CSE 100 Discussion 4

Wenjia Ouyang
Announcement

• 1. PA2:
  – Due Oct 22\textsuperscript{nd} 8pm (Thu)
  – Early submission @Oct 20\textsuperscript{th} 8pm (Tue)

• 2. Midterm:
  – Section A&B: Oct 19\textsuperscript{th} (Mon)
    • LAST NAME V-Z: A:WHL 2111; B:WLH2207
  – Section C: Oct 29\textsuperscript{th} (Thu)
PA2
RANDOM SEARCH TREES
Files

- benchtree.cpp
- BST.hpp
- BSTIterator.hpp
- BSTNode.hpp
- countint.cpp
- countint.hpp
- Makefile
- RST.hpp
- test_RST.cpp
#ifndef RST_HPP
#define RST_HPP

#include "BST.hpp"
#include <iostream>

using namespace std;

template <typename Data>
class RST : public BST<Data> {

public:

/* Insert new data into the RST if it is not already present.
 * Input
 *  item - the data to insert.
 *  return - true if the item was inserted. If the item was already contained in
 *            the tree then return false.
 */
  virtual bool insert(const Data& item) {
    // TODO: Implement this function!
    return true;
  }
};
rotateRight

```cpp
/* Perform a 'right rotation' that changes the structure of the tree without
* interfering with the size, data content, and left->right order of nodes in the RST.
* This rotation is done by changing the 'left', 'right', and/or 'parent' pointers
* of a subset of nodes without changing their 'priority' values.
*
* Input
*    par - a parent node in the RST with a left child
*    child - the left child of 'par'
*
* Output
*    The resulting tree will have 'par' and 'child' in the same order but switched
*    in level, meaning 'par' will effectively become the right-child of 'child'.
*
*/
void rotateRight( BSTNode<Data>* par, BSTNode<Data>* child ) {
    // TODO: Implement this function!
}
```

rotateLeft

```cpp
/* Perform a 'left rotation' that changes the structure of the tree without
* interfering with the size, data content, and left->right order of nodes in the RST.
* This rotation is done by changing the 'left', 'right', and/or 'parent' pointers
* of a subset of nodes without changing their 'priority' values.
*
* Input
*    par - a parent node in the RST with a right child
*    child - the right child of 'par'
*
* Output
*    The resulting tree will have 'par' and 'child' in the same order but switched
*    in level, meaning 'par' will effectively become the left-child of 'child'.
*
*/
void rotateLeft( BSTNode<Data>* par, BSTNode<Data>* child ) {
    // TODO: Implement this function!
}
```
Generate Random Numbers

• Random numbers can be generated in C++ using the `rand()` function from the `<stdlib.h>`.
  – `newNode->priority = rand();`

• `srand(seed)`
  – **Initialize random number generator**
  – Two different initializations with the **same seed** will generate the **same** succession of results in subsequent calls to `rand`.  
  – Only need to seed (call `srand`) once. **Don’t seed every time you insert**
  – Use it only in debugging, **not in the benchmarks calculation**.
```cpp
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    unsigned seed;
    printf("Enter seed: ");
    scanf("%u", &seed);
    //set seed
    srand(seed);
    //print generate 10 random number
    int myRnd;
    for (int i = 0; i <= 10; i++) {
        myRnd = rand();
        printf("%d", myRnd);
        printf("\n");
    }
}
```
benchtree.cpp

• Your benchtree.cpp must take the following four command line arguments.

  ./benchtree rst sorted 32768 5

• Performance: average comparisons
  – For each run $i$, compute the average number of comparisons for a successful find; call this $t_i$.

  \[ \hat{t} = \frac{1}{R} \sum_{i=1}^{R} t_i \]

• Standard Deviation

  \[ \sigma = \sqrt{\hat{s} - \hat{t}^2} \quad \text{, where} \quad \hat{s} = \frac{1}{R} \sum_{i=1}^{R} t_i^2 \]
Example

```cpp
std::vector<countint> v;
v.clear();
for(int i=0; i<N; ++i) {
  v.push_back(i);
}

std::vector<countint>::iterator vit = v.begin();
std::vector<countint>::iterator ven = v.end();

for(vit = v.begin(); vit != ven; ++vit) {
  s.insert(*vit);
}
countint::clearcount();
for(vit = v.begin(); vit != ven; ++vit) {
  s.find(*vit);
}
double avgcomps = countint::getcount()/(double)N;
```

• Do we miss something?
  – BST<countint> s = BST<countint>();
    A container to hold instances of the countint class

What's this "s"?
```cpp
#include "countint.hpp"

/**
 * Implementation of the countint class
 *
 * See: countint.hpp
 *
 * @author Paul Kube (c) 2010
 *
 */

unsigned long countint::count = 0; // define static count variable

void countint::clearcount() {
    count = 0;
}

unsigned long countint::getcount() {
    return count;
}
```
# Benchmarking average number of comparisons for successful find

- **Data structure:** rst
- **Data:** sorted
- **N** is powers of 2, minus 1, from 1 to 32768
- **Averaging** over 5 runs for each N

<table>
<thead>
<tr>
<th>N</th>
<th>avgcomps</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3.33333</td>
<td>0.365148</td>
</tr>
<tr>
<td>7</td>
<td>4.77143</td>
<td>0.714857</td>
</tr>
<tr>
<td>15</td>
<td>6.08</td>
<td>0.243676</td>
</tr>
<tr>
<td>31</td>
<td>8.2129</td>
<td>0.44726</td>
</tr>
<tr>
<td>63</td>
<td>10.873</td>
<td>1.8052</td>
</tr>
<tr>
<td>127</td>
<td>12.8882</td>
<td>0.790504</td>
</tr>
<tr>
<td>255</td>
<td>14.6525</td>
<td>0.901901</td>
</tr>
<tr>
<td>511</td>
<td>16.5119</td>
<td>0.623989</td>
</tr>
<tr>
<td>1023</td>
<td>19.5752</td>
<td>1.28953</td>
</tr>
<tr>
<td>2047</td>
<td>21.4011</td>
<td>0.735492</td>
</tr>
<tr>
<td>4095</td>
<td>22.0275</td>
<td>0.75499</td>
</tr>
<tr>
<td>8191</td>
<td>24.7844</td>
<td>1.07525</td>
</tr>
<tr>
<td>16383</td>
<td>26.9748</td>
<td>1.29925</td>
</tr>
<tr>
<td>32767</td>
<td>28.8381</td>
<td>0.546904</td>
</tr>
</tbody>
</table>
Analysis

• Would you expect **BST** to be faster (use less comparisons) for *sorted or shuffled* data?

• Would you expect a large difference in **performance** for RST/BST/Red-Black trees on *shuffled* data?

• Would you expect a greater **standard deviation** for *sorted or shuffled* data?
QUESTION?