新生入队积分赛模拟1 题解

E

签到题。
把区间按照左端点排序后，按题意模拟即可。
注意放木板后超出的部分可能会影响后面的木板，可以用一个变量 `cur` 来记录木板放到的最远距离。
时间复杂度 $O(N \log N)$，主要是排序的复杂度。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
void solve() {
    ll n, L;
    cin >> n >> L;
    vector<pair<ll, ll>> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i].first >> a[i].second;
    }
    sort(a.begin(), a.end());
    int ans = 0;
    ll cur = 0;
    for (int i = 0; i < n; ++i) {
        if (cur >= a[i].second) continue;
        cur = max(a[i].first, cur);
        ll x = (a[i].second - cur - 1 + L) / L;
        ans += x;
        cur += x * L;
    }
    cout << ans << "\n";
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T = 1;
    //cin >> T;
    while (T--) solve();
    return 0;
}
```

F

贪心。
把 t_i 按照时间升序排序，假设当前枚举到第 i 个人，已经用的总时间为 res，如果 $t_i \leq res$ 意味着我们只有将这个人与最后一个已经排了队的人交换位置才能没有负贡献，没有净贡献反而还增大了 res，所以 t_i 就不能拿，把他放最后，否则就加上 t_i。
时间复杂度 $O(N \log N)$，主要是排序的复杂度。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
void solve() {
    int n;
    cin >> n;
    vector<ll> t(n);
    for (int i = 0; i < n; ++i) {
        cin >> t[i];
    }
    sort(t.begin(), t.end());
    ll res = 0, ans = 0;
    for (int i = 0; i < n; ++i) {
        if (t[i] >= res) {
            ans++;
            res += t[i];
        }
    }
    cout << ans << "\n";
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T = 1;
    //cin >> T;
    while (T--) solve();
    return 0;
}
```

B

- $n = 1$ 时，显然后手赢。
- $n \neq 1$ 时，谁面对 " n 为奇数 "这个状态的人谁就能赢，所以双方都一定是尽可能地保留 n 为偶数。

具体地，设 $n = 2^x p$。

- 如果 p 不是质数，显然先手可以通过 x 来决定是否一次除掉 p 还是两次来转换自己面对 2^x 的先后手，所以先手必赢。
- 如果 p 是质数，那么先手第一步只能选择除掉 p，再讨论 x 看看先手赢还是后手赢。

时间复杂度 $O(T\sqrt{N})$

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
bool isprime(int x) {
    for (int i = 2; i * i <= x; ++i) {
        if (x % i == 0) return false;
    }
    return true;
}
void solve() {
    int n;
    cin >> n;
    int x = __builtin_ctz(n);
    if (n == 2) {
        cout << "Ashishgup\n";
    } else if (n == 1 || (n >> x) == 1) {
        cout << "FastestFinger\n";
    } else {
        if (n & 1) {
            cout << "Ashishgup\n";
        }else {
            int p = !isprime(n >> x) | isprime(n >> x) & (x > 1);
            cout << (p ? "Ashishgup\n" : "FastestFinger\n");
        }
    }
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T = 1;
    cin >> T;
    while (T--) solve();
    return 0;
}
```

C

看到异或应该想到考虑二进制位。
假设第 i 位是从高往低第一个 l 与 r 的对应位置不相等的位置，显然 a 和 b 的 i 高位都是一样的，异或起来为 0，没有贡献。
我们可以这样构造 a 和 b 的值：$a = 2^i$，$b = 2^i - 1$，答案就是 $2^{i+1} - 1$，是低 i 位能取到的最大值。a，b 显然在 $[l, r]$ 内，自己想想。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
void solve() {
    ll l, r;
    cin >> l >> r;
    for (int i = 60; i >= 0; --i) {
        if (((l >> i) & 1) != ((r >> i) & 1)) {
            cout << (1LL << i + 1) - 1 << "\n";
            return;
        }
    }
    cout << 0 << "\n";
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T = 1;
    //cin >> T;
    while (T--) solve();
    return 0;
}
```

D

二分答案 + 滑动窗口。
显然所有满足要求的花盆位置都能划归为一个点正好对齐花盆的一个端点的情况。所以我们二分花盆的长度 W，将点按照横坐标 x 排序，$check$ 时枚举每一个点的 x 作为区间右端点，此时只需要判断 $[x - W, x]$ 的最大值和最小值相减是否满足大于等于 D 就可以，这个是经典的单调队列滑动窗口。
时间复杂度 $O(N \log V)$。

```cpp
#include<bits/stdc++.h>
#define ll long long
#define llINF 0x7FFFFFFFFFFFFFFF
#define intINF 0x7fffffff
using namespace std;
const int maxn = 1e5+5;
int n,d,ans,ma,mi = intINF;
struct node{
	int x,y;
} a[maxn];
bool check(int len){
	deque<int> q1,q2;
	for(int i = 1; i <= n; ++i){
		while(q1.size() && a[i].y >= a[q1.back()].y) q1.pop_back();
		q1.push_back(i);
		while(q1.size() && a[q1.front()].x + len < a[i].x) q1.pop_front();
		while(q2.size() && a[i].y <= a[q2.back()].y) q2.pop_back();
		q2.push_back(i);
		while(q2.size() && a[q2.front()].x + len < a[i].x) q2.pop_front();
		if(a[q1.front()].y - a[q2.front()].y >= d) return true;
	}
	return false;
}
int main(){
	cin >> n >> d;
	for(int i = 1; i <= n; ++i){
		read(a[i].x,a[i].y);
		ma = max(ma,a[i].y);
		mi = min(mi,a[i].x);
	}
	if(ma - mi < d) return printf("-1")&0;
	sort(a+1,a+n+1,[](node x,node y){
		return x.x < y.x;
	});
	int L=1,R=1e6+5;
	while(L <= R){
		int mid = L+R >> 1;
		if(check(mid)){
			ans = mid;
			R = mid-1;
		}
		else L = mid+1;
	}
```

```
        cout << (ans ? ans : -1);
        return 0;
}
```

A

看到要维护 $\mathrm{mod}\,M$ 的信息时可以考虑根号分治。

- 若 $Y \leq \sqrt{V}$，我们可以对 $[1, \sqrt{V}]$ 所有模数都开一个桶，每加入一个 x，$O(\sqrt{V})$ 暴力更新每个桶，查询直接 $O(1)$ 输出该桶的最小值。
- 若 $Y \geq \sqrt{V}$, 则商不会超过 $\sqrt{V}$，用一个 `set` 来保存输入的 x，询问时，枚举商数 i，直接二分找第一个 $\geq Y * i$ 的数 P，用 $P - Y * i$ 更新答案，复杂度 $O(\sqrt{V}\log N)$。
  总时间复杂度 $O(N\sqrt{N}\log N)$。
  用值域分块代替 set 能做到$O(N\sqrt{V})$，自己上网研究研究。

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
void solve() {
    int n;
    cin >> n;
    set<int> s;
    const int V = 480;
    unordered_map<int, int> mp;
    for (int i = 0; i < n; ++i) {
        char c;
        int x;
        cin >> c >> x;
        if (c == 'A') {
            s.insert(x);
            for (int i = 1; i < V; ++i) {
                if (mp.count(i)) mp[i] = min(x % i, mp[i]);
                else mp[i] = x % i;
            }
        } else {
            int ans = 0x3f3f3f3f;
            if (x >= V) {
                for (int i = 0; i <= 625; ++i) {
                    auto t = s.lower_bound(i * x);
                    if (t == s.end()) break;
                    ans = min(ans, *t - i * x);
                }
            } else {
                ans = mp[x];
            }
            cout << ans << "\n";
        }
    }
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T = 1;
    //cin >> T;
    while (T--) solve();
    return 0;
}
```