拔河

30%的数据是给2^{2n}的暴力的；

接下来我们来分析一下题目性质，我们把绳子的两边看成二分图的两个部分A,B，对于一个人(l,r,s)，我们连一条$w(A_l,B_r)=s$的边。

如果存在点的度数为0那么肯定无解，如果存在点的度数为1那么这个点的方案就已经定了，从图中把这个点以及相关的边删除。重复上述操作直到所有点都是度数为2的为止。（是不是和往年有一届NOI很像）

对于剩下的度数为2的所有点，可以构成若干个偶环，显然对于每个环都只有两种方案，设奇数边权和为a，偶数边权和为b，我们只需要记录$|a-b|$即可。

对于60%的数据我们把每一个$|a-b|$作为元素，直接背包，时间复杂度是$O(n\cdot\sum s_i)$。

对于100%的数据，我们只需要更改一下背包的方式即可，原来我们背包的元素有$O(n)$个，现在我们把这个背包改成多重背包即可。

接下来我们来讲讲道理。

令(x,y)表示权值为x的物品有y个，假设一共有S种权值，那么有：

$$\sum_{i=1}^{S}x_iy_i\leq\sum_{i=1}^{2n}s_i\leq 4.5\cdot 10^5$$

S最大的情况下可以假设每种权值有且仅有一种，且为$1,2,\ldots,S$，那么有：

$$\frac{S(S+1)}{2}\leq 4.5\cdot 10^5$$

可以发现极限情况下$S\ll n$，而一次多重背包的时间复杂度是$O(\sum s_i)$的，再由于还要满足点数之和$\leq n$，状态数实际上还要少很多（算了算最多三四亿的时间复杂度，不过肯定达不到），这样我们就可以在$O(S\cdot\sum s_i)$的时间内通过这道题啦。

光盘

很明显这是一个括号序列的模型。

对于10%的数据，直接爆枚即可。

对于20%的数据，我们可以构造一个费用流：设源汇点为S,T，初始流量为k，S向所有左括号连代价为A_i的边，所有右括号向T连代价为B_i的边，第i个左括号向所有编号$\geq i$的右括号连一条代价为0的边，跑最小费用最大流即可。

对于30%的数据，我们只需要把模型稍微改一下，没有必要“第i个左括号向所有编号$\geq i$的右括号连一条代价为0的边”，我们直接从第i个左括号向第i个右括号连代价为0的边，然后第i个右括号向第$i+1$个右括号连边即可。

对于50%的数据，我们观察一下费用流的实现过程，实际上就是每次从剩下的括号中取出权值和最小的合法的一对"()"或")("加入当前的括号序列中，取出k次即可。暴力选择一次是$O(n)$的，时间复杂度$O(kn)$。

对于100%的数据，我们考虑用线段树维护选择方案。

设"("的位置为A，")"的位置为B。

设S_i为前缀和。

如果当前取出"()"，那么区间$[A,B-1]$的前缀和$+1$。

如果当前取出")("，那么区间$[B,A-1]$的前缀和-1，要保证不会减成负数。

对于区间$[L,R]$：

设m_1表示权值最小的A，m_2表示权值最小的B；

设m_3表示权值最小的满足$[L,A]$的前缀和最小值$>[L,R]$的A；

设m_4表示权值最小的满足$[B,R]$的前缀和最小值$>[L,R]$的B；

设min为区间前缀和最小值，add为区间加减标记；

设a_1表示权值和最小的"()"；

设a_2表示权值和最小的满足$[A,B-1]$的最小值$>[L,R]$的")(";

设a_3表示权值和最小的")("。

为啥需要记这么多东西呢~~（肯定是我太弱了）~~，由于一个区间的前缀和从0变成>0，又从>0变成0，会使得")("比较难维护，因此我们要记这些东西，如果脑补不出怎么搞的话，也可以看看代码。

其中$getmin(a,b,c)\Rightarrow a=\min(b,c)$，$merge(a,b)$相当于取出两种方案中小的那个。

```
inline friend data operator + (const data &p, const data &q) {
    data r;
    getmin(r.m1, p.m1, q.m1);
    getmin(r.m2, p.m2, q.m2);
    r.min = std::min(p.min, q.min);
    r.a1 = merge(merge(p.a1, q.a1), std::make_pair(p.m1, q.m2));
    r.a3 = merge(merge(p.a3, q.a3), std::make_pair(p.m2, q.m1));
    //m3, m4, a2
    if(p.min < q.min) {
        r.m3 = p.m3;
        getmin(r.m4, q.m2, p.m4);
        r.a2 = merge(merge(p.a2, q.a3), std::make_pair(p.m4, q.m1));
    }
    else if(p.min == q.min) {
        r.m3 = p.m3;
        r.m4 = q.m4;
        r.a2 = merge(merge(p.a2, q.a2), std::make_pair(p.m4, q.m3));
    }
    else {
        r.m4 = q.m4;
        getmin(r.m3, q.m3, p.m1);
        r.a2 = merge(merge(p.a3, q.a2), std::make_pair(p.m2, q.m3));
    }
    return r;
}
```

时间复杂度$O(k\log n)$，但是由于维护的东西较多，常数比较大。

树

对于10%的数据，我们直接暴力枚举每一种选边方案即可，$O(2^n)$。

对于另15%的数据，我们观察一下答案的计算方式$(\sum a[i])^k$，暴力展开后，是不是相当于在集合中可重地选出k个数的积之和，因此我们可以暴力枚举四个点，算出联通块包含它们的概率，累加起来即可，时间复杂度$O(n^4)$。

对于另25%的数据，我们按照上面的思路，设$f[i][j]$表示以i为根的子树中，联通块包含结点i，已经可重复地选出j个数的贡献。那么方程如下：

初始化$f[i][0] = 1, f[i][j] = f[i][j-1] \cdot a[i]$

我们把当前根结点和子树合并的时候，其实就是把两个有序序列合并成一个有序序列：

$$f[i][j] = (1-p) \cdot f[i][j] + p \cdot \sum_{x=0}^{j} f[son][x] \cdot f[i][j-x] \cdot \binom{j}{x}$$

每个点对答案的贡献就是$f[i][k] \cdot (1 - p_{fa})$，其中$p_{fa}$指的是父边的概率，直接暴力合并的时间复杂度就是$O(nk^2)$的。

对于100%的数据，我们考虑稍微变形一下：

$$f[i][j] = (1-p) \cdot f[i][j] + p \cdot j! \cdot \sum_{x=0}^{j} \frac{f[son][x]}{x!} \cdot \frac{f[i][j-x]}{(j-x)!}$$

是不是就可以NTT了啊，时间复杂度$O(nk \log k)$。