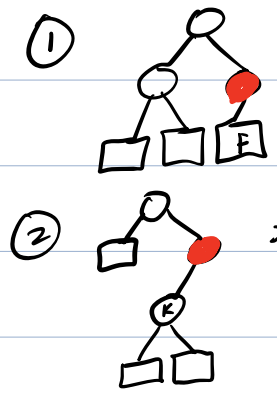
16.3-2

证明不满二叉树不可能对应一个最优前缀码

反证：设一棵不满二叉树T对应了一个C的最优前缀码

由于T是不满二叉树，存在某个中间结点，其只有一个孩子。（若无孩子则该结点无意义）

有2种情况：



① 该结点唯一孩子是叶子

② 该结点唯一孩子是中间结点

无论哪种，只要去把其孩子结点用于替代原结点，得到树T'

对于1情况，明显只有叶子F的编码受影响，上提一层，编码变短。$F.freq \cdot d_{T(F)}$ 变小

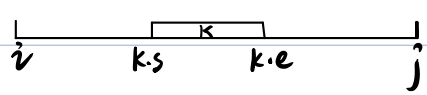
对于2情况，明显该子树下所有叶结点上提一层，编码变短 $\sum_{c\in 子树K} c.freq \cdot d_{T(c)}$ 变小

综上所述，T'比T有更好的编码性能，与T为最优矛盾

2. Give pseudo-code of weighted interval scheduling problem

weighted interval scheduling problem：each interval has a certain weight

✗ greedy ✓ dynamic programming



对于 max-weight[i, j]

$$= \begin{cases} 0 & i = j \\ \max\limits_{\substack{\text{intervals } k \\ k.start \le i \\ k.end \ge j}} (\text{max-weight}[i, k.start] + \text{max-weight}[k.end, j] + k.weight) \end{cases}$$

代码在下页

```java
class Solution {
    public void dynamicProgramming(ArrayList<Interval> intervals) {
        //对于开始时间和结束时间段进行动态规划，value[I][J]表示[I,J]区间的最大权重   一些初始化工作
        int n = intervals.size();
        //找到所有事件的最小开始时间
        int start = Integer.MAX_VALUE;
        for (int i = 0; i < n; i++) {...}
        //找到所有事件的最大结束时间
        int end = Integer.MIN_VALUE;
        for (int i = 0; i < n; i++) {...}
        int[][] value = new int[end+1][end+1];
        int[][] track = new int[end+1][end+1];
        for (int i = 0; i <= end; i++) {...}
        for (int i = 0; i <= end; i++) {...}
        //整体思路类似于矩阵乘法，每次求出[i,j]区间的最大权重，并且记录下来   动态规划核心逻辑
        for(int len = 1; len <= end; len++) {
            for (int i = 0; i <= end - len; i++) {
                int j = i + len;
                value[i][j] = 0;
                for(int index = 0; index < n; index++) {
                    if(intervals.get(index).start >= i && intervals.get(index).end <= j) {
                        if(value[i][j] < intervals.get(index).weight + value[i][intervals.get(index).start] + value[intervals.get(index).end][j]) {
                            value[i][j] = intervals.get(index).weight + value[i][intervals.get(index).start] + value[intervals.get(index).end][j];
                            track[i][j] = index;
                        }
                    }
                }
            }
        }
        System.out.println(value[0][end]);
        printTrack(intervals, track,  start: 0, end);
    }
```

定义事件的类 包含权重

```java
class Interval {
    public int start;
    public int end;
    public int weight;
    public Interval(int start, int end, int weight) {
        this.start = start;
        this.end = end;
        this.weight = weight;
    }
    public void print() {
        System.out.print("[" + start + ", " + end + "]  ");
    }
    public void printWithWeight() {
        System.out.print("[" + start + ", " + end + "] " + "weight: " + weight + "\n");
    }
}
```

输出最优解的代码

```java
//输出最优解的选择
public void printTrack(ArrayList<Interval> intervals, int[][] track, int start, int end) {
    if(start == end) {
        return;
    }
    if(track[start][end] == -1) {
        return;
    }
    int index = track[start][end];
    System.out.println("[" + intervals.get(index).start + " " + intervals.get(index).end+ "]");
    printTrack(intervals, track, start, intervals.get(index).start);
    printTrack(intervals, track, intervals.get(index).end, end);
}
```

测试用例与输出结果

```java
ArrayList<Interval> intervals = new ArrayList<>();
intervals.add(new Interval( start: 1,  end: 4,  weight: 1));
intervals.add(new Interval( start: 3,  end: 5,  weight: 2));
intervals.add(new Interval( start: 0,  end: 6,  weight: 3));
intervals.add(new Interval( start: 5,  end: 7,  weight: 4));
intervals.add(new Interval( start: 3,  end: 9,  weight: 5));
intervals.add(new Interval( start: 5,  end: 9,  weight: 6));
intervals.add(new Interval( start: 6,  end: 10,  weight: 7));
intervals.add(new Interval( start: 8,  end: 11,  weight: 8));
intervals.add(new Interval( start: 8,  end: 12,  weight: 9));
intervals.add(new Interval( start: 2,  end: 14,  weight: 10));
intervals.add(new Interval( start: 12,  end: 16,  weight: 11));
```

```
26
[3, 5] weight: 2
[5, 7] weight: 4
[8, 12] weight: 9
[12, 16] weight: 11
```