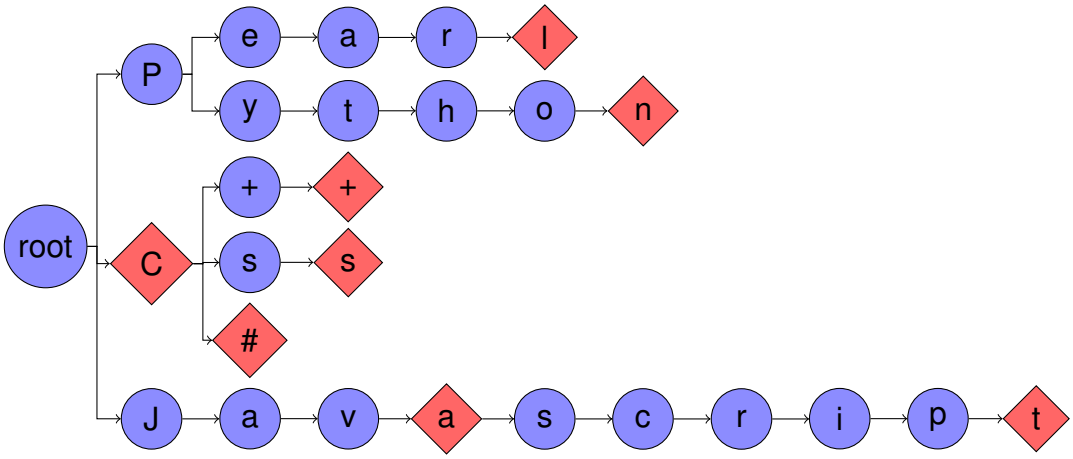
IMS
Institut für Mikroelektronische Systeme
Leibniz Universität Hannover

Programmierpraktikum Technische Informatik (C++)
Programmiertutorium Vorlesung 4

Pullen Sie von ppti-common. Sie finden dann in ihrem Repository den Ordner `tutorials/Lecture04/`, in dem die folgenden Aufgaben zu bearbeiten sind.

In diesem Tutorial sollen Sie einen Trie, einen speziellen Suchbaum zum Durchsuchen von Strings, implementieren. Dabei entspricht ein Knoten im Trie einem Zeichen im String und das nächste Zeichen wird durch das Kind des jeweiligen Knotens dargestellt. Sollte ein String vollständig eingetragen sein, so wird das letzte Zeichen mit einem Marker versehen. Eine Anwendung ist zum Beispiel die Autovervollständigung. Unten sehen Sie beispielhaft einen Trie, in dem einige Wörter eingetragen sind.



Teilaufgabe 1

Als erstes Ziel soll es möglich sein, Wörter in den Trie hineinzuschreiben. Dafür werden die Funktion `addWord` und die Hilfsfunktion `findChild` verwendet.

a) Implementieren Sie zunächst die Funktion `Node* Node::findChild(char c)`. Diese soll überprüfen, ob der Knoten ein Kind hat, dessen Inhalt `c` entspricht. Wenn ja soll einen Pointer auf das Kind zurückgegeben werden, ansonsten einen Nullpointer.

b) Zum Einfügen eines Wortes wird die Funktion `void Trie::addWord(const std::string& s)` benötigt. Diese soll, stets bei der Wurzel beginnend, den Pfad im Baum durchlaufen, der `s` entspricht. Sollten Sie

Appelstraße 4 | 30167 Hannover
fon 0511 762-19691 | fax 0511 762-19694
www.ims.uni-hannover.de | info@ims.uni-hannover.de

dabei auf einen fehlenden Knoten stoßen, soll dieser hinzugefügt werden. Bei dem letzten Character, egal ob bereits vorhanden oder erst erstellt, soll der `endMarker` des Knotens gesetzt werden, der das Ende eines Wortes anzeigt.

Teilaufgabe 2

Für die nächsten Aufgaben wird die Funktion `lastPossibleNode(const std::string& s)` benötigt. Diese gibt den letzten mit `s` übereinstimmenden Knoten und die Länge des gefunden Wortes zurück.

a) Implementieren Sie die Funktion `std::pair<Node*, size_t> Trie::lastPossibleNode(const std::string& s)`. Fangen Sie bei der Wurzel des Baumes an, einen Pfad zu suchen, der teiweise oder ganz mit `s` übereinstimmt. Ein Pointer auf den letzten übereinstimmenden Knoten und die Länge des gefundenen (Teil)Wortes sollen als `pair` zurückgegeben werden.

b) Mit der von Ihnen implementierten Funktion `lastPossibleNode(std::string& s)` ist es nun auch möglich, ganze Wörter im Baum zu suchen. Erstellen Sie dazu die Funktion `bool Trie::searchWord(const std::string& s)`. Diese soll `true` zurückgegeben, wenn das ganze Wort im Baum gefunden wurde. Ansonsten wird `false` zurückgegeben.

Teilaufgabe 3

Als nächstes soll es möglich sein, Wörter aus dem Baum auch wieder zu löschen Zu diesem Zweck soll die Funktion `void Trie::deleteWord(const std::string& s)` implementiert werden. Es sollen alle Knoten gelöscht werden, die zu dem übergebenen Wort gehören, ohne die Vollständigkeit anderer Wörter zu beeinflussen. Dafür ist es hilfreich, erst das Ende eines Wortes zu finden und sich von da Richtung Wurzel vorzuarbeiten. Achten Sie dabei auf mögliche Ausnahmefälle.

Teilaufgabe 4

Für den erwähnten Anwendungszweck der Autovervollständigung soll nun folgende Aufgabe umgesetzt werden.

a) Zunächst soll die Funktion `void Node::findPossibleWords(std::string s, std::vector<std::string>& result)` implementiert werden. Die Funktion soll für jedes Kind rekursiv aufgerufen werden. Dabei soll der übergebene String `s` jeweils um den Inhalt des Kindes ergänzt werden. Wenn einer der Knoten einen `endMarker` hat, wurde ein Wort gefunden und es wird `result` hinzugefügt.

b) Implementieren Sie nun die Funktion
`std::vector<std::string> Trie::autoComplete(const std:string& s)`. Sie soll einen String übergeben bekommen und alle möglichen Vervollständigungen in einem Vektor zurückgeben. Hierzu wird der letzte Knoten von `s` im Trie gesucht, um dort `findPossibleWords` auszuführen. Am Ende wird `result` zurückgegeben.