

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2021-2022** |
| **Major**: | Software Engineering |
| **Class**: | 2020 |
| **Student Name**: | 温长锟 |
| **Student ID:** | 222020321062106 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | The Height of Binary Search Tree | | | |
| Date | | Dec，2021 | Type | | □Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Understand the concept and property of binary search tree    2. Get familiar with the insert, delete and find operations on binary search tree    3. Grasp the design of recursive or iterative algorithms about binary search tree    4. Understand the concept of height of a binary search tree; can use iterative algorithms to compute the height of a binary search tree    5. Understand the relationship between the size and height of a binary search tree; can deduce the relationship both mathematically and by experiment. | | | | | |
| 1. **Experimental environment (**platform and software**)**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results)   Task 1   1. Complete the implementation of the BST container based on the code sent to you. In particular, the method you are required to implement includes:  * BinSearchTree(); // default constructor * int size() const; // get the number of stored elements * int height() const; // compute the height * Iterator insert(const T& item); //insertion * ~BinSearchTree(); //destructor   as well as the constructor of the associated iterator inner class. It is suggested that some of the methods be implemented in a recursive way.   1. Generate a series of integers randomly and insert them into an empty binary search tree, and compute the height of the tree. Repeat this for a number of times and compute the average height of a BST of size n. Try to discover the mathematical relationship between the size and the height of a BST based on your analysis.   For the test documents not given  First, create a write file stream to store the number of nodes and their corresponding tree height for later analysis. A binary tree with 1 node to 800 nodes is simulated here. 200 cases are randomly generated in each case to calculate the average tree height. In order to increase the randomness, the maximum value of random number is increased to 10000.  #ifndef BST\_H  #define BST\_H  #include <iostream>  #include <string>  #include <string.h>  #include <iomanip>  using namespace std;  #define NULL 0  #define LEFT 1  #define RIGHT 2  #define ROOT 0  template<typename T>  class BinSearchTree  {  class Iterator; //very important!!!  private:  struct Node  {  T item;  Node\* parent;  Node\* left;  Node\* right;  }; // Node  Node\* root;  Node\* largestNode;  Node\* smallestNode;  int count;  Iterator insertRec(Node\* curr, Node\* parent, const T& item, int side);  int getHeightRec(Node\* root)const;  void destroy(Node\* curr);  public:  BinSearchTree();  int size() const;  int height() const;  Iterator insert(const T& item);  Iterator find(const T& item) const;  void erase(Iterator itr);  ~BinSearchTree();  void printTreeRec(Node\* tempRoot);  void printTree();  class Iterator  {  friend class BinSearchTree<T>;  private:  Node\* curr;  Iterator(Node\* currNode);  public:  Iterator();  Iterator& operator++ ();  Iterator& operator-- ();  T& operator\* () const;  bool operator== (const Iterator& otherIterator) const;  };  Iterator begin();  Iterator end();  };  template<typename T>  BinSearchTree<T>::BinSearchTree()  {  root = NULL;  largestNode = NULL;  smallestNode = NULL;  count = 0;  }  template<typename T>  int BinSearchTree<T>::size() const  {  return count;  }  template<typename T>  int BinSearchTree<T>::height() const  {  return getHeightRec(root);  }  template<typename T>  typename BinSearchTree<T>::Iterator BinSearchTree<T>::insert(const T& item)  {  return insertRec(root, NULL, item, ROOT);  }  template<typename T>  void BinSearchTree<T>::destroy(Node\* curr)  {  if (curr != NULL)  {  destroy(curr->left);  destroy(curr->right);  delete curr;  }  }  template<typename T>  BinSearchTree<T>::~BinSearchTree()  {  destroy(root);  }  template<typename T>  BinSearchTree<T>::Iterator::Iterator(Node\* currNode)  {  curr = currNode;  }  #endif  #include <iostream>  #include <fstream>  #include "bst.h"  #include <time.h>  #include <string>  #include <iomanip>  #include <math.h>  using namespace std;  int main()  {  ofstream out;  out.open("C:\\Users\\温长锟\\Desktop\\data.txt");  double Num = 200;  int nodeNum = 800;    int MAX = 10000;  srand((unsigned)time(0));  for (int k = 1; k <= nodeNum; k++) {  double averageHeight = 0;    for (int j = 1; j <= Num; j++) {  BinSearchTree<int> mybst;  int item;  for (int i = 0; i < k; i++) {  item = rand() % MAX;    mybst.insert(item);  }  averageHeight += mybst.height();  }  cout << k << "\t" << averageHeight / Num << endl;  out << k<<"\t"<<averageHeight / Num << endl;  }  out.close();  cout << "FINISH!!" << endl; | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）       Y=3.6ln(x)-4  R²=0.9956  Using the Excel data analysis function, the data is curve fitted, and the goodness of fit is calculated. It is concluded that R²= 0.9956, very close to 1, and the goodness of fit is very good, indicating that the tree height of binary tree has a logarithmic relationship with the number of nodes. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |