云某人的博弈

打表的艺术

概念： 博弈论是研究具有斗争或竞争性性质现象的数学理论和方法。

假设玩家的集合是 N，第 i 个玩家的策略集合是 $\sum^i$，游戏的结果集合为 Γ，关于每个玩家使用不同的策略，游戏产生对应的结果，及
$\pi : \prod_{i \in n}(\sum^i) \to \Gamma$

组合游戏

> 博弈的子集

特征

- 两个玩家
- 一个状态集合
- 游戏规则是指明玩家在一个状态可以转移到其他状态

- 玩家轮流进行移动
- 如果当前处于某个状态，玩家根据规则无法移动，游戏结束。
- （**大部分时候**）无论玩家如何选择，游戏都会在有限步内结束。

可以将组合游戏理解为有向无环图

Bash游戏

在 N 堆石子中轮流取 $[1, M]$ 个，谁不能取输。

> 结论：当 $N = k \cdot (M + 1)$ A输，否则A赢

状态集合：

$S = 0, 1, 2, \ldots, N$，$i \rightarrow$ 当前剩 i 个石子

PN态

P状态：

- privious position
- 走到该状态的玩家胜利（离开失败）
- **不能**走到 **P状态** 的为P状态

N状态：

- nest position
- 离开该状态的玩家胜利
- **可以**走到 **P状态** 的为 N状态

根据结束状态（无法走到其他状态位置）为 P 或 N来反向转移

（普适）解题方法

- 打表，找规律（DP，搜索）
- `guess`，猜结论
- 证明（理由PN态）

Nim游戏

有 N 堆石子，第 i 堆有 a_i 个。轮流取，每次取**一堆**中的**至少一个**，不能取的失败。

P-N分析

> 默认 $x_1 \geq x_2 \geq x_3 \geq \cdots \geq x_n$

n	x	P-N
0		P
1	x	N
2	$x_1 = x_2$ $x_1 \neq x_2$	P N

n	x	P-N
3		

结论

- $x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_n = 0$ **P态**
- $x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_n > 0$ **N态**

推导
对于 $x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_n = 0$，一次操作后不可能保持 $= 0$，必然变为 > 0。
而对于 $x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_n > 0$，只需让存有最高位 1 的数，减去其他1组成的数，异或和就将变为 0

SG函数

图游戏

图游戏是一种组合游戏的抽象描述
给定一个有向图 $G = (V, E)$，V 是非空点集，E 是有向边集。两个人在图 G 上进行游戏。从起始节点 x_0 出发，轮流移动。每一轮当前玩家可以从当前节点 x 按照邮箱边集移动到下一个节点，把 x 可以一步移动到的节点集合记为 $F(x)$。如果 $F(x)$ 为空，则 x 为终止结点，此时玩家不能移动，游戏结束。

> 只讨论无环，且在有限次内终止的情况（否则无意义）

SG函数

定义有向无环图 $G = (V, E)$ 的SG函数是一个结点集 V 到非负整数 $Z \geq 0$ 的映射。

$g : V \to Z_{\geq 0}$，满足　$g(x) = mex\{g(x) | y \in F(x)\}$

SG函数与PN关系

- x 为P态当且仅当 $g(x) = 0$
- x 为N态当且仅当 $g(x) > 0$

组合游戏的和

给定多个组合游戏，可以用如下方式把他们进行组合。每个游戏都有一个初始状态，每次当前玩家可以选择一个游戏，并按照该游戏的规则移动一次，如果轮到当前玩家时，所有游戏都不能移动，则玩家输（正常规则）。称这个新的组合游戏为这些

> nim游戏就可以看成若干个游戏的和

图游戏的和

假设给定 n 个邮箱无换图 $G_i(V_i, E_i)$，则它们的图游戏的和 $G(V, E)$，满足

- $V = V_1 \times V_2 \times V_3 \times \ldots \times V_n$

> 笛卡尔乘积是指在数学中，两个集合 X 和 Y 的笛卡尔积（Cartesian product），又称直积，表示为 $X \times Y$，第一个对象是 X 的成员而第二个对象是 Y 的所有可能有序对的其中一个成员

- 结点 $x=(x_1,x_2,\ldots,x_n)$ 一步能到 $F(x)=U_{i=1}^{n}(x_1,x_2,\ldots,y_i,\ldots x_n)|y_i\in F_i(x_i)$

  > x_i 走向 y_i。

记 G 为 $G_1+G_2+\ldots+G_n$。

SG定理

假设有 n 个组合游戏 $G_1,G_2,\ldots,G_n$，记 g_i 为第 $i(1\le i\le n)$ 个组合游戏的SG函数，则它们的和 $G=G_1+G_2+\ldots+G_N$的SG函数。
$g(x)=g((x_1,x_2,\ldots))=g(x_1)\oplus g(x_2)\oplus\ldots g_n(x_n)$

推论

假设 $G=G_1+G_2+\ldots+G_n$

- $x=(x_1,x_2,\ldots,x_n)$ 是 P 态当且仅当 $g(x_1)\oplus g(x_2)\oplus\ldots g_n(x_n)=0$
- $x=(x_1,x_2,\ldots,x_n)$ 是 N 态当且仅当 $g(x_1)\oplus g(x_2)\oplus\ldots g_n(x_n)>0$

sg的相较比较PN态，优点在于： **可以求多个游戏的组合**

code

```
int get_sg(int p)// 记忆化搜索求sg
{
    // if(p == 0)   return 0;
    if(sg[p] != -1) return sg[p];
    unordered_set<int> st;
    for(auto &i : a[p]) st.insert(get_sg(i));
    int mex = 0;
    while(true)
    {
        if(!st.count(mex))  return (sg[p] = mex);
        mex ++;
    }
}
```

题目

组合游戏转移

递归转移求最值

题面
设计师们想出了一个名为“Rake It In”的新简单游戏。两个玩家，Alice和Bob，最初选择一个整数 k 并初始化一个分数指示器。创建一个 4×4 的棋盘，棋盘上有 16 个值。从玩家 Alice 开始，一轮中的每个玩家选择棋盘的 2×2 区域，将该区域中的值的总和添加到得分指示器，然后将这四个值逆时针旋转 90 度。

在总共 $2k$ 轮之后，每个玩家做了 k 次决定。 Alice的最终目标是最大化最终得分。然而对于 Bob 来说，他的目标是最小化最终得分。
sloution
对于 $Alice$

- 需要保证答案最大

- 每步选择最大矩形，不能保证答案最大（不能贪心）
- 所以现需要保证每次操作后，后续操作尽可能**不小**,也就是尽可能大
  Bob同理

如果 $k = 1$**:**
A 会给 B 传入 9 种矩形，且加上小矩形的和

然后 B 会从**每个矩形**中选择一个最小值返回

A 最终会选择 小矩形和 $+$ B返回值最大的结果

如果 $k \geq 2$**:**
那么就是重复多次上述步骤，只是每次 B 会选择小矩形 $+$ A返回值 最小的结果

```
int k;
int a[5][5];

int get(int x,int y)
{
    return a[x][y] + a[x+1][y] + a[x][y+1] + a[x+1][y+1];
}

void move(int x,int y)
{
    int tmp = a[x][y];
    a[x][y] = a[x][y+1];
    a[x][y+1] = a[x+1][y+1];
    a[x+1][y+1] = a[x+1][y];
    a[x+1][y] = tmp;
}

void rmove(int x,int y)
{
    int tmp = a[x][y];
    a[x][y] = a[x+1][y];
    a[x+1][y] = a[x+1][y+1];
    a[x+1][y+1] = a[x][y+1];
    a[x][y+1] = tmp;
}

int dfs(int p)
{
    if(p > k*2)
    {
        return 0;
    }
    int res = (p&1 ? 0 : M);
    for(int i=1;i<4;++i)
    {
        for(int j=1;j<4;++j)
        {
            int tmp = get(i,j);
            move(i,j);
            res = (p&1 ? max(res,tmp+dfs(p+1)) : min(res,tmp+dfs(p+1)));
            rmove(i,j);
        }
```

```
    }
    return res;
}

void func(void)
{
    cin >> k;
    for(int i=1;i<=4;++i)
    {
        for(int j=1;j<=4;++j)   cin >> a[i][j];
    }

    cout << dfs(1) << '\n';
}
```

枚举转移

题面
一串由0~9组成的数字，可以进行两个操作：
1、把其中一个数变为**比它小**的数；
2、把其中一个数字0及其右边的所以数字删除。

两人轮流进行操作，最后把所以数字删除的人获胜，问前者胜还是后者胜。
sloution
有 $\sum_{i=1}^{6}(10^i)$ 种数字，若是使用 `dfs` 或者说递归，每个状态都有$1 \sim 9$条分支，且还是字符串形式，很容易 `TLE`。

但若是使用**递推**，只需要 $\sum_{i=1}^{6}(10^i)$ 次。

字符串的数据较多，所以使用数字存储

> 可能不需要，之前看错题导致出现很多错误
> 但为了数字进位方便，还是用数字存储

1. **空字符串**为 P态
2. 数字无法存储前导0，且 0XXX 与 XXX 的结果不同
3. 0XXX 可以一步到空字符串，也就是 N态，所以对其特判
4. 其他数字按照组合游戏方法判断 PN 态。

设 P 态为 1， N 态为 0。
因为总状态 $\leq 10^8$打表计算后输出即可。

code

```
vector<int> a(N+10);
void func(void);

signed main(void)
{
    Start;
    int _ = 1;
    int z = 0;
    a[0] = 0;
    while(z <= N)
    {
```

```
        int op = 1, res = 0;
        while(z/op >= 10)
        {
            if((z/op) % 10)
            {
                for(int i=1;i<=(z/op) % 10;++i) res |= a[z-op*i];
            }
            else    res |= a[z/(op*10)];
            op *= 10;
        }
        res |= 0;// 不必要，代表0xxx
        for(int i=1;i<z/op;++i) res |= a[z-op*i];// 首位变成0 为 0xxx，而非xxx，所以
去除掉
        a[z] = !res;
        z ++;
    }
    cin >> _;
    while(_--)  func();
    return 0;
}

void func(void)
{
    string st;
    cin >> st;
    if(st[0] == '0')    cout << "Yes\n";
    else    cout << (a[stoi(st)] ? "No" : "Yes") << '\n';

}
```

dp打表

题面
有一个01串。两个人在一起做游戏，ALICE先手，Bob后手，他们可以从以下两种操作中选一个：

1. 任选某个为0的位置将其变为1，代价为1。
2. 若该01串目前不为回文串，且上一个操作不为操作2的情况下可以将整个01串翻转。

代价少的人获胜，否则平局。
sloution
根据题意可知，字符的位置不重要，对称的关系才需要关注。

且如果对称位置为 `11`，那我就不需要再考虑了，因为无法再被修改。

所以可以可以记录每个状态为变为 (`00`,`11`,`mid`,`r_ed`)
分别是 `00` 对的数目， `01` 对的数目, 正中字符是 `1` 或 `0`，是否翻转过。

存储代价的最小值。因为值越小越可能赢

转移则看下列代码

对于每个状态，可以由至多四种可能转移而来
code

```
int dp[N][N][2][2];
```

```
signed main(void)
{
    Start;
    int _ = 1;
    memset(dp,0x3f,sizeof dp);
    dp[0][0][0][1] = 0;
    dp[0][0][0][0] = 0;

    for(int i=0;i<N;++i)
    {
        for(int j=0;j<N;++j)
        {
            for(int k=0;k<=1;++k)
            {
                for(int l=1;l>=0;--l)
                {
                    if(i>0) dp[i][j][k][l] = min(dp[i][j][k][l],1-dp[i-1][j+1][k]
[0]);
                    if(j>0) dp[i][j][k][l] = min(dp[i][j][k][l],1-dp[i][j-1][k]
[0]);
                    if(k==1) dp[i][j][k][l] = min(dp[i][j][k][l],1-dp[i][j][0]
[0]);
                    if(l==0 && j>0) dp[i][j][k][l] = min(dp[i][j][k][l],-dp[i][j]
[k][1]);
                }
            }

        }
    }
    cin >> _;
    while(_--)  func();
    return 0;
}

void func(void)
{
    int n;
    string st;
    cin >> n >> st;
    vector<int> a(3);
    for(int i=0;i<(n>>1);++i)
    {
        a[0] += (st[i] == '0' && st[n-i-1] == '0');
        a[1] += st[i] != st[n-i-1];
    }
    a[2] = (n&1 && st[n/2] == '0');
    int z = dp[a[0]][a[1]][a[2]][0];
    cout << (z < 0 ? "ALICE\n" : (z == 0 ? "DRAW\n" : "BOB\n"));
}
```

Nim变体

题面
共有N堆石子，已知每堆中石子的数量，两个人轮流取石子，每次只能选择N堆石子中的一堆取一定数量的石子（最少取一个），取过子之后，还可以将该堆石子中剩余的石子随意选取几个放到其它的任意一堆或几堆上。等哪个人无法取子时就表示此人输掉了游戏。注意：一堆石子没有子之后，就不能再往此处放石子了。

假设每次都是小牛先取石子，并且游戏双方都绝对聪明，现在给你石子的堆数、每堆石子的数量，请判断出小牛能否获胜。
sloution

n	x	P-N
0	Null	P
1	x	N
2	$x_1 = x_2$ $x_1 > x_2$	P N
3	$x_1 \geq x_2 \geq x_3$	N
$\vdots$		
$2 \cdot k - 1$	$x_1 \geq x_2 \dots x_n$	N
$2 \cdot$	$x_1 = x_2, x_3 = x_4$ other	P N

> 详细推导就不写了

对于 $2 \cdot k - 1$
保证一次变成 $2 \cdot k$ 且 $x_1 = x_2 \dots$
需要的代价是$x_2 - x_3 + x_4 - x_5 + \dots + x_{n-1} - x_n$
$\because -x_3 + x_4 \leq 0, -x_5 + x_6 \leq 0 \dots$
且$x_2 \leq x_1, -x_n \leq 0$

$\therefore$ 代价 $\leq x_1$
code

```
void func(void)
{
    int n;
    while(true)
    {
        cin >> n;
        if(!n)  break;
        vector<int> a(n+1);
        for(int i=1;i<=n;++i)   cin >> a[i];
        if(n&1)
        {
            cout << "Win\n";
            continue;
        }
```

```
        sort(a.begin()+1,a.end());
        bool op = true;
        for(int i=1;i<=n;i+=2)
        {
            if(a[i] != a[i+1])  op = false;
        }
        cout << (op ? "Lose\n" : "Win\n");
    }
}
```

SG函数

模板题

题面

有一个 n 个节点的有向无环图，编号为 0 到 $n-1$。有若干轮游戏，每轮游戏给出 m 个棋子在图上的节点位置（可能存在多个棋子在同一个节点上）。两个玩家轮流移动棋子，每次可以选择一个棋子将其移动到它的后继节点。无法移动任何棋子的一方判输。假设双方都采取最优策略，如果先手获胜输出 `"WIN"`，否则输出 `"LOSE"`。

sloution

经典SG

使用 `记忆化搜索` 求出各点sg函数后，对每个回答求点的 $\oplus$ 值即可。

code

```
int n;
vector<int> a[N];
vector<int> sg(N,-1);

int get_sg(int p)// 记忆化搜索求sg
{
    // if(p == 0)   return 0;
    if(sg[p] != -1) return sg[p];
    unordered_set<int> st;
    for(auto &i : a[p]) st.insert(get_sg(i));
    int mex = 0;
    while(true)
    {
        if(!st.count(mex))  return (sg[p] = mex);
        mex ++;
    }
}

signed main(void)
{
    Start;
    cin >> n;
    for(int i=0;i<n;++i)
    {
        int k,x;    cin >> k;
        // if(k == 0)   sg[i] = 0;
        for(int j=0;j<k;++j)
        {
            cin >> x;
            a[i].push_back(x);
```

```
        }
    }
    for(int i=0;i<n;++i)    get_sg(i);
    // get_sg(0);
    int m,tmp;
    while(cin >> m && m)
    {
        int ans = 0;
        while(m --)
        {
            cin >> tmp;
            ans ^= sg[tmp];
        }
        cout << (ans ? "WIN" : "LOSE") << '\n';
    }
    return 0;
}
```

使用规律优化sg速度（减少递归）

题面

有 n 个箱子，第 i 个箱子的容量为 s_i，初始时装有 c_i 颗石子。两名玩家轮流往箱子里放石子，每次放入的石子数必须满足：

放入的数量不超过当前箱子内石子数的平方（例如，若某箱子当前有 3 颗石子，则可放入 1 到 9 颗石子）。每次操作可以选择任意一个未满的箱子放入石子，直到某一方无法进行任何合法操作时判输。假设双方均采取最优策略，判断先手是否能获胜。若能获胜输出 `"YES"`，否则输出 `"NO"`。

sloution

到最后可以一步走到的点，可以直接得出其 g函数，而不需要再求 mex

code

```
vector<int> s(65), c(65),g(N,-1);
// vector<vector<int>> g(65,vector<int>(N,-1));
int get_sg(int k,int p)
{
    if(p == s[k])   return (g[p] = 0);
    if(g[p] != -1)  return g[p];
    if(p*(p+1) >= s[k]) return s[k]-p;
    unordered_set<int> st;
    for(int i=1;p+i<=s[k] && i<=p*p;++i)    st.insert(get_sg(k,p+i));
    int mex = 0;
    while(true)
    {
        if(!st.count(mex))  return (g[p] = mex);
        mex ++;
    }
}

void func(void)
{
    int n;  cin >> n;
    int ans = 0;
    for(int i=1;i<=n;++i)
    {
        cin >> s[i] >> c[i];
```

```
        for(int j=c[i];j<=s[i];++j) g[j] = -1;
        ans ^= get_sg(i,c[i]);
    }
    cout << (ans ? "Yes\n" : "No\n");
}
```

分裂游戏（sg打表找规律）

题面

Alice 和 Bob 厌倦了标准规则下的 Nim 游戏，因此他们修改了规则：允许玩家在每次操作时选择**移除物品**或**将一堆分成两小堆**。具体规则如下：

1. **移除物品**：从任意一堆中移除至少一个物品（规则与传统 Nim 相同）。
2. **分割堆**：选择一堆，将其分成两堆较小的堆（堆的大小必须为正整数）。
   游戏以正规则进行，即执行最后一次有效操作（移除最后一个物品或完成最后一次分割）的玩家获胜。
   sloution
   n **很大时，sg函数只能作为打表的辅助，不能直接求解。**
   对于一组**分裂游戏**，虽然其可以分裂为多个子游戏，但是可以对其子游戏也求 Nim和，并作为更大单个游戏的子游戏，最后任可用sg函数求解。
   code

```
int g[N];

int get_sg(int p)
{
    if(g[p] != -1)  return g[p];
    unordered_set<int> st;
    for(int i=0;i<p;++i)    st.insert(get_sg(i)); // 取出
    for(int i=1;i<p;++i)    st.insert(g[i]^g[p-i]);// 分裂
    int mex = 0;
    while(true)
    {
        if(!st.count(mex))  return (g[p] = mex);
        mex ++;
    }
}

void func(void);

signed main(void)
{
    Start;
    int _ = 1;
    g[0] = 0;
    // 调用sg打表
    // for(int i=1;i<N;++i) g[i] = -1;
    // for(int i=1;i<N;++i) get_sg(i);
    cin >> _;
    while(_--)  func();
    return 0;
}

void func(void)
```

```
{
    int n,x,ans = 0;    cin >> n;
    while(n --)
    {
        cin >> x;
        if(x%4 == 0)    x --;
        else if(x%4 == 3)   x ++;
        ans ^= x;
    }
    cout << (ans ? "Alice\n" : "Bob\n");
}
```