CÂY NHỊ PHÂN TÌM KIẾM

# MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

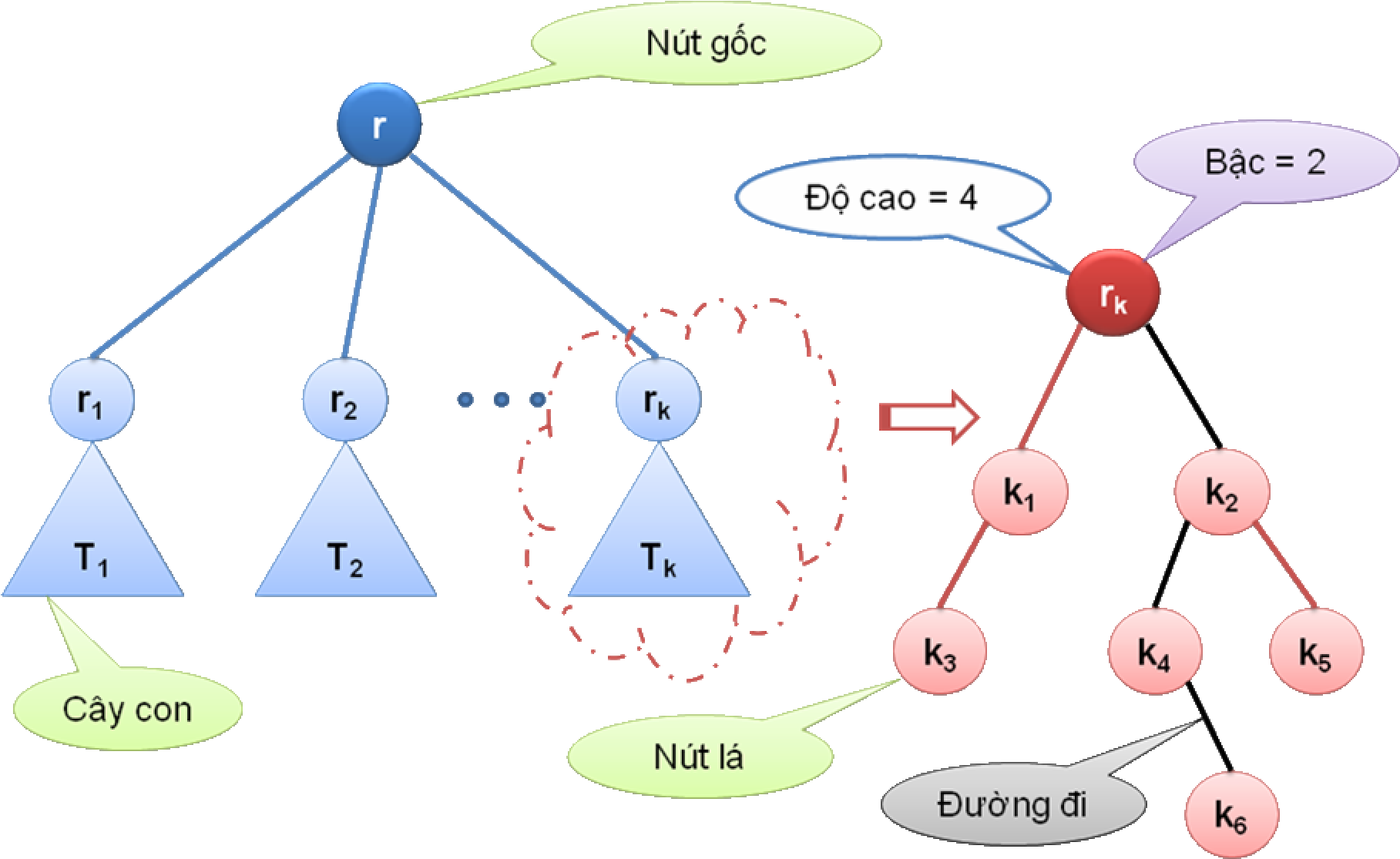
* Hiểu được các thành phần của cây nhị phân tìm kiếm.
* Thành thạo các thao tác trên cây nhị phân tìm kiếm: tạo cây, thêm phần tử, xóa phần tử, duyệt cây nhị phân tìm kiếm.
* Áp dụng cấu trúc dữ liệu cây nhị phân tìm kiếm vào việc giải quyết một số bài toán đơn giản.

Thời gian thực hành: **t**ừ **120 phút** đế**n 400 phút**

# TÓM TẮT

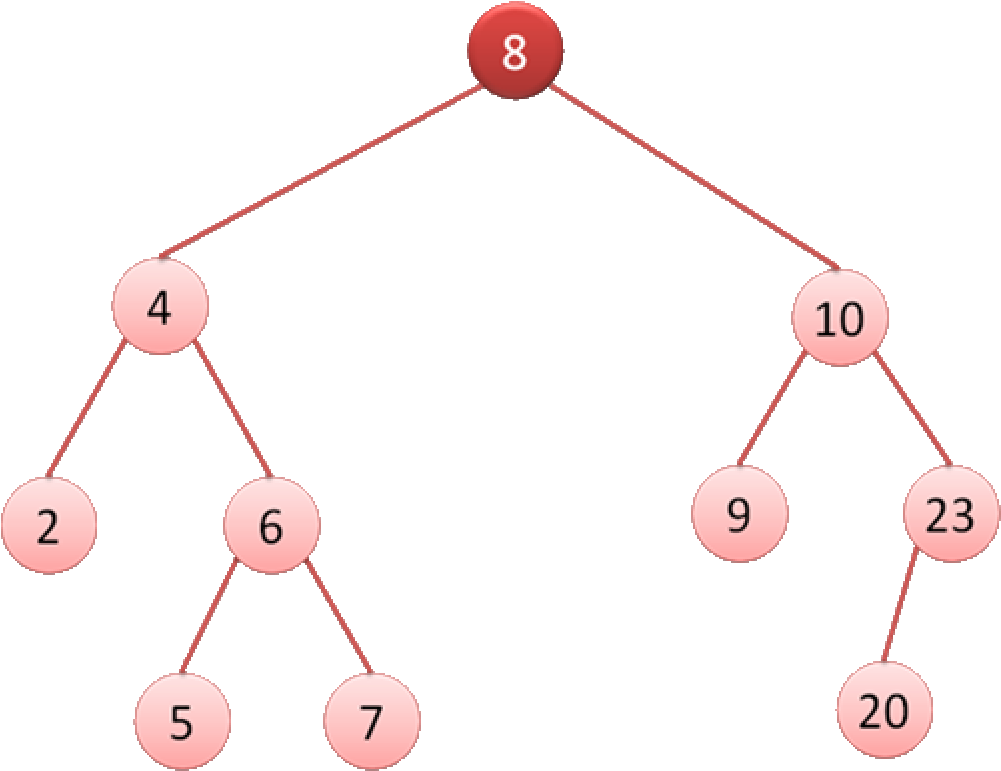
Cây nhị phân tìm kiếm là cây có tối đa 2 nhánh (cây con), nhánh trái và nhánh phải. Cây nhị phân tìm kiếm có các tính chất sau:

* Khóa của tất cả các nút thuộc cây con trái nhỏ hơn khóa nút gốc.
* Khóa của nút gốc nhỏ hơn khóa của tất cả các nút thuộc cây con phải.
* Cây con trái và cây con phải của nút gốc cũng là cây nhị phân tìm kiếm Một số khái niệm:



* Nút lá có độ cao bằng 1

*Ví d*ụ *cây nh*ị *phân tìm ki*ế*m:*



Trong mỗi nút của cây nhị phân tìm kiếm, thông tin liên kết là vô cùng quan trọng. Chỉ cần một xử lý không cẩn thận có thể làm mất phần liên kết này thì cây sẽ bị ‘**gãy**’ cây con liên quan ứng với liên kết đó (không thể truy xuất tiếp tất cả các nút của nhánh con bị mất).

Các thao tác cơ bản trên cây nhị phân tìm kiếm:

* Thêm 1 nút: dựa vào tính chất của cây nhị phân tìm kiếm để tìm vị trí thêm nút mới.

o Tạo cây: từ cây rỗng, lần lượt thêm các nút vào cây bằng phương thức thêm nút vào cây nhị phân tìm kiếm

* Xóa 1 nút: là nút lá, là nút có 1 nhánh con, là nút có 2 nhánh con.
* Duyệt cây nhị phân tìm kiếm: để có thể đi được hết các phần tử trên cây nhị phân tìm kiếm: duyệt trước (NLR), duyệt giữa (LNR), duyệt sau (LRN). Do tính chất của cây nhị phân tìm kiếm, phép duyệt giữa cho phép duyệt các khóa của cây theo thứ tự tăng dần

# NỘI DUNG THỰC HÀNH

## Cơ bản

Sinh viên đọc kỹ phát biểu bài tập và thực hiện theo hướng dẫn:

*T*ổ *ch*ứ*c m*ộ*t cây nh*ị *phân tìm ki*ế*m trong* đ*ó m*ỗ*i ph*ầ*n t*ử *ch*ứ*a thông tin d*ữ *li*ệ*u là s*ố *nguyên.*

### Phân tích

* Cây nhị phân tìm kiếm có mỗi nút chứa dữ liệu nguyên. Thông tin của mỗi nút được khai báo theo ngôn ngữ C/C++ như sau:

#ifndef NODE\_H

#define NODE\_H

class Node

{

public:

Node();

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;

};

#endif // NODE\_H

* Thao tác cần thực hiện:

o Khai báo, khởi tạo cây o (lập) **thêm** nút có khóa nguyên vào **cây nh**ị **phân tìm ki**ế**m** (**Insert**), o **in** các nút của cây nhị phân tìm kiếm (**NLR**), o **tìm** 1 giá trị, nếu có:

tính độ **cao** của nút đó (**Height**) **xóa** nút khỏi cây (**RemoveNode**) **in** các nút của cây sau khi xóa (**NLR**)

### Chương trình mẫu

| #ifndef NODE\_H  #define NODE\_H  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;  };  #endif // NODE\_H  /////////////////////////////////////////////////////////////////////////  #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  }  /////////////////////////////////////////////////////////////////////////////  #ifndef BST\_H  #define BST\_H  #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;  };  #endif // BST\_H  //////////////////////////////////////////////////////////////////////////  #include "BST.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);  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){  //sinh vien code  }  void BST::LRN(Node\*r){  //sinh vien code  }  void BST::TravelNLR(){  NLR(this->root);  }  void BST::TravelLNR(){  //sinh vien code  }  void BST::TravelLRN(){  //sinh vien code  }  Node\* BST::search\_x(int k){  //sinh vien code  }  void BST::deleteNode(Node\* n){  Node\* p=n;  if(p->Getleft()==nullptr&&n->Getright()==nullptr)  delete n;  else{  if(p->Getright()!=nullptr){  while(p->Getleft()!=nullptr)//  p=p->Getleft();  n->Setkey(p->Getkey());  //sinh vien code  delete p;  }else{  while(p->Getright()!=nullptr)//  p=p->Getright();  n->Setkey(p->Getkey());  //sinh vien code  delete p;  }  }  }  ////////////////////////////////////////////////////////////////////////////  #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;  } |
| --- |

### Yêu cầu

1. Biên dịch đoạn chương trình nêu trên.
2. Vẽ hình cây nhị phân tìm kiếm theo dữ liệu được câu 1.
3. Thực hiện hoàn thiện các hàm: có chú thích //sinh vien code

## 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**.

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.
2. 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**.

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**. 5. Sử dụng cây nhị phân tìm kiếm để giải bài toán:

* 1. Đếm có bao nhiêu giá trị phân biệt trong dãy số cho trước
  2. Với mỗi giá trị phân biệt, cho biết số lượng phần tử

# BÀI TẬP THÊM

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
2. 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