COGS 3719 有n种物品

- 每一种物品，两个人都会拥有一件，所以一旦某个人先选，那么另外一件物品必然是另外一个人的。

子任务1：$a_i \geq b_i$

- 每个人若想获得的物品价值最大，那么要尽可能的先选物品，而且要让对方的价值尽可能小，所以进行先取时，优先取择价值差别大的。
- 将物品按照$a_i - b_i$的顺序从大到小排序，依次确定先取即可。
- 时间复杂度：O($N \log N$)。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10;
struct Node
{
    int a, b;
} c[N];
bool cmp(Node p, Node q)
{
    return p.a - p.b > q.a - q.b;
}
int main()
{
    int n; cin >> n;
    for(int i = 1; i <= n; ++i) cin >> c[i].a >> c[i].b;
    sort(c + 1, c + n + 1, cmp); // 按照a-b的差值从大到小排序
    long long x = 0, y = 0;   // 两个人获得的价值
    for(int i = 1; i <= n; ++i)
    {
        if(i % 2) x += c[i].a, y += c[i].b; // A先取
        else x += c[i].b, y += c[i].a;    // B先取
    }
    cout << x << " " << y;
    return 0;
}
```

子任务2

- 物品先选不一定价值大，那么为了利益最大化，那么肯定先选$a_i \geq b_i$的物品，而对于$a_i < b_i$的物品，二者都希望对方先选。
- 首先考虑$a_i \geq b_i$的物品，二者依次进行先选。
- 等到物品开始出现$a_i < b_i$时，考虑最后一个$a_i \geq b_i$的物品是谁选的：
    - 如果是A选的，那么B此时会选择$a_i \geq b_i$中它没有选择的物品，然后接着A也会选择$a_i \geq b_i$中它没有选择的物品，直到所有的$a_i \geq b_i$的物品全部选完。此时，轮到A选了，那么后续A只能选$a_i < b_i$的物品a_i，B接着选b_i。
    - 如果是B选的，那么A也会会选择$a_i \geq b_i$中它没有选择的物品，然后接着B也会选择$a_i \geq b_i$中它没有选择的物品，直到所有的$a_i \geq b_i$的物品全部选完。此时，轮到A选了，那么后续A只能选$a_i < b_i$的物品a_i，B接着选b_i。

故而对于$a_i < b_i$的物品，A只能选a_i，而B选择所有的b_i。

- 时间复杂度：O($N \log N$)。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10;
struct Node
{
    int a, b;
} c[N];

bool cmp(Node p, Node q)
{
    return p.a - p.b > q.a - q.b;
}

int main()
{
    int n; cin >> n;
    for(int i = 1; i <= n; ++i) cin >> c[i].a >> c[i].b;
    sort(c + 1, c + n + 1, cmp); // 按照a-b的差值从大到小排序
    long long x = 0, y = 0;    // 两个人获得的价值
    for(int i = 1; i <= n; ++i)
    {
        if(c[i].a - c[i].b >= 0)
        {
            if(i % 2) x += c[i].a, y += c[i].b;
            else x += c[i].b, y += c[i].a;
        }
        else x += c[i].a, y += c[i].b;
    }
    cout << x - y;
    return 0;
}

```

C3264.魔法部落

算法1：模拟

- 按照题目要求一边求幂，一边累和。
- 注意无论是求幂还是累和，都需要多取余，注意用`long long`。
- 时间复杂度：O(N)。

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

const int M = 1e9 + 7;

int main()
{
    int n; cin >> n;
    long long ans = 1;

```

```
    long long p = 1;
    for(int i = 1; i <= n; ++i)
    {
        p = p * 3 % M;
        ans = (ans + p) % M;
    }
    cout << ans << endl;
    return 0;
}
```

算法2：快速幂

- $3^0+3^1+3^2+3^3+3^4+3^5=3^0+3^1+3^2+(3^0+3^1+3^2)\times 3^3$
- 令$S_n=3^0+3^1+3^2+\cdots+3^n$:
  - 若n为奇数，则$S_n=S_{\frac{n}{2}}+S_{\frac{n}{2}}\times 3^{\frac{n+1}{2}}$；
  - 若n为偶数，则$S_n=S_{n-1}+3^n$。
- 定义递归函数计算S_n，中间需要定义快速幂函数。
- 时间复杂度为：O($\log^2 N$)。

```
// 二分快速幂
#include <bits/stdc++.h>
using namespace std;

const long long M = 1e9 + 7;

// x^n % M
long long f(int n)
{
    if(n == 0) return 1;
    long long t = f(n / 2);
    if(n % 2 == 0) return t * t % M;
    else return t * t % M * 3 % M;

}

long long s(int n)
{
    if(n == 0) return 1;
    long long t = s(n / 2);
    if(n % 2 == 1) return (t + t * f((n + 1) / 2) % M) % M;
    else return (s(n - 1) + f(n)) % M;
}

int main()
{
    int n; cin >> n;
    cout << s(n);
    return 0;
}
```

错误解法

- 根据等比数列求和公式$S_n=\frac{3^{n+1}-1}{2}$，利用快速幂取余求$k=3^{n+1}\%M$，然后令$ans=(k-1)/2$。
- 错误原因：$(a\div b)\%M\neq(a\%M\div b\%M)\%M$。
- 注意以下公式都是成立的：
  - $(a+b)\%M=(a\%M+b\%M)\%M$
  - $(a-b)\%M=(a\%M-b\%M+M)\%M$(加$M$是为了避免出现负数)
  - $(a\times b)\%M=(a\%M\times b\%M)\%M$
  - $(a^b)\%M=(a\%M)^b\%M$

C1045.七夕祭

- 通过观察可以发现：交换左右相邻两个摊点只会改变列上感兴趣的摊点数量，交换上下相邻两个摊点只会改变行上感兴趣的的摊点数量。
- 于是问题可以分成两个独立的子问题：
  - 通过最少次数的左右交换，使得列上感兴趣的摊点数相同。
  - 通过最少次数的上下交换，使得行上感兴趣的摊点数相同。
- 以上下交换为例，首先统计每行中感兴趣的摊点总数，记为$b[1\sim n]$。
- 若T不能被n整除，那么不可能达到要求；若能整除，则调整后每一行的摊点数为$\frac{T}{n}$。
- 类似于均分纸牌模型，可以通过交换相邻两行的摊点(具体是谁不关注)使得每一行的摊点数为$\frac{T}{n}$，不过和均分纸牌模型不同的是：$b[1]$和$b[n]$也是相邻的。
- 通过思考可以发现，在最优解的情况下，环(行)上某两个相邻的位置之间一定不会有摊位交换。

朴素算法

- 枚举不会交换的相邻位置，然后让环断成链，那么转化为均分纸牌模型。
- 在所有枚举的均分纸牌模型中选择最小的结果即可。
- 时间复杂度：$\mathrm{O}(N^2+M^2)$。

算法

- 首先将b_i都减去$\frac{T}{n}$，然后求$b[]$前缀和$s[]$，如果从第1行开始均分纸牌那么$|s[i]|$就是在第i行需要交换的次数。
- 那么显然有$s[n]=0$，均分纸牌的答案为$\sum|s[i]|$

```
b[1], b[2], b[3], ..., b[n]
s[1], s[2], s[3], ..., s[n]
```

- 考虑在第k列和第$k+1$列之间没有交换，那么均分纸牌需要的情况如下：

```
b[k+1]     |s[k+1]-s[k]|
b[k+2]     |s[k+2]-s[k]|
...
b[n]       |s[n]-s[k]|
b[1]       |s[1]+s[n]-s[k]|
b[2]       |s[2]+s[n]-s[k]|
...
b[k-1]     |s[k-1]+s[n]-s[k]|
b[k]       |s[k]+s[n]-s[k]|
```

- 因为有$s[n]=0$，所以

```
b[k+1]      |s[k+1]-s[k]|
b[k+2]      |s[k+2]-s[k]|
...
b[n]        |s[n]-s[k]|
b[1]        |s[1]-s[k]|
b[2]        |s[2]-s[k]|
...
b[k-1]      |s[k-1]-s[k]|
b[k]        |s[k]-s[k]|
```

- 于是问题转化为有n个数$s[1,n]$，求使得$\sum|s[i]-s[k]|$最小的k，这其实是一个中位数模型。
- 将$s[1,n]$从小到大排序，然后求出中位数，从中位数位置的后面断开即为最小结果。

C1266.借教室

算法1：模拟

- 按照题目要求模拟即可。
- 时间复杂度：O(NM)。

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

const int N = 1e6 + 10;
int a[N];

int main()
{
    freopen("classroom.in", "r", stdin);
    freopen("classroom.out", "w", stdout);
    int n, m; scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
    for(int i = 1;i <= m; ++i)
    {
        int d, s, t; scanf("%d%d%d", &d, &s, &t);
        for(int j = s; j <= t; ++j)
            if(a[j] < d) { printf("-1\n%d", i); return 0; }
            else a[j] -= d;
    }
    puts("0");
    return 0;
}
```

算法2：二分答案

- 一般情况下人数较少时，能实现所有的安排；当人数变多时，可能无法实现所有的安排，所以能否实现安排是有单调性的，可以用二分答案。
- 定义`check(k)`函数判定能否实现前k个的安排；因为每个人的安排都是区间操作，可以用差分进行优化。
- 时间复杂度：O($N \log M$)。

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

const int N = 1e6 + 10;
int n, m;
int a[N], b[N];
int d[N], s[N], t[N];

// 前k个人是否合法
bool check(int k)
{
    for(int i = 1; i <= n; ++i) b[i] = a[i] - a[i - 1];
    for(int i = 1; i <= k; ++i) b[s[i]] -= d[i], b[t[i] + 1] += d[i];
    for(int i = 1; i <= n; ++i)
    {
        b[i] += b[i - 1];
        if(b[i] < 0) return false;
    }
    return true;
}

int main()
{
    freopen("classroom.in", "r", stdin);
    freopen("classroom.out", "w", stdout);
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
    for(int i = 1; i <= m; ++i) scanf("%d%d%d", &d[i], &s[i], &t[i]);
    int l = 0, r = m;
    while(l < r)
    {
        int k = (l + r + 1) / 2;
        if(check(k)) l = k;
        else r = k - 1;
    }
    if(l == m) puts("0");
    else printf("-1\n%d", l + 1);
    return 0;
}
```

C0532.Brownie Slicing

- 获得的最多的巧克力豆最小，典型的最大值最小，可以考虑二分答案。
- 二分能获得的巧克力豆k，下界为$\max\{a[i][j]\}$，上界为$\sum a[i][j]$。
- 判定函数：能否切成超过A行B列且每块和$\geq k$。
- 时间复杂度：$\mathrm{O}(NM\log(NM))$。

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

const int N = 1010;
int n, m, A, B;
int a[N][N];

int ask(int x1, int y1, int x2, int y2)
{
    return a[x2][y2] - a[x1 - 1][y2] - a[x2][y1 - 1] + a[x1 - 1][y1 - 1];
}

// 能否切成超过A行B列 且每块和都>=k
bool check(int k)
{
    int x = 0; // 行切多少块
    int r1 = 1;    // 当前块行起点
    for(int r2 = 1; r2 <= n; ++r2)
    {
        int y = 0; // 列切多少块
        int c1 = 1; // 当前块列起点
        for(int c2 = 1; c2 <= m; ++c2)
            if(ask(r1, c1, r2, c2) >= k) ++y, c1 = c2 + 1;
        if(y >= B) ++x, r1 = r2 + 1;
    }
    return x >= A;
}

int main()
{
    freopen("brownie.in", "r", stdin);
    freopen("brownie.out", "w", stdout);
    scanf("%d%d%d%d", &n, &m, &A, &B);
    int l = 1e9, r = 0;
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= m; ++j)
            scanf("%d", &a[i][j]), l = min(l, a[i][j]);
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= m; ++j)
            a[i][j] += a[i - 1][j] + a[i][j - 1] - a[i - 1][j - 1];
    r = a[n][m];
    while(l < r)
    {
        int k = (l + r + 1) / 2;
        if(check(k)) l = k;
        else r = k - 1;
    }
    printf("%d", l);
    return 0;
}
```

C3842.国旗计划

- 首先需要破环为链，将坐标范围扩大至$2m$，同时将战士们的奔袭区间也复制一份，然后按照起点终点从小到大排序。
- 对于每一个边防战士，定义$f[i][j]$表示从第i个人开始向右再挑选2^j个人最远能覆盖的位置。
- 首先求解$f[i][0]$，它表示从i开始再选一个人最远能覆盖的位置，因为已经按照起点终点排序，所以可以二分求解。
- 利用倍增求解剩余的状态，$f[i][j] = f[f[i][j-1]][j-1]$。
- 对于每一位战士，求解向右挑若干人覆盖长度$\geq m$的最少的人即可，可以用二分答案，评价函数为从i开始再选k个人后覆盖长度能否$\geq m$。

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

const int N = 2e5 + 10;

struct Node
{
    int x, y;
    int id;
}a[2 * N];
int n, m;
int T = 20; // 倍增
int ans[N];
int f[2 * N][25]; // 从第i个人向右再挑2^j个人能到哪

bool cmp(const Node &a, const Node &b)
{
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

int query(int i, int k)
{
    int t = i;
    for(int j = T; j >= 0; --j)
        if((k >> j) & 1) t = f[t][j];
    return a[t].y - a[i].x;
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; ++i)
    {
        int x, y; scanf("%d%d", &x, &y);
        if(x >= y) y += m;
        a[i] = {x, y, i};
    }
    sort(a + 1, a + n + 1, cmp);
    for(int i = 1; i <= n; ++i) a[i + n] = {a[i].x + m, a[i].y + m, a[i].id};
    for(int i = 1; i <= 2 * n; ++i)
    {
        int l = i, r = 2 * n;
        while(l < r)
        {
            int k = (l + r + 1) / 2;
```

```
            if(a[k].x <= a[i].y && a[k].y >= a[i].y) l = k;
            else r = k - 1;
        }
        f[i][0] = l;
    }
    for(int j = 1; j <= T; ++j)
        for(int i = 1; i <= 2 * n; ++i)
            f[i][j] = f[f[i][j - 1]][j - 1];
    for(int i = 1; i <= n; ++i)
    {
        int l = 0, r = n - 1;
        while(l < r)
        {
            int k = (l + r) / 2;
            if(query(i, k) >= m) r = k;
            else l = k + 1;
        }
        ans[a[i].id] = l + 1;
    }
    for(int i = 1; i <= n; ++i) printf("%d ", ans[i]);
    return 0;
}
```