwendet, welche verfügbare Libraries für GUI Entwicklung auflistet. In den folgenden Kapiteln werden Frameworks und Libraries, die dafür geeignet empfunden wurden, untersucht. Ein Kriterium besteht zunächst daraus, auf welchen Betriebssystemen diese laufen können. Hierbei ist es wichtig, ein Framework zu verwenden, welches auf allen Betriebssystemen funktioniert, da nutzende Personen unterschiedliche Hard- und Software besitzen können. In den folgenden Kapiteln werden zunächst die Libraries vorgestellt und anschließend mit Hilfe von Erfahrungswerten durch die Entwicklung von *Proof-of-Concepts* (PoCs) analsysiert. Ein PoC ermöglicht es, die Funktionalität und Möglichkeiten eines Frameworks in verschiedenen Szenarien zu testen, bevor eine endgültige Entscheidung festlegt wird. Hierbei werden verschiedene Aspekte wie Benutzerfreundlichkeit und Flexibilität evaluiert, um festzustellen, welches Framework am besten zu den Anforderungen des Projekts passt. Durch die Durchführung von PoCs können fundierte Entscheidungen getroffen werden. Dies stellt sicher, dass die gewählte Bibliothek eine geeignete Wahl für das Vorhaben ist.

3.1 Tkinter

Tkinter [7] ist das standardmäßige GUI-Toolkit für die Programmiersprache Python und ermöglicht die Entwicklung von plattformunabhängigen grafischen Benutzeroberflächen. Es ist in die Python-Standardbibliothek integriert, was bedeutet, dass es ohne zusätzliche Installation verfügbar ist und direkt verwendet werden kann. Tkinter basiert auf der Tk GUI-Bibliothek, die ursprünglich für die Programmiersprache Tcl entwickelt wurde, aber aufgrund ihrer Flexibilität und Leistungsfähigkeit auch in Python adaptiert wurde.

Mit Tkinter können Entwickler*innen Anwendungen mit visuellen Elementen wie Fenstern, Schaltflächen, Textfeldern, Menüs oder Listen erstellen. Diese Elemente werden Widgets genannt und Tkinter bietet eine Vielzahl davon, um nahezu jede Art von Benutzeroberfläche zu gestalten. Einfache Anwendungen wie Taschenrechner oder Texteditoren können ebenso entwickelt werden wie komplexere Programme mit mehreren Fenstern und interaktiven

Features.

Ein wesentlicher Vorteil von Tkinter ist seine Benutzerfreundlichkeit. Durch die Python-typische Einfachheit und Klarheit des Codes ist es besonders für Einsteiger*innen geeignet, die in die GUI-Programmierung einsteigen möchten. Gleichzeitig bietet es ausreichend Funktionalität und Flexibilität, um auch fortgeschrittene Anwendungen zu realisieren.

Ein weiteres Merkmal von Tkinter ist die plattformübergreifende Kompatibilität. Anwendungen, die mit Tkinter entwickelt wurden, laufen auf verschiedenen Betriebssystemen wie Windows, macOS und Linux, ohne dass umfangreiche Anpassungen notwendig sind.

3.2 Flet

Flet [1] ist ein innovatives, plattformübergreifendes GUI-Toolkit für die Programmiersprache Python, das auf die Entwicklung reaktiver Web- und Desktop-Anwendungen spezialisiert ist. Durch die intuitive API ermöglicht Flet es Entwickler*innen, benutzerfreundliche und optisch ansprechende Benutzeroberflächen zu erstellen, ohne tiefgehende Kenntnisse der Webentwicklung zu benötigen. Dies macht es besonders optimal für Python-Entwickler*innen, die schnell hochwertige Anwendungen entwickeln möchten.

Ein Merkmal von Flet ist die Nutzung von Flutter als Rendering-Engine. Flutter, ursprünglich von Google entwickelt, ist bekannt für seine hohe Leistung und die Fähigkeit, konsistente Benutzeroberflächen über verschiedene Plattformen hinweg zu bieten. Dies bedeutet, dass Anwendungen, die mit Flet entwickelt wurden, sowohl auf mobilen Geräten (iOS und Android) als auch auf Desktop-Systemen (Windows, macOS und Linux) einheitlich und flüssig laufen. Diese plattformübergreifende Kompatibilität erleichtert es Entwickler*innen, ihre Anwendungen einem breiteren Publikum zugänglich zu machen.

Ein weiteres wichtiges Feature von Flet ist die Unterstützung von Hot Reload. Diese Funktion erlaubt es Entwickler*innen, Änderungen des Codes sofort zu sehen, ohne die Anwendung neu starten zu müssen. Dies beschleunigt den Entwicklungsprozess erheblich und macht das Testen und Debuggen von Anwendungen wesentlich effizienter. Mit Hot Reload können Entwickler*innen iterativ arbeiten und sofortiges Feedback zu ihren Änderungen erhalten.

Flet eignet sich für Projekte, die sowohl eine Web- als auch eine Desktop-Präsenz erfordern. Dies ist in der heutigen Zeit, in der Benutzer nahtlose Erfahrungen über verschiedene Geräte hinweg erwarten, ein großer Vorteil. Die Fähigkeit, sowohl Web- als auch Desktop-Anwendungen mit demselben Codebase zu entwickeln, spart Zeit und Ressourcen und vereinfacht die Wartung der Anwendungen.

3.3 Pywebview

PyWebView [5] ist ein Python-Toolkit, das es Entwickler*innen ermöglicht, plattformübergreifende Desktop-Anwendungen mit Web-Technologien zu erstellen. Es nutzt native Web-Engine-Komponenten wie WebKit, Microsoft Edge (Chromium), oder MSHTML, um HTML/CSS und JavaScript in einem Desktop-Fenster zu rendern. Dies erlaubt es Entwickler*innen, die Flexibilität und Leistungsfähigkeit moderner Webentwicklung zu nutzen, während sie gleichzeitig die Vorzüge einer nativen Desktop-Anwendung beibehalten.

Ein weiteres nützliches Feature von PyWebView ist die Möglichkeit zur bidirektionalen Kommunikation zwischen Python und JavaScript. Dies ermöglicht es Entwickler*innen, die Stärken beider Welten zu kombinieren: die robuste Backend-Entwicklung in Python und die flexible, interaktive Benutzeroberfläche mit Web-Technologien. Diese Interaktion wird durch eine einfache API erleichtert, die es ermöglicht, Python-Funktionen aus JavaScript heraus aufzurufen und umgekehrt.

3.4 Fazit

Pywebview wurde ausgeschlossen, da es stark auf die Verwendung von JavaScript setzt. Dies widerspricht den Anforderungen der Aufgabenstellung, die eine reine Python-Lösung erfordert. Darüber hinaus würde die Integration des zusätzlichen JavaScript-Codes den Entwicklungsaufwand erheblich erhöhen.

Flet schien zunächst vielversprechend, doch es bietet keine native Unterstützung zur Darstellung der Baumstruktur. Diese Funktionalität ist jedoch zentral für das Projekt, da eine effiziente und klare Darstellung der Daten unerlässlich ist. Die Implementierung solcher Strukturen in Flet wäre mit erheblichem Mehraufwand verbunden und würde die Komplexität der Anwendung unnötig steigern.

Im Gegensatz dazu bietet Tkinter eine ausgereifte und stabile Bibliothek zur Erstellung von grafischen Benutzeroberflächen in Python. Es verfügt über die Möglichkeit, mithilfe des Canvas-Widget und dem Malen auf diesem, die Baumstruktur darzustellen, was die Entwicklung erheblich vereinfacht und beschleunigt. Zudem ist Tkinter umfangreich dokumentiert und weit verbreitet, was den Zugang zu Ressourcen und Community-Support erleichtert. Diese Vorteile machen Tkinter zur optimalen Wahl das Projekt.

4 Design der grafischen Benutzeroberfläche

Dieses Kapitel beleuchtet den Aufbau der GUI. Sie ist in vier Bereiche unterteilt. In den folgenden Abschnitten werden die verwendeten Tkinter-Komponenten der Bereiche und deren Funktion erläutert.

4.1 Toolbar

Die Toolbar, siehe Abbildung 4.1, befindet sich am oberen Rand mittig im Fenster. Sie wird für die Hauptoperationen auf dem Baum verwendet und wurde dort platziert, um Aufmerksamkeit zu erregen. Die ersten drei Tkinter-Widgets werden verwendet, um grundlegende Operationen zu starten. Dafür ist ein Entry-Widget vorhanden, welches erweitert wurde, um einen Platzhalter anzeigen zu können. Zudem kommt ein Dropdown-Menü, aus dem die Operationen ausgewählt werden können. Dazu kommt noch ein Tkinter-Knopf, welcher der Startknopf der gewählten Operation ist. Anschließend gibt es die Möglichkeit mehrere Werte in den Baum einzufügen. Dafür wird erneut ein erweitertes Entry-Widget verwendet in dem Werte, getrennt mit Semikola eingetragen werden können und mit dem Insert_Multiple-Knopf gestartet wird. Der nächste Bereich ermöglicht es, die Operationen schrittweise durchlaufen zu lassen zu können. Dafür werden zwei Knöpfe verwendet. Der Step_Wise-Knopf schaltet hierbei zwischen den Modi umher. Befindet man sich im Stepwise-Modus, wird zudem der Knopf Next_Step sichtbar. Das letzte Widget ist der Settings-Knopf, welcher das Fenster für das SettingsWindow [Kapitel 4.4] öffnet.

Abbildung 4.1: Toolbar

4.2 Bottombar

Die Bottombar, siehe Abbildung 4.2, befindet sich an dem unteren Rand des Fensters. Sie ist konzeptionell in drei Bereiche aufgeteilt. Der erste Bereich ermöglicht die Traversierung des Baums mit den in Abschnitt 2.2.4 erklärten Methoden durch drei Knöpfe.Öberhalb der Bottombar wird die Reihenfolge der Traversion dargestellt. Der nächste Abschnitt bietet die Funktion, aus einem Dropdown-Menü eine Anzahl an zufälligen Werten in den Baum einzufügen. Gestartet wird dies durch den angrenzedenen Knopf Generate_Random_Nodes. Abschließend folgt der Bereich, der es den nutzenden Personen ermöglicht Bäume abzuspeichern und aus Dateien zu laden. Dafür gibt es ein Entry-Widget und zwei Knöpfe: Export und Import.

Abbildung 4.2: Bottombar

4.3 MainFrame

Das MainFrame Objekt, siehe Abbildung 4.3 ist ein Tkinter-Frame , welches die Tkinter-Leinwand auf der linken Seite beinhaltet, auf der der aktuelle Schritt mithilfe der echten Werte des Baumes als Text-Element sowie der aktuelle Baum mithilfe von Text- und Oval-Elementen dargestellt wird. Auf der rechten Seite wird mithilfe eines Text-Widgets der vollständige Pseudocode für die aktuelle Operation angezeigt.

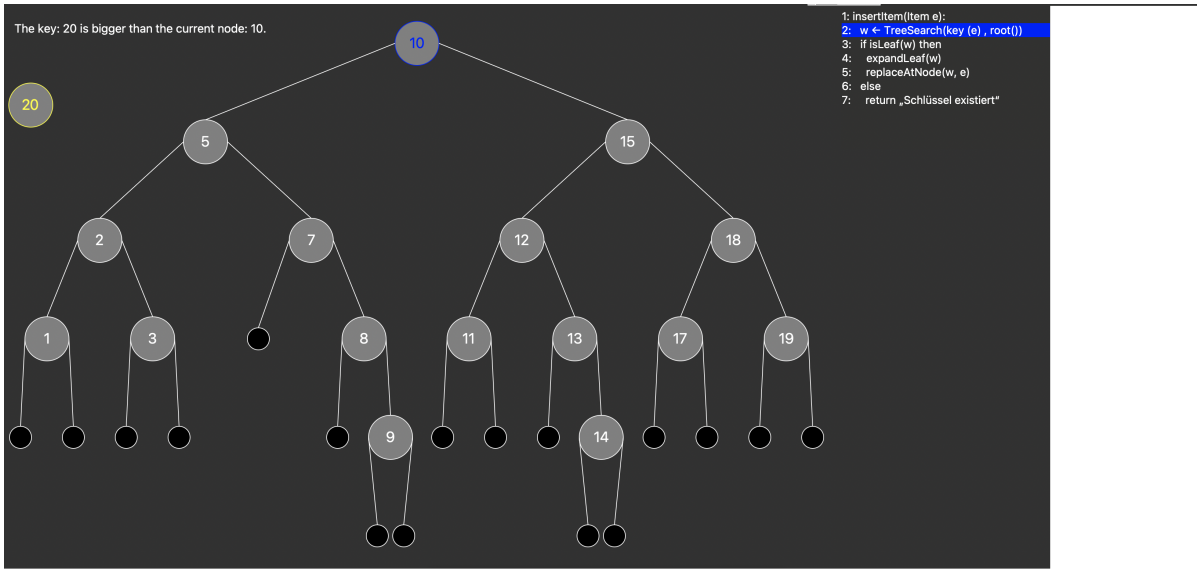

Abbildung 4.3: MainFrame

4.4 SettingsWindow

Das SettingsWindow, siehe Abbildung 4.4, ist ein Fenster, welches es ermöglicht die Einstellungen der GUI anzupassen. Dafür gibt es zwei Knöpfe, welche ein Colorchooser-Dialog öffnen, in dem die nutzende Person neue Farben für das Design der markierten Zeile im Pseudocode sowie die Farbe des Knotens, welcher eingefügt werden soll, verändern kann. Darauf folgt ein Slider-Widget, mit dem die Schriftgröße des Pseudocodes verändert werden kann. Darunter folgen zwei Widegts, die es ermöglichen die Reichweite der Werte der zufällig generierten Knoten anzupassen. Dafür werden ein erweitertes Entry-Widget und ein Knopf verwendet. Außerdem wird mithilfe einer Checkbox die Möglichkeit geboten, die Blätter des Baumes zu verbergen.

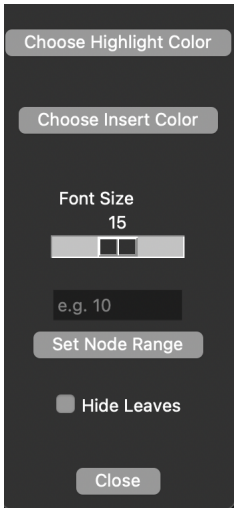

Abbildung 4.4: SettingsWindow

5 Implementierung

In den folgenden Kapiteln werden zunächst die grundlegenden Konzepte der Implementation beschrieben. Darauf folgt die Beleuchtung der Funktionen, die verwendet werden um, die GUI umzusetzen.

5.1 Konzept

Die OrdnerStruktur ermöglicht einen geordneten Überblick über die unterschiedlichen Komponenten.

- Innerhalb des *Pseudocode*-Ordners befinden sich die Dateien *delete_code*, *insert_code*, *search_code*, *inorder_code*, *postorder_code* und *preorder_code*. Diese beinhalten jeweils Listen des Pseudocodes der gleichnamigen Operationen, welche in der GUI dargestellt werden.
- In dem *Tree_files*-Ordner werden exportierte binären Suchbäume als Textdatei abgespeichert, um sie später wiederfinden zu können.
- Der *Tree_Struct*-Ordner enthält die Klassen *TreeNode*, die die Struktur eines Knotens vorgibt, sowie *BinaryTree*, welche die Logik des Baumes umsetzt. Zudem wird die Klasse für das Generieren des Inputs für die Exportation der Bäume genutzt.
- Der *UI*-Ordner enthält die Klassen *BinarySearchTreeGUI* und *Animations*. BinarySearchTreeGUI stellt die Oberfläche dar und ist aus den Widgets *Toolbar*, *MainFrame* und *Bottombar* zusammengesetzt. Animations setzt die Darstellung des Baumes sowie des richtigen Pseudocodes um.
- In dem Widget-Ordner befinden sich die Widgets *Bottombar*, *EntryWithPlaceholder*, *MainFrame*, *SettingsWindow* und *Toolbar*. Die Klasse EntryWithPlaceholder ist eine Erweiterung des Tkinter-Widgets Entry, welches es ermöglicht, einen Platzhalter in dem Widget zu realisieren. Dies erleichtert es nutzenden Personen der Software, nachvollziehen zu können, wie Einträge für das jeweilige Textfeld aussehen sollen. Auf die Widgets Bottombar, Toolbar, SettingsWindow und MainFrame und deren Komponenten wurde in Abschnitt 4 eingegangen.

Die Datei *main.py* startet nachdem ein Eventlistener an das Hauptfenster geheftet wurde, die Mainloop und somit die GUI. Der Eventlistener wird verwendet, um bei einer Anpassung der Größe des Fensters den Baum neu zeichnen zu lassen, damit dieser die passende Größe für das Fenster behält.

5.2 Darstellen des Pseudocodes

Die GUI ermöglicht einerseits die genaue Darstellung des aktuellen Schrittes mit der Funktion *display_canvas_pseudocode*. Durch diese Funktion wird der Text des *canvas_pseudocode* Widgets manipuliert. Der aktuelle Schritt zeigt im Gegensatz zu dem im *complete_pseudocode* dargestellten Pseudocode die verglichenen Werte während der Operation an. Andererseits existiert ein Widget, welches durch *display_pseudocode* verändert wird. Hierbei werden entsprechend zu der jeweils laufenden Operation die Listen aus den Dateien des Pseudocode-Ordners durchlaufen.

Darüber hinaus besteht die Möglichkeit, Zeilen hervorzuheben. Dies geschieht durch das Setzen des Parameters *highlighted_line*. Wenn dieser gesetzt ist, zählt eine Variable hoch, bis die gewünschte Zeile erreicht ist. Diese erhält die Markierung *highlighted*, um farblich hervorgehoben zu werden. Das Markieren der entsprechenden Zeile erleichtert das Nachvollziehen der momentan durchgeführten Schritte des Algorithmus.

5.3 Baumdarstellung

Der binäre Suchbaum wird auf einer Leinwand durch die Funktion *draw_tree* gezeichnet. Die Knoten bestehen aus einem Text mit dem entsprechenden Wert und einem Oval, welches diesen umschließt. Die Funktion hat die Parameter *current_node* und *inserted_key*. Der Parameter *current_node* legt fest, welcher Wert aktuell in der Animation überprüft wird und stellt ihn in der Farbe für hervorgehobene Knoten (*HIGHLIGHT_COLOR*) statt der normalen Farbe (*OUTLINE_COLOR*) dar. Dies ermöglicht es den Überblick zu behalten, welcher der aktuell zu vergleichende Knoten ist. Besonders wichtig während der Einfüg-Operation ist der Parameter *inserted_key*, welcher verwendet wird, um den einzufügenden Knoten zunächst unabhängig von dem Baum zu erzeugen und nach erfolgreichem Einfügen durch *animate_movement* zu dem richtigen Platz zu bewegen. Sollte die Wurzel nicht existieren und die Option, die Blätter zu sehen, ist aktiviert, stellt die Funktion ein leeres Blatt dar, da der Baum leer ist. Die Funktion ruft, sollte eine Wurzel existieren, eine rekursive Funktion *draw_tree_recursive* auf, welche zusätzlich noch x, y, dx sowie dy übergeben bekommt. Die Parameter x und y beschreiben die Koordinaten des Knotens auf der Leinwand. Die Parameter

dx und dy werden berechnet durch:

$$dx = \frac{\text{canvas width}}{4} \tag{5.1}$$

$$dy = \frac{\text{canvas height}}{\text{tree height}} \tag{5.2}$$

Die Position der Kinderknoten wird anschließend berechnet mit:

$$x\_left = x - dx \qquad y\_left = y + dy \tag{5.3}$$

$$x\_right = x + dx \qquad y\_right = x + dy \tag{5.4}$$

Die Funktion wird nun für den rechten und linken Kindknoten mit den errechneten x_left und y_left, beziehungsweise x_right und y_right Werten aufgerufen, sofern dieser Knoten existiert. Die neue Iteration der Kindknoten bekommt $dx = dx/2$ übergeben. Falls kein Knoten mehr folgt und die Option Blätter anzuzeigen gewählt ist, wird ein leeres Oval mit der in *BACKGROUND_COLOR_LEAF* festgelegten Farbe erstellt.

5.4 Visualisierung der Operationen

Die Animationen der Operationen sind in der *Animations*-Klasse implementiert und werden nach dem Drücken des Startknopfes ausgelöst. Sie bekommen den Eintrag der Textbox als Schlüssel *key* übergeben. Nach Betätigung des Knopfes wird überprüft, welche Operation aktuell ausgewählt ist, und diese wird durch *insert_element*, *delete_element* oder *search_element* gestartet. Während der Veranschaulichung werden die Funktionen *draw_tree* sowie *display_pseudocode* verwendet, um den jeweiligen Schritt darzustellen. Der hierbei gezeigte Pseudocode entspricht dem in Abschnitt 2.2 erläuterten. Dieser wird auf der rechten Seite dargestellt. Zusätzlich wird der momentane Schritt konkret links über dem Baum geschrieben.

5.5 Traversierung

Die Knöpfe, um die jeweiligen Traversionsmethoden zu starten, befinden sich in der Bottombar, welche unterhalb der Tree-Leinwand ist. Entsprechend der in Abschnitt 2.2.4 beleuchteten Methoden wird der Baum abgearbeitet. Dafür wird bei jedem Besuch eines Knotens der Baum mit der Funktion draw_tree neu gezeichnet. Dabei wird der aktuellen Knoten als Parameter für current_node übergeben, damit der entsprechende Knoten hervorgehoben wird. Dies macht es für die nutzende Peron kenntlich, welcher Knoten aktuell abgearbeitet wird. Zusätzlich wird die *print_order_var* Variable um den Wert des aktuellen Knotens erweitert, sodass oberhalb der Bottombar das Label die Ausgabe des Algorithmus anzeigt.

5.6 Animation

Die Animation wird mit der Funktion *animate_movement* ausgeführt. Die Parameter *duration* und *keys* werden hierbei übergeben. Bei duration handelt es sich um die Animationsgeschwindigkeit, welche über ein Slider-Widget angepasst werden kann. Dies wirkt sich auf die Anzahl der Schritte, sowie auf die Zeit zwischen den einzelnen Schritten aus, die während der Animation gemacht werden. Keys hingegen ist eine Liste, welche die Werte, die bewegt werden sollen, sowie die neuen X- und Y-Koordinaten beinhaltet. Die Funktionalität ist in zwei Schritte aufgeteilt. Zunächst wird in *calc_animation* berechnet, wie sich das Element bewegen soll. Dafür wird berechnet, wie viele Schritte (steps) gemacht werden müssen. Mit der Information kann nun ein Vektor für die Bewegung berechnet werden.

$$\text{steps} = \frac{\text{duration}}{\text{MS\_BETWEEN\_STEPS}} \tag{5.5}$$

$$\text{Vektoren der Bewegung} = \frac{\text{Ursprung} - \text{Ziel}}{\text{steps}} \tag{5.6}$$

Dies wird in einer Liste *pos* mit folgendem Schema für ein Element der Liste abgespeichert: $[[text, xt, yt, xst, yst], [oval, xo, yo, xso, yso]]$.

- text = Das Textelement, welches bewegt wird.
- xt,yt = X- und Y-Werte des Textes.
- xst,yst = Die Länge eines Schrittes des Textes in X und Y Richtung.
- oval = Das Oval, welches bewegt wird.
- xo,yo = X- und-Y Werte des Ovales.
- xso, yso = Die Länge eines Schrittes in X- und Y-Richtung.

Anschließend wird die Funktion *move_step* ausgeführt, startend mit step = 0. Hier werden die beiden Elemente der Leinwand bewegt. Dies wird durch

$$\text{Neue Koordinaten} = \text{Ursprung} + \text{Bewegungsvektor} * \text{step} \tag{5.7}$$

berechnet. Nachdem die Bewegung abgeschlossen ist, wird die Funktion mit step + 1 aufgerufen. Sollte der aktuelle Schritt der letzte sein, wird der Baum gezeichnet, um veraltete Verbindungen und Blätter zu löschen. Der letzte Schritt zeichnet sich dadurch aus, dass step = steps ist.

5.7 Import, Export und Multiple-Insert

Die Funktionen für das Einfügen mehrerer Knoten sowie das Importieren ganzer Bäume soll nutzende Personen unterstützen, spezifische Bäume effizienter als durch einzelne Eingaben darzustellen. Dies ermöglicht es, genutzte Bäume aus Vorlesungen und Übungen schneller nachzustellen und bestimmte Operationen auf spezifischen Bäumen zu visualisieren. Multiple-Insert verwendet das animierte Einfügen. Das Importieren ganzer Bäume aus Dateien, jedoch nur die Einfügfunktion des Baumes wird nicht animiert und nur, nachdem jedes Element eingefügt wurde, einmal gemalt.

Bei dem Exportieren wird eine Textdatei erstellt. Diese ist, wie im Export_Name-Entry benannt, spezifiziert. Der Inhalt der Datei wird mit der Funktion *generate_file_input* generiert. Diese schreibt zunächst die Werte des Baumes in Preorder, getrennt durch Semikola, in einen String. Dies wird durch die rekursive Funktion *generate_preorder* realisiert. Anschließend wird *generate_file_input_recursive* ausgeführt, um einen für Menschen lesbaren Baum zu dem String hinzuzufügen. Diese Methode folgt der Idee einer Antwort auf Stackoverflow [6]. Das Ergebnis dieser Funktionen wird in die Datei geschrieben.

5.8 Skalieren

Für das Skalieren wird die Funktion *find_collision* verwendet. Diese sucht nach Kollisionen von zwei Knoten anhand ihrer Ovale. Dafür wird die Methode *find_overlapping* von Tkinter verwendet. Diese gibt eine Liste an Objekten zurück, welche sich innerhalb von angegebenen Koordinaten befinden. Diese Liste wird auf Ovale und Blätter geprüft. Sollte es eine Kollision geben, wird die Variable *scale* angepasst und der Baum erneut gezeichnet. Anschließend wird in der *draw_tree* Methode find_collision erneut aufgerufen. Dadurch wird überprüft, ob weiterhin Kollisionen bestehen und sichergestellt, dass kein Oval über einem anderen ist, was zu der Lesbarkeit des Baumes beiträgt.

6 Future Work

Diese Arbeit liefert nicht nur die Möglichkeit, den nutzenden Personen binäre Suchbäume näher zu bringen, sondern auch das Potenzial, zusammen mit ähnlichen Programmen eine Komponente für ein neues Programm zu bieten, welches es ermöglicht, verschiedene Datenstrukturen und Algorithmen zu erforschen. Besonders im Hinblick auf andere Baumarten kann es hier sinnvoll sein, diese zusammenzuführen. Sofern keine neuen Operationen benötigt werden, muss die GUI nur insofern angepasst werden, dass ein Feld ergänzt wird, mit dem es den nutzenden Personen möglich ist, den gewünschten Baumtypen zu wählen. Zudem ist es nötig, die neue Baumstruktur in dem Ordner tree_struct zu ergänzen. Sofern es neue Animationen oder andere Abläufe der Operationen gibt, müssen diese ebenfalls angepasst werden. Alternativ könnte eine GUI entworfen werden, welche bestehende Programme mit anderen Datenstrukturen, die bereits realisiert wurden, zusammenfügt und die entsprechenden Programme mit der Auswahl der Sturktur startet. Mithilfe der Import-Methode ist es zudem möglich, einen Baum aus einer Datei in den diversen Baumarten zu visualisieren indem die Operation des Einfügens für ebendiese Art verwendet wird.

Literatur

[1] *Flet*. https://flet.dev. , letzter Zugriff am: 26.05.2024.

[2] *GUI Programming in Python*. https://wiki.python.org/moin/GuiProgramming. , letzter Zugriff am: 26.05.2024.

[3] Pat Morin. *Open Data Structures*. AU Press, 2013.

[4] Thomas Ottmann und Peter Widmayer. *Algorithmen und Datenstrukturen 6. Auflage*. Springer, 2017.

[5] *pywebview*. https://pywebview.flowrl.com. , letzter Zugriff am: 26.05.2024.

[6] *Stackoverflow: print binary tree level by level in python*. https://stackoverflow.com/questions/34012886/print-binary-tree-level-by-level-in-python. , letzter Zugriff am: 26.05.2024.

[7] *Tkinter*. https://docs.python.org/3/library/tkinter.html. , letzter Zugriff am: 26.05.2024.