Example: Binary Search Tree (BST) as Template

Posted by Alejandro G. Carlstein Ramos Mejia on October 15, 2010 November 12, 2010 About Programming / Algorithms / C++

Example of Binary Search Tree (BST) as template by using vectors, iterators, and time STL.

NOTIFICATION: These examples are provided for educational purposes. Using this code is under your own responsibility and risk. The code is given 'as is'. I do not take responsibilities of how they are used.

Please notice that as an example, the first tree will have all the nodes to the left, the the second tree will have all the nodes to the right, and the third tree will be balance.

Since the second tree have to check first to the left and then to the right it will take more time than the first tree. Also since the third tree is balance it will find faster the values than the second tree

BSTree.cc:

```
#ifndef BST_CC
#define BST_CC

#include <iostream>

using namespace std;

// Forward declaration of class BST
// By fowarding the declaration, the class BST
// can access to the private data members of BSTNode
// helping to maintain the concept of encapsulation
template <typename T> class BST;

/**
 * BSTNode
 * @description:
 */
template <typename T>
class BSTNode {

 // This is needed to grant class BST access
 // to the private members of BSTNode
 // and help to maintaining the concept of encapsulation
 friend class BST<T>;

 private:
  BSTNode<T>* pLeft;
  T data;
```

```
 BSTNode<T>* pRight;

 public:

 //Default Constructor
 BSTNode(void);

 //Constructor
 BSTNode(const T& newData);

 // **** Get Methods ****

 T getData() const;

 // **** Set Methods ****

 void setData(const T& newData);

 //Destructor
 ~BSTNode(void);

};

////////////////////////////////////////////////////////////////

/**
 * BST
 * @description:
 */
template <typename T>
class BST {

 private:

 int numberOfNodes;

 BSTNode<T> *pRoot;

 T insertNode(BSTNode<T> **pBSTNode,
        const T& newData);

 T findNode(BSTNode<T> **pBSTNode,
        const T& searchData);

 void displayPreOrderTraversal(BSTNode<T> *pBSTNode) const;

 void displayInOrderTraversal(BSTNode<T> *pBSTNode) const;

 void displayPostOrderTraversal(BSTNode<T> *pBSTNode) const;

 void removeAll(BSTNode<T> *pBSTNode) const;

 public:
```

```
 //Default Constructor
 BST(void);

 //Constructor
 BST(const T& newData);

 // **** Get Methods ****

 // **** Set Methods ****

 // **** Methods ****
 T insert(const T& newData);

 T find(const T& searchData);

 int size(void);

 void displayPreOrder(void) const;

 void displayInOrder(void) const;

 void displayPostOrder(void) const;

 //Destructor
 ~BST(void);

};

//*****************************************************************************
//**** BSTNODE METHODS ****
//*****************************************************************************

/////////////////////////////////////////////////////////////////////////////
//**** BSTNODE PRIVATE METHODS ****
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
//**** BSTNODE PUBLIC METHODS ****
/////////////////////////////////////////////////////////////////////////////

template <typename T>
BSTNode<T>::BSTNode(void){
 //cout << endl << '[Default BSTNode]' << endl;
}

template <typename T>
BSTNode<T>::BSTNode(const T& newData):
       pLeft(NULL),
       data(newData),
       pRight(NULL){
// cout << endl << '[BSTNode]' << endl;
}
```

```
// **** Get Methods ****

/**
 *
 */
template <typename T>
T BSTNode<T>::getData() const{
 return data;
}

// **** Set Methods ****

template <typename T>
void BSTNode<T>::setData(const T& newData){
 data = newData;
}

template <typename T>
BSTNode<T>::~BSTNode(void){
// cout << endl << '[~BSTNode(X)]' << endl;
}

//*****************************************************************************
//**** BST METHODS ****
//*****************************************************************************

/////////////////////////////////////////////////////////////////////////////
//**** BST PRIVATE METHODS ****
/////////////////////////////////////////////////////////////////////////////

/**
 *
 */
template <typename T>
T BST<T>::insertNode(BSTNode<T> **pBSTNode,
       const T& newData){

 // If subBST is empty then
 // create new BSTNode for the value
 if (*pBSTNode == NULL){

  *pBSTNode = new BSTNode<T>(newData);

  numberOfNodes++;

  return newData;

 }else{

  // If the data is not duplicated
  if (newData != (*pBSTNode)->data){
```

```
   // if newData is less than the current node's data
   // then insert a left BSTNode with newData
   // else insert a right BSTNode with newData
   if (newData < (*pBSTNode)->data ){

    insertNode( &( ( *pBSTNode) -> pLeft), newData);

   }else{

    insertNode( &( ( *pBSTNode) -> pRight), newData);

   }
  }else{

   T rtnT;

   return rtnT;
  }

 }
}

/**
 *
 */
template <typename T>
T BST<T>::findNode(BSTNode<T> **pBSTNode,
            const T& searchData){

 T rtnT;

 // If subBST is empty
 if (*pBSTNode == NULL){

  // Object not found

  return rtnT;

 }else{

  // If the data is not found
  // Keep searching
  if (searchData != (*pBSTNode)->data){

   // if newData is less than the current node's data
   // then insert a left BSTNode with newData
   // else insert a right BSTNode with newData
   if (searchData < (*pBSTNode)->data ){

    rtnT = findNode( &( ( *pBSTNode) -> pLeft), searchData);

Just to let you know   }else{
```

```
    rtnT = findNode( &( ( *pBSTNode) -> pRight), searchData);

   }

  }else{

   rtnT = (*pBSTNode)->data;
  }

 }

 return rtnT;
}

/**
 *
 */
template <typename T>
void BST<T>:😀isplayPreOrderTraversal(BSTNode<T> *pBSTNode) const{

 if (pBSTNode != NULL){

  // Process Node
  cout << pBSTNode->data << ' ';

  // Traverse left subBST
  displayPreOrderTraversal(pBSTNode->pLeft);

  // Traverse right subBST
  displayPreOrderTraversal(pBSTNode->pRight);

 }
}

/**
 *
 */
template <typename T>
void BST<T>:😀isplayInOrderTraversal(BSTNode<T> *pBSTNode) const{

 if (pBSTNode != NULL){

  // Traverse left subBST
  displayInOrderTraversal( pBSTNode->pLeft);

  // Process Node
  cout << pBSTNode->data << ' ';

  // Traverse right subBST
  displayInOrderTraversal( pBSTNode->pRight);

 }
```

```
}

/**
 *
 */
template <typename T>
void BST<T>: isplayPostOrderTraversal(BSTNode<T> *pBSTNode) const{

 if (pBSTNode != NULL){

  // Traverse left subBST
  displayPostOrderTraversal(pBSTNode->pLeft);

  // Traverse right subBST
  displayPostOrderTraversal(pBSTNode->pRight);

  // Process Node
  cout << pBSTNode->data << ' ';

 }

}

/**
 *
 */
template <typename T>
void BST<T>::removeAll(BSTNode<T> *pBSTNode) const{

 if (pBSTNode != NULL){

  // Traverse left subBST
  removeAll(pBSTNode->pLeft);

  // Traverse right subBST
  removeAll(pBSTNode->pRight);

  delete pBSTNode;

 }

}

////////////////////////////////////////////////////////////////////////////
//**** BST PUBLIC METHODS ****
////////////////////////////////////////////////////////////////////////////

/**
 *
 */
template <typename T>
BST<T>::BST(void){
 //cout << endl << '[Default BST]' << endl;
```

```
 pRoot = NULL;
 numberOfNodes = 0;
}

/**
 *
 */
template <typename T>
BST<T>::BST(const T& newData){
 //cout << endl << '[BST]' << endl;
 numberOfNodes = 0;
}

// **** Get Methods ****

// **** Set Methods ****

// **** Methods ****

/**
 *
 */
template <typename T>
T BST<T>::insert(const T& newData){
 return insertNode( &pRoot, newData);
}

/**
 *
 */
template <typename T>
T BST<T>::find(const T& searchData){

 //cout << endl << '[find : ' << searchData << ']' << endl;

 return findNode( &pRoot, searchData);
}

/**
 *
 */
template <typename T>
int BST<T>::size(void){
 return numberOfNodes;
}

/**
 *
 */
template <typename T>
void BST<T>: isplayPreOrder(void) const{
 displayPreOrderTraversal( pRoot );
}
```

```
/**
 *
 */
template <typename T>
void BST<T>: isplayInOrder(void) const{
 displayInOrderTraversal( pRoot );
}

/**
 *
 */
template <typename T>
void BST<T>: isplayPostOrder(void) const{
 displayPostOrderTraversal( pRoot );
}

/**
 *
 */
template <typename T>
BST<T>::~BST(void){

 cout << endl << '[~BST(X)]' << endl;

// removeAll(pRoot);

}

#endif
```

main.cpp:

```
#include <iostream>
#include <cstdlib>
#include <time.h>
#include <sys/time.h>
#include <vector>
#include <algorithm>
#include 'BSTree.cc'

const int MAX_NUMBERS = 10000;

int main(int argc, char *argv[]){

 cout << endl << '{main}' << endl;

 vector<int> numbers;
 vector<int> reverseNumbers;
 vector<int> randomNumbers;

 BST<int> tree;
 BST<int> reverseTree;
```

```
BST<int> randomTree;

//Declare at top of main...

struct timeval initialSimTime, postSimTime;

double totalSimTime;

int findMod = 10000;
int NTimes = 5000;

cout << 'Create Numbers...' << endl;

// Insert the first 10,000 odd numbers, in order, into it.
for (int i = 1; i < MAX_NUMBERS; i += 2)
  reverseNumbers.push_back(i);

numbers = reverseNumbers;

reverse(numbers.begin(), numbers.end());

randomNumbers = numbers;

random_shuffle(randomNumbers.begin(), randomNumbers.end());

while(!numbers.empty()){
 tree.insert(numbers.back());
 numbers.pop_back();
}

while(!reverseNumbers.empty()){
 reverseTree.insert(reverseNumbers.back());
 reverseNumbers.pop_back();
}

while(!randomNumbers.empty()){
 randomTree.insert(randomNumbers.back());
 randomNumbers.pop_back();
}

cout << 'Search...' << endl;

////////////////////////////////////////////////////////////////////

// Place before the thing you want to time

gettimeofday(&initialSimTime, NULL);

cout << endl << 'Values found: ';

for (int i = 1, j = 0; i < MAX_NUMBERS && j < NTimes; i += 2)
 if (i % findMod)
  if (tree.find(i) == i){
```

```
   cout << i << ' ';
   j++;
  }

 cout << endl;

 //Place after the thing you want to time

 gettimeofday(&postSimTime, NULL);

 totalSimTime = (double)(
       (double)(postSimTime.tv_sec - initialSimTime.tv_sec) * 1000000 +
       (double)(postSimTime.tv_usec - initialSimTime.tv_usec) ) /
       (double)1000000;

 //Report how long it takes your program to run each call to that function
 //Output the time it took to do the stuff you wanted to time
 cout << 'Simulation took '<< totalSimTime << ' seconds' << endl << endl;

////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////

 // Place before the thing you want to time

 gettimeofday(&initialSimTime, NULL);

 cout << endl << 'Values found: ';

 for (int i = 1, j = 0; i < MAX_NUMBERS && j < NTimes; i += 2)
  if (i % findMod)
   if (reverseTree.find(i) == i){
    cout << i << ' ';
    j++;
   }

 cout << endl;

 //Place after the thing you want to time

 gettimeofday(&postSimTime, NULL);

 totalSimTime = (double)(
       (double)(postSimTime.tv_sec - initialSimTime.tv_sec) * 1000000 +
       (double)(postSimTime.tv_usec - initialSimTime.tv_usec) ) /
       (double)1000000;

 //Report how long it takes your program to run each call to that function
 //Output the time it took to do the stuff you wanted to time
 cout << '(Reverse) Simulation took '
      << totalSimTime << ' seconds' << endl << endl;

////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////

 // Place before the thing you want to time

 gettimeofday(&initialSimTime, NULL);

 cout << endl << 'Values found: ';

 for (int i = 1, j = 0; i < MAX_NUMBERS && j < NTimes; i += 2)
  if (i % findMod)
   if (tree.find(i) == i){
    cout << i << ' ';
    j++;
   }

 cout << endl;

 //Place after the thing you want to time

 gettimeofday(&postSimTime, NULL);

 totalSimTime = (double)(
       (double)(postSimTime.tv_sec - initialSimTime.tv_sec) * 1000000 +
       (double)(postSimTime.tv_usec - initialSimTime.tv_usec) ) /
       (double)1000000;

 //Report how long it takes your program to run each call to that function
 //Output the time it took to do the stuff you wanted to time
 cout << '(Random) Simulation took '
    << totalSimTime << ' seconds' << endl << endl;

////////////////////////////////////////////////////////////////////////

/*
 // Search for the first 20 multiples of 1,000
 for (int i = 0; i <= NTimes; i++)
  if (i % findMod == 0)
   if (tree.find(i) == i)
    cout << 'Value ' << i << ' Found!' << endl;

 //Report how long it takes your program to run each call to that function
 //Output the time it took to do the stuff you wanted to time
 cout << 'Simulation took '<< totalSimTime << ' seconds' << endl;

 // Start over with a new BST and insert the same odd numbers,
 // but in reverse order.
 // Again, search for the first 20 multiples of 1,000 and
 // report how long each takes

 // Insert the first 10,000 odd numbers, in order, into it.
 for (int i = MAX_NUMBERS; i >= 0; i--)
  if (i % 2 != 0)
```

```
  reverseTree.insert(i);

findMod = 1000;
NTimes = 20;

// Place before the thing you want to time

gettimeofday(&initialSimTime, NULL);

// Search for the first 20 multiples of 1,000
for (int i = 0; i <= NTimes; i++)
 if (i % findMod)
  if (reverseTree.find(i) == i)
   cout << 'Value ' << i << ' Found!' << endl;

//Place after the thing you want to time

gettimeofday(&postSimTime, NULL);

totalSimTime = (double)(
      (double)(postSimTime.tv_sec - initialSimTime.tv_sec) * 1000000 +
      (double)(postSimTime.tv_usec - initialSimTime.tv_usec) ) /
      (double)1000000;

//Report how long it takes your program to run each call to that function
//Output the time it took to do the stuff you wanted to time
cout << 'Simulation (Reverse) took '<< totalSimTime << ' seconds' << endl;

// Insert the first 10,000 odd integers in random order into,
// yet another BST, and run the same test.

// Insert the first 10,000 odd numbers, in order, into it.

int i;
srand(time(NULL));

bool bMatch[MAX_NUMBERS];

for (int i = 0; i < MAX_NUMBERS; i++)
 bMatch[i] = false;

int max = MAX_NUMBERS;
int min = 0;

int counter = 0;
while(randomTree.size() < MAX_NUMBERS){

 i = rand() % (max - 1) + min;

 if ((max - 1) == i)
  max--;

 if ((min + 1) == i)
```

```
   min++;

  if (i >= max)
   i -= min;

  for (int j = 0; j < MAX_NUMBERS; j++)
   cout << ((bMatch[i])? 1: 0);

  if ( i % 2 != 0 && !bMatch[i]){
   bMatch[i] = true;
   cout << ' ' << i;
   randomTree.insert(i);
  }
  cout << endl;

 }

 cout << endl << 'SIZE: ' << randomTree.size() << endl;

 findMod = 1000;
 NTimes = 20;

 // Place before the thing you want to time

 gettimeofday(&initialSimTime, NULL);

 // Search for the first 20 multiples of 1,000
 for (int i = 0; i <= NTimes; i++)
  if (i % findMod)
   if (reverseTree.find(i) == i)
    cout << 'Value ' << i << ' Found!' << endl;

 //Place after the thing you want to time

 gettimeofday(&postSimTime, NULL);

 totalSimTime = (double)(
       (double)(postSimTime.tv_sec - initialSimTime.tv_sec) * 1000000 +
       (double)(postSimTime.tv_usec - initialSimTime.tv_usec) ) /
       (double)1000000;

 //Report how long it takes your program to run each call to that function
 //Output the time it took to do the stuff you wanted to time
 cout << 'Simulation (Random) took '<< totalSimTime << ' seconds' << endl;
*/

 return 0;
}
```

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks.