国庆集训 Day1（普及班）

福州第一中学 林国军

排序

- 题目描述
- 2023 年 CSP−J1成绩公布了。小毅手里有n名同学的成绩，他想得到其中前k个同学的成绩按从低分到高分排序后的结果。小毅今天参加提高班训练去了，所以把这个问题交给了你。
- 输入格式
- 第一行两个空格隔开的整数 n,k。
- 第二行n个空格隔开的整数，第i个整数a_i表示第i个同学的成绩。
- 输出格式
- 一行k个由空格隔开的整数。
- 对于100%的测试数据，$1≤k≤n≤5000,0≤a_i≤100$

排序

- 将int型a数组中的a[1]到a[n]按照升序排序。

左闭，右开。
时间复杂度O(nlogn)

- sort(a+1,a+n+1);
- 需要#include<algorithm>
- 参考程序

拓展问题：离散化基础

- 描述
- 在以后要学习使用的离散化方法编程中，通常要知道每个数排序后的编号(rank值)。
- 输入
- 第1行：一个整数N，范围在[1…10000]。
- 第2行：有N个不相同的整数，每个数都是int范围的。
- 输出
- 依次输出每个数的排名。
- 样例输入
- 5
- 8 2 6 9 4
- 样例输出
- 4 1 3 5 2

拓展问题：离散化基础

- 分析
- 排序是必须的，关键是怎样把排名写回原来的数“下面”。
- 程序使用了分别对数值和下标不同关键字2次排序的办法来解决这个问题，一个数据“节点”应该包含：数值，排名，下标3个元素，用结构体比较好。

拓展问题：离散化基础

- 说明
- sort函数是<algorithm>提供的快速排序函数，可以自己定义比较函数。

拓展问题：离散化基础

- 比较函数（T为自定义的结构体类型）
- T d[N+10];
- ...
- sort(d+1,d+N+1,cmp);
- bool cmp(T x,T y){
- ....
- }
- 比较函数规则：若cmp(a,b)为true，则a排在b的前面。特别地，要求a=b时，返回值应为false

拓展问题：离散化基础

```
#include<bits/stdc++.h>
using namespace std;
struct tNode{
        int data,rank,index;//依次表示数值、排名、下标
}a[10010];
int n;
bool cmpData(tNode x,tNode y){
        return x.data<y.data;
}
bool cmpIndex(tNode x,tNode y){
        return x.index<y.index;
}
int main(){
        scanf("%d",&n);
        //输入数据
        for(int i=1;i<=n;++i){
                scanf("%d",&a[i].data);
                a[i].index=i;
        }
        //按照值排序，求rank
        sort(a+1,a+n+1,cmpData);
        for(int i=1;i<=n;++i) a[i].rank=i;
        //按照下标排序，回复“原始顺序”
        sort(a+1,a+n+1,cmpIndex);
        //输出
        for(int i=1;i<=n;++i) printf("%d ",a[i].rank);
        return 0;
}
```

打怪物

- 题目描述
- 怪物正在入侵Steve的家，Steve需要打败怪物。
- Steve的体力为 a，怪物的体力为 b。每个时刻Steve可以选择攻击或者治疗，若选择攻击，那么怪物的体力将会被扣到 b-p；若选择治疗，那么Steve的体力将会恢复到 a+s，不过如果恢复后的体力比战斗开始时的体力还大，那么只会恢复到战斗开始时的体力。无论选择攻击还是治疗，该时刻结束时Steve都会受到怪物攻击，Steve的体力会被扣到 a-p，之后怪物的体力会恢复到 b+q（同样，如果恢复到比开始时还大则只会恢复到开始时的体力）。
- 一旦某一方体力被扣到 0 或以下时，战斗结束，该方战败，另一方战胜。作为对战爱好者，Steve希望战胜对方时自己的体力恰好为 k。Steve想知道战胜至少要多少时刻。
- 输入格式
- 输入仅一行，包含六个非负整数 a,b,p,q,s,k，每两个整数之间用一个空格隔开。
- 输出格式
- 输出仅一行，包含一个整数，表示最少需要在第几个时刻击败怪物。若无法满足条件击败怪物，输出-1。

打怪物

- 对于 100% 的数据，$1 \le a \le 500$，$1 \le b \le 500$，$p \le 100$，$q \le 100$，$s \le 100$，$1 \le k \le a$

广度优先搜索

给定一个**N*N**方格的迷宫，迷宫里有若干处障碍，障碍处不可通过。在迷宫中移动有上下左右四种方式。给定起点坐标和终点坐标（保证有路径到达）。问从起点到终点的要经过的最少步数。

输入：第一行**N**（**N<=1000**），下面是一个**N*N**的**01**矩阵，**0**表示可以通过，**1**表示障碍。

最后一行四个数：起点坐标**SX,SY**，终点坐标**FX,FY**。

输出：一个整数，表示起点到终点要经过的最少步数。

广度优先搜索

零步可达：**(1,1)**

一步可达：**(2,1)**

两步可达：**(2,2) , (3,1)**

三步可达：**(4,1)**

四步可达：**(4,2)**

五步可达：**(4,3)** 找到答案！

S	*		
		*	
	*		
		T	

广度优先搜索

图示，标红为“待扩展节点”：

点	(1,1)								
步数	0								

点	(1,1)	(2,1)							
步数	0	1							

点	(1,1)	(2,1)	(2,2)	(3,1)					
步数	0	1	2	2					

待拓展节点（标红），似乎在排着队，等待“被拓展”，“被拓展”结束后，即离开队列。则队列里的节点就是“待拓展”节点。

队列

那么如何用程序实现队列呢？

只需要：

(1)一个数组q

(2)一个头指针head指向队列中的头元素

(3)一个尾指针tail指向队列中最后一个元素的后一个位置

q数组

点	(1,1)	(2,1)	(2,2)	(3,1)					

head=2　　　　**tail=4**

广度优先搜索

例题一：迷宫问题（最少步数）

队列的基本操作：

(1)初始化:head=tail=0.这时候队列中没有元素，为空。

(2)x元素入队:

q[tail++]=x;

(3)队首元素赋值给a并出队:

a=q[head++];

(4)队列为空的条件:

head==tail

(5)队列中元素个数:

tail-head

广度优先搜索

```
BFS()
{
        初始结点入队
        记录状态
        while(队列非空)
        {
                取出队首元素u
                遍历所有相邻且还未访问过的元素v
                {
                        将v点加入队列
                        记录状态
                        如果v点为目标结点，返回相关信息。结束。
                }
                u点出队
        }
}
```

打怪物

- a,b,p,q,s,k均为输入数据。没有太好的贪心或动归算法。
- 考虑爆搜。
- 不难发现这是一个广度优先搜索问题。
- 画出搜索树。
- 搜索状态数仅500*500，每个状态出边至多两条（对应选择攻击或治疗两种决策）。
- 参考程序

子序列

- 题面简述：
- 爱好数学的小 B 开始研究子序列的问题。他称一个子序列的权值为序列中不同数的种数。
- 特别地，空序列的权值为 0。现给定一个 n 项的数列 $a_1,a_2,\dots,a_n$，小 B 想知道：该数列所有子序列的权值之和是多少？该数列所有连续子序列的权值之和是多少？
- 对于 100% 的数据，$1 \le n \le 100000$，$1 \le a_i \le 1000000000$。

子序列

- 暴力算法：
- $O(2^n)$边DFS枚举子序列，并累加统计贡献：
- DFS过程中用一个数组动态记录每个数字出现的次数，用一个变量动态记录当前不同数字个数。
- （出现次数由0变1，则不同数字个数加一；由1变0则不同数字个数减一）
- 连续子序列仅需要$O(n^2)$即可枚举，并$O(n)$统计贡献。
- 可过：对于 50% 的数据，$1 \le n \le 25$，$1 \le a_i \le 100$；（如果$O(2^n*n)$仅可过30%）

子序列

- 题目要求求所有满足条件的子序列和子段的数字种类数之和。
- 对于100%的测试数据，因为暴力枚举的复杂度是肯定承受不了的
- 那此时就要转变答案的统计方式，转换枚举对象？

子序列

- 常见的做法之一是，累加序列中每个元素对答案的贡献。
- 每个元素存在于某个序列时，对种类数的贡献最多为1？
- 如何避免相同元素重复贡献呢？
- 等价地，我们可以认为一个序列中相同的数中只有第一个能对答案做出贡献。
- 这样就不会重复、也不遗漏了。

子序列

- 先考虑子序列(不一定连续)
- 第i个数对答案的贡献？
- 即为所有子序列中，满足第i个数为子序列中首次出现的子序列的方案数。
- 因此原序列中，每个数有取或不取两种选择。而在第i个数之前出现的与第i个数相等的数字一定不能取！
- 方案数？
- pow(2,n-1-前i个数中与第i个数相等的数字个数)

子序列

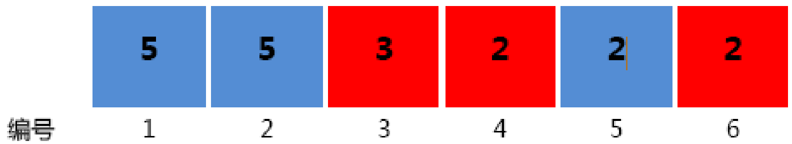
- 再思考子段（连续）？
- 也同样的思路，把求第i个数对答案的贡献，转化为求满足第i个数是子段中首次出现的子段的方案数。
- 根据分步计数原理，我们可以把确定子段分为两步走。
- 1.确定左端点。方案数为x
- 2.确定右端点。方案数为y
- 则答案为x*y。那么x=?y=?

子序列

- x=i-原数列中前一个与第i个数相等的数的位置
- y=n-i+1(右端点只要位置大等于i即可。)
- 需要处理出：
- 1.前i个数中与第i个数相等的数的个数
- 2.原数列中前一个与第i个数相等的数的位置？
- 可以在从小到大枚举i时O(1)维护。
- 如何操作？
- 需要**离散化**!
- 参考程序

求和

- 一条狭长的纸带被均匀划分出了n个格子，格子编号从 1 到 n。每个格子上都染了一种颜色$color_i$（用 [1,m] 当中的一个整数表示），并且写了一个数字$number_i$。



- 定义一种特殊的三元组：(x,y,z)，其中 x，y，z 都代表纸带上格子的编号，这里的三元组要求满足以下两个条件：
- 1. x,y,z都是整数, $x<y<z,y-x=z-y$
- 2. $color_x=color_z$
- 满足上述条件的三元组的分数规定为$(x+z)*(number_x+number_z)$。
- 整个纸带的分数规定为所有满足条件的三元组的分数的和。这个分数可能会很大，你只要输出整个纸带的分数除以 10,007 所得的余数即可。
- 对于100%的数据，$1\le n\le 100000,1\le m\le 100000,1\le color_i\le m,1\le number_i\le 100000$。

求和

- 分析问题，发现每个三元组对答案的贡献只和**x**及**z**有关，和**y**无关，而**y**只需要存在即可。
- 易得**y=(x+z)/2**，可知：**x**和**z**奇偶性必须相同
- 因此我们称**x**与**z**状态相同，当且仅当：
- **a. x**与**z**奇偶性相同　**b.x**与**z**颜色相同。
- 原题转化为求所有格子中状态相同的格子对**(x,z)**的**(x+z)(number$_x$+number$_z$)**值之和。

求和

- 算法一
- 枚举z，查找z之前与它状态相同的所有格子x，并将格子对(x,z)对答案的贡献累加进ans。
- 时间复杂度：
- $O(n^2)$ 只能通过40%的测试数据。

求和

- 算法二
- 将式子$(x+z)(number_x+number_z)$展开，得：
- $x*number_x+x*number_z+z*number_x+z*number_z$
- 记与z状态相同，且编号小于z的格子编号依次为$x_1,x_2,x_3…x_i$.
- 考虑枚举z，并计算每个z作为格子对(x,z)时，对答案的贡献之和。

求和

- 算法二
- z对答案的贡献：
- $x_1 * number_{x1} + x_1 * number_z + z * number_{x1} + z * number_z$
- $x_2 * number_{x2} + x_2 * number_z + z * number_{x2} + z * number_z$
- ...
- $x_i * number_{xi} + x_i * number_z + z * number_{xi} + z * number_z$
- 每项第一列： $x_1 * number_{x1} + x_2 * number_{x2} + ... + x_i * number_{xi}$
- 每项第二列：$(x_1 + x_2 + ... + x_i) * number_z$
- 每项第三列：$(number_{x1} + number_{x2} + ... + number_{xi}) * z$
- 每项第四列：$i * z * number_z$

求和

- 算法二（x为编号，$number_x$为格子数字）
- 第一列是z之前与z状态相同格子的$x*number_x$的前缀和
- 第二列是z之前与z状态相同格子的x的前缀和乘上$number_z$
- 第三列是z之前与z状态相同格子的$number_x$数组的前缀和乘上z
- 第四列是z之前与z状态相同格子个数乘上 $z*number_z$
- 则只需要对于每一种状态（颜色和奇偶性），记录：
- 1.$x*number_x$的前缀和
- 2.x的前缀和
- 3.$number_x$数组的前缀和
- 4.状态相同格子个数

求和

- 算法二
- 每种状态用一个二维数组表示：　**[col][w] , col**表示颜色，**w=0,1**存储奇偶性。
- 所以开这几个数组（初值均为**0**）：
- （**1**）**psum[col][w]**记录颜色为**col**，位置奇偶为**w**（**w=0,1**）的所有格子$x*number_x$之和；
- （**2**）**isum[col][w]**记录颜色为**col**，位置奇偶为**w**（**w=0,1**）的所有格子编号**x**之和；
- （**3**）**nsum[col][w]**记录颜色为**col**，位置奇偶为**w**（**w=0,1**）的所有格子$number_x$之和；
- （**4**）**cnt[col][w]**记录颜色为**col**，位置奇偶为**w**（**w=0,1**）的格子个数；
- 则对于枚举到的每一个**z**，其对答案的贡献之和等于：
- $psum[col][w]+isum[col][w]*number_z+z*nsum[col][w]+cnt[col][w]*z*number_z$
- 其中$col=color_z$，**w=z%2**

求和

- 然后将枚举到的z格子，统计到它所属类别中去。令
- （1）psum[col][w]加等于$z*number_z$；
- （2）isum[col][w]加等于z；
- （3）nsum[col][w]加等于$number_z$；
- （4）cnt[col][w]加等于1；
- 这样枚举完z以后，答案就算出来了，时间复杂度O(n)
- 参考程序

营救行动

- 题目描述
- 4748 年，天猫城的地表下发生了崩塌。天猫城和周围的城市面临着极大的危险。营救行动迫在眉睫。
- 天猫城和周围的城市总共有 n 座，n-1条无向道路连接着它们，使得任意两个城市都能方便地到达。每个城市都住着许多人，第 i个城市的人数为 w_i。现在一切能到达外界的快速通道已经被破坏，只能通过空中传送器这种老旧的方式进行传送，而仅有的 k 座空中传送接口分布在这 n 座城市的 k 个地方。
- 为了防止交通堵塞，你，营救行动的总指挥，决定切断一些道路，将这些城市城市分成若干个连通的部分，每个部分都必须至少有一个传送接口。然后你再对每个连通部分，控制仅仅一个空中传送器营救这个连通部分中的所有人。
- 一个传送器有一个规格 lim，表示能传送的最大总人数，如果超过这个规格，传送器就不能工作。由于一些原因，你派出的所有空中传送器必须具有相同的规格。因此你希望找到一个最小的 lim，使得在这种规格下，存在一种划分城市的方式，让所有人都能被顺利营救。

营救行动

- 输入格式
- 输入数据的第一行包含两个整数 n,k，表示城市的数量和空中传送接口的数量；
- 第二行包含 n 个整数w_i，表示每个城市的人数；
- 第三行包含 k 个整数 p_i，表示每个传送接口分别在哪个城市中；
- 接下来 n-1行，每行两个整数，表示这 n-1 条连接城市的无向道路。
- 输出格式
- 输出数据的第一行包含一个整数 lim，表示能使得划分城市方案存在的最小规格。

营救行动

<table>
<tr><th>测试点编号</th><th>n</th><th>k</th><th>w_i</th><th>其他限制</th></tr>
<tr><td>1</td><td>$=2$</td><td rowspan="6">$\leq n$</td><td rowspan="6">≤ 5</td><td rowspan="13">无其他限制</td></tr>
<tr><td>2</td><td>$=3$</td></tr>
<tr><td>3</td><td>$=10$</td></tr>
<tr><td>4</td><td>$=15$</td></tr>
<tr><td>5</td><td rowspan="2">$=20$</td></tr>
<tr><td>6</td></tr>
<tr><td>7</td><td rowspan="4">≤ 100000</td><td rowspan="2">$=n$</td><td rowspan="4">$\leq 10^9$</td></tr>
<tr><td>8</td></tr>
<tr><td>9</td><td rowspan="2">$=1$</td></tr>
<tr><td>10</td></tr>
<tr><td>11</td><td>≤ 50</td><td rowspan="10">$\leq n$</td><td rowspan="3">$=1$</td></tr>
<tr><td>12</td><td>≤ 70</td></tr>
<tr><td>13</td><td>≤ 100</td></tr>
<tr><td>14</td><td>≤ 20000</td><td rowspan="7">$\leq 10^9$</td><td rowspan="4">所有道路(x,y)满足
$|x-y|=1$</td></tr>
<tr><td>15</td><td>≤ 50000</td></tr>
<tr><td>16</td><td rowspan="5">≤ 100000</td></tr>
<tr><td>17</td></tr>
<tr><td>18</td><td rowspan="3">无其他限制</td></tr>
<tr><td>19</td></tr>
<tr><td>20</td></tr>
</table>

营救行动

- 部分分算法？
- k=1时？
- 只有一个空中传送接口，答案即为所有城市人数和。可以通过10%测试数据。
- k=n时？
- 贪心，启用所有空中传送接口，答案即为所有城市人数最大值。可以通过10%测试数据。

营救行动

- 暴力算法？
- n个点n-1条边：一棵树。城市为结点，道路为边。
- $O(2^{n-1})$枚举每条边删除/不删除，对于每种方案，O(n)遍历整张图，找人数和最大的连通块，并用打擂法动态刷新答案。
- 总的时间复杂度：$O(2^{n-1}*n)$.
- 可以再拿30%的分数。思考如何实现？
- 本题基本得分：50分。

营救行动

- lim表示能传送的最大总人数。要求lim最小值。
- 最大值最小问题，考虑二分答案+贪心判断求解。
- 答案显然满足单调性，如果x行，那么大于x在同样划分下也一定行。如果x不行，那么小于x也一定不行。
- 如何check(ans)?

营救行动

- 部分分：所有道路(x,y)满足|x-y|=1。
- 即原图为一条链：
- 1-2-3-...-n
- 显然将所有传送接口均开启生效不会更劣。（即每个传送接口独立一个连通块，并安放空中传送器）
- 可以很容易check：

营救行动

- 贪心地分段：
- 对于当前第一个（最左边）传送口，如果它之前（不含它本身）的所有结点（城市）人口已经超过ans，则不合法。
- 否则：
- 1.二分找到当前第一个传送口右边（含其本身）尽量远的位置k，使得ans恰好大于这段结点人口总和。（需要预处理人口前缀和）。
- 2.将当前段终点置为min(k,pos-1)，其中pos为当前第二个传送口的位置，若不存在两个传送口，pos=inf。
- 3.删除该段，后重复分段操作，直到处理完所有传送口后，检查终点是否为n，若是则可判定为合法。
- 具体实现时需要注意细节。

营救行动

- 又再拿下20分。
- 水过70分...很高了

营救行动

- 那不是一条链该如何check呢？
- 已经确定传送人数ans，可以考虑用树形DP检验其合法性。
- 记f[i]表示以i为根的子树被全部覆盖后的运输量结余。（如果不能全部被覆盖，则记为0）.
- 记g[i]表示若以i为根的子树全部被覆盖，至少要从别处引进多少运输量（不需要引进记为0）.
- 以上“被覆盖”指该子树内的人口全部能被送走。

营救行动

- 对于f值为正的子树的根结点，它可能可以连向其父节点，去“帮助他人”;
- 还可能切断连向其父节点的边，“自成一体”。
- 让f值为正的子树从别处引进运输量一定是不优的。（因为别处引进运输量需要在子树外还有一个传送口，那么由于一个连通块只能一个传送口工作，则该f值为正的子树内正在生效的传送口就不能工作。那么与其让这棵子树成为他人负担不如“自产自销”）。
- 对于g值为正的子树根结点，它必须连向它的父亲结点，寻求它人帮助。

营救行动

- 对于点i，在计算f[i]和g[i]时，显然要先判断它是运输量有结余（可能可以去帮助他人）还是需要从别处引进运输量（不能自产自销，需要别人帮助）。
- 则要先计算出i点最大的运输承载力v和i点需要承载的运输量c。
- 则如果i点是传送接口，贪心，一定将接口开起来（启用）。
- v=ans。（因为它儿子给它的结余量显然不可能大于ans）

营救行动

- 如果i点不是传送接口，那么要从它儿子中选取一个“余量”最大的，连向它来给它提供最大运输能力。
- 则此时$V=\max\{f[j] \mid j \in son_i\}$
- 思考i点需要承载的运输量c？
- 刚分析过了，对于i结点儿子中，g值为正的点，它们必须连向i，成为i的“负担”。
- i结点儿子中，f值为正的（即g值为0）的点一定不会成为i点的负担。
- 则不难得到$c=\text{sum}\{g[j] \mid j \in son_i\}+w[i]$

营救行动

- 如果v>=c，即i点承载能力大于所需要的承载量：
- f[i]=v-c,g[i]=0
- 如果v<c，即i点承载能力小于所需要的的承载量：
- 那么全部所需承载量都要向外寻求帮助（儿子结点不可帮扶，因为一个连通块只能有一个生效的传送接口）。
- f[i]=0,g[i]=c

营救行动

- 对于每一个二分出的ans：
- 都O(n)dfs整棵树，最后ans不可行当且仅当g[root]>0。（root可随机选取）
- 参考程序
- 总结：
- 学会写暴力，学会拿部分分！

括号树

- 题目描述
- 一个大小为 n 的树包含 n 个结点和 n−1条边，每条边连接两个结点，且任意两个结点间有且仅有一条简单路径互相可达。
- 小 Q是一个充满好奇心的小朋友，有一天他在上学的路上碰见了一个大小为 n 的树，树上结点从 1~n 编号，1 号结点为树的根。除 1 号结点外，每个结点有一个父亲结点，u（$2 \le u \le n$）号结点的父亲为 f_u（$1 \le f_u < u$）号结点。
- 小 Q发现这个树的每个结点上恰有一个括号，可能是 '(' 或')'。小 Q 定义 s_i 为：将根结点到 i号结点的简单路径上的括号，按结点经过顺序依次排列组成的字符串。
- 显然 s_i是个括号串，但不一定是合法括号串，因此现在小 Q 想对所有的 i（$1 \le i \le n$）求出，s_i中有多少个互不相同的子串是合法括号串。
- 这个问题难倒了小 Q，他只好向你求助。设 s_i 共有 k_i 个不同子串是合法括号串，你只需要告诉小 Q 所有 $i \times k_i$ 的异或和，即： $(1 \times k_1)$ xor $(2 \times k_2)$ xor $(3 \times k_3)$ xor ⋯ xor $(n \times k_n)$
- 其中 xor是位异或运算。

括号树

- 输入格式
- 第一行一个整数 n，表示树的大小。
- 第二行一个长为 n的由 '(' 与 ')' 组成的括号串，第 i 个括号表示 i号结点上的括号。
- 第三行包含 n−1个整数，第 i（$1 \le i < n$）个整数表示 i+1 号结点的父亲编号 。
- 输出格式
- 仅一行一个整数表示答案。

括号树

- 这是2019年提高组的一道真题。
- 甚至难倒了一些很不错的选手。但其实本身算法并不复杂。
- 近年的许多真题都给出了较高的部分分。而且有些部分分特殊化为某种情况。很多时候是在启发大家思考的方向。
- 如本题$f_i=i-1$其实就是一条链，有55分的部分分。

括号树

- 我们先来分析链的情况。这还是一条非常友好的“链”，f_i=i-1
- 其实就是转化为序列上的问题。
- 先一重for 枚举 i：代表从根节点走到了 i 号节点。
  - 枚举左端点 l和右端点 r，表示枚举到区间为[l,r]的子括号序列
    - 对于每一个子序列，用栈来暴力判断其是否为合法括号序列，如是，则计数器加一。

回顾：括号匹配

- 描述：
- 给定一个只包含左右括号的合法括号序列，按右括号从左到右的顺序输出每一对配对的括号出现的位置(括号序列以0开始编号)。
- 输入：
- 仅一行，表示一个合法的括号序列。
- 输出：
- 设括号序列有n个右括号。则输出包括n行，每行两个整数l，r，表示配对的括号左括号出现在第l位，右括号出现在第r位。

回顾：括号匹配

- 样例输入：
- （（））（）
- 样例输出
- 1 2
- 0 3
- 4 5

回顾：括号匹配

- 对于每一个左括号，一定是越靠后出现的越先匹配。符合栈的性质。
- 考虑用栈来维护当前未匹配的左括号：
- 维护一个栈，从左到右扫序列，如果当前括号是左括号则将该位置加入栈中，如果是右括号，则该右括号与栈顶位置的左括号匹配，输出这对匹配括号的位置并删除栈顶的左括号。

括号树

- 时间复杂度为$O(n^4)$，可以拿到20分了。
- 我们发现其中还是存在大量的重复运算：
- 每次枚举1~i的子段，并判断每个子段是否为合法括号序列。
- 对于计数类问题，很常见的优化方式是不直接枚举+判断，而是处理出每个元素对答案的贡献。
- 或者对本题而言，就是处理出以每个括号结尾的合法子段是多少(w[i])。
- 这样，只要对w数组求一遍前缀和，sum[i]=w[1]+w[2]...+w[i]
- 最终的sum数组即为原题描述中要求的k_1，$k_2...k_n$

括号树

- 那么如何求出w[i]呢，我们来手玩几组样例看看：（注意，以下的例子第一个字符的下标均为1）
- 例子1：
- ()()()
- 我们发现，i=2 的时候，对答案的贡献值为 1。而 i=4的时候，本身 [3,4]就有一个满足要求的括号序列，在合并上前面的成为[1,4]，同样满足，于是对答案的贡献值就为2，再加上前面[1,2]本身有的括号序列，总共为 3。
- i=6时同理，总共的贡献值为 3，加上前面的有 3+3=6 种。其他位置均没有贡献。
- 换句话说，i为1−6时对答案的贡献分别为0,1,0,2,0,3，合并后的总答案为0,1,1,3,3,6

括号树

- 例子2:
- ())()
- 继续前面的思想，i=2时，对答案贡献1。而i=3时，由于不满足成匹配的括号序列，所以没有贡献。而i=5时，由于i=3多了一个后括号，[1,3]不匹配，导致[1,5]成不了一个匹配的括号序列。故对答案的贡献仍为 1
- i为1–5时对答案的贡献分别为0,1,0,0,1，合并后的总答案为0,1,1,1,2

括号树

- 例子3:
- ()(())
- 接着刚刚的分析，i=2时，贡献为1，而i=5时，由于i=3在中间断开，使[1,5]不能匹配，所以贡献仍为1。
- 当i=6情况有了变化。我们发现[1,2]是匹配的。故[1,2],[3,6]能合成一个匹配的序列，故对答案贡献为2。
- i为1−6时对答案的贡献分别为0,1,0,0,1,2，合并后的总答案为0,1,1,1,2,4

括号树

- 有没有发现什么规律？
- 我们发现，一个后括号如果能匹配一个前括号，假设这个前括号的前1位同样有一个已经匹配了的后括号，那么我们势必可以把当前的匹配和之前的匹配序列合并，当前的这个后括号的贡献值，其实就等于前面那个后括号的贡献值+1！
- 要想知道每个括号匹配的括号是谁只需要用栈维护即可。（刚才回顾的那道题）

括号树

```
//s是栈，top是栈顶。
if(c[i] == ')') //是后括号
{
    if(top == 0) continue; //栈为空，则没有匹配,w值为初值0
    int t = s[top]; //匹配的前括号的位置
    w[i] = w[t - 1] + 1; //结论计算贡献值
    top --;
}
else if(c[i] == '(') s[++ top] = i; //是前括号，就压入它的位置
sum[i] = sum[i - 1] +  w[i]; //计算总和
```

括号树

- 再把树上问题转化为序列上的问题就不难了：(只需要在dfs树的同时，维护一个“根到当前结点的链”对应的栈即可。)
- 而递推式也更改一下：
- w[i] = w[t - 1] + 1;//t为与第i个括号匹配的左括号位置
- 改为：w[i] = w[fa[t]] + 1;
- sum[i] = sum[i - 1] + w[i];
- 改为：sum[i] = sum[fa[i]] + w[i];
- 因为在“根到当前结点的链”中t的前一个结点不再是t-1而是fa[t]，同理i的前一个结点不再是i-1而是fa[i]。

括号树

- 那么如何在dfs树的同时，维护一个“根到当前结点的链”对应的栈呢？
- 很简单，利用我们刚才介绍到的回溯的原理。在遍历到一个新的括号结点时：
- 1.如果是左括号入栈。
- 2.如果是右括号并且栈非空，让栈顶元素出栈，并匹配，处理。
- 3.如果是右括号并且栈为空，不作任何处理。w值为默认的0.

括号树

- 而当遍历完这棵子树时，需要消除这个结点的影响。即如果第一次遍历到该节点时：
- 1.执行的是左括号入栈操作，则弹出当前栈顶元素
- 2.执行的是右括号匹配弹出栈顶元素的操作，那么把原来的栈顶元素“补”压回栈中

括号树

```
//参考程序，有根树
void dfs(int k){
        int tmp=0;
        if(s[k]=='(')
        {
                st[++top]=k;
                w[k]=0;//可省
        }
        else if(top){
                tmp=st[top--];
                w[k]=w[fa[tmp]]+1;
        } else {
                w[k]=0;//可省
        }
        sum[k]=sum[fa[k]]+w[k];//红色部分为处理该节点，统计该节点对答案的贡献。
        for(int i=first[k];i;i=nxt[i])
        dfs(to[i]);
        if(s[k]=='(') --top;
        else if(tmp) st[++top]=tmp;//紫色部分为遍历完k结点后"消除影响"的操作
}
```