A Робот

Первый выигрывает, когда количество клеток четно, иначе второй.

Тут можно написать перебор для небольших N и найти закономерность. То есть, по сути, бектрекинг. Строим все пути и смотрим, если можем пойти в проигрышную клетку то путь выигрышный (при чем путь до этой клетки надо сохранять).

```
#include <stdio.h>
#include <sstream>
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <list>
#include <iomanip>
#include <map>
#include <set>
#include <cmath>
#include <queue>
#include <cassert>
#include <string.h>
using namespace std;
#pragma comment(linker, "/STACK:20000000")

typedef vector<int> vi;
#define sz(a) int((a).size())
#define all(c) (c).begin(),(c).end()


int u[116][116];
int n,m;
int can(int x, int y)
{
	if (x<0 || y<0 || x>=n || y>=m || u[x][y]) return 0;
	return 1;
}

int dx[4] = {1,0,0,-1};
int dy[4] = {0,1,-1,0};

int go(int x,int y)
{
	u[x][y]=1;
	for (int i=0;i<4;i++) {
		if (can(x+dx[i],y+dy[i]) )
		if (!go(x+dx[i],y+dy[i]))
		{
			u[x][y]=0;
			return 1;
		}
	}

	u[x][y]=0;
	return 0;

}

string res[22][22];
void solve()
{
```

```
	cin >> n >> m;
	memset(u,0,sizeof(u));

	if ((n*m)%2 == 0 ) printf("first\n"); else
		printf("second\n");
}



int main()
{

	solve();

	return 0;
}
```

B Саша и палочки

Заметим, что не важно с какой стороны зачеркиваются палочки. Всего игроки сделают $\lfloor \frac{n}{k} \rfloor$ ходов. Если это число нечетное, то Саша сделал на 1 ход больше и выиграл. Иначе Саша и Лена сделали одинаковое число ходов, а значит Саша не выиграет.

Решение C++

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 17, lg = 17;

ll n, k;
int main(){
	ios::sync_with_stdio(0), cin.tie(0);
	cin >> n >> k;
	cout << ((n / k) & 1 ? "YES" : "NO") << '\n';
	return 0;
}
```

Решение Python

```
n, k = map(int, input().split())
print('YES' if n//k % 2 != 0 else 'NO')
```

С Конан и Агаса играют в карточную игру

Let $A = max\ (a_1, a_2, ..., a_n)$. Observe that if A occurs an odd number of times, Conan can simply begin by removing one instance of A. If there are any cards left, they all have the same number A on them. Now each player can only remove one card in their turn, and they take turns doing so. Since there were an odd number of cards having A on them initially, this keeps continuing until finally, in one of Agasa's turns, there are no cards left.

However, if A occurs an even number of times, Conan cannot choose a card having A on it because it will leave Agasa with an odd number of cards having A. This will result in both players picking cards one by one, ending with Agasa picking the last card, and thus winning. In such a case, Conan can consider picking the next distinct largest number in the array, say B. If B occurs an odd number of times, then after Conan's turn there will be an even number of cards having B and an even number of cards having A. If Agasa takes a card having A then it becomes the same as the previous case and Conan wins. Otherwise, they take turns choosing a card having B until finally, on one of Agasa's turns, there are no cards having B and Agasa is forced to pick a card having A. Now it is Conan's turn and there are an odd number of cards having A, so it is again the same as the first case and Conan wins.

By a similar argument, we can show that if Conan plays optimally, he starts by picking a card having the greatest number that occurs an odd number of times. Conan loses if and only if there is no such number, i.e., Conan loses if and only if every number occurs an even number of times.

Решение C++

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int cnt[100005];

int main() {
  ios::sync_with_stdio(false);

  int n;
  cin >> n;
  while(n--) {
    int x;
    cin >> x;
    cnt[x]++;
  }

  for (int i = 1; i <= 1e5; i++) {
    if (cnt[i] % 2 == 1) {
      cout << "Conan\n";
      return 0;
    }
  }
  cout << "Agasa\n";
  return 0;
}
```

Решение Python

```
n=int(input())
b=[0]*100001
a=list(map(int,input().split()))
for i in a:
    b[i]+=1
for i in b:
    if i%2==1:
        print('Conan')
        exit()
print('Agasa')
```

D Игра с тарелками

Если первый игрок своим ходом не может поставить тарелку на стол (стол слишком маленький, и тарелка не помещается, т.е. $2r > min(a, b)$), выигрывает второй игрок.

Иначе выигрывает первый игрок. Выигрышная стратегия такова: первый игрок ставит свою первую тарелку в центр стола, а затем симметрично отражает ходы соперника относительно центра стола стола. Легко видеть, что если в этом случае второму игроку удастся совершить ход, первому это также удастся. А если не удастся, первый игрок победит, что ему и было нужно.

Решение C++

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int a,b,r;
    cin>>a>>b>>r;
    r*=2;
    if(r>a||r>b)
        cout<<"Second";
    else
        cout<<"First";
}
```

Решение Python

```
a, b, c = map(int, input().split())
if a < c * 2 or b < c * 2:
    print("Second")
else :
    print("First")
```

E Мишень

Эту задачу не решить стандартным методом с помощью выигрышных и проигрышных состояний, просто их очень много. Также нет какой-то простой зависимости от N.

Поэтому решаем функцией Гранди.

После первого хода круг превращается в цепочки.

Состоянием будет длина цепочки. На каждом шаге перебираем все варианты хода. Цепочка распадается на 2 . Хорим результаты функции Гранди для обеих цепочек. Выбираем минимальное число которое не встречалось после всех возможных ходов. Это и будет результатом функции.

Если функция возвращает положительный результат то выигрывает первый, если 0 то второй.

```
#include <stdio.h>
#include <sstream>
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <list>
#include <iomanip>
#include <map>
#include <set>
#include <cmath>
#include <queue>
#include <cassert>
#include <string.h>
using namespace std;
#pragma comment(linker, "/STACK:20000000")

typedef vector<int> vi;
#define sz(a) int((a).size())
#define all(c) (c).begin(),(c).end()

string problem_name = "game";

void init(){
	freopen((problem_name+".in").c_str(),"rt",stdin);
	freopen((problem_name+".out").c_str(),"wt",stdout);
}

int g[1010];  // числа Гранди

int go(int len){
	if (len<0) return 0;
	if (g[len]!=-1) return g[len]; // если посчитано
	set <int> st;                  // храним результаты возможных ходов
	for (int i=1;i<=len;i++)           // перебираем возможное деление
		st.insert(go(i-2)^go(len-i-1));
	int res=0;  // ищем первое число которое не встречается в множестве
	while (st.find(res)!=st.end()) res++;
	return g[len]=res;
}

int main()
{
	init();

	//srand(5);
```

```
	memset(g,-1,sizeof(g));


	int n;

	scanf("%d\n",&n);

	if (n<3 || n>1000) return 1;

	if (go(n-3)) printf("%d\n",2);  else
		printf("%d\n",1);


	return 0;
}
```

F В поисках локального минимума

We maintain by binary search a range $[l, r]$ which has a local minimum. Moreover, we assume that $a_{l-1} > a_l$ and $a_r < a_{r+1}$. Initially, $[l, r] = [1, n]$.

In each iteration, let m be the midpoint of l and r.

Case 1. If $a_m < a_{m+1}$, then the range becomes $[l, m]$.

Case 2. If $a_m > a_{m+1}$, then the range becomes $[m + 1, r]$.

When $l = r$, we have found a local minimum a_l.

The number of queries to a_i is at most $2\lceil \log_2 n \rceil \leq 34 < 100$.

Решение C++

```
#include <bits/stdc++.h>

using namespace std;

const int MAXN = 100010;

int n;
int a[MAXN];

int query(int x)
{
	if (1 <= x && x <= n)
	{
		printf("? %d\n", x);
		fflush(stdout);
		scanf("%d", &a[x]);
	}
}

int main()
{
	scanf("%d", &n);
	a[0] = a[n + 1] = n + 1;
	int L = 1, R = n;
	while (L < R)
	{
		int m = (L + R) / 2;
		query(m);
		query(m + 1);
```

```
            if (a[m] < a[m + 1])
                  R = m;
            else
                  L = m + 1;
      }
      printf("! %d\n", L);
      fflush(stdout);
      return 0;
}
```

Решение Python

```
def interact(i):
    print("?",i,flush=True)
    return int(input())

n=int(input())
l=0
r=n
while r-l>1:
    m=(l+r)//2
    if interact(m)<interact(m+1):
        r=m
    else:
        l=m
print("!",r,flush=True)
```

G Игра для бобров

Если *n* четно, то всегда выигрывает Марсель - он просто симметрично повторяет ходы Тимура.

Теперь рассмотрим случай, когда *n* нечетно. Если Тимур не может сделать хода - он проигрывает автоматически. Если же он может сделать ход, то он всегда может разгрызть одно из бревен так, чтобы далее его части нельзя было разгрызть на меньшие части. Чтобы это сделать, нужно найти наименьшее число *t* такое, что $k \le t < m$ и $t | m$, и разгрызть бревно на части длины *t*. После этого Тимур симметрично повторяет ходы Марселя и выигрывает.

Чтобы проверить может ли Тимур сделать ход, нужно перебрать все делители *m*. Если хоть для одного из них (предположим, *t*), выполняется $k \le t < m$, то ход сделать возможно. Проверку всех делителей можно сделать за время $O(\sqrt{m})$.

Решение C++

```
#include<iostream>

using namespace std;

int main(){
      int n, m, k;
      cin >> n >> m >> k;
      bool div=false;
      if(m>1 && k==1){
            div=true;
```

```
        }
    for(int j=2; j*j<=m; j++){
        if (m%j==0   && (m/j)>=k){
                    div=true;
                    break;
                }
        }
        if(div && n%2==1){
                cout << "Timur" << endl;
        }
        else{
                cout << "Marsel" << endl;
        }
        return 0;
}
```

Решение Python

```
n,m,k=list(map(int,input().split()))
if n&1:
    i=1
    flag=False
    while(i*i<=m):
        if m%i==0 and (k<=i<m or k<=m//i<m):
            flag=True
            break
        i+=1
    print("Timur") if flag else print("Marsel")
else:
    print("Marsel")
```