# COIS 3020H Assignment 3: R-Way Tries README Documentation

## Description

The R-Way Trie is a C# implementation of a Radix Trie data structure and associated nodes (RTrieNode). This implementation allows efficient storage and retrieval of strings with support for optional associated values. The code provides functionalities for insertion, search, removal, and traversal of the trie.

## Contributors:

Adam Kassana
Alexander Wolf
Carmen Tullio

## Content Table

# RTrieNode Class

## Overview:

The RTrieNode class represents a node in a Radix Trie (RTrie). Each node holds information about a specific character in a string and is part of the overall trie structure. The class contains private fields to store the value associated with the node, the number of children it has, and a dictionary to efficiently store child nodes based on individual characters. The RTrieNode class is responsible for creating nodes and organizing them in a tree-like structure.

## Class Fields:

- private int nValue
  - Represents the value associated with the node.
- private int childQuantity
  - Represents the number of children nodes.
- public Dictionary<char, RTrieNode> children
  - A dictionary to store child nodes based on individual characters.

## Constructors

public RTrieNode()

Parameters: None

Time Complexity: O(1)

Description: Creates a new instance of RTrieNode. Initializes the node with default values.

Example of use: `RTrieNode node = new RTrieNode();`

# RTrie Class

## Overview

The RTrie class is the main implementation of the Radix Trie data structure. It manages a collection of RTrieNode instances to represent a trie efficiently. The class includes constructors to initialize an empty trie or create a trie from an array of words or a file. The trie supports operations such as insertion, search, and removal of words. It also provides methods for traversing and printing the trie, particularly for words with a specific prefix.

## Class Fields

private RTrieNode root: The root node of the trie.

## Constructors

### String Constructor

public RTrie(string[] words, int value = -1)

Parameters:

string words - An array of strings to initialize the trie.

int value (optional) - Default value to associate with each word.

Time Complexity: O(L * M), where L is the total length of words and M is the average length of a word.

Description: Initializes an RTrie with an array of words. Optionally accepts a default value for each word.

Example of use:

```
string[] wordsArray = { "apple", "banana", "orange" };
RTrie trie = new RTrie(wordsArray, 42);
```

### Path Constructor

public RTrie(string path, int value = -1)

Parameters:

string path - Path to a file containing words to initialize the trie.

int value (optional) - Default value to associate with each word.

Time Complexity: O(L * M), where L is the total length of words and M is the average length of a word.

Description: Initializes an RTrie by reading words from a file specified by the provided path. Optionally accepts a default value for each word.

Example of use:

```
string filePath = "path/to/words.txt";
RTrie trieFromFile = new RTrie(filePath, 10);
```

### Default Constructor

public RTrie()

Parameters: None

Time Complexity: O(1)

Description: Initializes an empty RTrie.

Example of use: `RTrie emptyTrie = new RTrie();`

## Methods

### Prefix Match

Header: public void PrefixMatch(string prefix, RTrieNode current = null)

Parameters:

string prefix - The prefix to match.

RTrieNode current (optional) - The current node in the trie. Used for recursion.

Time Complexity: O(P + K), where P is the length of the prefix and K is the total number of nodes matching the prefix.

Description: Prints all words in the trie with a given prefix.

Example of use:

```
RTrie trie = new RTrie(new string[] { "apple", "banana",
"orange", "app" }, 42);
trie.PrefixMatch("app");
```

### Print

Header: public void Print(RTrieNode current = null)

Parameters:

RTrieNode current (optional) - The current node in the trie. Used for recursion.

Time Complexity: O(N), where N is the total number of nodes in the trie.

Description: Prints all words in the trie.

Example of use: `trie.Print();`

### Insert

Header: public bool Insert(string key, int value)

Parameters:

key - The string key to insert.

value - The associated value to insert.

Time Complexity: O(L), where L is the length of the key.

Description: Inserts a key-value pair into the trie. Returns true if successful, false otherwise.

Example of use: `bool inserted = trie.Insert("apple", 42);`

### Search

Header: public int Search(string key)

Parameters:

key - The string key to search for.

Time Complexity: O(min(L, M)), where L is the length of the key and M is the length of the longest key in the trie.

Description: Searches for a key in the trie. Returns the associated value or -1 if not found.

Example of use: `int result = trie.Search("banana");`

### Remove

public bool Remove(string key)
Header: public bool Remove(string key)

Parameters:

key - The string key to remove.

Time Complexity: O(min(L, M)), where L is the length of the key and M is the length of the longest key in the trie.

Description: Removes a key from the trie. Returns true if successful, false otherwise.

Example of use: `bool removed = trie.Remove("banana");`