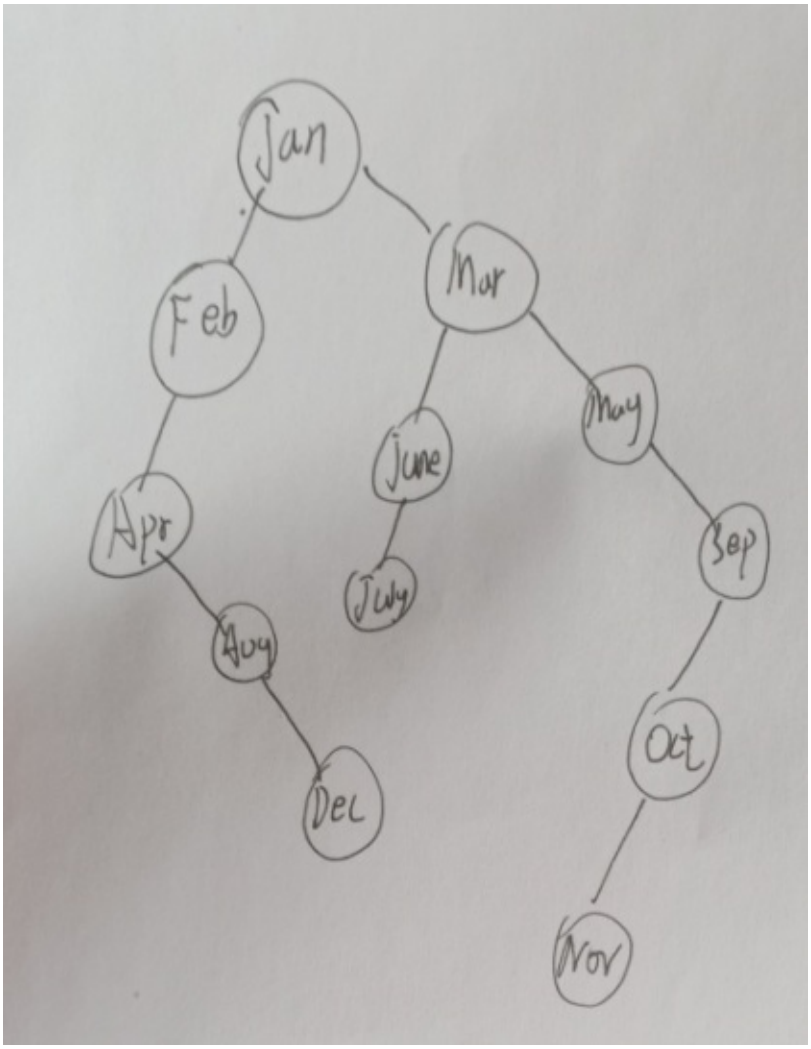
chapter 9

9.9

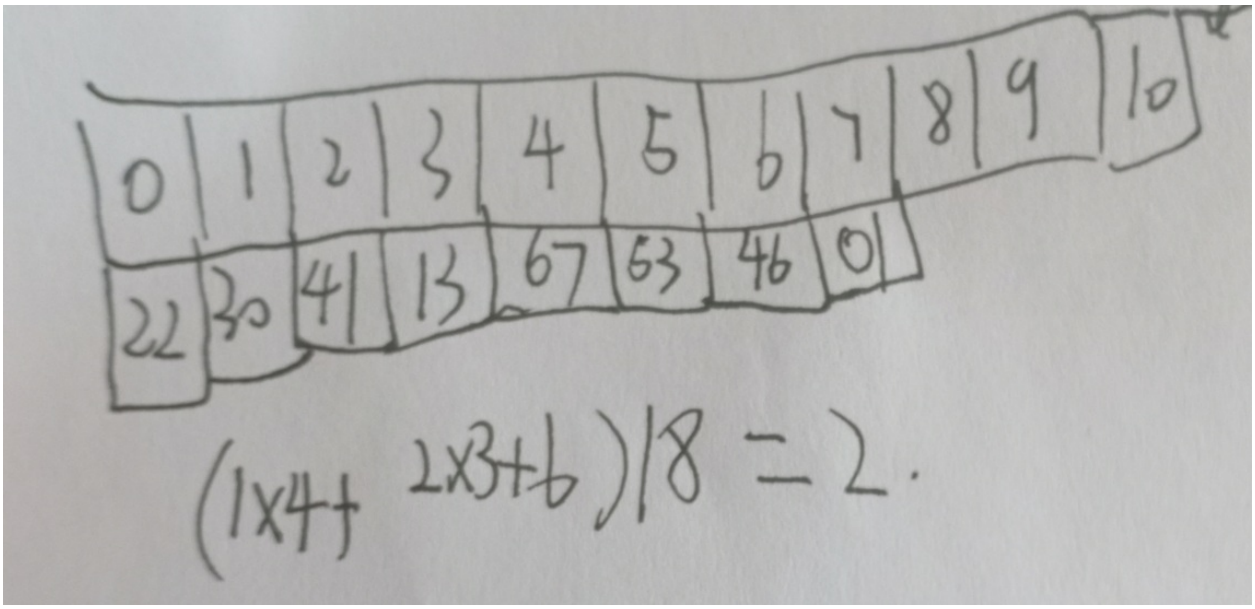


平均查找长度为$\frac{1+2+2+3+3+3+4+4+4+5+5+6}{12}=3.5$

更改为有序表后，平均查找长度为$\frac{1*1+2*2+3*4+4*5}{12}=\frac{37}{12}$

平衡树中，平均查找长度为$\frac{1*1+2*2+3*4+4*4+5*1}{12} = \frac{38}{12}$

9.19



9.21

H(Jan) = 5;
H(Feb) = 3;
H(Mar) = 6;
H(Apr) = 0;
H(May) = 6; 6 + 1 = 7;
H(June) = 5; 5 + 1 = 6; 6 + 1 = 7 7 + 1 = 8;
H(July) = 5; 5 + 1 = 6; 6 + 1 = 7; 7 + 1 = 8; 8 + 1 = 9;
H(Aug) = 0; 0 + 1 = 1;
H(Sep) = 9; 9 + 1 = 10;
H(Oct) = 7; 7 + 1 = 8; 8 + 1 = 9; 9 + 1 = 10; 10 + 1 = 11;
H(Nov) = 7; 7 + ... = 12;
H(Dec) = 2;
ASL = $\frac{31}{12}$

0 -> Apr -> Aug
2 -> Dec
3 -> Feb
5 -> Jan -> June -> July
6 -> Mar -> May
7 -> Oct -> Nov
9 -> Sep
ASL = $\frac{3}{2}$

9.25

```
int search(ST st, Type key) {
    st.elem[st.length].key = key;
    int i = 0;
    while (key < st.elem[i].key) i++;
    if (i == st.length) return 0;
    if (key == st.elem[i].key) return i;
    else return 0;
}
```

判定树为一条链，每个节点只有右孩子

ASL成功 = $\frac{n+1}{2}$
ASL失败 = $\frac{n+2}{2}$

9.26

```
int search(int key, vector<int> &st, int l, int r) {
    int mid = (l + r) / 2;
    if (st[mid] < key) {
        return search(key, st, mid, r);
    } else if (st[mid] > key) {
        return search(key, st, l, mid);
    } else {
        return mid;
    }
}
```

9.28

```
int search(ST st, Type key) {
    int l, r, mid;
    if (key < st.block[1].min) l = 0;
    else if (key >= st.block[st.num].min) l = st.num;
    else {
        l = 1, r = st.num;
        while (r - l > 1) {
            mid = (l + r) / 2;
            if (st.block[mid].min <= key) l = mid;
            else r = mid;
        }
    }
    if (l) {
        if (l == st.num) {
            r = st.length;
        } else {
            r = st.block[r].beg - 1;
        }
        while (r - l > 1) {
            mid = (l + r) / 2;
            if (key <= st.t.elem[mid].key) r = mid;
            else l = mid;
        }
    }
    return r;
}
```

9.31

```
bool check(TreeNode *root) {
    if (!root) return true;
    if (root->left != nullptr && root->left->val > ro
    if (root->right != nullptr && root->right->val <
    return check(root->left) && check(root->right);
}
```

9.33

```
void find(TreeNode *root, int key) {
    if (!root) return;
    if (root->val >= key) {
        find(root->left);
        del(root->right);
        delete root;
    } else {
        find(root->right);
    }
}
void del(TreeNode *root) {
    if (!root) return;
    del(root->left);
    del(root->right);
    delete root;
}
```

9.36

```
void insert(TreeNode *root, Type e) {
    TreeNode *p;
    find(root, e, &p);
    TreeNode *newnode = new TreeNode();
    newnode->data = e;
    newnode->ls = NULL;
    newnode->RTag = Thread;
    if (!p) {
        root->ls = newnode;
        newnode->rs = root;
    } else if (e < p->data) {
        p->ls = newnode;
        newnode->rs = p;
    } else {
        newnode->rs = p->rs;
        p->RTag = Link;
        p->rs = newnode;
    }
}
```