BÀI THỰC HÀNH SỐ 8

1. MỤC TIÊU:

- Giúp SV làm quen với các giải thuật trên cây:
  - Tạo cây
  - Duyệt cây
  - Thêm node mới
  - Xóa node

2. LÝ THUYẾT CẦN GHI NHỚ

Cây nhị phân tìm kiếm là cây nhị phân mà giá trị (khóa) của phần tử bên trái của một node có giá trị nhỏ hơn giá trị (khóa) của node, giá trị (khóa) của các phần tử bên phải của một node thì lớn hơn giá trị (khóa) của node đó.Trong một cây nhị phân tìm kiếm sẽ không có 2 phần tử có giá trị bằng nhau

Các thao tác trên cây trong bài thực hành 8:

- Xóa node
- Tính tổng giá trị các node
- Đếm các node trong cây/đếm các node đặc biệt trong cây

3. BÀI TẬP THỰC HÀNH CƠ BẢN

SV tạo project "Win32 Console Application" trên VS 2015, với tên là "**Lab8**".

Tạo file source .cpp có tên: **hovaten_mssv_lab8.cpp**

*** Cách tạo project và cấu trúc chương trình C++: Xem lại file hướng dẫn thực hành lab 1

Sử dụng lại bài thực hành 7

Bài 1:

1. Viết thủ tục xóa một nut trong cây (dùng đệ quy).
   Tìm node cần xóa, nếu tìm không thấy ➔ trả về kết quả không xóa
   Nếu tìm thấy:
   - Nếu node cần xóa là node lá ➔ delete node này (chuyển liên kết của node cha đến node này là NULL và xóa node)
   - Nếu node cần xóa là node cha của 1 hoặc 2 node lá (node bậc 1) thì chỉnh liên kết từ node cha của node cần xóa đến node con của node cần xóa, sau đó xóa node cần xóa đi

- Nếu node cần xóa là node bậc 2 và có 2 con: tìm node cực trái của nhánh con bên phải – gọi là node thế mạng (là node có giá trị gần với node cần xóa nhất). Chép giá trị của node thế mạng vào giá trị của node cần xóa. Sau đó xóa node thế mạng

  Code như sau:

```
int Delete(Node*& T, int x)
{
    if (T == NULL) return 0;
    if (T->info == x) {
        Node* p = T;
        if (T->left == NULL) T = T->right;
        else if (T->right == NULL) T = T->left;
        else searStandfor(p, T->right);
        delete(p);
        return 1;
    }
    if (T->info < x) return Delete(T->right, x);
    else return Delete(T->left, x);

}
void searStandfor(Node*& p, Node*& q) {
    if (q->left == NULL) {
        p->info = q->info;
        p = q;
        q = q->right;
    }
    else
        searStandfor(p, q->left);
}
```

2. Viết thủ tục đếm số node lá của cây

- Node lá là node mà có thành phần left -> NULL và right->NULL
- Sử dụng đệ quy để đếm node lá ở cây bên trái và cây bên phải

```
int demnodela(Node* p) {
    if (p != NULL) {
        if (p->left == NULL && p->right == NULL) return 1;
        return demnodela(p->left) + demnodela(p->right);
    }
    return 0;
}
```

3. Viết thủ tục tính tổng các khóa trên cây/ Tính tổng giá trị của các node có giá trị lẻ/node có giá trị chẵn/node là số nguyên tố/node là số chính phương/node là số hoàn thiện/node là số chia hết cho 3/5/4….
   - Tính tổng giá trị của node hiện tại và node bên trái, node bên phải của node đó

```
int tongnodetrencay(Node *p) {
    if (p == NULL) return 0;
    return p->info + tongnodetrencay(p->left)
        + tongnodetrencay(p->right);
}
```

4. Đếm số node có duy nhất một cây con/số node chỉ có cây con trái, số node chỉ có cây con phải

   Số node có duy nhất một cây con là node có cây con trái hoặc cây con phải NULL

   Sử dụng dụng đệ quy để đếm số node có một cây con bên trái và số node có một cây con bên phải

```
int demsonode1caycon(Node* p) {
    if (p == NULL) return 0;
    int count = ((p->left == NULL && p->right != NULL) ||
        (p->left != NULL && p->right == NULL)
        ) ? 1 : 0;
    return count + demsonode1caycon(p->left)
        + demsonode1caycon(p->right);
}
```

5. In ra những node mà chỉ có cây con trái /phải

   Tương tự câu 4, nhưng kiểm tra những node nào chỉ có cây con phải (cây con trái =NULL) thì in giá trị (info) ra màn hình

   Sinh viên tự làm

6. Đếm số node có 2 cây con/in ra những node có 2 cây con

   Node có cây con bên trái khác NULL và cây con phải khác NULL

```
void insonode2caycon(Node* p) {
    if (p == NULL) return;
    if (p->left != NULL && p->right != NULL) cout << p->info << "\t";
    insonode2caycon(p->left);
    insonode2caycon(p->right);
}
```

   SV tự viết code đếm số node có cả cây con trái và cây con phải

7. Viết thủ tục kiểm tra 2 cây nhị phân tìm kiếm bằng nhau
   - Nếu cả 2 cây rỗng ➔ 2 cây giống nhau
   - Nếu cả 2 cây khác rỗng:
     - Kiểm tra info của node gốc của 2 cây (1)
     - Dùng đệ quy để kiểm tra cây con bên trái (2)

- Dùng đệ quy để kiểm tra cây con bên phải (3)

Nếu (a), (b), (c) đều đúng ➔ 2 cây giống nhau

Ngược lại thì kết luận 2 cây khác nhau

8. Viết thủ tục tính chiều cao của cây nhị phân tìm kiếm

Chiều cao của cây nhị phân tìm kiếm là số cạnh lớn nhất đi từ node gốc đến node lá

Nếu node =NULL ➔ chiều cao là 0

Gọi đệ quy tính chiều cao node bên trái (1)

Gọi đệ quy tính chiều cao node bên phải(2)

So sánh (2) và (1) chọn (giá trị lớn hơn +1)

Chiều cao của cây là giá trị nhận được -1 (vì cây có 1 node ➔ chiều cao cây là 0)

```
int chieucaocay(Node* p) {
    if (p == NULL) return 0;
    int h1 = chieucaocay(p->left);
    int h2 = chieucaocay(p->right);
    return (h1 > h2 ? h1 : h2) + 1;
}
```

Chiều cao của cây =chieucaocay(root) – 1;

9. Viết thủ tục tính độ sâu (depth) của một node (giá trị node do người dùng nhập vào) trong cây nhị phân tìm kiếm

(Sinh viên tự viết code)

Độ sâu của một node là số cạnh đi từ node gốc đến node đó

10. Viết thủ tục tính mức của các node lá trong cây nhị phân tìm kiếm

(Sinh viên tự code)

4. BÀI TẬP NÂNG CAO

Bài 1:

a. Viết thủ tục tìm một phần tử trong cây (không dùng đệ quy).

```
Node* search3(Node* pNode,  int x)
{
    Node* p = pNode;
    while (p != NULL) {
        if (p->info == x) return p;// return 1 // return true;
        else
            if (p->info > x)
                p = p->left;
            else p = p->right;
    }
    return NULL; // return 0; //return false;
}
```

b. Viết thủ tục duyệt cây theo thứ tự LNR (dùng stack)

- Khai báo cấu trúc stack:

```
struct stack {
    int sp;
    Node* A[sizeof(int)];
};
```

- Xây dựng các hàm sử dụng trong stack:init, Push, pop, isemtpy (hoặc isfull)

```
void initstack(stack& s) {
    s.sp = -1;
}
int isfull(stack s) {
    if (s.sp == sizeof(int)) return -1;
    return 1;
}
int isempty(stack s) {
    if (s.sp == -1) return 1;
    return -1;
}
int  push(stack& s, Node* p) {
    if (isfull(s) == -1) return 0;
    s.A[++s.sp] = p;
    return 1;
}
int pop(stack& s, Node*& p) {
    if (isempty(s) == 1) return 0;
   p= s.A[s.sp--];
    return 1;
}
```

- Xây dựng hàm duyệt cây theo thứ tự

```
void LNRStack( Node* p) {
    stack s;
    initstack(s);
    if (p != NULL) {
        while (isempty(s)==-1 || p != NULL) {
            if (p != NULL)
            {
                push(s, p);
                p = p->left;
            }
            else {
                pop(s, p);
                cout << p->info << "\t";
                p = p->right;

            }
        }
    }
}
```

c. Viết thủ tục duyệt cây theo thứ tự NLR (dùng stack)
Sinh viên tự viết code

d. Viết thủ tục duyệt cây theo thứ tự LRN (dùng queue)
   Sinh viên tự viết code