Shakespearean Words

Goals

The purpose of this assignment is to learn to

1. Access Shakespeare's work with BRIDGES
2. Write a Dictionary using BRIDGES
3. Write your own tree based data structure: A Binary Min-Heap

You will generate a visualization that looks like that!

Purpose

The purpose of the assignment is to analyze out of Shakespeare works which words are used most often. The assignment is in two pieces. First, we will need to count how many times each words appear. This will be accomplished using a Dictionary implemented as a Binary Search Tree. Then, we will extract the words used most often. This will be accomplished using a Min Heap.

Programming part

Counting word appearances using a Dictionary and a custom implementation

A Dictionary (sometimes called associative arrays) enable to store and retrieve (key, value) pairs. In this assignment they will be useful to count how many times a particular word appear in Shakespeare's work. The keys are going to be words. And the value associated with that key is going to be how many times that word appears. Counting the words then becomes:

```
Dictionary d

for each word w in document
  entry = d.get(w)
  if (entry is NULL)
    d.insert (w, 1)
  else
    entry.value += 1
```

Getting Started

1. Open your scaffolded code.
2. Plug in your credentials.
3. Observe the Dictionary interface.

Tasks

1. Use the Dictionary implementation to compute the number of occurencecs of each word.
2. Implement you own Dictionary using a Binary Search Tree leveraging the BSTElement of BRIDGES.
3. Visualize the Dictionary using BRIDGES.

If you have time

1. Implement the Dictionary using a HashTable.
2. Use BRIDGES SymbolCollection to generate a visualization of the HashTable.
3. Use all of Shakespeare work and measure the performance difference between the HashTable and the BST implementations.

Extracting the most frequent words using a Min Heap

The purpose of this task is to build a MinHeap in BRIDGES represented as a binary tree (as opposed to the

more common array representation of a heap)

Recall that as a binary tree, a heap defined recursively as a root and two subheaps. The invariant of a min heap is that the root of any heap should have a lower (or equal) key than any node contained in the heap.

Getting Started

1. Observe the `MyHeapElement` class that extends BRIDGES's `BinTreeElement`.
2. Observe the `MyHeap` class that provide Min Heap features.

Build a Binary Min Heap

1. Write the insert function in MyHeap.

The algorithm for inserting in a heap is as follows. (This algorithm ignores that there is a key and a value.) Note that it uses information about the size of the subheaps being stored at each node of the heap.

```
Heap {
  Key
  HeapLeft
  SizeHeapLeft
  HeapRight
  SizeHeapRight
}

insert (Heap h, k) {
  if (h is empty)
    return makenewheap (k)

  if (k < h.Key)
    swap k and h.Key

  if (SizeHeapLeft < SizeHeapRight)
    //push left
    SizeHeapLeft = SizeHeapLeft + 1
    HeapLeft = insert (h.HeapLeft, k)
  else
    //push right
    SizeHeapRight = SizeHeapRight + 1
    HeapRight = insert (h.HeapRight, k)

  return h
}
```

2. Write a pop function that return the element with the lowest key.
3. Use the heap to identify the most occuring words in Shakespeare work.

If you have time

1. Use all of Shakespeare's works.
2. Using the pop function, keep only the 100 most occuring words in the heap at any time.
3. Measure the performance difference between keeping all entries in the heap and only the top-100 most occuring words.
4. Style the heap so that words with more than 1000 occurences are highlighted.

Help

for Java

Color documentation

BinTreeElement documentation

Shakespeare documentation

Bridges class documentation

for C++

Color documentation

BinTreeElement documentation

DataSource documentation

Shakespeare documentation

For Python

Bridges documentation

Color documentation

BinTreeElement documentation