2022 天梯赛选拔赛 题解

----by lr580

> 以下所有题解仅提供一种或多种正确解法。并不必然代表下面提供的解法是最优解，且并不必然代表其他的解法不可行。因此，如果有别的思路，也欢迎各位大佬在 SCNUOJ 讨论区分享你的解法。若题解有误，欢迎指正~ 。˚ · (/>/ ㅅ /</) · ˚ 。

星月诞生

值得一提的是，天梯赛是允许使用万能头的。(事实上不允许用的比赛少之又少)

C++ 参考程序

```cpp
#include <bits/stdc++.h>
int main()
{
    printf("Xing Yue\n");
    printf("Xing Yue\n");
    printf("Xing Yue\n");
    return 0;
}
```

Python 参考程序

```python
print('''Xing Yue
Xing Yue
Xing Yue''')
```

星月摸鱼

可以直接计算答案 $\frac{n(a+b)}{2}$，也可以循环逐天模拟

C++ 参考程序

```cpp
#include <bits/stdc++.h>
typedef long long ll;
#define sc(x) scanf("%lld", &x)
ll n, a, b;
signed main()
{
    sc(n), sc(a), sc(b);
    printf("%lld", n / 2 * (a + b));
    return 0;
}
```

Python 参考程序

```python
n, a, b = [int(i) for i in input().split()]
print(n // 2 * (a + b))
```

星月划水

直接分类讨论即可，周一 24 ，周二到周五 20 ，否则 16

C++ 参考程序

```cpp
#include <bits/stdc++.h>
typedef long long ll;
#define sc(x) scanf("%lld", &x)
ll x;
signed main()
{
    sc(x);
    if (x == 1)
        printf("24");
    else if (x <= 5)
        printf("20");
    else
        printf("16");
    return 0;
}
```

Python 参考程序

```python
x = int(input())
print(24 if x == 1 else 20 if x <= 5 else 16)
```

星月训练

直接按照题意说的去做即可，使用 `float` 或 `double` 输出两位以上的小数都可以过题；对 `C/C++/Java` 如果计数变量是整数，除法的时候要先强转浮点数。

C++ 参考程序

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
ll n, a, b, tp, tn, fp, fn;
signed main()
{
    sc(n);
    while (n--)
    {
        sc(a), sc(b);
        if (a == 1 && b == 1)
            tp++;
```

```
        else if (a == 0 && b == 0) // 其实本题不需要计算tn
            tn++;
        else if (a == 1 && b == 0)
            fp++;
        else
            fn++;
    }
    printf("%lf %lf", 1.0 * tp / (tp + fp), 1.0 * tp / (tp + fn));
    return 0;
}
```

Python 参考程序

```
tp = tn = fp = fn = 0
for i in range(int(input())):
    a, b = [int(i) for i in input().split()]
    tp += a * b
    tn += (1 - a) * (1 - b)
    fp += a * (1 - b)
    fn += (1 - a) * b
print(tp / (tp + fp), tp / (tp + fn))
```

星月选课

根据数学期望的性质：$E(a+b)=E(a)+E(b)$，可知两点结论：

1. 每门课可选可不选，选上一定比不选更优，因为数学期望多一项非负数。所以全选能得到最大的期望值
2. 设第 i 门课的期望为 f_i，则全选时，$E(\sum_{i=1}^{n} f_i)=\sum_{i=1}^{n} E(f_i)=\sum_{i=1}^{n} w_i \cdot p_i$

C++ 参考程序

```
#include <bits/stdc++.h>
typedef long long ll;
typedef double db;
#define sc(x) scanf("%lld", &x)
ll w[10010], n;
db p, e;
signed main()
{
    sc(n);
    for (ll i = 1; i <= n; ++i)
        sc(w[i]);
    for (ll i = 1; i <= n; ++i)
    {
        scanf("%lf", &p);
        assert(p >= 0.1 && p <= 1.);
        e += w[i] * p;
    }
    printf("%lf", e);
    return 0;
}
```

Python 参考程序

```python
n = int(input())
w = [int(i) for i in input().split()]
p = [float(i) for i in input().split()]
print(sum(map(lambda i: w[i] * p[i], range(n))))
```

星月学语

解法一：本题可以直接模拟，要寻找的即为 s 里的子串 t 的数量，且满足找到的位置的前后一个位置都不能是字母(可以是空、空格、标点符号)。匹配子串可以用 C 风格的 `strncpy` 函数，判断是否为字母可以用 `isalpha` 或 `islower` 。为了避免处理首位匹配特判，可以在 s 头部插入一个非字母字符。可以用 C 风格字符串或 `std::string` 实现，没有多大差别。时间复杂度为 $O(n \cdot |s| \cdot |t|)$

另一种模拟的思路是先批量把标点符号全部替换为空格，然后把 s, t 首尾加上空格，然后统计 s 里出现了多少次 t 即可。这种方法思维上更加简单，但是实现起来相对更复杂。时间复杂度同上。

解法二：也可以用正则表达式匹配，表达式为 t 加上首尾 `\b` ，即正则表达式里表示单词边界的转义符。统计匹配了多少次即可(本题 Python 该解法只需四行代码)

解法三：也可以使用 KMP 算法匹配，时间复杂度为 $O(n(|s| + |t|))$

解法四：也可以使用字符串哈希匹配，时间复杂度为 $O(n(|s| + |t|))$

> 由于通常不建议使用 Python 写算法题(非算法知识签到题还是可以的)，所以不给出 Python 的 KMP 算法代码；从本题往后涉及算法知识的解法均不提供 Python 参考代码 {>~<}

输入时读行注意读走最后的换行符，否则读 s 的时候会读一个空行而不是目标行。可以用各种方法，如 `getchar`, 占位符 `%*c` ，或 `cin.igore()` 实现。需要注意的是，在绝大多数 OJ ，都不支持 `gets` 函数，因为这是个被新版本 `C/C++` 废弃的函数，请使用 `fgets` 代替。

C++ 模拟

思路一：`isalpha`

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 105
ll n, cnt, ns, nt;
char s[mn], t[mn];
signed main()
{
    scanf("%lld%*c", &n);
    while (n--)
    {
        cnt = 0;
        fgets(s + 1, 101, stdin);
        scanf("%s%*c", t + 1);
        ns = strlen(s + 1);
        nt = strlen(t + 1);
        for (ll i = 1; i <= ns; ++i)
        {
            if (strncmp(s + i, t + 1, nt) == 0 && !isalpha(s[i + nt]) &&
!isalpha(s[i - 1]))
```

```
            {
                ++cnt;
            }
        }
        printf("%lld\n", cnt);
    }
    return 0;
}
```

思路二：替换标点符号

> 来自验题人 bobby285271 的代码

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

void solve()
{
    int n;
    int ans = 0;
    string a;
    getline(cin, a);
    a = ' ' + a + ' ';
    for (auto &i : a)
    {
        if (i == ',' || i == '?' || i == '!' ||
            i == '.' || i == '\'' || i == '"' || i == '-')
        {
            i = ' ';
        }
    }
    string b;
    cin >> b;
    getchar();
    b = ' ' + b + ' ';
    for (int i = 0; i < a.size(); i++)
    {
        if (a.substr(i, b.size()) == b)
        {
            ans++;
        }
    }
    cout << ans << endl;
}

int main()
{
    int t;
    cin >> t;
    getchar();
    while (t--)
    {
        solve();
    }
}
```

C++ 正则表达式

一种写法是正则表达式匹配：

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll n;
string s, t;
signed main()
{
    scanf("%lld", &n), getchar();
    while (n--)
    {
        getline(cin, s);
        cin >> t;
        cin.ignore();
        regex reg("\\b" + t + "\\b");
        smatch m;
        auto lf = s.cbegin(); // begin(),end()参数不匹配
        auto rf = s.cend();
        ll ans = 0;
        for (; regex_search(lf, rf, m, reg); lf = m.suffix().first)
            ++ans;
        cout << ans << '\n' ;
    }
    return 0;
}
```

由于 C++ 正则表达式匹配比较难写，所以可以尝试写容易写的正则表达式替换，然后统计替换了多少次即可。

> 来自验题人 dayuanx 的代码：

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

using namespace std;

using ll = long long;

int main()
{
    int n;
    cin >> n;
    while (getchar() != '\n');
    while (n--)
    {
        string s, t;
        getline(cin, s);
        getline(cin, t);
        regex pattern("\\b" + t + "\\b");
        s = regex_replace(s, pattern, "@");
        int ans = count(s.begin(), s.end(), '@');
        cout << ans << "\n";
    }

    return 0;
```

```
}
```

C++ KMP

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 105
ll n, kmp[mn], ns, nt, ans, j;
char s[mn], t[mn];
signed main()
{
    cin >> n, cin.ignore();
    while (n--)
    {
        cin.getline(s + 1, 101), cin >> (t + 1), cin.ignore();
        ns = strlen(s + 1), nt = strlen(t + 1);
        memset(kmp, 0, sizeof kmp), ans = j = 0;
        for (ll i = 2; i <= nt; ++i)
        {
            while (j && t[i] != t[j + 1])
                j = kmp[j];
            if (t[j + 1] == t[i])
                ++j;
            kmp[i] = j;
        }
        for (ll i = 1, j = 0; i <= ns; ++i)
        {
            while (j > 0 && t[j + 1] != s[i])
                j = kmp[j];
            if (t[j + 1] == s[i])
                ++j;
            if (j == nt) //匹配成功
            {
                ans += (!isalpha(s[i - nt]) && !isalpha(s[i + 1]));
                j = kmp[j];
            }
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

C++ 字符串哈希

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
ll n, cnt;
ull pw[110], p = 131, hs[110], ht;
signed main()
{
```

```
    pw[0] = 1;
    for (ll i = 1; i < 110; ++i)
    {
        pw[i] = pw[i - 1] * p;
    }
    cin >> n, cin.ignore();
    while (n--)
    {
        string s, t;
        getline(cin, s), getline(cin, t);
        s = ' ' + s + ' ', t = ' ' + t + ' ', cnt = 0, ht = 0;
        for (char i : ",?!.'\"-")
        { //也可以像上文解法一样不替换，直接用isalpha
            replace(s.begin(), s.end(), i, ' ');
        }
        for (ll i = 1; i <= (ll)s.size(); ++i)
        {
            hs[i] = hs[i - 1] * p + s[i - 1];
        }
        for (ll i = 1; i <= (ll)t.size(); ++i)
        {
            ht = ht * p + t[i - 1];
        }
        for (ll i = 0; i < (ll)(s.size() - t.size() + 1); ++i)
        {
            if (ht == hs[i + t.size()] - pw[t.size()] * hs[i])
            {
                ++cnt;
            }
        }
        printf("%lld\n", cnt);
    }
    return 0;
}
```

Python 模拟

```
for i in range(int(input())):
    s, t, ans = input(), input(), 0
    s = ' ' + s + ' '
    ns, nt, prev = len(s), len(t), 0
    while True:
        lf = s.find(t, prev)
        if lf == -1:
            break
        if not s[lf + nt:lf + nt + 1].isalpha() and not s[lf -
1:lf].isalpha():
            ans += 1
        prev = lf + nt
    print(ans)
```

Python 正则表达式

```python
from re import findall
for i in range(int(input())):
    s, t = input(), input()
    print(len(findall('\\b' + t + '\\b', s)))
```

星月观星

解法一：枚举每张照片的左上角，然后统计含月亮的每张照片的星星数，求最值即可，时间复杂度 $O((n-k+1)(m-k+1)k^2)$, $m=n=2k$ 时，最大 $O(\frac{nm(n+2)(m+2)}{16})$

解法二：分别维护星星，月亮的二维前缀和，直接查询即可，复杂度 $O(nm)$

C++ 枚举

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 55
char s[mn][mn];
ll n, m, k, ans, cnt, hasMoon;
signed main()
{
    sc(n), sc(m), sc(k);
    for (ll i = 1; i <= n; ++i)
    { //即从s[i][1]地址开始读,使下标和坐标一致
        scanf("%s", s[i] + 1);
    }
    for (ll i = 1; i <= n - k + 1; ++i)
    {
        for (ll j = 1; j <= m - k + 1; ++j)
        {
            cnt = hasMoon = 0;
            for (ll x = i; x < i + k; ++x)
            {
                for (ll y = j; y < j + k; ++y)
                {
                    if (s[x][y] == 'o')
                        hasMoon = 1;
                    else if (s[x][y] == '*')
                        ++cnt;
                }
            }
            if (hasMoon && cnt > ans)
                ans = cnt;
        }
    }
    printf("%lld", ans);
    return 0;
}
```

C++ 前缀和

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 55
char s[mn][mn];
ll n, m, k, a[mn][mn], b[mn][mn], ans;
signed main()
{
    sc(n), sc(m), sc(k);
    for (ll i = 1; i <= n; ++i)
    {
        scanf("%s", s[i] + 1);
        for (ll j = 1; j <= m; ++j)
        {
            a[i][j] += a[i - 1][j] + a[i][j - 1] - a[i - 1][j - 1] + (s[i][j] == '*');
            b[i][j] += b[i - 1][j] + b[i][j - 1] - b[i - 1][j - 1] + (s[i][j] == 'o');
        }
    }
    for (ll i = k; i <= n; ++i)
    {
        for (ll j = k; j <= m; ++j)
        {
            if (b[i][j] - b[i - k][j] - b[i][j - k] + b[i - k][j - k] == 1)
            {
                ans = max(ans, a[i][j] - a[i - k][j] - a[i][j - k] + a[i - k][j - k]);
            }
        }
    }
    printf("%lld", ans);
    return 0;
}
```

Python 枚举

```python
n, m, k = [int(i) for i in input().split()]
s, ans = [input() for i in range(n)], 0
for i in range(n - k + 1):
    for j in range(m - k + 1):
        cnt = hasMoon = 0
        for x in range(i, i + k):
            hasMoon += s[x][j:j + k].count('o')
            cnt += s[x][j:j + k].count('*')
        ans = max(ans, hasMoon * cnt)
print(ans)
```

星月寻人

解法一：多维数组枚举。用三维 `string` 或四维 `char` 数组存每个名字，然后遍历，如果找到所询问的名字输出对应下标，鉴于字符串相等暴力匹配暴力复杂度为 $O(|s|)$，故复杂度为 $O(nmkq|s|)$

解法二：`C/C++` 使用 `std::map` 或 `std::unordered_map`（下文简称 `unmap`），`key` 是名字，`value` 是 (a,b,c) 三元对。复杂度为 $O(nmk|s|+q\log(nmk)|p|)$（对 `map` 而言；若对 `unmap`，没有 $\log$）；对应于 Python，则为 `dict`，复杂度类似。实际结果表明真实用时解法二不如解法一。

一个热知识：对 `map` 或 `unmap` 使用 `[]` 运算符时，若本来不存在要找的键，会原地新建一个元素，返回值是 `value`；而使用 `find` 时，若不存在返回迭代器尾，存在返回迭代器。不会出现不存在就新建的情况。所以假设有很多可能不存在的键来查询时，理论上用 `find` 比 `[]` 更优。在下文 `云烟科技争锋` 这题中(如果你足够砬能做到这题)，该技巧能够缩短数倍的时间用时和十倍多的空间开销。

C++ 枚举

```cpp
#include <bits/stdc++.h>
typedef long long ll;
typedef double db;
#define re register
using namespace std;
#define sc(x) scanf("%lld", &x)
#define mn 42
#define an 22
char s[mn][mn][mn][an], p[an];
ll sn[mn][mn], m, n, k, q, a, b, c;
signed main()
{
    scanf("%lld%lld%lld", &n, &m, &k);
    for (ll i = 0; i < m * n; ++i)
    {
        scanf("%lld%lld%lld", &a, &b, &c);
        sn[a][b] = c;
        for (ll j = 1; j <= c; ++j)
            scanf("%s", s[a][b][j]);
    }
    scanf("%lld", &q);
    while (q--)
    {
        scanf("%s", p);
        ll found = 0, i, j, k;
        for (i = 1; i <= n && !found; ++i)
        {
            for (j = 1; j <= m && !found; ++j)
            {
                for (k = 1; k <= sn[i][j]; ++k)
                {
                    if (strcmp(p, s[i][j][k]) == 0)
                    {
                        found = 1;
                        printf("%lld %lld %lld\n", i, j, k);
                        break;
                    }
                }
            }
        }
        if (!found)
```

```
            printf("not found\n");
        }
        return 0;
}
```

C++ map

> `unmap` 比 `map` 快将近一倍；写法不唯一，当然可以用 struct 或三个 map 来代替 tuple。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
ll n, m, k, q, a, b, c;
char s[22];
map<string, tuple<ll, ll, ll>> h;
signed main()
{
    sc(n), sc(m), sc(k);
    for (ll i = 1; i <= n * m; ++i)
    {
        sc(a), sc(b), sc(c);
        for (ll j = 1; j <= c; ++j)
        {
            scanf("%s", s);
            h[s] = {a, b, j};
        }
    }
    for (sc(q); q--;)
    {
        scanf("%s", s);
        auto r = h.find(s);
        if (r == h.end())
        {
            printf("not found\n");
        }
        else
        {
            tie(a, b, c) = r->second;
            printf("%lld %lld %lld\n", a, b, c);
        }
    }
    return 0;
}
```

Python 枚举

```
n, m, k = [int(i) for i in input().split()]
h = [[[None for z in range(k)] for y in range(m)] for x in range(n)]
for i in range(n * m):
    s = input().split()
    a, b, c = int(s[0]), int(s[1]), int(s[2])
    for ss, j in zip(s[3:], range(c)):
```

```
        h[a - 1][b - 1][j] = ss
for _ in range(int(input())):
    p = input()
    for i in range(n):
        for j in range(m):
            for g in range(k):
                if h[i][j][g] == p:
                    print(i + 1, j + 1, g + 1)
                    p = ''
                    break
            else:  #for: ... else: ... 语法可用于跳出多重循环
                continue
            break  #if-break后else-continue失效,break掉j这层
        else:
            continue
        break  #同理连锁反应break掉i这层
    if p != '':
        print('not found')

```

Python dict

```
n, m, k = [int(i) for i in input().split()]
h = dict()
for i in range(n * m):
    s = input().split()
    a, b, c = int(s[0]), int(s[1]), int(s[2])
    for ss, j in zip(s[3:], range(1, 1 + c)):
        h[ss] = (a, b, j)
for i in range(int(input())):
    p = input()
    try:
        print(h[p][0], h[p][1], h[p][2])
    except KeyError:
        print('not found')
```

弥明爱跑步

前置知识：无

这题的核心和难点在于如何枚举 4 的 A_4^4 个全排列。之后只需把所有情况都枚举一遍然后求符合条件的最小值即可，本题的其他部分代码相信对于能做到这里的你来说应该不是特别大的障碍。

一种容易理解的方法是直接开四层循环。循环中一种检验全排列的方法为：可以证明 $\forall a,b,c,d\in Z, 1\le a,b,c,d\le 4$，若 a,b,c,d 是 $(1,2,3,4)$ 的排列，一定满足：

$$a+b+c+d=1+2+3+4=10,\ abcd=1\cdot 2\cdot 3\cdot 4=24$$

而其他任何情况都不满足。这条性质可以帮助代码的简化。(当然不用这条性质也是可以枚举的，只需能判断 a,b,c,d 里 $1,2,3,4$ 各出现一次即可，排序、数组计数、直接做 10 次判断等方法均可，其中排序参见 Python 参考代码)

> 拓展：这个方法对小于 9 的全排列也都成立，验证程序见此，对 9 的全排列，有反例：

> $$1+2+4+4+4+5+7+9+9=\sum_{i=1}^{9} i=45$$
>
> $$1\cdot 2\cdot 4\cdot 4\cdot 4\cdot 5\cdot 7\cdot 9\cdot 9=\prod_{i=1}^{9} i=362880$$

由于本题精度要求比较高，对 C/C++ 选手，使用 float 可能会导致代码无法过题。请使用 double 完成本题。

> 本题解法很多，也可以使用 DFS 或 C++ 的 `std::next_permutation` 枚举。如果完全没有想法的话，也可以手写一个长为 A_4^4 的常量二维数组记录全排列，然后枚举这个二维数组即可枚举全排列。在这里不给出这三种解法的代码，有兴趣的可自行尝试。

> 附：本题本来是 2021 AK 杯网络赛的备用题目。本题的强化版 `部道乐跑 II` 已发布在 SCNUOJ 公共题库。感兴趣者可以去挑战。

C++ 参考代码 (枚举)

```cpp
#include <stdio.h>
#include <math.h>
int t;
double x[5], y[5], ans, v;
double dis(int i, int j)
{
    return sqrt((x[i] - x[j]) * (x[i] - x[j]) + (y[i] - y[j]) * (y[i] - y[j]));
}
int main()
{
    scanf("%d", &t);
    while (t--)
    {
        ans = 1e9; //设一个比最大输入得到的结果还大的值代表不存在
        for (int i = 0; i <= 4; ++i)
        {
            scanf("%lf%lf", &x[i], &y[i]);
        }
        for (int a = 1; a <= 4; ++a)
        {
            for (int b = 1; b <= 4; ++b)
            {
                for (int c = 1; c <= 4; ++c)
                {
                    for (int d = 1; d <= 4; ++d)
                    {
                        if (!(a + b + c + d == 10 && a * b * c * d == 24))
                        {
                            continue;
                        }
                        v = dis(0, a) + dis(a, b) + dis(b, c) + dis(c, d);
                        if (v >= 2500.0 && v < ans)
                        {
                            ans = v;
                        }
                    }
                }
            }
```

```
        }
        if (ans != 1e9)
        {
            printf("%.1f\n", ans);
        }
        else
        {
            printf("no\n");
        }
    }
    return 0;
}
```

C++ 参考代码 (next_permutation)

```
#include <bits/stdc++.h>
#pragma GCC optimize(2)
using namespace std;
typedef long long ll;
typedef double db;
#define sc(x) scanf("%lld", &x)
#define il inline
#define big 1e18
ll t, x[6], y[6], p[6];
db ans;
signed main()
{
    sc(t);
    while (t--)
    {
        ans = big;
        for (ll i = 0; i <= 4; ++i)
            sc(x[i]), sc(y[i]), p[i] = i;
        //不会dfs/next_permutation的选手可以枚举24种p形成二维数组
        do
        {
            db dis = 0;
            ll a = 0, b;
            for (ll i = 1; i <= 4; ++i)
            {
                b = p[i];
                dis += sqrt((x[a] - x[b]) * (x[a] - x[b]) + (y[a] - y[b]) *
(y[a] - y[b]));
                a = b;
            }
            if (dis >= 2500.0)
            {
                ans = min(ans, dis);
            }
        } while (next_permutation(p + 1, p + 5));
        if (ans == big)
            printf("no\n");
        else
            printf("%.0lf\n", ans);
    }
    return 0;
}
```

C++ 参考代码 (DFS)

> 来自验题人 dakta 的代码

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[6],b[6];
bool is[6];
vector<double>v;
void dfs(int x,int y,int deep,double dis)
{
  if(deep==4){v.push_back(dis);return;}
  for(int i = 2; i <= 5; i ++)
  {
    if(!is[i])
    {
      is[i]=1;
      dfs(a[i],b[i],deep+1,dis+sqrt((a[i]-x)*(a[i]-x)+(b[i]-y)*(b[i]-y)));
      is[i]=0;
    }
  }
}
signed main()
{
  cin.tie(0)->ios::sync_with_stdio(0);
  int T;cin>>T;
  while(T--)
  {
    v.clear();
    for(int i = 1; i <= 5; i++)
      cin>>a[i]>>b[i],is[i]=0;
    dfs(a[1],b[1],0,0);
    sort(v.begin(),v.end());
    if(v.back()>=2500)cout<<(int)(*lower_bound(v.begin(),v.end(),2500)+0.5)
<<'\n';
    else cout<<"no\n";
  }
  return 0;
}
```

Python 参考代码(枚举)

```python
for t in range(int(input())):
    x, y, ans = [], [], 1e9

    def dis(i, j):
        return ((x[i]-x[j])**2+(y[i]-y[j])**2)**0.5
    for i in range(5):
        x0, y0 = [int(i) for i in input().strip().split()]
        x.append(x0)
        y.append(y0)
    for a in range(1, 5):
        for b in range(1, 5):
            for c in range(1, 5):
                for d in range(1, 5):
```

```
                    h = [a, b, c, d]
                    h.sort()
                    if h[0] != 1 or h[1] != 2 or h[2] != 3 or h[3] != 4:
                        continue
                    v = dis(0, a)+dis(a, b)+dis(b, c)+dis(c, d)
                    if v >= 2500.0:
                        ans = min(ans, v)
    if ans != 1e9:
        print('%.f' % ans)
    else:
        print('no')

```

Python 参考代码(DFS)

> 来自 AK 杯网络赛验题人 CLAMorz 的代码

```
def dis(a, b):
    return ((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2) ** .5


def solve(points):
    seen = [0]
    ans = 0

    def dfs(u, length):
        if len(seen) == len(points):
            if length >= 2500:
                nonlocal ans
                if not ans:
                    ans = length
                ans = min(ans, length)
            return
        for v in range(1, len(points)):
            if v in seen:
                continue
            seen.append(v)
            dfs(v, length + dis(points[u], points[v]))
            seen.remove(v)

    dfs(0, 0)
    return ans


t = int(input())
for _ in range(t):
    points = []
    for _ in range(5):
        points.append(tuple(map(int, input().split())))
    ans = solve(points)
    print(round(ans) if ans else "no")
```

弥明破阵法

前置知识：BFS

这是十分经典的 BFS 求连通块的模板题。不难发现术式就是八方向连通块，术系就是块大小相同的连通块的集合。

本题可以遍历整个矩阵，当发现当前 $a_{i,j}$ 是非负数时，从这个点开始 BFS ，每遍历一个点将其从矩阵上删掉，即可以赋值 $a_{i,j}=-1$ 。当相邻且元素符号相同时，就继续入队 BFS 即可。

实现上，注意到 $n,m\le 10^6$ ，所以不能开二维静态数组。而 $n\times m\le 10^6$ ，所以可以开动态数组，用 `new` 或 `vector` 等方法均可，可以选择 10^6 个一维 `vector` 或一个二维 `vector` (代码只展示后者，前者比较简单，若有需要请读者自行实现)。

因为最大的术式复杂度不会超过 $n\times m$ ，所以存储每个术系可以直接使用长为 10^6 的静态数组，下标为 i 代表该术系里术式的复杂度，值代表术系的丰富度。

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
#define sc(x) scanf("%lld", &x)
#define mn 1000010
ll n, m, bin[mn], mx, num;
ll dx[] = {-1, -1, -1, 0, 1, 1, 1, 0}, dy[] = {1, 0, -1, -1, -1, 0, 1, 1};
signed main()
{
    sc(n), sc(m);
    vector<vector<ll>> a(n + 1, vector<ll>(0)); //构造函数(长度,初始值)
    for (ll i = 1; i <= n; ++i)
    {
        a[i].emplace_back(-1);
        for (ll j = 1, x; j <= m; ++j)
        {
            sc(x);
            a[i].emplace_back(x);
        }
    }
    for (ll i = 1; i <= n; ++i)
    {
        for (ll j = 1; j <= m; ++j)
        {
            if (a[i][j] != -1)
            {
                ll cnt = 0, a0 = a[i][j];
                queue<pair<ll, ll>> q;
                q.push({i, j});
                while (!q.empty())
                {
                    auto p = q.front();
                    q.pop();
                    ll x = p.first, y = p.second;
                    if (a[x][y] == -1)
                    {
                        continue;
                    }
                    ++cnt;
                    a[x][y] = -1;
```

```
                    for (ll h = 0; h < 8; ++h)
                    {
                        ll ax = x + dx[h], ay = y + dy[h];
                        if (ax > 0 && ay > 0 && ax <= n && ay <= m && a[ax]
[ay] == a0)
                        {
                            q.push({ax, ay});
                        }
                    }
                }
                ++bin[cnt];
            }
        }
    }
    for (ll i = 1; i < mn; ++i)
    {
        if (bin[i])
        {
            ++num;
            mx = max(mx, bin[i] * i);
        }
    }
    printf("%lld %lld\n", num, mx);
    return 0;
}
```

弥明超时空

前置知识：图，DFS

电路图是无向图，电线是无向边，变电箱是节点，我们来翻译翻译限制条件：

1. 不含自环
2. 任一点与任意其他点连通 $\rightarrow$ 任意两点连通 $\rightarrow$ 连通图
3. 无环 (这同时也要求不能重边，不能自环)

总结下来，即要求：给定无向图是简单无环连通图。

我们再来看看无根树的定义：无向无环连通图。

这也就是说，只要给定是树，就满足限制；否则不满足。

我们可以用任意一点开始 DFS ，如果 DFS 过程不能走回任意已遍历的点，那么就一定是无环的(即满足条件 $1, 3$)。判断遍历与否可以直接标记布尔值数组。具体实现细节参考代码。

当遍历完这次 DFS 后，如果发现每个点都被遍历了一次，那证明只有一个连通分量，是连通图。如果存在未被遍历的点，就是非连通图。

建图可以用邻接表、链式前向星。本题用邻接矩阵的话时空开销是 $O(n^2)$ ，所以不可行。邻接矩阵常用于完全图等稠密图。

本题有多次询问，记得每次询问完后要初始化。

时间复杂度 $O(\sum n + \sum m)$

C++ 链式前向星

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 200010
ll t, n, m, hd[mn], cnt, vis[mn], loop, sum;
struct edge
{
    ll to, nx;
} e[400010];
void adde(ll u, ll v)
{
    e[++cnt] = {v, hd[u]};
    hd[u] = cnt;
}
void dfs(ll u, ll fa)
{
    vis[u] = 1;
    ++sum;
    for (ll i = hd[u], v; i; i = e[i].nx)
    {
        v = e[i].to;
        if (v == fa)
        {
            continue;
        }
        if (vis[v])
        {
            loop = 1;
            return;
        }
        dfs(v, u);
    }
}
signed main()
{
    for (sc(t); t; --t)
    {
        sc(n), sc(m);
        memset(vis, 0, sizeof vis);
        memset(hd, 0, sizeof hd);
        loop = sum = cnt = 0;
        for (ll i = 1, u, v; i <= m; ++i)
        {
            sc(u), sc(v);
            adde(u, v), adde(v, u);
        }
        dfs(1, 0);
        if (loop == 0 && sum == n)
        {
            printf("Chronosphere ready\n");
        }
        else
        {
            printf("Low power\n");
```

```
        }
    }
    return 0;
}
```

C++ 邻接表

> 来自验题人 wumeibanfa 的代码

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

#define ll long long
const int N = 2e5 + 5;

int n, m;
vector<int> e[N];
int col[N];
bool ok = 1;

void dfs(int u, int fa) {
    col[u] = 1;
    for (auto v: e[u]) {
        if (col[v] != -1) {
            if (v != fa) ok = 0;
            continue;
        }
        dfs(v, u);
    }
}

int main(){
    ios::sync_with_stdio(false), cin.tie(0);
    int T; cin >> T;
    assert(T <= 10);
    while (T--) {
        cin >> n >> m;
        assert(n <= (int)2e5 && m <= 1ll * n * (n - 1) && n >= 2);
        for (int i = 1; i <= n; i++) e[i].clear(), col[i] = -1;
        ok = 1;
        for (int i = 1; i <= m; i++) {
            int u, v; cin >> u >> v;
            assert(u >= 1 && u <= n && v <= n && v >= 1);
            e[u].push_back(v), e[v].push_back(u);
        }

        dfs(1, 1);

        for (int i = 1; i <= n; i++) {
            if (col[i] == -1) {
                ok = 0;
            }
        }

        if (ok) cout << "Chronosphere ready\n";
```

```
        else cout << "Low power\n";
    }
    return 0;
}
```

弥明闯异界

前置知识：二分

直接暴力枚举天数 d 和 m，然后逐天计算的话，复杂度是 $O(Tn^2m_0)$，显然不可行。考虑进行优化。

不难发现，设教授 $d(1 \le d \le n)$ 天，教室容量为 $m(1 \le m \le m_0)$，所求即函数 $f(d,m)=\sum_{i=1}^{d}\min(a_i,m)$ 满足 $f(d,m) \ge k$ 的最小的 d 以及该条件下最小的 m。不难发现以 d 为自变量，m 为常数时，为增函数(固定教室容量，增加天数不可能使得总授课学生减少)；而固定 d 为常数，m 为常数时，也为增函数(固定天数，增加教室容量，也不可能使得总授课学生减少)。d,m 均满足单调性，所以都可以用二分答案法来优化。

根据题意，d 越小越好，且 d 相同时 m 越小越好。所以先二分枚举 d，在这个条件下，再二分枚举 m，然后计算一遍 $f(d,m)$，若符合条件则记录，不符合条件则更改 m 的二分边界。若 m 没有符合条件的就更改 d 的二分边界。

时间复杂度为 $O(Tn\log n\log m_0) \le O(\log m_0 \sum n \log \sum n)$。具体实现细节参见代码(二分套二分参考代码)。

事实上，还可以进一步优化。因为 d 越小越好，可以采用这样的策略：假定 $m=m_0$，然后从小到大枚举 d，直到达到目标为止。那么这个 d 一定是最小满足条件的 d。然后以当前 d 为基础，二分枚举 m 并计算金币总数，最差在 m_0 时枚举得到答案。这样只需要单次二分，时间复杂度为 $O(T(n+n\log m_0))=O(Tn\log m_0)=O(\log m_0 \sum n)$。

C++ 参考代码(单次二分)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 100010
ll t, n, m0, k, a[mn];
signed main()
{
    for (sc(t); t--;)
    {
        sc(n), sc(m0), sc(k);
        ll lf = 1, rf = m0, cf, day = -1, m, ans = 0;
        for (ll i = 1; i <= n; ++i)
        {
            sc(a[i]);
            ans += min(m0, a[i]);
            if (day == -1 && ans >= k)
            {
                day = i;
            }
        }
        if (day == -1)
        {
```

```
            printf("tai ruo xiao le, mei you li liang\n");
        }
        else
        {
            while (lf <= rf)
            {
                cf = (lf + rf) >> 1;
                ll res = 0;
                for (ll i = 1; i <= day; ++i)
                {
                    res += min(cf, a[i]);
                }
                if (res >= k)
                {
                    ans = min(ans, res), m = cf, rf = cf - 1;
                }
                else
                {
                    lf = cf + 1;
                }
            }
            printf("%lld %lld %lld\n", day, m, ans);
        }
    }
    return 0;
}
```

C++ 参考代码 (二分套二分)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 100010
ll t, n, m0, k, a[mn];
signed main()
{
    for (sc(t); t--;)
    {
        sc(n), sc(m0), sc(k);
        for (ll i = 1; i <= n; ++i)
        {
            sc(a[i]);
        }
        ll lf = 1, rf = n, cf, day = -1, m, ans;
        while (lf <= rf)
        {
            cf = (lf + rf) >> 1;
            bool ok = false;
            ll lf2 = 1, rf2 = m0, cf2;
            while (lf2 <= rf2)
            {
                cf2 = (lf2 + rf2) >> 1;
                ll cnt = 0;
                for (ll i = 1; i <= cf; ++i)
```

```
                {
                    cnt += min(cf2, a[i]);
                }
                if (cnt >= k)
                {
                    ok = true, m = cf2, ans = cnt, rf2 = cf2 - 1;
                }
                else
                {
                    lf2 = cf2 + 1;
                }
            }
            if (ok)
            {
                day = cf, rf = cf - 1;
            }
            else
            {
                lf = cf + 1;
            }
        }
        if (day != -1)
        {
            printf("%lld %lld %lld\n", day, m, ans);
        }
        else
        {
            printf("tai ruo xiao le, mei you li liang\n");
        }
    }
    return 0;
}
```

云烟团队激励

前置知识：ST表

~~恭喜你能到这里，从这里开始就不是算法基础题了 >_<~~

本题需要求静态区间任意子段的 $\gcd$ 。首先注意一个性质：

$$\gcd(a,b,c)=\gcd(\gcd(a,b),c)$$

那么只需要不断两两求 $\gcd$ 即可完成任务。

> $\gcd$ 满足结合律、封闭性，存在单位元 1 ，题目又是区间查询，不难想到线段树，但是线段树询问的复杂度是 $O(\log n)$ 的，而本题的询问足足有 2.3×10^6 个，这意味着稳稳当当的 TLE。所以常规线段树不能用 (当然如果您足够神犇您可以使用常数接近树状数组的 zkw线段树 卡常过本题，虽然这么做的复杂度仍然是 $O(n\log n+q\log n)$)。

考虑能够实现 $O(1)$ 询问的数据结构。不难发现 ST 表这一数据结构。

其实这道题就是 ST 表的模板题啦……相信会 ST 表的选手应该都知道怎么写。值得一提的是，虽然 $\gcd$ 的复杂度是对数级的，但是本题的复杂度并不是 $O(n\log n\log a+q\log a)$ ，而是 $O(n(\log n+\log a)+q\log a)$ ，具体证明过程见 这里 。

C++ ST表

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef int ll;
#define sc(x) scanf("%d", &x)
#define mn 2000010
#define lg 23
ll n, m, st[mn][lg], l, r, lg2[mn];
signed main()
{
    sc(n), sc(m);
    for (ll i = 1; i <= n; ++i)
    {
        sc(st[i][0]);
    }
    for (ll i = 2; i <= n; ++i)
    {
        lg2[i] = lg2[i / 2] + 1;
    }
    for (ll j = 1; j < lg; ++j)
    {
        for (ll i = 1; i + (1 << j) - 1 <= n; ++i)
        {
            st[i][j] = __gcd(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
        }
    }
    while (m--)
    {
        sc(l), sc(r);
        ll p = lg2[r - l + 1];
        ll ans = __gcd(st[l][p], st[r - (1 << p) + 1][p]);
        printf("%d\n", ans);
    }
    return 0;
}
```

C++ zkw线段树

> 来自验题人 wumeibanfa 的代码

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

#define lson p << 1
#define rson p << 1 | 1
#define pii pair<int,int>
#define ll long long

const int N = 2e6 + 5;
inline int read(){
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-')
```

```
            f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9'){
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
inline void write(int x)
{
    char F[200];
    int tmp=x>0?x:-x ;
    if(x<0)putchar('-') ;
    int cnt=0 ;
       while(tmp>0)
       {
           F[cnt++]=tmp%10+'0';
           tmp/=10;
       }
       while(cnt>0)putchar(F[--cnt]) ;
}
int n, M, q, d[N << 1];

int main(){
    // ios::sync_with_stdio(false), cin.tie(0);
    // scanf("%d %d", &n, &m);
    n = read(), q = read();

    for (M = 1; M < n; M <<= 1);
    for (int i = M + 1; i <= M + n; i++) d[i] = read();

    for (int i = M - 1; i; i--) d[i] = __gcd(d[i << 1], d[i << 1 | 1]);

    for (int i = 1; i <= q; i++) {
        int l, r;
        l = read(), r = read();

        int ans = 0;
        for (l = l + M - 1, r = r + M + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
            if (~l & 1) ans = __gcd(ans, d[l ^ 1]);
            if (r & 1) ans = __gcd(ans, d[r ^ 1]);
        }

        write(ans);
        puts("");

    }
    return 0;
}
```

云烟科技争锋

前置知识：分块、并查集 、离散化

这是一道比较复杂的数据结构题。考虑使用分块，将原数组分为 $\sqrt{n}$ 个块(根据喜好直接分为 $\sqrt{10^5}$ 也行(题解即如此)，下同)，每块长度为 $\sqrt{n}$ 。不放设下标从 0 开始(为了整除方便)，下标为 i 的块(i 也从 0 算起)维护原数组下标范围 $[i\sqrt{n},(i+1)\sqrt{n}-1]$ 。对每一块，维护整块的数值和，并用 `unmap` 维护这一块内每个数出现的次数。

对于每次修改，遍历每一个块，如果这个块内有 t 个 x，那么删掉 `unmap` 里的 x，并标记 y 增加 t 个(注意不是赋值为 t 个，可能本来有 y)。整块数值和先减去 xt 再加上 yt 。每次修改 $\sqrt{n}$ 个块，所以 `unmap` 的复杂度是 $O(\sqrt{n})$， `map` 的复杂度是 $O(\sqrt{n}\log\sqrt{n})$ 。

对于每次查询，定义一个块是整块当且仅当该块维护的下标范围是 $[l,r]$ 的子集；否则是散块。区间 $[l,r]$ 一定可以拆分为至多 $\sqrt{n}$ 个整块以及不多于 2 个散块 (分别是左端 $[l,?]$ 和右端 $[?,r]$)。对于每个整块，直接加上整块数值和，所以整块上单次查询复杂度为 $O(\sqrt{n})$ 。

但是对于散块，该块的数值和并不是散块之和，本来维护的 `unmap` 也用不上。因此，只能暴力地遍历散块的每个值然后累加。因为对原数组 a，每次修改时不可能真的找到每个 $a_i=x$ 改成 y，所以遍历时当前所看到的 a_i 并不一定是最后的值。下面重点分析如何处理散块：

由于 $n,m\le 10^5$，所以所有不同的值最多可能有 2×10^5 个(一开始 10^5，且每次询问改一个新的值)。而值的取值范围为 10^9，为了下文的操作方便，可以考虑离散化，将其压缩至 2×10^5 个需要考虑的单位(可以用 `unmap` 或 `map` 实现)。例如将第 i 个不同的值编号设为 i，为表述简便，设 x 值对应的编号为 h_x 。根据上述定义有 $1\le h_x\le 2\times 10^5$ 。

如果设一个有 2×10^5 个节点的图，节点的编号设为上述的编号(与数值一一对应)，对于每次操作 1，可以视为对编号为 h_x 的节点向 h_y 节点连一条有向边。那么对散块的 $a_i=x$，要找到最终等于什么时，即相当于在图上从 h_x 开始，沿着有向边一直走直到不能走，得到的节点编号就是 a_i 最终被修改成的值。

然而，直接这么做是不可行的。考虑多次操作一(下文简记一次操作 1 为 $(x\to y)$)，例如 $(1\to 2),(2\to 3),(3\to 1),(1\to 2),\cdots$，这样会形成环，永远无法终止。

为了解决上述问题，我们重定义图。设一个长为 $n+m$ 的序列 b，前 n 个元素有 $b_i=a_i$，后 m 个元素有 $b_i=y$ (当然可以不设 m，设为 $c=1$ 的出现次数即可)。接着，重新定义 x 值的 h_x 为其在 b 序列中最后一次出现的下标。如果有操作 $(x\to y)$，y 对应的 b 下标可以直接根据现在是第几次操作直接对应过去。对 x 也能找到 h_x (下文说明如何找)。

初始化时对所有相同的值，每出现一个相同的值 x 就让之前那个相同的值的 h_x 指向新的 h_x，通过 `unmap` 或 `map` 能够实现。这样才能实现所有相同的 x 一并更改为 y 。

然后以 b 的下标范围编号，设 $n+m$ 个点的图。如果 x 存在，每次操作让节点 h_x 指向 h_y，由于 h_y 每次都不一样，所以不可能成环。那么，对散块每个 a_i，只需要找到其 h_{a_i}，再沿着图不断地走就能找到 h_y，再通过下标对应直接取得 $b_{h_y}=y$ (即离散化公式)。

特别地，这样的图还需要考虑一种情况，如果 y 本来存在，还需要让本来的 h_y 向现在的 h_y 连一条边。

对 x 找 h_x，可以用 `unmap` 或 `map` 来维护。 然后对每次 $(x\to y)$，直接对 x 覆盖，如果本来有 x 就会更新 h_x，如果没有就会新建。

由于可能存在诸如： $(1\to 2),(1\to 3)$，这时图就会出现多岔，导致从图上找终点时无法选择。所以规定：当执行操作 $(x\to y)$ 后，直接把图上的点 h_x 锁定，锁定后的点不允许加边。那么当图上没有再出现过新的 1 (如 $(2\to 1)$)时，就不会执行 $(1\to 3)$，直接忽略本次操作。

至此，解决了实现问题，但未解决复杂度问题。虽然通过 `unmap` 从 x 找 h_x 是 $O(1)$ 的(`map` 是 $O(\log(n+m))$)，但是从图上一点找终点，当图退化为链时，可以达到 $O(n+m)$ 。所以接下来解决优化问题：

对上述限制下的图，有如下性质：从任一点出发，能沿着唯一有向边走到确定的终点(孤立点终点是自己)。这个性质显然跟题意是一致的。根据这个性质， 可以用并查集优化这个找终点的过程。每次连一条边时，就把当前弱连通块的根指向 h_y 。即把 h_x 所在并查集弱连通块与 h_y 合并，并使其成为根。这样下来均摊能够把图上找终点优化成 $O(1)$ 。

根据上文推导，下面总结初始化、操作 1 和操作 2 要做的事情。

初始化：

- 初始化分块的块长、边界、块的数值和、块内各数值出现频次

  复杂度 $O(\sqrt{n})$ (`unmap`) 或 $O(\sqrt{n}\log\sqrt{n})$ (`map`)
- 离散化 a_i ，初始化每个值 x 指向的 h_x

  复杂度 $O(n)$ (`unmap`) 或 $O(n\log n)$ (`map`)
- 初始化长为 $n+m$ 的并查集，值相同的 a 数组元素连成块

  复杂度 $O(n+m)$

每次操作 1 需要做的事情如下：

- 将原本 x,y 对应的 h_x,h_y (若存在且未锁定)所在连通块上的根赋值为新 h_y ，然后锁定 h_x 并更新离散化

  复杂度 $O(1)$ (`unmap`) 或 $O(\log(n+m))$ (`map`)
- 更新每个分块的数值和以及数值出现频次

  复杂度 $O(\sqrt{n})$ (`unmap`) 或 $O(\sqrt{n}\log\sqrt{n})$ (`map`)

每次操作 2 需要做的事情如下：

- 求和所有整块数值和

  复杂度 $O(\sqrt{n})$
- 枚举每个散块的最初数值 a_i ，根据下标 i 所在并查集找到其最后对应的 h_y ，然后通过逆向离散化找到 y ，即 a_i 现在的值是 y ，累加求和

  复杂度 $O(\sqrt{n})$ (`unmap`) 或 $O(\sqrt{n}\log(n+m))$ (`map`)

有 m 次操作，总复杂度为 $O(n+m\sqrt{n})$ (`unmap`) 或 $O(n\log n+m\sqrt{n}\log(n+m))$ (`map`)

一个优化细节：使用 `unmap` 或 `map` 来查找时，不要使用 `[]` 运算符，因为不存在时会新建(而对每个分块不存在是绝大多数情况，会使得最快新建 m 次，每次共 $\sqrt{n}$ 块都插入一个元素)，要使用 `.find()` 方法。实践表明，该优化使得内存空间占用少 10 倍，将 `unmap` 从 $2\sim3s$ 优化到 $1s$ 内；将 `map` 从 $5\sim10s$ 优化到约 $2s$ 。

具体实现细节参考代码。

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 50630
#define mb 225
#define mf 100010
ll n, m, c, x, y, ans;
ll b[mb], lf[mb], rf[mb], bel[mn];  //分块专用
ll died[mf], hi[mf], fa[mf], a[mn]; //并查集专用
unordered_map<ll, ll> h, s[mb];
```

```
ll findf(ll x)
{
    while (x != fa[x])
    {
        x = fa[x] = fa[fa[x]];
    }
    return x;
}
void brute(ll lf, ll rf)
{
    for (ll i = lf; i <= rf; ++i)
    {
        ans += hi[findf(i + 1)]; //起点第i+1块的根导出是什么值
    }
}
signed main()
{
    sc(n), sc(m);
    for (ll i = 1; i < mf; ++i)
    {
        fa[i] = i;
    }
    for (ll i = 0; i < n; ++i)
    {
        sc(a[i]);
        if (h[a[i]] == 0) //首次出现a[i]
        {
            h[a[i]] = i + 1;  //根据a[i]值可以找到第i+1块
            hi[i + 1] = a[i]; //第i+1块为并查集根可以导出a[i]
        }
        else //不是第一次出现a[i]
        {
            fa[i + 1] = findf(h[a[i]]); //将第i+1块并入a[i]值所在块
        }
    }
    for (ll i = 0; i < mb; ++i)
    {
        lf[i] = i * mb;
        rf[i] = (i + 1) * mb - 1;
    }
    for (ll i = 0; i < n; ++i)
    {
        bel[i] = i / mb;
        b[bel[i]] += a[i];
        ++s[bel[i]][a[i]];
    }
    for (ll j = n + 1; j <= n + m; ++j)
    {
        sc(c), sc(x), sc(y);
        if (c == 1)
        {
            if (h[x] == 0 || died[h[x]]) //未出现过x或x被移走了
            {
                continue;
            }
            if (!(h[y] == 0 || died[h[y]])) //本来有y值且没有被移走
            {
                fa[findf(h[y])] = j; //本来的y值所在块归到新的并查集块j
```

```
            }
            fa[findf(h[x])] = j; //本来的x值所在块归到第j块
            died[h[x]] = 1;      // x被移走了
            hi[j] = y;           //第j块为并查集根可以导出值y
            h[y] = j;            //更新值y可以找到的并查集块
            for (ll i = 0; i < mb; ++i)
            {
                ll t = s[i][x];
                b[i] = b[i] - x * t + y * t;
                s[i][x] = 0, s[i][y] += t;
            }
        }
        else if (c == 2)
        {
            --x, --y, ans = 0;
            if (bel[x] == bel[y])
            {
                brute(x, y);
            }
            else
            {
                brute(x, rf[bel[x]]);
                brute(lf[bel[y]], y);
            }
            for (ll i = bel[x] + 1; i < bel[y]; ++i)
            {
                ans = ans + b[i];
            }
            printf("%lld\n", ans);
        }
    }
    return 0;
}
```

云烟蓝星对决

前置知识：博弈论 Nim 游戏

先解决 b_i 的计算。如果使用暴力计算，可以二进制枚举所有子集然后求和，复杂度为 $\mathrm{O}(n2^n)$，考虑使用组合数学优化。从集合里提取出第 i 个数，还剩下 $n-1$ 个数，每个数都有选与不选两种情况，根据乘法原理，一共有 2^{n-1} 个子集。所有这些子集都加上第 i 个数，即第 i 个数可以贡献 2^{n-1} 次。设 $p=1009$。因此，有：

$$b_i = 2^{n-1}\sum_{i=1}^{n} a_i \bmod p$$

暴力计算上面的式子，求幂、求和复杂度均为 $\mathrm{O}(n)$，总复杂度为 $\mathrm{O}(n^2)$。

可以预处理(累积) 2 的 1 到 n 次幂，且预处理(累积) $\forall 1 \le i \le n$，前 i 项和 $\sum_{j=1}^{i} a_j \bmod p$，则总复杂度为 $\mathrm{O}(n+n+n)=\mathrm{O}(n)$。

双方均采取最优策略，即这是一个**博弈论**问题。题意转化为：有 n 堆石子，最开始第 i 堆有 b_i 个。双方轮流拿石子，每次只能从一堆里拿任意正整数个。轮到谁无法拿谁失败，另一方获胜。这就是经典的**Nim游戏**。结论是：当 b_i 异或和不为 0 时，先手必胜，否则先手必败。即 $b_1 \oplus b_2 \oplus \cdots \oplus b_n \neq 0$ 时先手必胜，否则先手必败。

考虑找到一种策略，总回合数不超过 $2n$ 。结论为必然能找到这样的策略。具体策略如下：

1. 若当前异或和 x 不为零，当前回合方必胜，当前方目的是将异或和变为 0 ，使得下一回合方异或和为零必败。

   > 如果不这么做，下一回合方可以将异或和变为零，之后己方无论如何操作都会改变异或和使其不为零，对方再将异或和变为零，直到取完(取完异或和也是 0)，将变为下一回合方必胜。

   根据异或性质，必然能找到一个数 b_i ，使得 $b_i \oplus x < b_i$ ，这等价于 $b_i - (b_i \oplus x) > 0$。那么将 b_i 更改为 $b_i \oplus x$ ，即在第 i 堆取走 $b_i - (b_i \oplus x)$ 个石子。由于 $x \neq 0$ ，所以 $b_i \oplus x \neq b_i$ (只有 0 异或一个数得它自身)，也就是说这种取法不可能取走 0 个石子。

   此时，等效于取走 b_i 个，再放回 $b_i \oplus x$ 个，新异或和为 $x \oplus b_i \oplus (b_i \oplus x) = 0$ ，即这种取法必然会让原本异或和变为零。

   必然能找到这样的数 b_i ，理由如下：设 x 的最高位为 j ，则一定在 b 数组里存在一个数 b_i ，第 j 位为 1 ，否则 x 的第 j 位不可能为 1 。那么 b_i 与 x 异或后，大于 j 的位不变 (因为 x 的这部分全 0 ， 0 异或任何数位得到本身)，第 j 位变为 0 ，那么小于 j 的位无论如何变化，最终结果都不会大于 b_i ，即 $b_i \oplus x < b_i$ 得证，即一定会取走正数个石子。当且仅当 $b_i = x$ 时把 b_i 全部取走。读者可自己模拟，容易得知。

   根据分析，可知，只要找到第 j 位是 1 的数即可。由于对 p 取模，最大值不会超过 p ，则有至多 $\lceil \log_2 p \rceil$ 二进制位，可以设 $\lceil \log_2 p \rceil$ 个 set ，第 k 个 set 存所有第 k 二进制位为 1 的数的下标。维护这些 set ，即可以 set 的复杂度实现该操作。则该回合需要做的事情如下：

   求 j ，依据 j 找任意一个 i ，将 b_i 更改为 $b_i \oplus x$ ，将 x 更改为 0 ，若 b_i 更新后为零，堆数减一并从 set 里删除 i

   单次操作复杂度为求 j 复杂度加上查找 i 复杂度加上 $\log_2 p$ 个 set 单点删除的复杂度，由于 set 用红黑树实现，复杂度是对数的，即总复杂度为
   $\mathrm{O}(\log p + \log n + \log p \log n) = \mathrm{O}(\log p \log n)$

   该回合操作后，$x = 0$ ，堆数可能减一，也可能不变

2. 若当前异或和 x 为零，当前回合方必败。必败状态下无论怎么操作，都不会导致胜利。为了让总回合数最小，可以直接取完任意一堆石子。即找到第一个非空 set ，取任一元素 i ，删掉 i ，设 $b_i = 0$ ，由于取走了 b_i 个石子，所以 $x = x \oplus b_i = 0 \oplus b_i = b_i$ ，石子堆数减一

   最坏情况下要找 $\log p$ 次找到非空，然后取 set 元素并删除 $\log p$ 个 set 的单个元素，总复杂度为
   $\mathrm{O}(\log p + \log n + \log p \log n) = \mathrm{O}(\log p \log n)$

   该回合操作后，$x \neq 0$ ，堆数减一

不难发现，上述操作总是使得异或和为零与不为零间隔出现。异或和不为零必然删一堆石子；最优回合数为异或和不为零时每次都删石子，总回合数为 n ；最差回合数为异或和不为零时每次都不删石子，总回合数为 $2n$ 。所以该策略下，回合数取值范围是 $[n, 2n]$ ，所以必然有解。

最多有 $2n$ 个回合，所以求回合情况的总复杂度为 $\mathrm{O}(2n \log p \log n)$ 。发现 $\log p \log n \approx 496$ ，所以可以过题。

总时间复杂度为 $\mathrm{O}(n + 2n \log p \log n) = \mathrm{O}(n \log p \log n)$

> 也可以用 $unordered\_set$ 代替 set ，最后结果快于 set 。

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 100010
#define digit 30
ll n, a[mn], b[mn], p = 1e9 + 7, xsum, m, x, y;
```

```cpp
vector<pair<ll, ll>> ans;
set<ll> s[digit + 3];
void modify(ll pos, ll v, ll ope) // ope=1 插入; ope=-1删除
{
    for (ll i = 0; i < digit; ++i)
    {
        if (v & 1)
        {
            if (ope == 1)
            {
                s[i].insert(pos);
            }
            else
            {
                s[i].erase(pos);
            }
        }
        v >>= 1;
    }
}
signed main()
{
    scanf("%lld", &n);
    assert(n >= 1 && n <= 1e5);
    for (ll i = 1, n2 = 1; i <= n; ++i, n2 = n2 * 2 % p)
    {
        scanf("%lld", &a[i]);
        assert(a[i] >= 1 && a[i] < p);
        a[i] = (a[i] + a[i - 1]) % p;
        b[i] = n2 * a[i] % p;
        modify(i, b[i], 1);
        xsum ^= b[i];
    }
    printf(xsum ? "first wins\n" : "second wins\n");
    printf("yes\n");
    for (m = n; m;)
    {
        if (xsum)
        {
            ll j = digit - 1; // xsum最高位
            for (; j >= 0; --j)
            {
                if (xsum & (1 << j))
                {
                    break;
                }
            }
            ll i = *s[j].begin(); //任取一个
            modify(i, b[i], -1);
            ans.push_back({i, b[i] - (xsum ^ b[i])});
            b[i] = b[i] ^ xsum;
            if (b[i])
            {
                modify(i, b[i], 1);
            }
            else
            {
                --m;
```

```
            }
            xsum = 0;
        }
        else
        {
            for (ll h = 0; h < digit; ++h)
            {
                if (s[h].size())
                {
                    ll i = *s[h].begin();
                    ans.push_back({i, b[i]});
                    modify(i, b[i], -1);
                    xsum = b[i];
                    --m;
                    b[i] = 0;
                    break;
                }
            }
        }
    }
    printf("%lld\n", ans.size());
    for (auto i : ans)
    {
        printf("%lld %lld\n", i.first, i.second);
    }
    return 0;
}
```