Boolean Function

题目大意

给你一个逻辑表达式，包含A, B, C, D这4个变量和&, |这2个运算符。现在这个逻辑表达式的一些变量和运算符看不清了。

已知在n种不同的A, B, C, D取值下，表达式的值，求有多少可能的表达式。

数据范围

表达式长度≤ 500

$n \leq 16$

解题思路

考虑到$n \leq 16$，将一个表达式每一种情况下当前的答案是0还是1记为一个$g(s)$,$g(s)$是一个n位的二进制数，（$0 \leq g(s) < 2^n$，s为一个表达式）。

显然的，$g((s1)|(s2)) = g(s1)|g(s2)$，$g(((s1)\&(s2)) = g(s1)\&g(s2)$。

接着考虑题目，一个表达式可以被建为一个二叉树，其节点个数为$\frac{|s|}{6}$级别的。转换为树后开始考虑树形DP，记录$dp[v][x]$，表示所有可以被以v为根的子树代表的表达式中（即通过把'?'变为确定符号得到），有多少表达式s的$g(s) = x$。该DP的转移为$dp[v][x|y] += dp[ls][x] * dp[rs][y]$或$dp[v][x\&y] += dp[ls][x] * dp[rs][y]$，其中$ls$为左儿子，$rs$为右儿子。

现在的时间复杂度为$O(|s| * 4^n)$，无法通过题目。

观察时间复杂度的瓶颈——DP转移。可以发现该DP转移的形式是一个或卷积或者与卷积的形式，可以使用fwt进行优化。

```
void fwt_and(int*f,int tag){
    for(int mid=1;mid<1<<n;mid<<=1)
        for(int l=0;l<1<<n;l+=mid<<1)
            for(int i=l;i<l+mid;i++){
                f[i]=(f[i]+f[i+mid]*tag)%mod;
            }
}
void fwt_or(int*f,int tag){
    for(int mid=1;mid<1<<n;mid<<=1)
        for(int l=0;l<1<<n;l+=mid<<1)
            for(int i=l;i<l+mid;i++){
                f[i+mid]=(f[i]*tag+f[i+mid])%mod;
            }
}
```

此时，时间复杂度被优化为$O(|s| * 2^n * n)$，可以通过题目。

Pairing Points

题目大意

在一个圆上顺序排列着$2N$个点，保证任意3个点对的连线不交于一点。求有多少中将$2N$个点分成N个点对的方法，使得N个点对间的连线连通且无环，并且有一些点对不允许出现。

数据范围

$1 \leq N \leq 20$

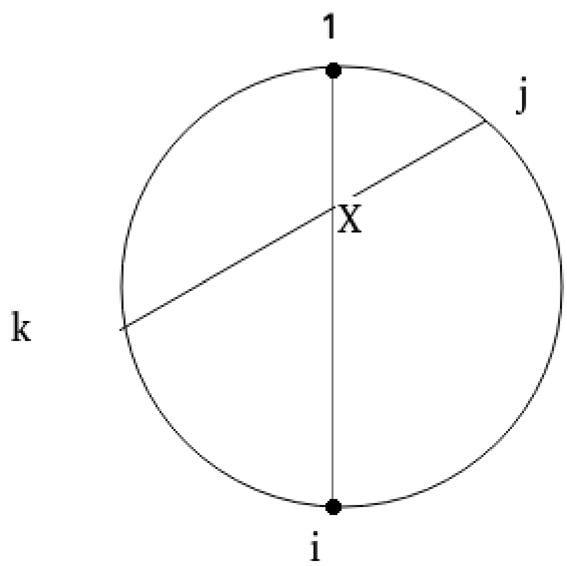
解题过程

令$n = 2N$，即总点数。

先考虑1号点的对应点，假设为i。现在问题变成了在$[2,i) \cup (i,n]$中连一些点对，至少有一组点对(P,Q)满足$P \in [2,i), Q \in (i,n]$并且所有$(P_1,Q_1),(P_2,Q_2), P_1,P_2 \in [2,i), Q_1,Q_2 \in (i,n]$，满足$[P_1 < P_2] == [Q_1 < Q_2]$。我们现在用$F(2,i,n)$来表示上述问题。

考虑如何解决这个问题:

因为至少有一个点对横跨两边。所有考虑枚举横跨两边的点对(P,Q)中，P最小的点对，设该点对为(j,k)。



可以发现，在(j,i)中，有一部分点会和线段（i,X）连通，有一部分点会和线段(j,X)连通，并且它们存在一个分界点p。

同样的，在(i,k)中有一部分点会和点i连通，有一部分点会和点k连通，并且它们存在一个分界点q。

由于$[2,j)$和$(j,p]$都不会连出与$(1,i)$相交的点，所以区间$[2,j) \cup (j,p]$的问题为$F(2,j,p)$，由于$[q,k)$和$(k,n]$都不会连出与$(1,i)$相交的点，所以区间$[q,k) \cup (k,n]$的问题为$F(q,k,n)$。

再来考虑$[p+1,i)$和$(i,q-1]$，它们不会和(j,k)相交，所以这个问题也是$F(p+1,i,q-1)$。

考虑记忆化搜索，状态数为$O(n^3)$，转移为$O(n^4)$，总复杂度$O(n^7)$。现在的时间复杂度已经通过这道题了，但是比较慢，考虑优化这个算法。

因为记忆化搜索不太好优化，我们先把记忆化搜索转化为DP。现在我们的状态为(l,i,r)，转移时枚举了(j,k,p,q)。因为DP转移依赖性的必要，先按$r-l+1$的大小枚举(l,r)。考虑优化DP，转移时枚举(p,q)，因为原DP转移是$F(l,i,r)+=F(l,j,p)*F(q,k,r)*F(p+1,i,q-1)$，转移条件为$l\leq j\leq p<i<q\leq k\leq r$，$A_{j,k}=1$，所以考虑记$Fs(l,x,r)$表示$\sum_{A_{x,i}=1}F(l,i,r)$,计算$S=\sum F(l,j,p)*F(q,k,r)=\sum Fs(l,k,p)*F(q,k,r)$。计算时只需枚举$k$。

对于DP转移的另一项$F(p+1,i,q-1)$，因为p,q已经枚举了，可以直接枚举i，并将$F(p+1,i,q-1)\times S$转移到$F(l,i,r)$。每一轮转移完后更新Fs。

此时的复杂度为$O(n^5)$。

Addition and Andition

题目大意

给你两个正整数X,Y，进行K次以下操作：

- 令Z=X and Y，and为位运算与。
- X+=Z,Y+=Z

输出最终的X,Y。

数据范围

X,Y二进制表示下长度不超过10^6。

$1 \le K \le 10^6$

解题过程

定义S_i表示X二进制表示下从低到高第i位的值，T_i表示Y二进制表示下从低到高第i位的值。

定义一次对i进行的**进位**为：

```
S[i]=T[i]=2
while S[i]==2||T[i]==2 {
    if S[i]==2
    then
        S[i]=0,S[i+1]+=1
    endif
    if T[i]==2
    then
        T[i]=0,T[i+1]+=1
    endif
    i+=1
}
```

定义一次**加法**为

```
for(i=max(|S|,|T|);i>=0;i-=1){
    if S[i]==1&&T[i]==1
    then
        对i进行进位操作
    endif
}
```

现在要做的就是进行K次加法操作。

现在我们考虑交换循环顺序：从大到小枚举每一个$S_i = T_i = 1$的i，然后重复K次以下操作：

找到最小的$j \geq i$，使得$S_j = T_j = 1$，对j进行进位操作，若不存在这样的j，则停止操作。

由于在第i个位置一次进位的增量大于所有$j < i$的j进位产生的增量之和，所以交换循环顺序后的结果不变。

用x表示进位的情况，now表示当前还能操作几次。初始是先令$S_i = T_i = 0, x = 11, now = K$

对于$j = i, i+1, ..$进行以下操作:

如果$(S_j, T_j, x) = (0, 0, 10)$，那么使$(S_j, T_j) = (1, 0)$，结束操作。
如果$(S_j, T_j, x) = (0, 0, 01)$，那么使$(S_j, T_j) = (0, 1)$，结束操作。
如果$(S_j, T_j, x) = (0, 0, 11)$，那么使$(S_j, T_j, x) = (0, 0, 11)$，$now \to now - 1$，如果$now < 0$，结束操作。
如果$(S_j, T_j, x) = (1, 0, 11)$，那么使$(S_j, T_j, x) = (0, 1, 10)$。
如果$(S_j, T_j, x) = (1, 0, 10)$，那么使$(S_j, T_j, x) = (0, 0, 10)$。
如果$(S_j, T_j, x) = (1, 0, 01)$，那么使$(S_j, T_j, x) = (0, 0, 11)$，$now \to now - 1$，如果$now < 0$，结束操作。

$(S_j, T_j) = (0, 1)$和$(S_j, T_j) = (1, 0)$的情况类似。

因为可以交换循环顺序，所以并不会遇到$(S_j, T_j) = (1, 1)$。

可以发现，如果有连续一段$(S_j, T_j) = (0, 0)$，可以快速处理。所以考虑使用一个栈记录所有$(S_j, T_j) \neq (0, 0)$的位置，j在栈中从顶到底递增。当我们找到一个$(S_i, T_i) = (1, 1)$时，弹出栈顶，快速处理当前位置到栈顶所在位置间的$(0, 0)$，如果能处理到栈顶的位置，就把他暴力处理掉，并把中途新增的$(S_j, T_j) \neq (0, 0)$的位置记录在一个临时数组中，重复执行上述操作。执行完后把临时数组中$(S_j, T_j) \neq (0, 0)$的位置加进栈中。

撇开$(0, 0)$,$x = 11$时,X, Y中1的数量不增，$x \neq 11$时,X, Y中1的数量减少，并且如果$x = 11$，那么一次操作后x一定不等于11,而当$(S_j, T_j) = (0, 0)$时，可以快速处理。所以时间复杂度为$O(N + M + K)$。