Tree (Cont)

1 Cây nhị phân

Mỗi Node của một cây nhị phân (tìm kiếm) được định nghĩa như sau:

```
struct BSTNode{
    int key;
    NODE* p_left;
    NODE* p_right;
};
```

Sinh viên cần thực hiện cài đặt các hàm sau:

1. Tìm và trả về một `NODE` với giá trị cho trước từ một cây nhị phân tìm kiếm: (Không sử dụng đệ quy)
   - `NODE* Search(NODE* pRoot, int x)`
2. Thêm một `NODE` với giá trị cho trước vào cây nhị phân tìm kiếm: (Không sử dụng đệ quy)
   - `void Insert(NODE* &pRoot, int x)`
3. Xoá một `NODE` với giá trị cho trước từ một cây nhị phân tìm kiếm: (Không sử dụng đệ quy)
   - `void Remove(NODE* &pRoot, int x)`
4. Kiểm tra cây BST có tồn tại 2 giá trị a, b sao cho a + b = k hay không
   - `bool check(NODE* &pRoot, int k)`
5. Xác định một cây nhị phân có phải là cây AVL không:
   - `bool isAVL(NODE* pRoot)`

2 Cây Red-Black

Mỗi Node của một cây Red-Black được định nghĩa như sau:

```
struct NODE{
      int data;
      Node *parent;
      Node *left;
      Node *right;
      int color;

};
```

Sinh viên cần cài đặt các hàm sau:

1. Khởi tạo một NODE từ một giá trị cho trước:

- `NODE* createNode(int data)`

2. Thêm một `NODE` có giá trị cho trước vào cây RED-BLACK cho trước (Chú ý giá trị cho trước có tồn tại hay chưa):
   - `void Insert(NODE* &pRoot, int x)`
3. Xóa một `NODE` với giá trị cho trước từ một cây RED-BLACK cho trước (Chú ý giá trị đó có tồn tại hay không):
   - `void Remove(NODE* &pRoot, int x)`