**NỘI DUNG THỰC HÀNH**

**Cơ bản**

|  |
| --- |
| //NODE.h  #pragma once  class Node  {  public:  Node();  Node(int);  virtual ~Node();  Node\* Getleft() { return left; }  void Setleft(Node\* val) { left = val; }  Node\* Getright() { return right; }  void Setright(Node\* val) { right = val; }  Node\* Getparent() { return parent; }  void Setparent(Node\* val) { parent = val; }  int Getkey() { return key; }  void Setkey(int val) { key = val; }  protected:  private:  Node\* left;  Node\* right;  Node\* parent;  int key;  };  //NODE.cpp  #include "NODE.h"  Node::Node()  {  //ctor  this->key = 0;  this->left = nullptr;  this->right = nullptr;  this->parent = nullptr;  }  Node::Node(int k) {  //ctor  this->key = k;  this->left = nullptr;  this->right = nullptr;  this->parent = nullptr;  }  Node::~Node()  {  //dtor  }  //BST.h  #pragma once  #include "NODE.h"  class BST  {  public:  BST();  virtual ~BST();  Node\* Getroot() { return root; }  void Setroot(Node\* val) { root = val; }  bool InsertNode(Node\*);  bool InsertNodeRe(Node\*, Node\*);  void deleteNode(Node\*);  void TravelNLR();  void TravelLNR();  void TravelLRN();  void NLR(Node\*);  void LNR(Node\*);  void LRN(Node\*);  Node\* search\_x(int);  protected:  private:  Node\* root;  };  //BST.cpp  #include "BST.h"  #include "NODE.h"  #include <iostream>  using namespace std;  BST::BST()  {  //ctor  this->root = nullptr;  }  BST::~BST()  {  //dtor  }  bool BST::InsertNode(Node\* n) {  Node\* p = this->root;  Node\* T;  if (root == nullptr)  {  this->root = n;  return true;  }  while (p != nullptr) {  T = p;  if (p->Getkey() > n->Getkey())  p = p->Getleft();  else  if (p->Getkey() < n->Getkey())  p = p->Getright();  else  if (p->Getkey() == n->Getkey())  return false;  }  if (T->Getkey() > n->Getkey())  T->Setleft(n);  else T->Setright(n);  n->Setparent(T);  return true;  }  bool BST::InsertNodeRe(Node\* root, Node\* p) {  if (root == nullptr) {  root = p;  return true;  }  if (root->Getkey() == p->Getkey())  return false;  else if (root->Getkey() > p->Getkey())  return InsertNodeRe(root->Getleft(), p);  else return InsertNodeRe(root->Getright(), p);  }  void BST::NLR(Node\* r) {  if (r != nullptr) {  cout << r->Getkey() << "\n";  NLR(r->Getleft());  NLR(r->Getright());  }  }  void BST::LNR(Node\* r) {  if (r != nullptr) {  LNR(r->Getleft());  cout << r->Getkey() << "\n";  LNR(r->Getright());  }  }  void BST::LRN(Node\* r) {  if (r != nullptr) {  LRN(r->Getleft());  LRN(r->Getright());  cout << r->Getkey() << "\n";  }  }  void BST::TravelNLR() {  NLR(this->root);  }  void BST::TravelLNR() {  LNR(this->root);  }  void BST::TravelLRN() {  LRN(this->root);  }  Node\* BST::search\_x(int k) {  Node\* r = this->root;  queue <Node\*> q;  q.push(r);  while (!q.empty()) {  Node\* f = q.front();  q.pop();  if (f->Getleft() != nullptr) q.push(f->Getleft());  if (f->Getright() != nullptr) q.push(f->Getright());  if (f->Getkey() == k) {  return f;  }  }  return nullptr;  }  void BST::deleteNode(Node\* n) {  Node\* p = n;  if (p->Getleft() == nullptr && n->Getright() == nullptr) {  if (p->Getparent()->Getleft() == n) {  p->Getparent()->Setleft(nullptr);  }  else {  p->Getparent()->Setright(nullptr);  }  delete n;  }  else {  if (p->Getright() != nullptr) {  p = p->Getright();  while (p->Getleft() != nullptr)//  p = p->Getleft();  n->Setkey(p->Getkey());  if (p->Getright() != nullptr) p->Getparent()->Setleft(p->Getright());  delete p;  }  else {  p = p->Getleft();  while (p->Getright() != nullptr)//  p = p->Getright();  n->Setkey(p->Getkey());  if(p->Getleft() != nullptr) p->Getparent()->Setright(p->Getleft());  delete p;  }  }  }  //main.cpp  #include <iostream>  #include "BST.h"  using namespace std;  int main()  {  BST\* tree = new BST();  Node\* n;  n = new Node(10);  tree->InsertNode(n);  n = new Node(19);  tree->InsertNode(n);  n = new Node(9);  tree->InsertNode(n);  n = new Node(3);  tree->InsertNode(n);  n = new Node(19);  tree->InsertNode(n);  n = new Node(8);  tree->InsertNode(n);  n = new Node(4);  tree->InsertNode(n);  n = new Node(1);  tree->InsertNode(n);  n = new Node(15);  tree->InsertNode(n);  tree->TravelNLR();  return 0;  } |

**Áp dụng – Nâng cao**

1. Bổ sung chương trình mẫu cho phép tính **t**ổ**ng giá tr**ị các nút trên cây nhị phân gồm các giá trị nguyên.

Gợi ý: tham khảo hàm **NLR** để viết hàm **SumTree**.

|  |
| --- |
| int BST::SumOfBST()  {  int sum = 0;  Node\* r = this->root;  if (r != nullptr) {  NLR(r->Getleft());  NLR(r->Getright());  sum += r->Getkey();  }  return sum;  } |

1. Bổ sung chương trình mẫu cho phép tìm **giá tr**ị **nguyên l**ớ**n nh**ấ**t và nh**ỏ **nh**ấ**t** trong số các phần tử nguyên trên cây nhị phân tìm kiếm gồm các giá trị nguyên. Gợi ý: dựa vào tính chất 1, 2 của cây nhị phân tìm kiếm.

|  |
| --- |
| int BST::LargestInt()  {  Node\* r = this->root;  if (r != nullptr) {  r = r->Getright();  }  return r->Getkey();  }  . . .  int BST::SmallestInt()  {  Node\* r = this->root;  if (r != nullptr) {  r = r->Getleft();  }  return r->Getkey();  } |

1. Bổ sung chương trình mẫu cho phép tính **s**ố **l**ượ**ng các nút** của cây nhị phân gồm các giá trị nguyên.

Gợi ý: tham khảo hàm **NLR** để viết hàm **CountNode**.

|  |
| --- |
| int BST::CountNode()  {  int cnt = 0;  Node\* r = this->root;  if (r != nullptr) {  NLR(r->Getleft());  NLR(r->Getright());  cnt++;  }  return cnt;  } |

1. Bổ sung chương trình mẫu cho biết **s**ố **l**ượ**ng các nút lá** trên cây nhị phân.

Gợi ý: tham khảo thao tác duyệt cây nhị phân **NLR**.

|  |
| --- |
| int BST::CountLeaf()  {  int cnt = 0;  Node\* r = this->root;  if (r != nullptr) {  NLR(r->Getleft());  NLR(r->Getright());  if (r->Getleft() == nullptr && r->Getright() == nullptr) {  cnt++;  }  }  return cnt;  } |

1. Sử dụng cây nhị phân tìm kiếm để giải bài toán:
2. Đếm có bao nhiêu giá trị phân biệt trong dãy số cho trước
3. Với mỗi giá trị phân biệt, cho biết số lượng phần tử

|  |
| --- |
| //BST.cpp  bool BST::InsertNode(Node\* n) {  . . .  if (this->search\_x(n->Getkey()) != nullptr) {  this->search\_x(n->Getkey())->Upfre();  return true;  }  . . .  }  //NODE.cpp  . . .  Node::Node(int k) {  //ctor  this->key = k;  this->fre = 1;  this->left = nullptr;  this->right = nullptr;  this->parent = nullptr;  }  int Getfre() { return fre; }  void Upfre() { fre++; return; }  **. . .**  //main.cpp  #include <iostream>  #include "BST.h"  using namespace std;  int main()  {  int arr[7] = {1, 3, 4, 3, 2, 3, 4};  BST\* tree = new BST();    for (auto i : arr) {  Node\* n = new Node(i);  tree->InsertNode(n);  cout << "---\n";  tree->TravelLNR();  }  return 0;  } |

**BÀI TẬP ỨNG DỤNG**

1. Sử dụng cây nhị phân tìm kiếm để giải bài toán đếm (thống kê) số lượng ký tự có trong văn bản (Không dấu).
   1. Xây dựng cây cho biết mỗi ký tự có trong văn bản xuất hiện mấy lần
   2. Nhập vào 1 ký tự. Kiểm tra ký tự đó xuất hiện bao nhiêu lần trong văn bản

|  |
| --- |
| //main.cpp  #include <iostream>  #include "BST.h"  using namespace std;  int main()  {  string s = "aaavsbsbsfs";  cout << s << "\n";  BST\* tree = new BST();  for (int i = 0; i < s.length(); i++) {  Node\* n = new Node(s[i] - 'a');  tree->InsertNode(n);  }  tree->TravelLNR();  char c;  cout << "Nhap vao ky tu muon kiem tra : "; cin >> c;  Node\* tmp = tree->search\_x(c - 'a');  if (tmp == nullptr) {  cout << "0";  }  else {  cout << tmp->Getfre();  }  return 0;  } |

1. Bài toán tương tự như trên nhưng thống kê số lượng tiếng có trong văn bản (không dấu)

Ví dụ:

Văn bản có nội dung như sau: “hoc sinh di hoc mon sinh hoc” Kết quả cho thấy như sau:

di: 1 hoc: 3 mon: 1 sinh: 2