# 【题解】2020 牛客 NOIP 赛前集训营-普及组（第四场）

## A - 时间

**题目大意**

给出一个时间，求出在 24 小时制下的 3 小时 30 分后的时间为多少。

**简要题解**

直接模拟即可，注意两天间时间的变化。

具体可以把时间都换算成分钟，然后对一天的分钟数取模，然后再换算成标准时间。

**参考代码**

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

int main()
{
    int h, m;
    scanf("%d:%d", &h, &m);
    m = h * 60 + m;
    m += 3 * 60 + 30;
    m %= 24 * 60;
    h = m / 60, m = m % 60;
    printf("%02d:%02d\n", h, m);
    return 0;
}
```

# B - 石子

**题目大意**

给出 n 堆石子，Alice 每次可以从一堆石子中取偶数个石子，Bob 每次可以从一堆中取奇数个石子，每次操作至少要取走一个石子，无法操作的人输掉游戏。

假设 Alice 和 Bob 都是绝顶聪明的，请问 Alice 先手时，Alice 是否有必胜策略。

**简要题解**

**结论：当且仅当石子堆数为 1 堆，且石子数量为偶数时，Alice 有必胜策略，否则一定是 Bob 获胜。**

证明：Bob 的策略为：若场上存在偶数堆石子，那么将这一堆取走奇数个，使其变成奇数堆石子；否则，取走一整堆石子（因为此时每一堆石子数量均为奇数）。由此可见，无论 Alice 如何操作，场上的偶数石子堆的数量一直减少，因而无法使得 Bob 无法操作。

**参考代码**

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    while (~scanf("%d", &n))
    {
        int d;
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &d);
        }
        if (n == 1 && d % 2 == 0)
            puts("YES");
        else
            puts("NO");
    }
    return 0;
```

```
}
```

# C - 卡片

## 题目大意

给出两个正多边形，其中一个绕着另一个旋转，求回到原来位置时所需要的旋转次数。

## 简要题解

考虑将旋转操作分为两类：

第一类是在地面上旋转，如样例中的第一次旋转；

第二类是在角上旋转，如样例中的第二次旋转。

然后分别讨论两类旋转的次数，求和输出即可。

## 参考代码

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

typedef long long ll;

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    ll a, m, b, n;
    while (cin >> a >> m >> b >> n)
    {
        ll x = a * (n * b) / __gcd(a, n * b);
        ll c1 = x / a;
        ll y = a * b / __gcd(a, b);
        ll c2 = x / y * (y / b - 1);
        cout << c1 + c2 << '\n';
    }
    return 0;
}
```

# D - 项链

我们可以称项链为点的子段和。记最大子段和为$ans1$，次大子段和为$ans2$。

**如何求最大值**

对于$ans1$ 其实我们已经有了明确的解法。

设当前结点的编号为$u$，我们可以维护两个值$mn, sum$分别表示从起点到$u$的前缀和的最小值以及到$u$的前缀和。那么显然我们用$sum - mn$去更新$ans1$就可以得到正解。

**如何求次大值**

然后我们也应该注意到，最大子段和必然是一条连续的路径，也意味着它存在起点$L$和终点$R$。所以$ans2$出现的区间必然在$L$的左侧或者$R$的右侧。

**对于链的情况**

我们只需要在更新$mn$的时候更新出$L$，更新$ans1$的时候更新$R$，之后注意边界重复两次即可（时间是允许的）。此外，由于我们在每个点都会用$sum - mn$更新$ans1$，我们可以额外变量$pmx$维护一个从起点到$u$的所有$sum - mn$的最大值，以降低复杂度。

## 对于$n \le 200$的情况

由于图中没有环，那么我们从任意一个点 dfs 都可以得到若干条链，不难转换为情况 1。

**对于完整的数据**

我们首先需要明确，利用拓扑序得到的答案肯定不会比从某个点暴力 dfs 得到的答案更差，且利用拓扑序我们化解的图的问题，如此我们就有个拓扑序上 dp

的想法。

仍然需要维护$sum, mn, L, R$这些变量。不同的是在更新$ans1$时需要利用$pmx$更新$ans2$，选取$ans2$更大的路径。

对于$R$之后的路径。

最后由于在图中可能得到很多处$ans1$，我们记录所有出现的位置，以每个出现的位置$R$为起点记忆化搜索跑出一个最大值即可。

### 参考代码

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

typedef long long ll;
const int maxn = 100010;
const ll inf = 0x3f3f3f3f3f3f3f3f;

int deg[maxn], pre[maxn];
ll w[maxn], mnsum[maxn], mxsum[maxn], sum[maxn], val[maxn], dp[maxn];
vector<int> v[maxn];

queue<int> q;
vector<int> pos;

ll ans1 = 0, ans2 = 0;

inline void dfs(int x, ll pmnsum, ll psum)
{
    ans2 = max(ans2, psum - pmnsum);
    if (psum - pmnsum < dp[x]) return;
    dp[x] = psum - pmnsum;
    for (int i = 0; i < v[x].size(); ++i) {
        dfs(v[x][i], min(pmnsum, psum + w[v[x][i]]), psum + w[v[x][i]]);
    }
}

int main()
{
    int n, m;
    scanf("%d %d", &n, &m);
```

```
    for (int i = 1; i <= n; ++i) {
        scanf("%lld", &w[i]);
    }
    for (int i = 1, x, y; i <= m; ++i) {
        scanf("%d %d", &x, &y);
        v[x].push_back(y);
        ++deg[y];
    }
    for (int i = 1; i <= n; ++i) {
        val[i] = sum[i] = -inf;
    }
    for (int i = 1; i <= n; ++i) {
        if (deg[i] == 0) {
            q.push(i);
            mnsum[i] = min(0LL, w[i]);
            sum[i] = w[i];
            val[i] = max(0LL, w[i]);
            pre[i] = (w[i] > 0 ? 0 : i);
            ans1 = max(ans1, val[i]);
        }
    }
    while (q.size()) {
        int now = q.front(); q.pop();
        for (int i = 0; i < v[now].size(); ++i) {
            int to = v[now][i];
            if (val[to] <= sum[now] - mnsum[now] + w[to]) {
                val[to] = sum[now] - mnsum[now] + w[to];
                sum[to] = sum[now] + w[to];
                pre[to] = (sum[to] > mnsum[now] ? pre[now] : to);
                mnsum[to] = min(mnsum[now], sum[to]);
                mxsum[to] = max(mxsum[now], sum[to] - mnsum[to]);
                if (val[to] > ans1) {
                    ans1 = val[to], ans2 = mxsum[pre[to]];
                    pos.clear();
                    pos.push_back(to);
                } else if (val[to] == ans1) {
                    ans2 = max(ans2, mxsum[pre[to]]);
                    pos.push_back(to);
                }
            }
            --deg[to];
            if (deg[to] == 0) {
                q.push(to);
            }
```

```
        }
    }
    for (int i = 0; i < pos.size(); ++i) {
        for (int j = 0; j < v[pos[i]].size(); ++j) {
            dfs(v[pos[i]][j], 0, 0);
        }
    }
    printf("%lld %lld\n", ans1, ans2);
    return 0;
}
```