算法基础（三十一）： 数学基础 - 博弈论 -SG函数

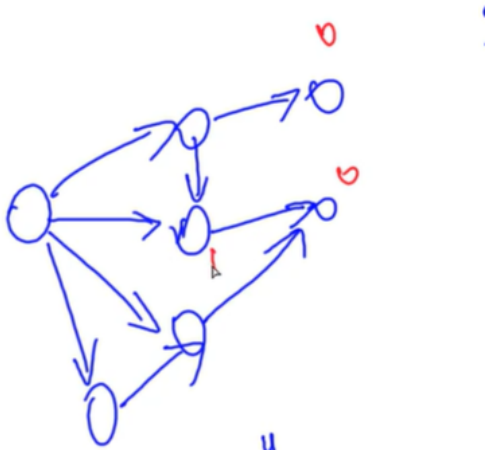
首先，我们先定义一个`mex`运算：

`mex`运算是一个跟集合有关的运算，这个运算是为了找到一个集合中最小的不存在的自然数，比如集合`mex({1, 2, 3}) = 0`

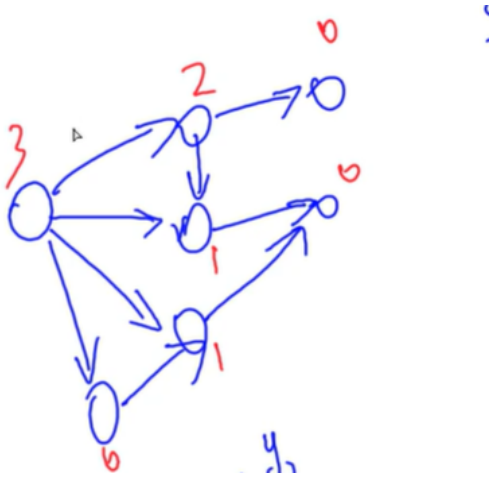
然后是`SG()`函数的定义：

在有向图游戏中,对于每个节点`x`,设从`x`出发共有`k`条有向边,分别到达节点`y1,y2,····yk`,定义`SG(x) = mex{SG(y1), SG(y2)......SG(yk)}`，也就是当前节点的`SG()`函数是后续结点的所有`SG()`函数然后做`mex`运算，特别地,整个有向图游戏`G`的`SG`函数值被定义为有向图游戏起点`s`的`SG`函数值,即`SG(G)=SG(s)`

举一个例子，假如有这么一个有向图游戏：



最后两个结点没有出边是为终结态，定义`SG(终点) = 0`那么从后往前经过计算，这些点到不了的最小自然数值，得到这些点的`SG()`函数的值为：

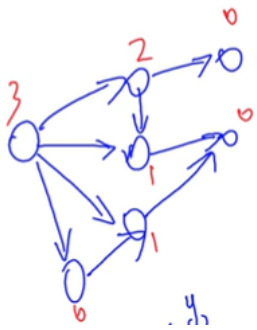


那么我们再看看这个石子游戏：

给定 n 堆石子，两位玩家轮流操作，每次操作可以从任意一堆石子中拿走任意数量的石子（可以拿完，但不能不拿），最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

当从这`n`堆石子中拿走一些石子的时候就到达了下一个状态，所以这些状态就可以构成一个游戏状态图，我们定义`SG(终结) = 0`：

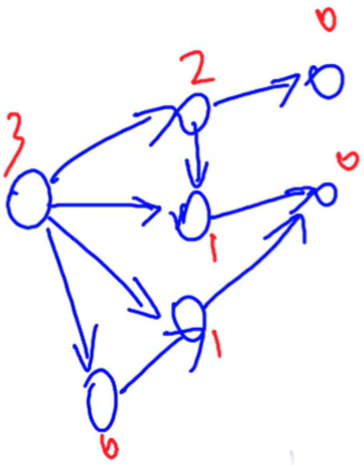


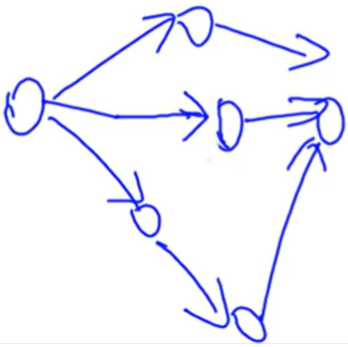
对于其中一个一个状态`x`，如果`SG(x) = 0`，就说明此时状态是一个必败状态，如果此时`SG(x) != 0`就说明此时的状态是一个必胜状态

> 因为对于`SG(x) = 0`的状态，说明其后续状态中不存在`SG() = 0`的状态，而对于`SG(x) != 0`的状态，其后续的状态中一定存在一个`SG`值为`0`的状态，所以当先手到达`SG(x) != 0`的状态时，他一定可以走到`SG(x) = 0`的后续状态，此时后手只能继续走到一个`SG(x) != 0`的状态，从而一步步走向终结状态，这个状态的`SG`值一定是`0`，所以面对的一定是后手，所以后手必输，从而`SG(x) != 0`这个状态一定是必胜状态

对于多个游戏状态图的情况：

对于`SG`函数来说其使用最多的地方是多个图游戏的时候，也就是说有`n`个图游戏，每个游戏的走法都是一个状态图，每个图游戏目前遇到的状态`SG`函数是`SG(x1), SG(x2) .....SG(xn)`，当所有的游戏状态图都到达终点状态的时候，所有游戏都无路可走，这时游戏结束，面对这个状态的人表示失败。





那么跟前面的`nim`游戏一样，若先手遇到的是$SG(x1) \oplus SG(x2) \oplus SG(x3) ..... SG(xn) \neq 0$这时先手必胜，反之若先手遇到的状态是$SG(x1) \oplus SG(x2) \oplus SG(x3) ..... SG(xn) = 0$这时先手必败

- 对于所有游戏最后的终结状态，这时所有游戏的局面`SG`函数有
  $SG(x1) \oplus SG(x2) \oplus SG(x3) ..... SG(xn) = 0$
- 对于先手，若遇到的所有游戏局面为$SG(x1) \oplus SG(x2) \oplus SG(x3) ..... SG(xn) = x \neq 0$，那么他一定可以通过操作某个图游戏的状态使得
  $SG(x1) \oplus SG(x2) \oplus SG(x3) ..... SG(xn) = 0$

> 假设对于某个游戏，此时的状态是`xk`，`SG`函数是`SG(xk)`，那么此时一定有`x ^ SG(xk) < SG(xk)`，由于`x ^ SG(xk) < SG(xk)`，那么对于这个游戏来说根据`SG`函数的定义`SG(xk)`表示达不到后续的最小自然数，而`x ^ SG(xk)`又小于这个值，所以，这个游戏必然可以从`xk`这个状态达到`SG(xi) = x ^ SG(xk)`这个状态
>
> 于是将这个游戏的`SG(xk)`替换为`x ^ SG(xk)`，再代入异或，结果就必然是`0`了

- 对于后手，若遇到的所有的游戏局面为$SG(x1) \oplus SG(x2) \oplus SG(x3) ..... SG(xn) = 0$，那么不管他操作哪一个游戏，最后留给先手的游戏局面一定是
  $SG(x1) \oplus SG(x2) \oplus SG(x3) ..... SG(xn) = x \neq 0$

> 可以用反证法进行证明，假设后手可以操作当前的一个游戏状态`xk`使得到达另外一个状态`SG(xi)`使得$SG(x1) \oplus SG(x2) \oplus SG(x3) ... \oplus SG(xi) .... SG(xn) = 0$，这个式子与$SG(x1) \oplus SG(x2) \oplus SG(x3) ... \oplus SG(xk) .... SG(xn) = 0$进行异或可以得到：$SG(xk) \oplus SG(xi) = 0$，说明这两个状态的值是相同的，说明$SG(xk) = SG(xi) = k$，而`xi`又是`xk`的后续状态，这与`SG`函数的定义矛盾，所以不成立

所以再整个的操作过程中，只有先手遇到的情况异或值不为`0`，那么他就一定可以操作某个游戏局面使得所有游戏的局面的异或值为`0`，而后手不管怎么操作留给先手的都是异或值不为`0`的情况，这样一步步下去，最后到达所有游戏的终结状态的时候必然是后手遇到，后手必输

一个集合`nim`游戏的例子

给定 n 堆石子以及一个由 k 个不同正整数构成的数字集合 S。

现在有两位玩家轮流操作，每次操作可以从任意一堆石子中拿取石子，每次拿取的石子数量必须包含于集合 S，最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

输入格式

第一行包含整数 k，表示数字集合 S 中数字的个数。

第二行包含 k 个整数，其中第 i 个整数表示数字集合 S 中的第 i 个数 s_i。

第三行包含整数 n。

第四行包含 n 个整数，其中第 i 个整数表示第 i 堆石子的数量 h_i。

输出格式

如果先手方必胜，则输出 `Yes`。

否则，输出 `No`。

数据范围

$1 \leq n, k \leq 100,$
$1 \leq s_i, h_i \leq 10000$

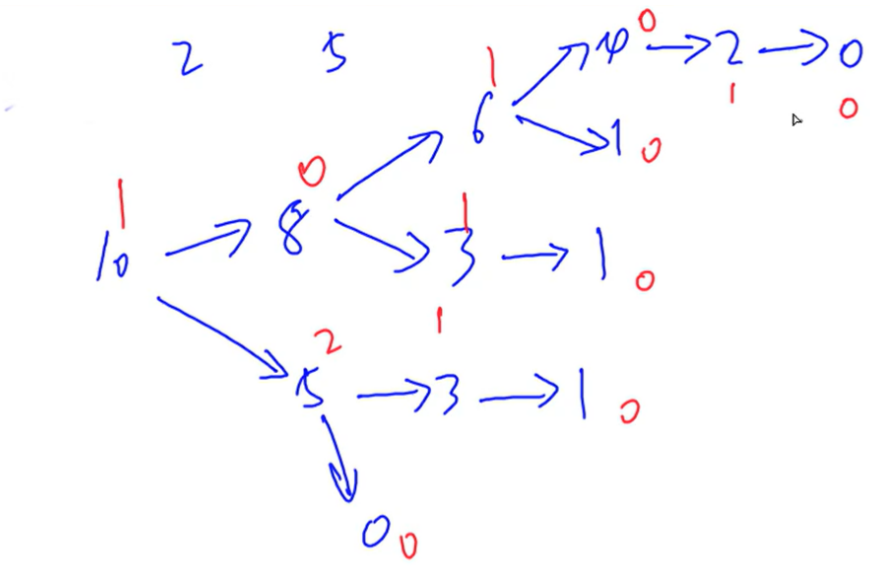
输入样例：

```
2
2 5
3
2 4 7
```

输出样例：

```
Yes
```

假设有一堆石子的个数是`10`，集合为`2，5`，那么我们每次拿`2`个或者`5`个形成的状态图就是，红色标注的是每个状态的`SG()`函数，终止状态的`SG()`函数的值定义为`0`：



那么对于`n`堆石子，那么我们可以认为有`n`个这样的图，从而求得`n`个图的起始状态的`SG`函数，然后将起点的`SG()`值异或起来，就知道先手是必输还是必胜，代码如下：

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<unordered_set>
using namespace std;
//最多有100堆石子,每堆石子的个数最多是10000个
const int N = 110, M = 10010;
int n, m;

//用s[]来表示可以操作的集合,用f[]来表示SG函数的值
int s[N], f[M];

int sg(int x)
{
    if(f[x] != -1) return f[x];

    //用一个哈希表来存当前他可以到达的局面
    unordered_set<int> S;
    //使用深度优先来进行搜索
    for(int i = 0; i < m; i ++)
    {
        //从集合中取出所有可以移动的石子个数,进行遍历
        int sum = s[i];
        //将下一个状态的sg函数加入到S中
        if(x >= sum) S.insert(sg(x - sum));

    }
    //退出for循环后我们就得到了当前的状态x可以到达的所有的其他的状态的sg函数
    //从小到达来枚举所有的自然数,来求得当前状态的sg函数
    for(int i = 0; ; i ++)
    {
        //如果这个自然数不在后面的状态中,就得到了当前状态的sg函数,并返回给上一层调用的地方
```

```
        if(!S.count(i)) return f[x] = i;
    }
}

int main()
{
    //读入集合中m个数
    cin >> m;
    for(int i = 0 ; i < m; i ++) cin >> s[i];

    //读入n表示有n堆石子
    cin >> n;
    int res = 0;
    memset(f, -1, sizeof f);

    for(int i = 0; i < n; i ++)
    {
        //输入每堆石子的个数
        int x;
        cin >> x;
        //这里使用记忆化搜索
        //异或上每堆石子的初始SG函数的值
        res ^= sg(x);
    }

    if(res) puts("Yes");
    else puts("No");

    return 0;

}
```

这里主要解释一下记忆化搜索这个东西，对于这`n`堆石子，每堆石子的最多个数是`10000`个，假设有两堆石子，这两堆石子的个数相同，那么集合`S`是公用的，所以对于这两堆石子来说从集合中取数操作是完全相同的，从而这两堆石子后面的状态图是完全相同的，于是对于不同堆的石子，若剩下了相同的石子`k`，那么`SG(k)`是完全可以公用的，统一记录到数组`f[]`中，前面堆的石子构建好了游戏图，后面的石子，可以直接使用前面的游戏图得到的`sg`的值，这样就大大加快了搜索的时间

对于一堆石子来说最多有`10000`个石子，也就最多有`10000`个状态，也就最多搜索`10000`次，一共有`100`堆石子，所以总的最多搜索次数就是`10 ^ 6`次，但是由于记忆化搜索，最终搜索的次数肯定远远小于这个结果