Terence Henriod

Test 3 Challenges

CS 302: Data Structures

December 5, 2013

|  |  |
| --- | --- |
| The Question: | Write the Insert function for this maxheap of integers. |
|  |  |
| My Assertion: | I was marked incorrect for not checking to see if the parent index variable was >=0 in the while loop condition. I believe that the instructor claimed that without this check the loop could become an infinite one. If the markdown was due to a bounds checking error, I assert that the bounds checking is unnecessary due to the impossibility of certain variables receiving a value of less than 0.  void Heap::insert( const int newItem ) throw( logic\_error )  {  int child = size; // size will always conceivable be >= 0  int parent = ( ( size – 1 ) / 2 ); // truncates to 0 when size == 0  int temp;  if( size >= maxSize )  {  throw logic\_error( “Can’t insert into a full heap.” );  }  else  {  table[ size ] = newItem;  size++;  while( data[ parent ] < data[ child ] )  {  Temp = data[ parent ];  Data[ parent ] = data[ child ];  Data[ child ] = temp;  child = parent;  int parent = ( ( child – 1 ) / 2 ); //truncates to 0 when child==0  }  }  } |
|  |  |
| Reasoning: | I assert that that is not true because no matter what, if the loop continues, at some point the parent and child variables will both become 0, and then the loop’s comparison statement will compare the top item of the heap to itself. Once this happens, the comparison statement will evaluate to false (an item cannot be < itself) and the loop will break.  Also, if a segmentation-fault/bounds checking operation was the issue, this code should never cause a segmentation-fault because the size, parent, and child variables should conceivably always be >= 0. |

|  |  |
| --- | --- |
| The Question: | Write the Remove function for a Binary Search Tree (BST) of integers. |
|  |  |
| My Assertion: | The result variable does not need to be updated in the part of the code pertaining to each and every removal case; the variable only need be updated in the case that it is known the removal will occur. In the case that an in-order successor is being removed, there is no reason to store this result. |
|  |  |
| Reasoning: | See the comments at the pertinent areas of the code.  bool BSTree::remove( const int key ) throw( logic\_error )  {  bool result = false;  if( root != NULL )  {  Result = remove\_sub( root, key );  }  else  {  throw logic\_error( “Can’t remove from empty tree.” );  }  return result;  }  bool BSTree::remove\_sub( Node\*& current, const int key )  {  Node\* temp = NULL;  bool result = false;  if( current != NULL )  {  if( current->data < key )  {  result = remove\_sub( current->left, key ):  }  else if( current->data > key )  {  result = remove\_sub( current->right, key ):  }  else  {  // current node’s data is the same as the key if we end up here  result = true; // a removal will be made in one of the following . // ways  if( ( current->left == NULL ) && ( current->right == NULL ))  {  delete current;  current = NULL;  }  else if( ( current->left == NULL ) || ( current->right == NULL ))  {  temp == current;  if( current->left == NULL )  {  Current = current->right;  }  else  {  current = current->left;  }  delete temp;  temp = NULL;  }  else  {  temp = current->left;  while( temp->right != NULL )  {  temp = temp->right;  }  current->data = temp->data;  remove\_sub( current->left, key ); // no need to store result  // we know removal will occur  }  }  }  return result;  }  Also, a snippet of psuedocode from the book. My code uses similar logic to produce the same result, in the case that we are removing, set a flag to say the node was found and will be removed without regard to the removal case:  “  else if (subTreePtr->getItem() == target)  {  // Item is in the root of some subtree  subTreePtr = removeNode(subTreePtr) // remove the item  success = true // does not depend on which version of removal occurs  return subTreePtr  }  “  Data Abstraction and Problem Solving with C++: Walls and Mirrors, by Carrano and Henry, page 478 |

|  |  |
| --- | --- |
| The Question: |  |
|  |  |
| My Assertion: |  |
|  |  |
| Reasoning: |  |