bracket

对给定字符串 S 建立括号树，并且维护区间 $[l, r]$ 对应结点的儿子个数，代码如下：

```
void solve(int l, int r) {
    if(l > r)return ;
    int cur = r - 1, cnt = 0;
    while(cur > l) {
        solve(mat[cur], cur);
        cur = mat[cur] - 1; cnt++;
    }
    ans += 1ll * cnt * (cnt + 1) / 2;
    return ;
}
```

其中 $\text{mat}[cur]$ 表示 cur 位置的右括号所匹配的左括号，可以通过栈求出。

则合法括号子串由若干个在下标区间上连续的兄弟结点拼接而成，所以答案是 $\sum \binom{cnt+1}{2}$，直接累加进答案即可。

perm

观察下标在 x 的数的下标的变化。

- 对于操作一，变化相当于 $x \leftarrow x \oplus 2^{n-1}$。
- 对于操作二，变化相当于把 x 在二进制下循环移位。

那么考虑原排列的逆排列，显然原排列逆序对数等于逆排列逆序对数。那么操作相当于：

- 整体循环移位若干位。
- 选择任意二进制位整体异或 1。

枚举决策可以得到 $\mathcal{O}(4^n n)$ 的做法。

进一步观察，枚举循环移位之后，发现逆排列 p_i^{-1} 和 p_j^{-1} 的大小关系只和它们二进制下不同的最高位是否异或有关。因为最高位一定不同，且比最高位更高的位置一定相同。所以决策每一位是否异或是相互独立的。所以对于每一位判断是异或前还是异或后逆序对数更少即可，精细实现可以做到 $\mathcal{O}(2^n n^2)$。

book

暴力的做法是：枚举每层塌的顺序，如果书架的某层没有被压塌那么就在上面补球，复杂度 $\mathcal{O}(n! \cdot n)$。

注意到被压塌的书架构成若干个区间，而不同区间之间相互独立，联想到区间 DP。设 $f_{i,j,k}$ 表示压塌 $[i, j]$ 区间，最后落下来 k 个球，最少需要放多少球，枚举区间内最后一个被压塌的位置，设为 l，转移为：

$$f_{i,l-1,x} + f_{l+1,r,y} + \max(a_l - x, 0) \to f_{i,j,\frac{\max(x,a_l)}{2}+y}$$

分析复杂度，前两维的规模显然是 n，下面证明最后一维的规模为 $\max a_i$。等价于证明任意时刻同时落下来的球不超过 $\max a_i$ 个。归纳证明，对于 $n = 1$ 显然成立，对于 $n > 1$，第 n 层书架上最多放了 a_n 个球，下面最多落下来 $\max a_i$ 个球，所以如果第 n 层被压塌，那么最多落下来 $\frac{a_n + \max a_i}{2} \le \max a_i$ 个球，所以结论成立。

总复杂度为 $\mathcal{O}(n^3 \max a_i^2)$，注意实现的常数即可通过。

MST

下文设 $\mathrm{LCP}(i,j)$ 表示 $a[i\dots n]$ 和 $a[j\dots n]$ 的最长公共前缀长度，$\mathrm{LCS}(i,j)$ 表示 $a[1\dots i]$ 和 $a[1\dots j]$ 的最长公共后缀长度。

求最小生成森林，考虑 kruskal 的过程，将所有 k 按照 w_k 排序，一次考虑所有长度为 k 的重复串带来的边。

注意到本题连边的要求是出现重复串，考虑仿照 NOI2016 优秀的拆分 一题的套路，枚举重复串的长度 k，在序列上将 $k,2k,\dots,\lfloor\frac{n}{k}\rfloor k$ 标为关键点，显然一对重复串会经过恰好两个相邻的关键点。

枚举相邻的关键点 ik 和 $(i+1)k$，设 $P=\min\{k,\mathrm{LCP}(ik,(i+1)k)\},S=\min\{k-1,\mathrm{LCS}(ik-1,(i+1)k-1)\}$。如果 $P+S\ge k$，那么对于 $l\in[i-S,i+P-k]$，都有 $a[l\dots l+k-1]=a[l+k\dots l+2k-1]$。也就是说，对于 $\forall t\in[i-S,i+P-1]$，t 和 $t+k$ 之间都有一条权值为 w_k 的边。

暴力将这些边加入并查集即可做到 $O(n^2)$ 的复杂度。

考虑优化 kruskal 中合并连通块的过程，事实上，真正有效的合并只有最多 $O(n)$ 次，但我们加入 $O(n^2)$ 条边时，每条都需要判断一遍，浪费了很多时间。

仿照 [SCOI2016] 萌萌哒 的套路。注意到我们的操作相当于对于两个区间 $[l_1,r_1],[l_2,r_2]$，每一个位置对应连边。考虑区间长度为 2^c 的情况，此时我们对每个 c 建立一个并查集 dsu_c，如果在 dsu_c 中 (x,y) 属于一个连通块，就代表对于任意 $i\in[0,2^c)$，在原并查集中 $x+i$ 与 $y+i$ 属于同一个连通块。那么我们一次合并时就可以快速判断当前 $[l_1,r_1],[l_2,r_2]$ 还有没有必要连边（如果在 dsu_c 中 l_1,l_2 在同一个块，显然没有必要再连边）。如果还需再连边，那么在 dsu_c 中将 l_1,l_2 合并，并将一个区间拆分为两个 2^{c-1} 的区间递归下去连边即可。

这部分操作的核心代码如下：

```
inline void Union(int x,int y,int dep) { // [x,x + 2^{dep}) , [y,y + 2^{dep})
    if(S[dep].Same(x,y)) return; // 如果这个区间已经没有必要连边
    if(dep == 0) {
        if(!S0.Same(x,y)) ans += w[y - x],S0.Merge(x,y); // 如果 x 和 y 在原并查集中不在一个连通块里
        return;
    }
    S[dep].Merge(x,y);
    Union(x,y,dep - 1);
    if(y + (1 << dep - 1) <= n) Union(x + (1 << dep - 1),y + (1 << dep - 1),dep - 1);
}
```

这样，所有连边操作的时间复杂度就是 $O(n\log n\alpha(n))$ 的了，因为 Union 函数每发生一次递归，对应的并查集的连通块个数就会减少 1。而最开始所有并查集的连通块个数之和为 $O(n\log n)$，所以总共只会递归 $O(n\log n)$ 次。

接下来考虑区间长度不是 2^c 的情况。仿照 ST 表的处理方法，设 $c=\lfloor\log_2(r_1-l_1+1)\rfloor$，将 $[l_1,r_1]$ 拆分成两个长度为 2^c 的区间 $[l_1,l_1+2^c-1]$ 和 $[r_1-2^c+1,r_1]$，$[l_2,r_2]$ 同理。对拆出来的两对区间分别进行 Union 即可。

上文略去了如何快速求出任意两个后缀的 LCP，这是 SA/SAM 等字符串算法的经典应用，这里不再赘述。