**ÔN TẬP TREE**

**Bài 1:** Trình bày định nghĩa Cây nhị phân tìm kiếm (Binary Search Tree, viết tắt: BST). Giả sử cho cây BST gồm N nút, trong trường hợp xấu nhất/ tốt nhất chiều cao của cây BST sẽ bằng bao nhiêu? Giải thích và cho ví dụ minh họa.

**Bài 2:** Ưu điểm của cây BST so với Danh sách liên kết, Mảng 1 chiều.

**Bài 3:** Viết 1 số hàm cơ bản như tìm kiếm, tính tổng, đếm node với điều kiện đơn giản.

**Bài 4:** Viết hàm tính Chiều cao cây. (nhớ ghi thêm định nghĩa chiều cao cây khi tính chọn h>=0 hay h>=1)

**Bài 5:** Thực hiện xóa 1 số node trên cây

**Bài 6:** Cho 1 dãy số theo kiểu duyệt cơ bản nào đó, hãy xác định 1 kiểu duyệt cơ bản khác.

**Bài 7:** **7.1** In cây theo 3 cách duyệt cơ bản.

**7.2** In từng mức và in hết tất cả các mức trong cây (duyệt theo chiều rộng).

**Bài 8: 8.1** Đếm node có số chiều cao trái phải lệnh nhau 1 đơn vị

**8.2** Đếm node có số node con trái phải lệnh nhau 1 đơn vị

**Bài 9:** Dùng cây BST để đếm số lần xuất hiện của các số trong 1 dãy số.

**Bài 10:** Xoá 1 nút có khoá bằng X trên cây.

**Bài 11:** In cây theo từng mức.

**Duyệt theo chiều sâu:**

* **NLR (preorder)**
* **LNR (inorder)**
* **LRN (postorder)**

|  |  |
| --- | --- |
| C1:  int SumPrime(NODE\* T){  if(T== NULL) return **0**;  int a=Sum(T->pLeft);  int b=Sum(T->pRight);  if(checkPrime(T->key))  return a+b+1;  else return a+b;  } | C2:  void SumPrime(NODE\* T, int &s){  if(T==NULL) return;  Sum(T->pLeft);  if(checkPrime(T->key))  s=s+T->key;  Sum(T->pRight);  }  int SumPrime(NODE\* T){  int s=0; SumPrime (T, s);  return s;  } |
| void printlevel(NODE \*T, int i){  if(T==NULL) return;  if (i==0) {cout << T->key; return;}  printlevel (T->pLeft, i-1);  printlevel (T->pRight, i-1);  }  void Print (NODE \*T) { // h>=0  for(i=0->h)  printlevel(T, i);  } | 8 level=0  5 30 level=1  4 7 25 35 level=2 |