二叉查找树/二叉搜索树

性质:

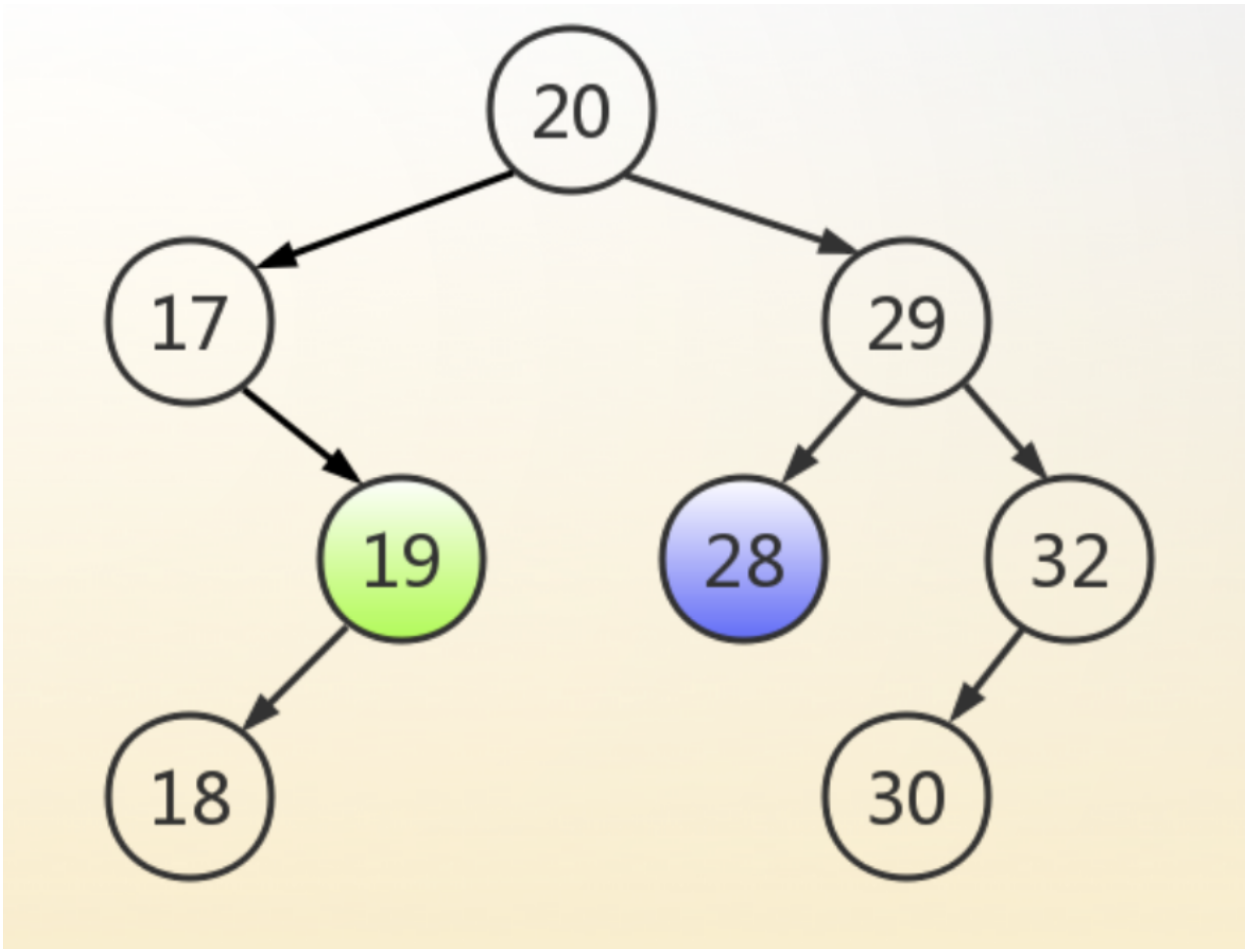
右子树 > 根节点 > 左子树

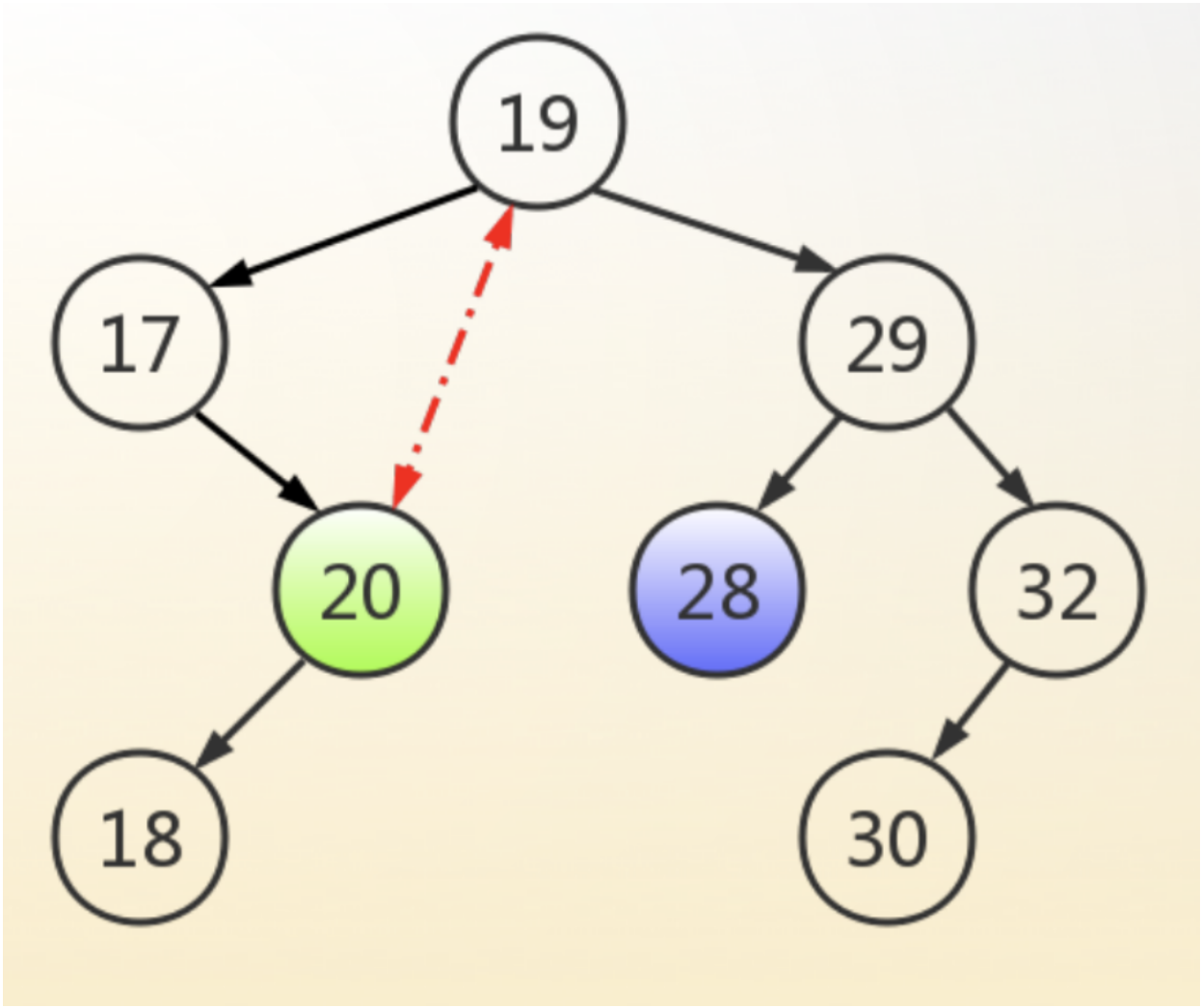
插入:

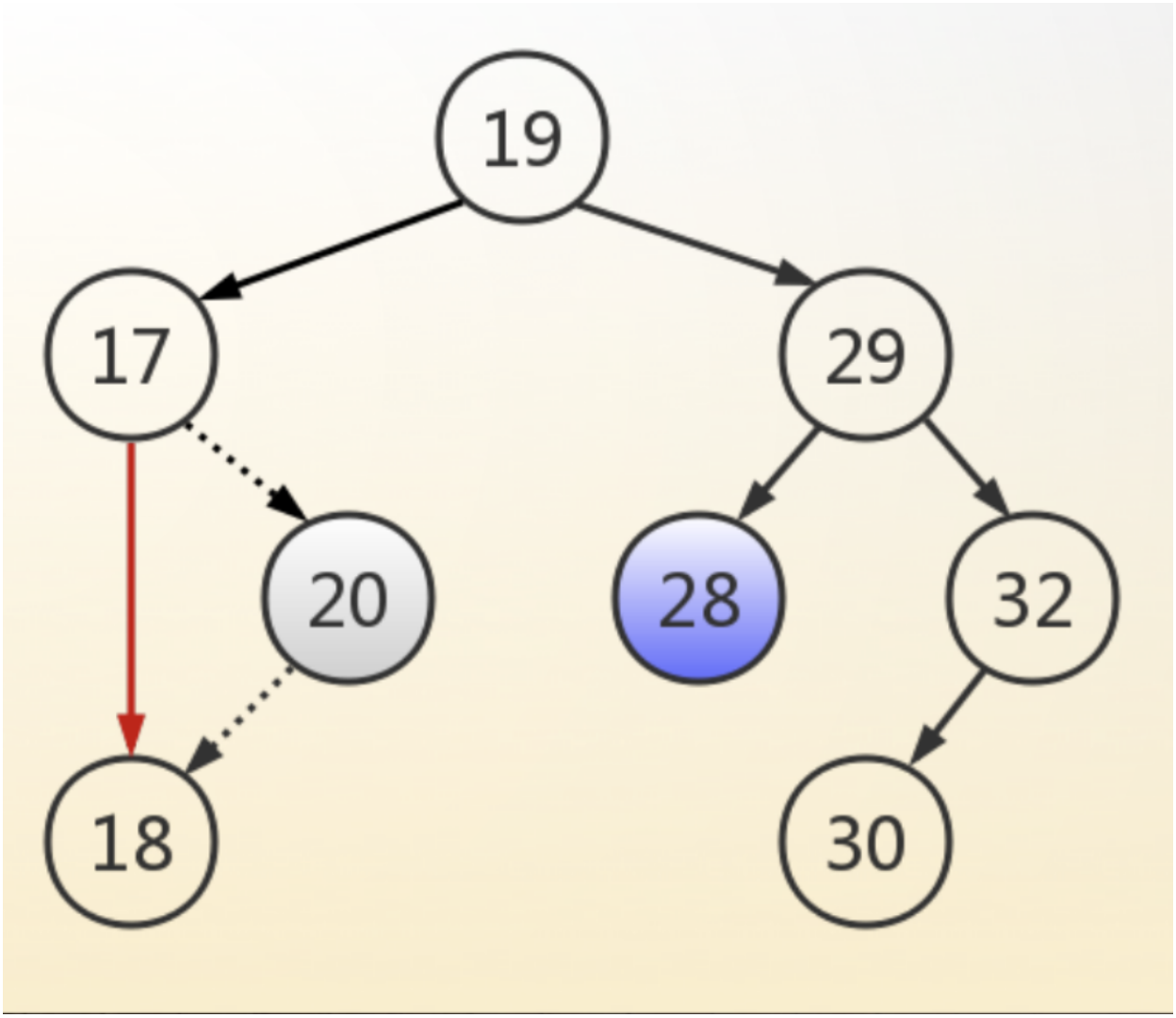
略

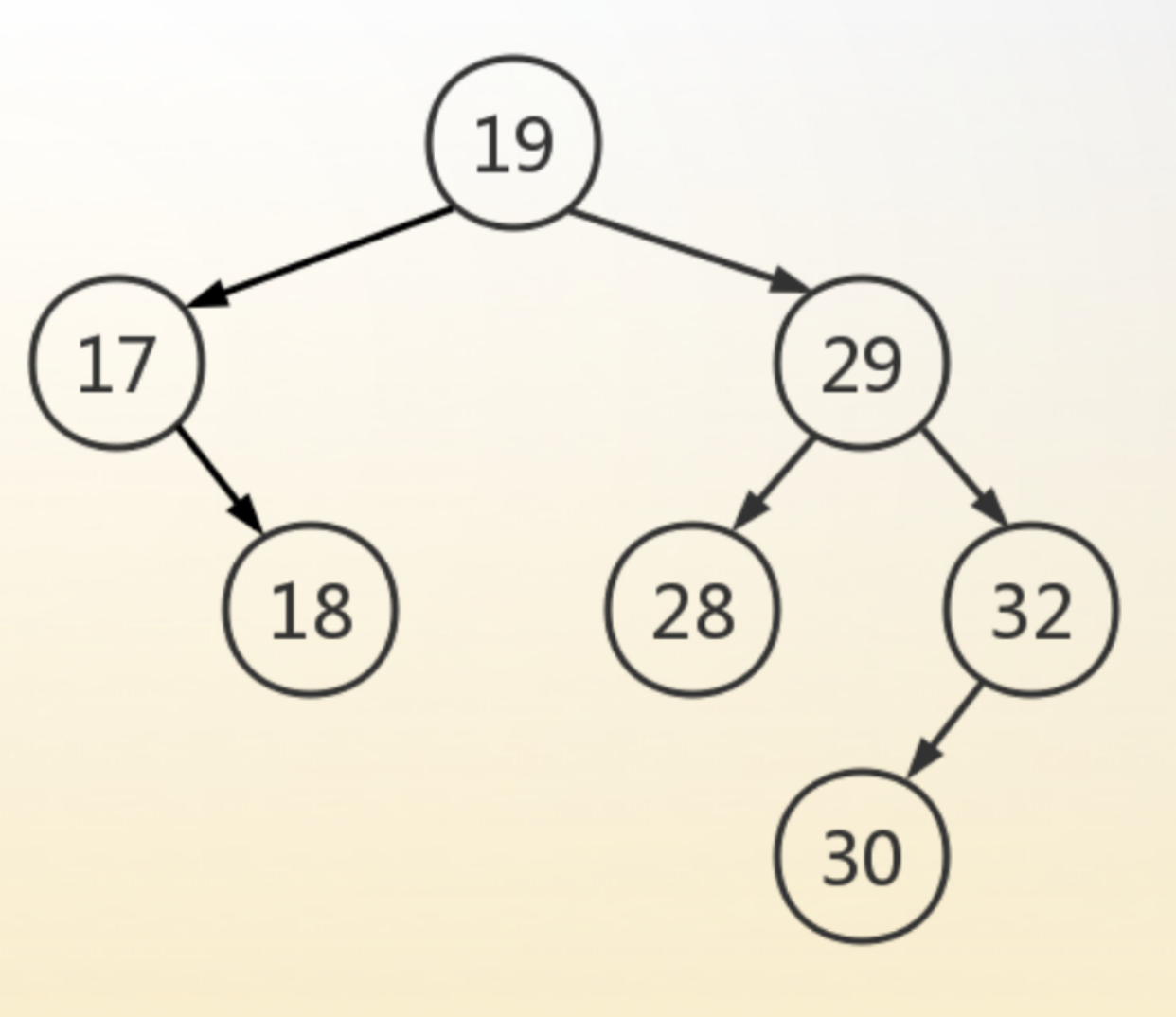
删除:

1 删除出度为0的节点（叶子节点）（左右子树为空）： 直接删除
2 删除出度为1的节点 （左子树或右子树有一个不为空）： 提升唯一子树
3 删除出度为2的节点 （左子树和右子树都不为空）： 找到前驱（左子树中最右面的值）或者后继（右子树中最左面的值）替换后转换为度为1的节点




19
17
29
20
28
32
18
30


19
17
29
20
28
32
18
30



代码

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


typedef struct BSTNode {
    int data;                           // 二叉搜索树节点中的值
    struct BSTNode *lchild, *rchild;   // 二叉搜索树的左右节点
} BSTNode;

BSTNode *init(int data) {
    BSTNode *p = (BSTNode *)malloc(sizeof(BSTNode));
    p->data = data;
    p->lchild = p->rchild = NULL;
    return p;
}

BSTNode *insert_node(BSTNode *root, int data) { // 可能改变根节点 返回 BSTNod
e *类型
```

```
    if (root == NULL) return init(data);    // 建立新节点
    if (root->data == data) return root;      // data存在直接返回root
    if (root->data < data) root->rchild = insert_node(root->rchild, data);
 // data大于根节点 插入右子树
    if (root->data > data) root->lchild = insert_node(root->lchild, data);
 // data小于根节点 插入左子树
    // else return root; // root->key == key
    return root;
}

BSTNode *predecesor(BSTNode *node) {  // 找到前驱节点
    BSTNode *temp = node->lchild;
    while (temp->rchild) temp = temp->rchild; // 找到左子树中最右边的值
    return temp;
}

BSTNode *delete_node(BSTNode *root, int data) {
    if (root == NULL) return root;  // 待删除数据在二叉搜索树中不存在 直接返回
    if (root->data < data) root->rchild = delete_node(root->rchild, data);
    if (root->data > data) root->lchild = delete_node(root->lchild, data);
    if (root->data == data) {  // 不能写 else （不存在直接返回）
        if (root->lchild == NULL && root->rchild == NULL) {  // root出度为0
直接删
            printf("%d\n", root->data);
            free(root);
            return NULL;
        } else if (root->lchild == NULL || root->rchild == NULL) { // root
出度为1 提升唯一子节点 然后删除
            BSTNode *temp = root->rchild ? root->rchild : root->lchild; //
 获得root的唯一子节点
            printf("%d\n", root->data);
            free(root);
            return temp;
        } else {
            BSTNode *temp = predecesor(root); // 获得root的前驱节点
            /*
                交换root节点和前驱节点的数据
                ^ 异或:位运算符 相同为0 不同为1
                a ^ a = 0; a ^ 0 = a;
                a = a ^ b;
                b = a ^ b; b = a ^ b ^ b = a ^ 0 = a;
                a = a ^ b = a ^ b ^ a = b ^ 0 = b;
                此方法不需要临时变量即可完成两个数的交换
            */
            temp->data ^= root->data;
            root->data ^= temp->data;
            temp->data ^= root->data;
            root->lchild = delete_node(root->lchild, data);
            // 删除root原前驱节点 转换为上述两种删除方法
        }
    }
    return root;
```

```
}

void output(BSTNode *root) {  // 中序遍历
    if (root == NULL) return ;
    output(root->lchild);       // 先遍历lchild
    printf("%d ", root->data);  // 在遍历root
    output(root->rchild);       // 最后遍历rchild
    return ;
}

int main() {
    srand(time(0));
    BSTNode *root = NULL;
    #define OP_NUM 20

    for (int i = 0; i < OP_NUM; i++) {
        int data = rand() % 30;
        root = insert_node(root, data);
        printf("insert_node %d to tree\n", data);
        output(root), printf("\n");
    }
    for (int i = 0; i < OP_NUM; i++) {
        int data = rand() % 30;
        root = delete_node(root, data);
        printf("delete_node %d from tree\n", data);
        output(root), printf("\n");
    }

    #undef OP_NUM
    return 0;
}
```