Department of Computer Science
Trent University

COIS 3020H: Data Structures and Algorithms II

Assignment 2: Ropes

(100 marks)

(due Monday, March 8, 2021 at 23:59 EST)

Note

Late assignments will be deducted 10% per day up to five days (including weekends).

Background

A string is a sequence of characters and is often implemented as a fixed-length array of characters. Therefore, operations like concat, split, insert and remove can consume $O(n)$ time where n is the total length of the string(s)[1].

One alternative implementation for very long strings is called a **rope**. A rope is an augmented binary tree whose leaf nodes represent parts of the overall string (see example on the last page). In this example, each node is augmented with an integer that represents the length of the string in its left subtree. However, to ensure that concatenation runs in $O(1)$ time, it may be better to store the total length rooted at a particular node. Why?

One of the big advantages of the rope data structure is its support of immutable strings. An immutable string is one that cannot be modified. Therefore, any modifications result in the creation of a new string as in C#. However, until a modification is done, only one copy of the string is necessary. As soon as the string is modified, the changes are made and a new string is created. This is known as copy-to-write. The rope data structure can maintain the various versions of a string as an overlay which saves a great deal on space. For this assignment though, the string represented by the Rope class can be modified and is therefore, not immutable. But it is interesting to think about how a new string (rope) can be built upon a previous version without destroying the original.

Requirements

Your task is to design, implement, test, and document the following methods for the Rope class. The notation $S[i, j]$ is used to represent the substring that begins at index i and ends at index j. Assume as well that the maximum size of a leaf substring is 10.

Rope( ) : Create an empty rope (1 mark).

Rope(string S) : Create a balanced rope from a given string S (8 marks).

Rope Concatenate(Rope R1, Rope R2) : Return the concatenation of ropes $R1$ and $R2$ (4 marks).

void Split(int i, Rope R1, Rope R2) : Return the two ropes split at index i (10 marks).

void Insert(string S, int i) : Insert string S at index i (6 marks).

[1] Review the C# library class for string.

void Delete(int i, int j) : Delete the substring $S[i, j]$ (6 marks).

string Substring(int i, int j) : Return the substring $S[i, j]$ (8 marks).

char CharAt(int i) : Return the character at index i (4 marks).

int IndexOf(char c) : Return the index of the first occurrence of character c (4 marks).

void Reverse() : Reverse the string represented by the current rope (6 marks).

int Length() : Return the length of the string (1 mark).

string ToString() : Return the string represented by the current rope (4 marks).

void PrintRope() : Print the augmented binary tree of the current rope (4 marks).

Optimizations

The ideal rope maintains a height of $O(\log n)$, but that can be quite a challenge. To help limit the height of the tree, try to implement the following optimizations.

1. After a Split, compress if possible the path back to the root (4 marks).
2. Combine left and right siblings whose total string length is 5 or less (4 marks).
3. Rebalance the rope periodically (4 marks).

Grading Scheme

Rope methods	66
Optimizations	12
Testing	16
Documentation	6

References

Hans-J. Boehm, Russ Atkinson, and Michael Plass, *Ropes: an Alternative to Strings*, Software - Practice and Experience, Volume 25(12), December 1995, pp 1315-1330

Dale King, *Ropes - Fast Strings*, January 2, 2017

Rope (data structure), *Wikipedia*, retrieved on February 10, 2021