ACM 第二场新生赛题解

Peterlits Zo

2020 年 12 月 19 日

目录

1 1578 题 – pzgg 玩炉石

1.1 题面

1.1.1 Time Limit

5s

1.1.2 Memory Limit

128M

1.1.3 题目

小心你的魔法把你给毁了。

《炉石传说》是 pzgg 最喜欢玩的游戏 , 并且 pzgg 已经达到了炉火纯青的地步。前两天，炉石的策划师给 pzgg 打电话，告知他要出一张新的魔法卡牌，想听听 pzgg 的意见。卡牌的作用如下:

假设敌方战场上有 n 只随从，第 i 只随从（ $1 \leq i \leq n$ ）具有生命值 a_i 。

当 $n \geq 3$ 时，己方英雄可以打出该卡牌，然后选择一个敌方随从 i（ $2 \leq i \leq n-1$ ），可以直接将该随从秒杀 (之后移出战场) ，并为己方英雄回复 $a_{i-1} \times a_i \times a_{i+1}$ 的血量。

为了测试这张卡牌会不会违反游戏平衡，策划师给了 pzgg 若干个战场，并且给了 pzgg 无数张该魔法卡牌，策划师想知道在每个战场上，己方英雄最多可回复多少滴血量。

pzgg 是 acm 高手，觉得这太简单了 , 所以现在他想来考考聪明的你。

1.1.4 输入

输入包括 $2 \times T + 1$ 行;

第一行包括一个整数 T（ $1 \leq T \leq 10$ ），即以下有 T 个战场需要你去求出结果;

接下去 T 组数据，每组数据有两行，第一行是一个整数 n （ J1 $\leq n \leq 200$ ）表示场上有 n 个随从，第二行有 n 个数，第 i 个数 a_i（ $1 \leq a_i \leq 1 \times 10^5$ ）表示第 i 个随从的生命值。

1.1.5 输出

对于每组数据输出一行，表示己方英雄可以最多回复的血量。

1.1.6 示例

输入:

```
2
3
1 2 3
4
1 2 3 4
```

输出:

```
6
32
```

1.2 题解

动态规划: $dp[i][j]$ 表示对 $a[i], a[i+1], ..., a[j]$ 这一段进行操作时所获得的最大价值, 即消除 $a[i+1], a[i+2], ..., a[j-1]$ 这些值并留下 $a[i], a[j]$ 所获得的最大价值。

状态转移方程: $dp[i][j] = max(dp[i][k] + dp[k][j] + a[i] \times a[k] \times a[j])\ for\ k\ from\ i+1\ to\ j-1$ 。

先枚举长度再枚举开始点进行区间 dp 。

这里就不细讲区间 dp 了，不了解区间 dp 的同学可以去学习一下。

时间复杂度: $O(n^3)$ 。

2 1579 题 – peter 的质数分解

2.1 题面

2.1.1 Time Limit

2s

2.1.2 Memory Limit

128M

2.1.3 题目

有一天老彼得对小彼得说:“小彼得，我这里有 t 个数组，每一个数组 A_j（$1 \le j \le t$），不妨设数组 A_j 的长度为 L_j，数组的上界为 R_j（上界不一定在数组里，但是大于等于所有的数组元素），那么我可以断言它的每一个元素 a_i（$1 \le i \le L_j$）满足 $2 \le a_i \le R_j$。”

老彼得知道小彼得最近学习了有手就行的算数基本定理（即: 任何整数都存在，且只存在一种的分解为质数乘积的形式），老彼得的妻子安娜· 卡特琳娜因为一个让人羞愤的理由离开了老彼得，老彼得突然对儿子的教育燃起了新的热情，他补充到:“我知道，给定一个数 $E = p_1^{b_1} \cdot p_2^{b_2} \cdot \cdots \cdot p_n^{b_n}$，它有且只有这么一种表达形式，所以 $S(E) = b_1 + b_2 + \cdots + b_n$ 的值也是唯一的，比如说 $72 = 2^3 \times 3^2$，那么 $S(72) = 3 + 2 = 5$。小彼得，我希望你把这个数组对应的 S 值求出来。

“小彼得，我希望你好好学习，要多看看一些数学书什么的！比如《初等数论》！在这周五我会来检查你的作业情况的!”

但是在周五，小彼得收到了一台由 AMD 强力驱动的，并搭载了 Ubuntu 的计算机，根本无心写那个即愚蠢又简单的、有手就行的问题。请问你能帮帮他吗?

2.1.4 输入

输入的第一行包含了一个整数 t（$1 \le t \le 10$），代表了有 t 个测试样例，每一个测试样例具有如下的格式:

每一个测试样例的第一行包含了两个整数，分别代表了 A_j 数组的数组长度 L_j（$1 \le L_j \le 5 \times 10^2$，且 $1 \le t \times L_j \le 5 \times 10^4$），和 R_j（每一个测试样例的数字范围边界，$1 \le R_j \le 1 \times 10^9$）。

每一个测试样例的第二行包含 L_j 个元素 $a_1, a_2, \ldots, a_{L_j}$，表示 A_j，对于任意 $i \in [1, L_j]$，有: $a_i \le R_j$。

2.1.5 输出

输出 t 个结果, 每个结果对应于输入的每一个测试样例。对于每一个结果包含了 L_j 个结果 $b_1, b_2, \ldots, b_{L_j}$。其中 b_i 为对应的 $a_i = p_1^{c_1} \cdot p_2^{c_2} \cdot \cdots \cdot p_n^{c_n}$ 中所有素数对应的指数之和 $S(a_i) = c_1 + c_2 + \cdots + c_n$。

2.1.6 示例

输入:

```
1
10 100
3 12 4 34 25 78 90 45 23 21
```

输出:

```
1 3 2 2 2 3 4 3 1 2
```

2.1.7 说明

对于 3，有：$3 = 3 \Longrightarrow S(3) = 1$。
对于 12，有：$12 = 2^2 \cdot 3 \Longrightarrow S(12) = 3$。
对于 4，有：$4 = 2^2 \Longrightarrow S(4) = 2$。
对于 34，有：$34 = 2 \cdot 17 \Longrightarrow S(34) = 2$。
对于 25，有：$25 = 5^2 \Longrightarrow S(25) = 2$。
对于 78，有：$78 = 2 \cdot 3^2 \cdot 13 \Longrightarrow S(78) = 3$。
对于 90，有：$90 = 2 \cdot 3^2 \cdot 5 \Longrightarrow S(90) = 4$。
对于 45，有：$45 = 3^2 \cdot 5 \Longrightarrow S(45) = 3$。
对于 23，有：$23 = 23 \Longrightarrow S(23) = 1$。
对于 21，有：$21 = 3 \cdot 7 \Longrightarrow S(21) = 2$。

2.2 题解

不保证题解的完全正确，如果有错误、或希望补充，请访问 *https://github.com/PeterlitsZo/Answer* 修改、评论

对于每一个测试样例，我们有上界 R 和数据 $a_1, a_2, \ldots, a_L$。

2.2.1 筛法

使用筛法，我们可以在 $O(R + L)$ 的时间复杂度中完成操作（证明略）。但是明显时间复杂度度过大。

2.2.2 朴素分解质因数

本来这个是过不去的，但是学长说不能出太难了，所以改了一下，这个应该能过得过去。

注意，因为 米勒罗宾 *+rho* 算法只适用于 S 值较小的大合数分解（因为它的常数太大了），所以更改了数据之后的，它的解只能用朴素的分解质因数的方法

利用反证法我们可以知道, 如果一个数 N 有非平凡因子 a，那 N 一定存在一个非平凡因子 $b \le \sqrt{N}$。

> 如果 $a \le \sqrt{N}$，那么令 $b = a$；如果 $a > \sqrt{N}$，因为 a 为 N 的因子，有存在：$ab = N$，其中 b 即为所求：b 为 N 的因子，且 $b \le \sqrt{N}$。

我们可以分类讨论：如果 N 不存在素因子 $a > \sqrt{N}$，那么我们可以在 $[2, \lfloor\sqrt{N}\rfloor]$ 的循环中解出所有的素因子；如果 N 存在素因子 $a > \sqrt{N}$，那么可以证明（见下）满足这个条件的素因子 a 有且只有一个，我们可以在循环之后判断。

如果有一个素因子 $a > \sqrt{N}$，那么 $\frac{N}{a} \leq \sqrt{N}$，因为素数整除性质，如果有另一个素因子 $b \mid N$，那么有 $b \mid a$，或者 $b \mid \frac{N}{a}$。

如果 $b \mid a$，因为素数的性质，则 $a = b > \sqrt{N}$，有 $ab > N$，矛盾；如果 $b \mid \frac{N}{a}$，那么有 $b \leq \frac{N}{a} < \sqrt{N}$。

综上，如果有一个素因子 $a > \sqrt{N}$，那么有且只有素因子 a 满足 $a > \sqrt{N}$。

可以写代码如下：

```cpp
using ll = long long;
ll solve(ll N) {
    assert (N >= 2);
    ll result = 0;
    for (ll i = 2; i*i < N; i ++) {
        ll sub = 0;
        if (N % i == 0) {
            while(N % i == 0) N /= i, sub ++;
        }
        result += sub;
    }
    if (N > 1) {
        result ++;
    }
    return result;
}

/*
int main() {
    cout << solve(ll(1e9)+7);
}
*/
```

或者 python 代码

```python
def RR(num: int):
    if num <= 1:
        return 0
    result = 0
    for i in range(2, num):
        if i**2 > num:
            break
        while num % i == 0:
            num /= i
            result += 1
    if num != 1:
        result += 1
    return result

def solve():
    _, _ = [int(i) for i in input().split()]
    nums = [int(i) for i in input().split()]
    for i, v in enumerate(nums):
        nums[i] = RR(v)
    print(*nums)

if __name__ == '__main__':
    T = int(input())
    for i in range(T):
        solve()
```

分析可知，平均的时间复杂度为 $O(\sqrt{N})$。如果每一个元素分布得比较平均，则一个测试样例的时间复杂度应为 $O(LR^{\frac{1}{2}})$（需要一点微积分知识，证明略），时间复杂度还是有点大。

2.2.3 优化的朴素分解质因数

我们注意到 $[2, \lfloor\sqrt{N}\rfloor]$ 的循环中，只有 i 为素数时才可能满足 $i \mid N$（证明略）。给定了上界 R，我们可以用欧拉筛在 $O(\sqrt{R})$ 的时间复杂度中筛出所有的小于等于 $\sqrt{N}$ 的素数（证明略），根据素数分布定理

，$\sqrt{N}$ 的素数数量约为 $\frac{\sqrt{N}}{\ln\sqrt{N}}$。我们可以在 $O(\sqrt{R}+L\frac{\mathrm{Ei}(\frac{3\ln R}{2})}{R})$ 的复杂度中完成（由 https://www.wolfra-malpha.com/ 给出，我也不大会），$\mathrm{Ei}(x)$ 可以近似地看作 e^x，那么有近似 $O(\sqrt{R}+LR^{\frac{1}{2}})$。复杂度没有变化，但的确有明显的优化，常数比较小。

```
using ll = long long;
template<typename T>
using vc = vector<T>;

list<ll> get_primes(ll R) {
    vc<bool> is_prime(R, true);
    list<ll> primes;
    for (ll i = 2; i <= R; i ++) {
        if (is_prime[i]) {
            primes.push_back(i);
        }
        for (auto j = primes.begin(); j != primes.end(); i ++) {
            if (i * (*j) >= R) break;
            is_primes[i * (*j)] = false;
            if (i % *j == 0) break;
        }
    }
    return primes;
}

ll solve(ll N, vc<ll>& primes) {
    assert (N >= 2);
    ll result = 0;
    for (auto i = primes.begin(); (*i)*(*i) < N; i ++) {
        ll sub = 0;
        if (N % *i == 0) {
            while(N % i == 0) N /= *i, sub ++;
        }
        result += sub;
    }
    if (N > 1) {
        result ++;
    }
    return result;
}

/*
int main() {
    list<ll>&& primes = get_primes((ll)sqrt(1e9 + 7) + 10);
    cout << solve(ll(1e9)+7, primes);
}
*/
```

2.2.4 rho 算法与 miller-rabin 素数检测

rho 算法是一种用来分解非平凡因子的随机算法，平均的时间复杂度较低。关于 rho 算法，可以查阅 https://www.cs.colorado.edu/ srirams/courses/csci2824-spr14/pollardsRho.html 或者 https://oi-wiki.org/math/polla rho/。

对于数 N，如果它不是素数的话，那我们可以使用 rho 算法将它分解为两个非平凡因子 p 与 $q=\frac{N}{p}$，有：$S(N)=S(p)\times S(q)$。

出人意料的是，虽然目前没有一个能在多项式复杂度里分解质因数的算法（不过网上说量子计算机可以在多项式的时间复杂度里分解质因数?），但是却已经可以在多项式里证明一个数是否为素数。RSA 算法就是依托于两者难度不匹配的情况设计了一个简单的公私匙系统保证通话安全。这里使用易于实现的 miller-rabin 随机算法。miller-rabin 算法可以在 https://oi-wiki.org/math/prime/#miller-rabin 进行学习。它的时间复杂度大抵在 $O(lnn)$ 左右。

```
using ll = long long;

ll fmul(ll a, ll b, ll N) {
    ll result = 0;
    while (b) {
```

```
        if (b & 1) {
            result = (result + a) % N;
        }
        b >>= 1, a = a * 2 % N;
    }
    return result;
}

ll fpow(ll b, ll p, ll N) {
    ll tmp = b, result = 1;
    while (p) {
        if (p & 1) {
            result = fmul(result, tmp, N);
        }
        p >>= 1, tmp = fmul(tmp, tmp, N);
    }
    return result;
}

const ll TEST_TIMES = 20;
bool is_prime(ll N) {
    if (N <= 1) return false;

    ll s=0, d = N-1;
    while (d % 2 == 0) d/=2, s++;
    for (int _ = 0; _ < TEST_TIMES; _ ++) {
        ll a = fmul(rand(), rand(), N-2) + 2;
        ll s_ = s, d_ = d;
        ll R = fpow(a, d_, N);
        while (s_ >= 1 && R != N-1 && R != 1) {
            R = fmul(R, R, N);
            s_ --;
        }
        if (s_ == 0) return false;
    }
    return true;
}
```

不过，这种素数检测方法的常数较高，在数较小的时候不比优化后的试除法，但是题目给的数据范围较大，所以应该使用这种素数检测方法，下面是仅供参考的优化试除法：

```
using ll = long long;

bool is_prime(ll N) {
    if (N <= 1) return false;
    if (N <= 3) return true;
    for (ll i = 5; i * i <= N; i += 6) {
        if (N % i == 0 || N % (i + 2) == 0) {
            return false;
        }
    }
    return true;
}
```

接下来是 rho 算法，它期望得到一个合数，并且会返回一个非平凡因子，在函数体里用到了一些上面的函数，和 ‘gcd‘ 函数：

```
ll gcd(ll a, ll b) {
    return a ? gcd(b % a, a) : b;
}
```
```

下面是 rho 算法的定义：
``` C++
ll rho(ll N) {
    while (true) {
        ll c = fmul(rand(), rand(), N-1) + 1;
        ll l = 2, cnt = 0;
        ll x = 0, y = 0;
        while (true) {
            x = (fmul(x, x, N) + c) % N;
            if (x == y) break;
            ll G = gcd(abs(x - y), N);
```
```

```
            if (G == N) break;
            if (G != 1) return G;
            if ((++cnt) == l) {
                y = x, cnt = 0, l *= 2;
            }
        }
    }
}
```

使用 rho 和检测素数的算法 is_prime，时间复杂度大约在 $O(N^{\frac{1}{4}})$。

2.2.5 参考代码

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

ll fmul(ll a, ll b, ll N) {
    ll tmp = a, result = 0;
    while (b) {
        if (b & 1) {
            result = (result + a) % N;
        }
        b >>= 1, a = (a << 1) % N;
    }
    return result;
}

ll fpow(ll b, ll p, ll N) {
    ll tmp = b, result = 1;
    while (p) {
        if (p & 1) {
            result = fmul(result, tmp, N);
        }
        p >>= 1, tmp = fmul(tmp, tmp, N);
    }
    return result;
}

bool is_prime(ll n) {
    if (n <= 20) {
        for (ll i : {2, 3, 5, 7, 11, 13, 17, 19})
            if (n == i) return 1;
        return 0;
    } else for (ll i : {2, 3, 5, 7, 11, 13, 17, 19}) {
        if (n % i == 0) return 0;
    }

    if (n < 3) return n == 2;
    ll a = n - 1, b = 0;
    while (a % 2 == 0) a /= 2, ++b;
    for (ll i = 1, j; i <= 16; ++i) {
        ll x = rand() % (n - 2) + 2, v = fpow(x, a, n);
        if (v == 1 || v == n - 1)
            continue;
        for (j = 0; j < b; ++j) {
            v = fmul(v, v, n);
            if (v == n - 1) break;
        }
        if (j >= b) return 0;
    }
    return 1;
}

ll gcd(ll a, ll b) {
    return a ? gcd(b % a, a) : b;
}

ll rho(ll N) {
    while (true) {
        ll c = fmul(rand(), rand(), N - 1) + 1;
        ll l = 2, cnt = 0;
        ll x = 0, y = 0;
```

```
61          while (true) {
62              x = (fmul(x, x, N) + c) % N;
63              if (x == y)
64                  break;
65              ll G = gcd(abs(x - y), N);
66              if (G == N) break;
67              if (G != 1) return G;
68              if ((++cnt) == l)
69                  y = x, l *= 2, cnt = 0;
70          }
71      }
72  }
73
74  ll rr(ll N) {
75      if (is_prime(N)) {
76          return 1;
77      } else {
78          ll R = rho(N);
79          return rr(R) + rr(N / R);
80      }
81  }
82
83
84  #define F(X) (#X) << ": " << (X) << ", "
85
86  int main() {
87      ll t, n, _, N;
88      for(cin >> t; t--; ) {
89          cerr << F(t) << endl;
90          for (cin >> n >> _; n --; ) {
91              cin >> N;
92              cerr << F(n) << F(_) << F(N) << endl;
93              cout << rr(N) << " ";
94          }
95          cout << endl;
96      }
97  }
```

3 1580 题 – cgg 的秀发

3.1 题面

3.1.1 Time Limit

2s

3.1.2 Memory Limit

128M

3.1.3 题目

“啊!!! 我的秀发”

众所周知，cgg 有一头乌黑透亮的秀发。在期中复习前夕，cgg 在自习室复习计网时又掉了一些头发，这让他十分伤心。巧合的是，cgg 掉落下来的头发有规律的排成了 '(' 和 ')' 组成的序列，也就是左右括号。作为一个字符串选手，cgg 最不能容忍这个序列不是合法括号序列了，你帮助他使得这个括号序列变为合法括号序列吗?

对于给定的括号序列，你可以进行任意次数以下操作:

1. 删除字符串中任意一个字符，消耗能量 a 点。举例: '())'→'()'
2. 将字符串的最前面一个字符移动到最后面，消耗能量 b 点。举例: ')()()'→'()())'

以上操作不限顺序，不限次数。

合法括号序列:

1. 空字符串是合法括号序列
2. 如果 s 是合法括号序列，'('+s+')' 也是合法括号序列
3. 如果 s 和 t 是合法括号序列，那么 s+t 也是合法括号序列

例：'(()())()'，'(())'，'()()'，'()' 是合法括号序列，'())'，')('，')' 不是合法括号序列

输入保证字符串中仅含有左括号 '(' 和右括号 ')' 你可以消耗尽量少的能量使其变为合法括号序列吗?

输出消耗的最小能量，若无法消除，输出 -1 。

3.1.4 输入

第一行 t，表示共有 t 组样例

每组输入两行，第一行为 l，a，b，$1 \le l \le 100000$，$1 \le a$，$b \le 1000000000$，分别为 cgg 掉落头发的根数，进行 1 操作需要的能量，进行 2 操作需要的能量，第二行为一个字符串，输入保证字符串中仅含有'('，')'。

输入保证 $\sum l \le 100000$

3.1.5 输出

对于每组样例输出需要消耗的最小能量，若无法消除输出 -1 。

3.1.6 示例

输入:

```
3
5 2 1
))((( 
2 5 12
)(
2 1 10
()
```

输出:

```
4
10
0
```

首先对于两种操作，即将最左边的字符移动到最右边和删除一个字符，两者是完全独立的，两个操作之前不会相互影响，所以可以看做：先将最左边的 i 个字符移动到最右边，再对这个字符串进行删除操作，求出对于每个 i 所需的最小代价，取最小值，即为答案。注意：答案不会是 -1，因为你总能够删除所有字符使其变为空串 (空串也是合法括号序列)

这里有一个小技巧, 你可以将两个相同的字符串进行拼接, 像滑动窗口一样每次向右滑动一个, 即可得到所有情况。例:$abcbc \to abcbcabcbc$, 所有的情况为 $[abcbc]abcbc, a[bcbca]bcbc, ab[cbcab]cbc, abc[bcabc]bc, abcb[cabcb]c$

所以现在问题变成：对于一段括号序列仅进行删除操作，如何使得删除次数最少并使得其变为合法括号序列?

记 $f(s)$ 为使字符串 s 变为合法括号序列的最小删除次数, 由上面的讨论可知答案为:$i \times b + f(s_i),\ for\ i\ from\ 0\ to\ n-1$，$s_i$ 为进行完移动操作以后的字符串。

对于一个括号序列，显然可以 $O(len(s))$ 判断最小删除次数，方法：从前往后扫，遇到相匹配的 (和) 就消除，最终变为)...)(...(的形式，维护当前不匹配的) 和最后剩下的 (即可，如果每次都这样判的话，总复杂度为 $O(n^2)$ ，会 tle

对于上面的方法，他的一个等价方法是：

1. (代表 1 ，) 代表 -1 ，用一个数组 a 维护当前的前缀和。比如))()(对应的 a 为 $[-1,-2,-1,-2]$

2. 当前删除的最小字符个数满足

$$num=\begin{cases}a[n] & if\ min(a)\geq 0\\ a[n]-2\times min(a) & if\ min(a)<0\end{cases}$$

所以滑动窗口每次往右移，你只需要求出这个窗口中的最小值以及最后一个值即可，注意因为当前 $a[i]...a[i+n-1]$ 的值还受 $a[1]...a[i-1]$ 的影响，所以还要维护一个 $base$ ，每次询问消除这个影响。

当前窗口的最后一个值显然为 $a[i+n-1]+base$ ，可以 $O(1)$ 询问

对于当前窗口中的最小值，有以下两种方法：

1. 线段树维护区间最值，复杂度 $O(logn)$ ，不会的同学可以去学习一下 2. 考虑滑动窗口从 $a[i]...a[i+n-1]$ 滑动到 $a[i+1]...a[i+n]$ 这个过程中最小值的变化 - 假如 $s[i]=($ ，则对于 $s[i+1]...s[i+n]$ 的前缀和中 $s[i]...s[i+n-1]$ 的前缀和相对之前减去了 1 ，这一段的最小值也减去了 1 ，$s[i+n]=s[i]=($ ，所以 $s[i+n]$ 的前缀和值比 $s[i+n-1]$ 的前缀和值大 1 , 所以当前窗口最小值尽可能取这两个值, 即为 $nowmin=min(premin-1,a[i]+base)$ - 假如 $s[i]=)$ ，证明与上面类似，$nowmin=min(premin+1,a[i]+base)$ ，$nowmin$ ：当前字符串前缀和数组的最小值，$premin$ ：上一个状态字符串前缀和数组的最小值 - 每次循环过后更新 $base,nowmin$ - 时间复杂度：$O(1)$

所以第一种写法的复杂度：$O(nlogn)$

第二种写法的复杂度：$O(n)$

4 1581 题 – 韭菜

4.1 题面

4.1.1 Time Limit

5s

4.1.2 Memory Limit

128M

4.1.3 题目

LFgg 有一间非常大非常大非常大的大别墅。

这天，LFgg 突然特别特别特别想吃韭菜。

LFgg 非常非常非常挑剔，他只吃自家别墅地里由女仆长 Cgg 种的韭菜。

LFgg 并不是特别特别特别能吃，所以他想知道，一块田中的一部分上有多少韭菜，能让他吃痛快却不至于浪费。

LFgg 叫来女仆长 Cgg ，在地图上画给了 Cgg 很多区域，希望 Cgg 能去调查究竟区域中有多少韭菜。

女仆长 Cgg 已经很累很累很累了，他希望聪明的你们能帮帮他，他已经快被压榨到犭

田地可以被抽象为一个二维平面，每一个平面上的点上有一些韭菜。

区域可以被抽象为一个二维平面上的一片矩形。

4.1.4 输入

第一行一个正整数 T, 代表田地块数。

对于每块田地，第一行三个正整数 n,m,q 分别代表田地的长、宽以及询问次数。

接下来 n 行，每行 m 个正整数 aij ，表示该点的韭菜数。

接下来 q 行，每行四个正整数 x1,y1,x2,y2 代表待查矩形的左上角与右下角。

T 10,n,m 1000,q 10000,0 aij 1e9,x1 x2,y1 y2. 注意：该平面的编号方式为，从上到下 x 从小到大，从左到右 y 从小到大。如下为 n=2,m=3 时的编号情况。

(1,1) (1,2) (1,3)

(2,1) (2,2) (2,3)

4.1.5 输出

对于每块田地，首先输出一个“ Field #t: ”，表示第 t 组田地。(注意，Field 后有空格，冒号后没有空格)

对于每次询问，首先输出“ Case #q: ”，表示第 q 组询问，然后输出 i=x1x2 j=y1y2aij ，即矩形内所有点的和（包括边上与顶点上）。（注意，Case 后与冒号后均有空格）

4.1.6 示例

输入:

```
1
4 4 3
1 2 3 1
1 5 3 2
2 3 1 4
3 5 6 2
1 1 1 1
2 2 3 3
3 4 4 4
```

输出:

```
Field #1:
Case #1: 1
Case #2: 12
Case #3: 6
```

4.2 题解

二维前缀和，开一个二维数组

记 $sum[x][y]$ 表示左上角为 $(1,1)$ 右下角为 (x,y) 的矩阵 a 中所有元素的和

即 $sum[x][y]=\sum_{i=1}^{x}\sum_{j=1}^{y}a[i][j]$

$sum[i][j]=sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1]$ ，可由此式求出整个 sum 数组

对于每次询问 :

$\sum_{i=x_1}^{x_2}\sum_{j=y_1}^{y_2}a[i][j]=sum[x_2][y_2]-sum[x_1-1][y_2]-sum[x_2][y_1-1]+sum[x_1-1][y_1-1]$

即为每次询问的答案

具体的讲解和证明可以百度学习,这里放一个 [链接](https://blog.csdn.net/Zeolim/article/details/86770827)

时间复杂度：$O(max(n^2,q))$

注意：如果维护每一行 (列) 的一维前缀和再每次累加也可过 (因复杂度略大，可能只有 C++ 可过) ，时间复杂度：$O(max(nq, nm))$

5 1582 题 – 卷王的终结

5.1 题面

5.1.1 Time Limit

1s

5.1.2 Memory Limit

128M

5.1.3 题目

内卷指不能从外部渠道获取资源，没有产生整体增量，只能在存量分配上做文章，往往损害内部一部分群体利益来补偿少数群体利益，最终整体利益没有增加或减少，内部持续性内耗的一种状态。在第一次内卷战争结束后，苏州大学的计科院决出了卷王选手。这位卷王选手要和别的学校的卷王同台竞技，争夺卷神的名号。(Involution-God)

想要在内卷战争中取得胜利，就必须有过人的“超码力”(Super-Programming Ability)。本次内卷战争共有 n 位卷王，他们的“超码力”分别为 2,3,4, ... n+1 。(其中第 i 个人的码力为 i+1)

卷王选手们所不知道的是，在无止境的内卷游戏中是不会有真正的胜利者的。

有一位超码力为 i 的卷王，他同时有一个“内卷终结者”的身份，对于超码力为 j 的卷王，在满足 gcd(i,j)=1(即 i , j 的最大公约数是 1) 的时候，卷王 j 就会被感化而放弃内卷，同时卷王 j 也变成了内卷终结者，将在下一天去感化其他人。

如果有多个卷王变成了内卷终结者，他们所有人都会在下一天去感召其他人。

lfgg 为了结束这场内卷游戏，选定了第 k 个人（超码力为 k+1 ）作为最开始的内卷终结者。但是他不知道要经过几天才能让所有的卷王都放弃内卷。他拜托聪明的你来计算出最终的答案，才能即时阻止这场战争。

可以证明的是，一定存在某一天，所有的卷王都会放弃内卷。如果答案过大，请你对 998244353 取模输出答案。

5.1.4 输入

输入第一行有一个整数 t, 满足 1 t 15 ，接下来有 t 个测试样例

每个测试样例的第一行包括了两个整数 n,k 满足 2 n 1014,1 k n

5.1.5 输出

一行一个正整数，表示答案。如果答案过大，请你对 998244353 取模输出答案。

5.1.6 示例

输入:

```
2
3
1 2 3
4
1 2 3 4
```

输出:

```
6
32
```

5.2 题解

对于当前的值 $val = k + 1$，将所有情况归纳与一下几种情况:

1. val 是合数，所有比它小的数中必然存在他的因子 p, $p < val, p > 2$，和一个素数。$gcd(p, val) = p > 1$，第一轮必然不会被感召，必定大于 1 轮。对于所有 $gcd(val, x) > 1$，$gcd(val, val - 1) = 1$，第一轮 $val - 1$ 一定会被感召，$gcd(val - 1, x) = 1$，所以所有数都会在第二轮被全部感召。答案为 2 2. val 是素数 - 如果不存在 val 的倍数，对于所有 x，$gcd(val, x) < val$ 为 val 的因子，因为 val 是素数，因子只有 $1, val$ 所以 $gcd(val, x) = 1$，一轮就可以传染全部。答案为 1 - 如果存在 val 的倍数，对于所有，$gcd(val, x) < val$ 为 val 的因子的情况同上，对于 $gcd(val, x) = val$ 即 x 为 val 的倍数的情况，需要两次，证明同情况 1

判别素数：$O(\sqrt{n})$，判别倍数：$O(1)$

时间复杂度：$O(\sqrt{n})$

6 1583 题 – 食堂打饭

6.1 题面

6.1.1 题目

食堂有 n 个窗口，每个窗口都有一个美味值 ai，小 C 今天来到食堂吃饭，由于小 C 刚运动完很累，不想端着盘子走很远，他会选择 m 个连续的窗口，获取这 m 个窗口的美味值。现在小 C 想知道他最多能获得多少美味值。

6.1.2 输入

第一行整数 T，表示数据组数。

接来来共 3T 行:

第一行两个整数 n 和 m。(1 m n) 第二行 n 个整数表示每个窗口的美味值 ai。

6.1.3 输出

T 行，每行表示小 C 可以获取的最大美味值。

6.1.4 示例

输入:

```
1
5 2
1 2 3 4 5
```

输出:

```
1 9
```

6.2 题解

求一个数组中连续 m 个数的和的最大值
利用前缀和可以对连续 m 个数的和进行 $O(1)$ 查询
$sum[x] = \sum_{i=1}^{x} a[i]$，就是前缀和
易证：$\sum_{i=l}^{r} a[i] = sum[r] - sum[l-1]$
可以先求出 sum 数组在循环一遍求连续 m 个数的最大值
时间复杂度：$O(n)$

7 1584 题 – 完全二叉树

7.1 题面

7.1.1 Time Limit

4s

7.1.2 Memory Limit

128M

7.1.3 题目

一个二叉树，如果每一个层的结点数都达到最大值，则这个二叉树就是满二叉树。对于深度为 D 的，有 N 个结点的二叉树，若其结点对应于相同深度满二叉树的层序遍历的前 N 个结点，这样的树就是完全二叉树。

给定一棵完全二叉树的后序遍历，请你给出这棵树的层序遍历结果。

定义：后序遍历：在二叉树中，先左后右再根，即首先遍历左子树，然后遍历右子树，最后访问根结点。层序遍历：对某一层的节点访问完后，再按照他们的访问次序对各个节点的左孩子和右孩子顺序访问。

7.1.4 输入

第一行一个正整数 T, 表示接下来有 T 组测试样例；

接下来 T 组数据，每一组数据的第一行有一个整数 N，即树中结点个数。每一组数据的第二行给出后序遍历序列，为 N 个不超过 100 的正整数。同一行中所有数字都以空格分隔。

数据范围：1 N 30

7.1.5 输出

在一行中输出该树的层序遍历序列。

7.1.6 示例

输入:

```
1
8
91 71 2 34 10 15 55 18
```

输出:

```
18 34 55 71 2 10 15 91
```

7.2 题解

树的遍历方式等等知识大一下学数据结构会系统地学

对于一颗完全二叉树，你可以用一个数组来模拟他的编号，若当前节点的编号为 i ，则他的左节点 (左儿子) 的编号为 $2 \times i$ ，他的右节点 (右儿子) 的编号为 $2 \times i + 1$ ，这样编号会简化遍历的顺序，像题目中给的那个图一样

根据后序遍历的定义模拟它的做法 (递归) :

1. 若左节点编号存在，先遍历左子树 2. 若右节点编号存在，再遍历右子树 3. 读入当前值

时间复杂度：$O(n)$

8 1585 题 – 我真的不想拿外卖

8.1 题面

8.1.1 Time Limit

5s

8.1.2 Memory Limit

128M

8.1.3 题目

人世间，唯有爱与美食不可辜负，爱已经辜负的太多了，美食就不能再辜负了。

一次训练赛结束以后，为了庆祝自己队伍在训练时的完美表现，wygg 和 6+gg（ six plus gg ）和楠哥决定点两个 kfc 全家桶来奖励一下自己。眼看外卖就要到了，忙碌的楠哥却突然被实验室喊去肝项目了。拿外卖的任务自然就落到了 wygg 和 6+gg 身上，但是 wygg 在和 npy（ ）视频聊天，6+gg 在王者峡谷里屠杀，两个人谁也不想去拿外卖。无奈外卖小哥的催促，两人决定玩一局小游戏决定谁去拿外卖。由于对楠哥天天爆肝项目的不满，wygg 和 6+gg 打开了楠哥的电脑，他们决定删掉最近楠哥写的 n 个项目文件。具体游戏规则如下：一共有 n 个项目文件 ; wygg 和 6+gg 轮流操作，wygg 先操作；每次操作可以删除其中 1 m 个文件；最先删光文件的人获胜。

8.1.4 输入

输入包括 T+1 行；第一行包括一个整数 T (1 T 105) ，即以下有 T 组数据需要你判断结果；接下去 T+1 行，每行有两个数 n,m ，分别表示楠哥的项目文件数量，和每轮操作至多删去的文件数，其中 1 m n 1014 。

8.1.5 输出

如果 wygg 去拿外卖，请输出 ”wygg,yyds”; 如果 6+gg 去拿外卖就输出 ”6+gg,yyds”, 每个实例的输出占一行。

8.1.6 示例

输入:

```
2
23 2
4 3
```

输出:

```
6+gg,yyds
wygg,yyds
```

8.2 题解

将先手取的玩家称为先手玩家，后手取的玩家称为后手玩家

把问题归结于一下几种情况:

1. 若 $n\%(m+1)=0$，对于每次先手玩家删除文件的个数 $x,\ 1\le x\le m$，后手玩家只需删除个数为 $m+1-x$ 即可，由 $1\le x\le m$，有 $1\le m+1-x\le m$，所以这种删法必定合法。因为每次先手玩家和后手玩家去完文件都少了 $m+1$ 个，故最后一次后手玩家一定能删完。后手必胜。2. 若 $n\%(m+1)\neq 0$，对于先手玩家来说，他只需删 $n\%(m+1)$ 个文件，即可使文件总量满足情况 1 的条件，并且情况 2 的先手相当于先手删完一次之后情况 1 的后手，转化为情况 1，所以先手必胜。

时间复杂度：$O(1)$

9 1586 题 – 噪音控制

9.1 题面

9.1.1 Time Limit

4s

9.1.2 Memory Limit

128M

9.1.3 题目

曼哈顿城正在施工!

城市里一共有 n 个施工点，施工点总是会发出莫名其妙的噪音，特别是当两个施工点距离足够远时还会发生神奇的反应，让噪音变得更大。

所以现在曼哈顿的区长请你来计算一下两两施工点之间曼哈顿距离的最大值是多少。

坐标为 (1, 1)(x1,y1) 的施工点和坐标为 (2, 2)(x2,y2) 的施工点曼哈顿距离为 | 2− 1|+| 2− 1|

9.1.4 输入

第一行一个整数 T 表示数据组数一行一个 n 接下来 n 行，每行两个整数表示施工点的坐标。
其中 2 n 200,0 x,y 1000 数据不超过 10 组。

9.1.5 输出

一行一个整数表示最大的曼哈顿距离。

9.1.6 示例

输入:

```
2
5
1 1
3 5
2 7
8 1
4 4
10
-64307 91234
-59333 48648
-58988 -28348
4217 67940
-13054 -53848
7249 -88917
79497 57607
-43065 72341
38901 73995
-63577 86518
```

输出:

```
12
251707
```

9.2 题解

签到题
暴力枚举任意两个点，求出他们的曼哈顿距离，取最大值
时间复杂度：$O(n^2)$

10 1587 题 – 走出自己的舒适区

10.1 题面

10.1.1 Time Limit

1s

10.1.2 Memory Limit

128M

10.1.3 题目

小明决定走出自己的舒适圈，但是他很懒。所以，他决定让 wygg 和 sngg 帮忙督促他。

舒适区的范围可以假想为二维平面中以坐标轴原点为圆心，半径为 d 的圆。刚开始，小明的位置在 $(0,0)$ 位置（ wygg 和 sngg 轮流督促他，第一天是 wygg ）。每天 wygg（ sngg ）督促小明往 x 轴的正方向或者 y 轴的正方向移动 k 个单位距离。（如果小明距离原点的距离超过了 d，则代表小明走出了自己的舒适圈，游戏结束）。

由于小明会暗中记恨那个让他走出舒适区的人，wygg 和 sngg 都不想被小明记恨，所以他们都不想走最后一步。wygg 和 sngg 都是绝顶聪明的人，保证他们每一步都做出最优的选择，看看谁会走最后一步，即让小明走出舒适圈的人是谁。

10.1.4 输入

第一行包括一个数字 t（代表测试数据的数量）。

每一行包括两个被空格分开的数字 d（ $1 \le d \le 100000$ ），k（ $1 \le k \le d$ ）。

10.1.5 输出

对于每一组数据，如果 wygg 走最后一步输出 `"wygg"`，否则则输出 `"sngg"`。

10.1.6 示例

输入：

```
3
2 1
5 2
10 3
```

输出：

```
wygg
sngg
wygg
```

10.2 题解

这是一个博弈论的题目。我们可以给每一个点打上 必输 或者 必赢 状态。可以列矩阵如下：

$$\begin{bmatrix} & & \cdots & Y & X \\ & & \cdots & X & Y \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ Y & X & \cdots & & \\ X & Y & \cdots & & \end{bmatrix}$$

博弈论的状态图是一个有向无环图，且又性质为：

- 没有后继状态的状态是必败状态。
- 一个状态是必胜状态，当且仅当至少存在一个必败状态为它的后继状态。
- 一个状态是必败状态，当且仅当它的所有后继状态均为必胜状态。

其中标记 X 的都是 wygg 面临的初始状态。如果最后一个 X 之后没有后继状态，那么所有的 X 标记上的状态就是必输状态。否则它能有向地到达 Y 状态，而因为该 X 状态是最后一个 X 状态，所以它无法到达下一个坐标为 $(x+1,\ x+1)$ 的点，又通过简单的几何学，$(x,\ x+1)$ 也不能到达 $(x,\ x+2)$，同理 $(x+1,\ x)$ 也没有后继状态，为必输状态。

不用考虑其他状态，这个带子上所有的状态点即为判断起始点为是否为必赢状态 / 必输状态的充分条件。我们可以断言：

- 如果最后一个 X 点无后继节点，则所有的 X 节点，包含起始点均为必败节点。
- 如果最后一个 X 点有后继节点 Y，且根据几何知识能证明该 Y 节点无后继节点，即该 Y 节点为必输节点。又因为 X 为的所有后继节点为必输节点，所以说该 X 节点为必赢节点，且所有 X 都为必赢节点。

10.3 参考代码

如下，其中 t 满足 $|(tk,\ tk)| \le d$。而 more 表示最后的 X 节点之后是否有下一个 Y 节点。

```
using ll=long long;

void swaper() {
    ll t;
    for(scanf("%lld",&t);t--;){
        ll d,k;
        scanf("%lld %lld",&d,&k);

        ll t=(ll)floor(d*1.0/sqrt(2)/k)-1;
        while((t+1)*k*(t+1)*k+(t+1)*k*(t+1)*k<=d*d)t++;

        bool more=(t*k*t*k+(t+1)*k*(t+1)*k<=d*d);

        printf((!more)?"wygg\n":"sngg\n");
    }
}
```

11 1588 题 – cgg 的秀发（Easy Version）

11.1 题面

11.1.1 Time Limit

1s

11.1.2 Memory Limit

128M

11.1.3 题目

“啊!!! 我的秀发”

众所周知，cgg 有一头乌黑透亮的秀发。在期中复习前夕，cgg 在自习室复习计网时又掉了一些头发，这让他十分伤心。巧合的是，cgg 掉落下来的头发有规律的排成了（‘和）组成的序列，也就是左右括号。cgg 将这些头发视为珍宝。

dyyyyyyyy 想捉弄一下 cgg，他想拿走一些头发 (数量可能为 0)，使得头发序列不是合法括号序列。为了不被 cgg 发现，要使拿走的头发竟可能的少。你能帮助他完成这个任务吗?

合法括号序列:

空字符串是合法括号序列如果 s 是合法括号序列，（‘+s+‘）也是合法括号序列如果 s 和 t 是合法括号序列，那么 s+t 也是合法括号序列例：(()())() ，(()) ，()() ，() 是合法括号序列，()) ，)(，) 不是合法括号序列

输入保证字符串中仅含有左括号（‘和右括号 ）你可以使其变为不合法括号序列吗？输出你要拿走头发的最小数量。

11.1.4 输入

第一行 t ，表示共有 t 组样例

每组输入两行，第一行为 l ，1 l 1000000 ，第二行为一个字符串，输入保证字符串中仅含有（‘,）。

输入保证 l 1000000

11.1.5 输出

对于每组样例输出需要拿走头发的最小数量。

11.1.6 示例

输入:

```
3
3
()(
2
()
1
(
```

输出:

```
0
1
0
```

11.2 题解

对于一个括号序列，考虑一下两种情况

1. 是合法括号序列，那么通过一次改变必定可以使他变为不合法括号序列。删除第一个字符即可。输出 1 。2. 不是合法括号序列，那么不需要进行删除操作。输出 0 。

时间复杂度：$O(n)$

12 1589 题 – 初识网络流

12.1 题面

12.1.1 Time Limit

1s

12.1.2 Memory Limit

128M

12.1.3 题目

网络流问题是 XCPC 比赛中重要的图论算法问题，一般用于解决流网络中的最大流量问题。考虑到许多同学没有学过图论，我们可以把这个问题形象地描述为如下问题：

网络中由两台计算机 s 和 t，现在想从 s 传输数据到 t。该网络中一共有 n 台计算机，其中一些计算机之间连有一条单向的通信电缆，每条通信电缆都有对应的 1 秒钟内所能传输的最大数据量。当其他计算机之间没有数据传输时，在一秒钟内 s 最多可以传送多少数据到 t？*

单纯的贪心算法，不断找到一条从 s 到 t 的电缆路径，并尽可能多的传输数据，并不能总是找到该类问题的最优解。

像如下的网络，如果我沿着 s->1->2->t 传输 5M 数据，s->1->3->t 传输 5M 数据 , 每秒一共只能传输 10M 数据，但是如果沿着 s->1->3->t 传输 6M 数据，s->1->2->t 传输 4M 数据，s->2->t 传输 1M 数据。就可以传输 11M 数据。

[图片（待上传）]

网络流算法需要对于每一个电缆，建立一个虚拟的方向相反的电缆，然后循环进行以下基本步骤：

1) 只利用当前流量小于最大数据量的电缆或者有流量流过的电缆的对应反向电缆，找到一条从 s 到 t 的电缆路径。

2) 如果不存在满足条件的路径，则结束。否则沿着该路径尽可能多的传输数据，返回上一步。（当然，反向电缆上的流量不可大于正向电缆当前的流量。）

具体证明方法并不在此赘述。

现在，我将在某一个图中进行上述算法每次循环找到的路径和路径上每秒发送的数据包大小告诉你，需要你告诉我最后形成的实际方案是什么?

注意：对于某一条单向电缆 E，设其起点和终点为 u 和 v, 如果在给定的路径中从 u 到 v 每秒传输 a 大小的数据包，从 v 到 u 到传输大小为 b 的数据包，则在实际解决方案中我们给这条电缆安排的流量为 a-b 。

12.1.4 输入

第一行两个整数 n 和 m，表示 n 台计算机和 m 条电缆。(1 n 200,1 m n∗(n−1)/2) 接下来 m 行，每行 3 个整数 u,v,c，表示存在一条从 u 到 v 的单向电缆，其最大数据量为 c 。(1 u,v n,1 c 100) 。保证不存在两条电缆他们的起点和终点与另一条相同或他们的起点和终点与另一条相反，也不存在起点终点相同的电缆。

之后一行一个整数 k，表示提供的路径数目。保证都是合法的路径，且起点和终点始终一致。

第 i 行有两个整数 ai 和 bi，表示第 i 条流量为 ai，会经过 bi 个计算机。之后跟随 bi 个整数，表示依次经过的计算机编号 .(1 ai 100,1 bi n)

12.1.5 输出

输出 m 行，每行三个整数 u,v,f，表示 u 到 v 的电缆最终空闲数据包为 f 。

不允许行末空格以及多余的回车。

电缆的输出顺序请按照输入顺序输出。

12.1.6 示例

输入：

```
5 7
4 1 10
1 3 6
```

```
3 5 8
1 2 6
3 2 3
2 5 5
4 2 2
3
5 4 4 1 2 5
5 4 4 1 3 5
1 5 4 2 1 3 5
```

输出：

```
4 1 0
1 3 0
3 5 2
1 2 2
3 2 3
2 5 0
4 2 1
```

12.2 题解

（网络流好难，但是这个题倒是很简单）。

与其说这个题目是一个网络流题，不如说这是一个阅读理解题。给定一个有向图和有向图生成的最大流路径，我们需要求它的剩余网络，本质上就是一个减法题：前者减去后者即为题目的解。

12.2.1 参考代码的解释

解释 ?? 节的参考代码，我们首先读入一个图到序列 A 中，并生成图 G（使用邻接矩阵表示）；之后 t 次，每一次读入一个求解最大流过程中的路径，并减去 G 图中对应边的流量（同时增加对应边的反向边的流量，从而维持两者和不变）；再使用 A 的顺序，输出相减后的结果。

12.3 参考代码

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

using namespace std;
using ll = long long;

#define F(x) (#x) << ": " << (x) << ", "

struct {
  ll u, v, c;
} A[20010];
ll B[210];
ll G[210][210];

int main() {
  ll n, m;
  memset(G, 0, sizeof(G));
  scanf("%lld %lld", &n, &m);
  for (ll i = 0; i < m; i++) {
    scanf("%lld %lld %lld", &A[i].u, &A[i].v, &A[i].c);
    G[A[i].u][A[i].v] += A[i].c;
  }

  ll t;
  for (scanf("%lld", &t); t--;) {
    ll a, b;
    scanf("%lld %lld", &a, &b);
    for (ll i = 0; i < b; i++) {
      scanf("%lld", &B[i]);
    }
    for (ll i = 1; i < b; i++) {
      G[B[i - 1]][B[i]] -= a;
```

```
32        G[B[i]][B[i - 1]] += a;
33      }
34    }
35
36    for (ll i = 0; i < m; i++) {
37      printf("%lld %lld %lld
38  ", A[i].u, A[i].v, G[A[i].u][A[i].v]);
39    }
40
41    return 0;
42  }
```