数据结构与算法 I 作业 21

2019201409 于倬浩

2020 年 12 月 23 日

18.2-1

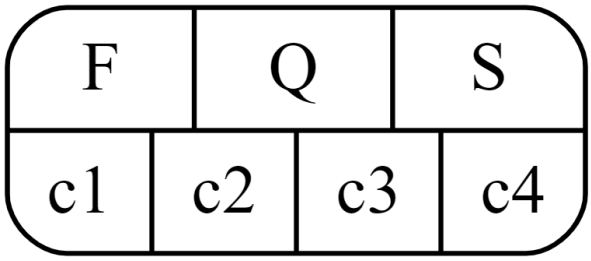

图 1: 插入 Q 后

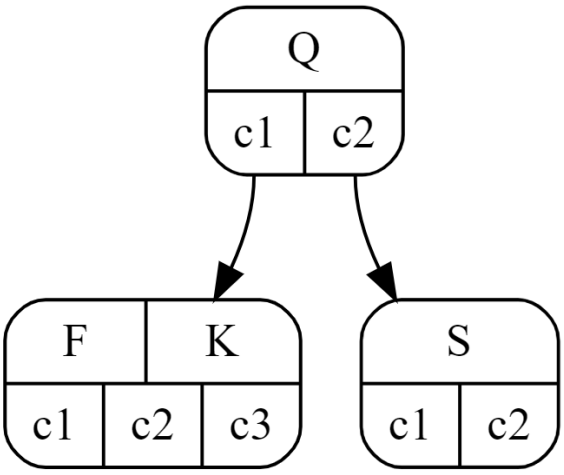

图 2: 插入 K 后

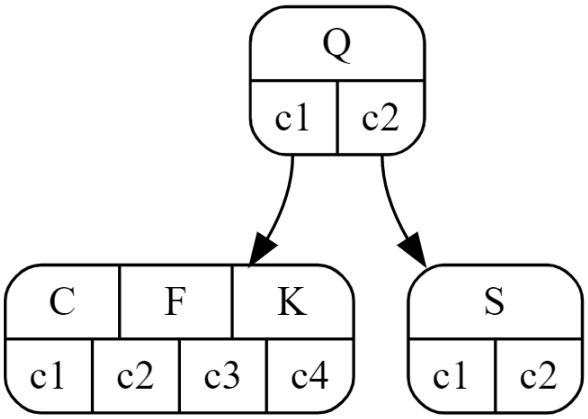

图 3: 插入 C 后

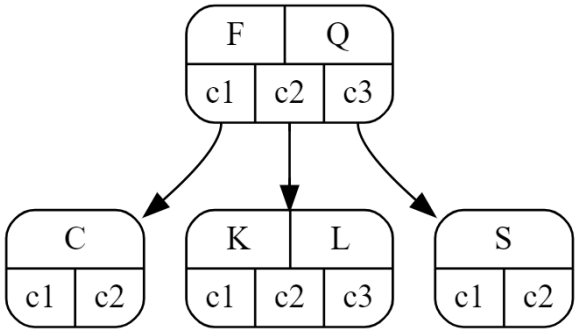

图 4: 插入 L 后

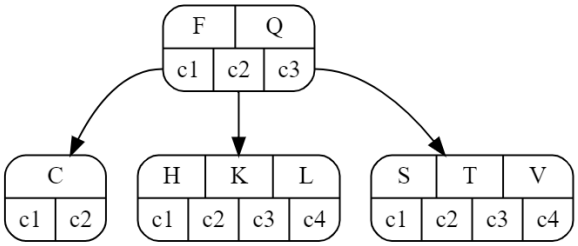

图 5: 插入 V 后

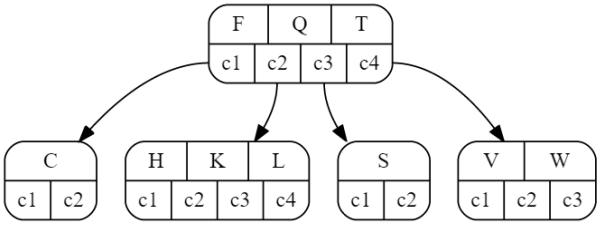

图 6: 插入 W 后

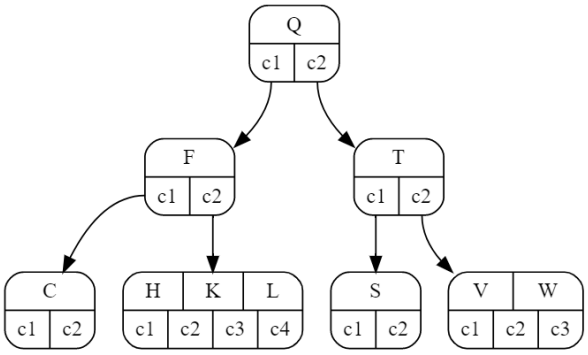

图 7: 插入 M 前（分裂）

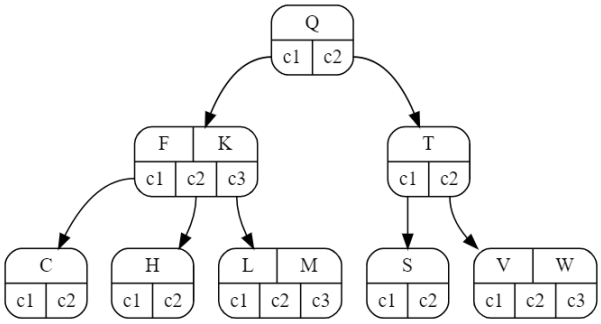

图 8: 插入 M 后

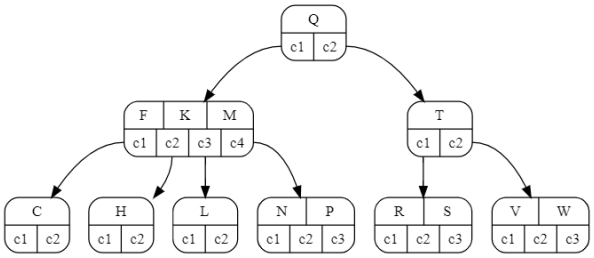

图 9: 插入 P 后

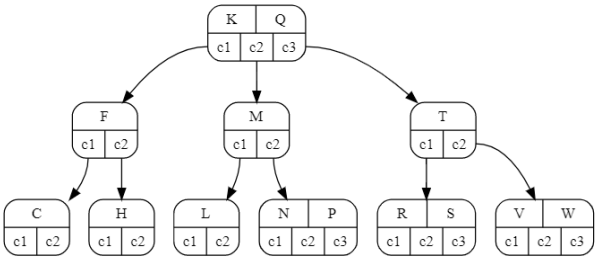

图 10: 插入 A 前（分裂）

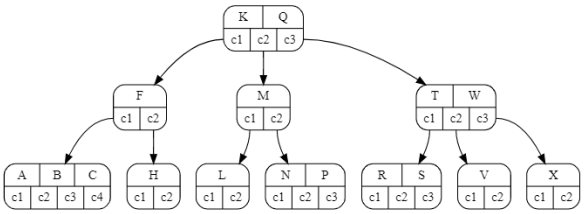

图 11: 插入 Y 前（分裂）

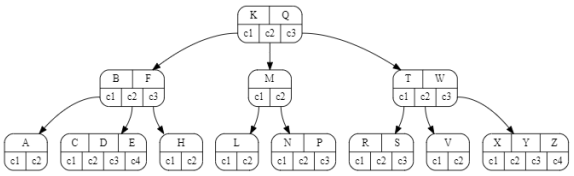

图 12: 最终结果

18.3-2

```
void B_Tree_Delete(Node *x, int k) {
    B_Tree_Read(x);
    if(x->leaf == true)
        delete x->key[k];
    else {
        if(k in x->key[]) {
            node *&y = find_predecessor(x, k), *&z = find_successor(x, k);
            node *kk = find_child(x, k);
            if(y->size >= t) {
                node *yy = find_predecessor(y, k);
                B_Tree_Delete(yy, k);
                y = yy;
            }
            else {
```

```
                if(z->size >= t) {
                    node *zz = find_successor(z, k);
                    B_Tree_Delete(zz, k);
                    z = zz;
                }
                else {
                    if(z->size + y->size == 2*t - 1) {
                        merge(y, kk, z);
                        x->key[].erase(kk), x->key.erase(z);
                        B_Tree_Delete(y, k);
                    }
                }
            }
        }
        else {
            node *xk = Find_K_recursively(x, k), *xkf = xk;
            node *last = Find_father_of_xk(x, k);
            if(xk->size == t - 1 && (xk->left->size >= t || xk->right->size >= t)) {
                xk->key.insert(last);
                if(xk->left->size >= t) x->key.insert(xk->left);
                else x->key.insert(xk->right);
            }
            if(xk->size == t - 1 && xk->left->size == t - 1 && xk->right->size == t - 1) {
                merge(xk->left, last, xk);
            }
            B_Tree_Delete(last, k);
        }

    }
    B_Tree_Write(x);
}
```