Atcoder abc 383

https://atcoder.jp/contests/abc383/

难绷，打得超级臭，被 D 耗了很久。赛时只做到 D，后面三个开都没开。。。

D - 9 Divisors

Find the number of positive integers not greater than N that have exactly 9 positive divisors.

可笑的是，在做这场的前几个小时，刚好学了点小儿科数学，学了约数个数的公式推导、结论。但是这里居然在草稿纸上瞎摸了半个小时才反应过来，然后就写完心态崩了就去洗澡了。

下面是我对于约数个数学习的时候做的笔记：

约数个数

这题是说，给定 n 个正整数 a_i，请你输出这些数的乘积的约数个数，答案对 10^9+7 取模。

- 通过 *1* 我们知道了一个数的约数怎么求，自然就能求出一个数的约数的个数有多少，而这里是求 n 个数的乘积的约数个数，可想而知，对于整个乘积来说，这 n 个数的各自的约数能够相互组成很多新的约数， 有没有什么好的方法计算这些约数个数呢。
- 首先，任何一个数都能被质因数分解，也就是说，N = $p_1^{\alpha_1} * p_2^{\alpha_2} * p_3^{\alpha_3} * \cdots * p_k^{\alpha_k}$，其中 p_i 是质数，α_i 是各自的幂次。我们现在也只讨论一个数，即使用试除法就能解决一个数的情况，但是由这里的结论最终可以解题。
- 对于 N 的任何一个约数 d，它其实是由不同的、不同个数的 p_i 组合而来的
- 对于 N 的任何一个约数 d，自然也能被质因数分解，也就是说，$d = p_1^{\beta_1} * p_2^{\beta_2} * p_3^{\beta_3} * \cdots * p_k^{\beta_k}$
- 当 β_1、$\beta_2 \ldots \beta_k$ 取不同的值时，能够组成不同 d_i，也就是形成不同的约数。
- 而 对于 β_i 来说，它的取值范围是 $0 \le \beta_i \le \alpha_i$，换句话来说，我可以选择 0 个 p_i 组成 d，也可以选择 2 个，3 个......最多 α_i 个 p_i 组成 d。
- 那么一个数 N = $p_1^{\alpha_1} * p_2^{\alpha_2} * p_3^{\alpha_3} * \cdots * p_k^{\alpha_k}$，它的所有不同的约数 d 的个数为：$(\alpha_1+1)*(\alpha_2+1)*\cdots*(\alpha_k+1)$ 个，也就是对于 N 的每个质因数 p_i，我有0 ~ α_i 总共 (α_i+1) 种选择来组成某个约数 d，乘法原理的结果就是不同的 d 的个数了。

那么回到这题，n 个正整数 a_i 的乘积的质因数分解情况，其实就是每个 a_i 的质因数分解情况的累加，举个例子：

比如 $2*3*6$，2 的质因数分解为 1 个 2、3 的质因数分解为 1 个 3、6 的质因数分解为 1 个 2 和 1 个 3，那么 36 的质因数分解就是 2、3、6 分解结果的累加：有 2 个 2 和 2 个 3。

那么我们对 n 个数统计所有的质因数分解情况，对质数的个数做收集，最终结果就是 n 个数乘积的质因数分解。

再利用上面的结论，可以求出这个乘积的不同约数的个数了。

通过这个结论我们知道，一个数的约数个数为：

如果 N = $p_1^{\alpha_1} * p_2^{\alpha_2} * p_3^{\alpha_3} * \cdots * p_k^{\alpha_k}$，那么 N 的约数个数为：$(\alpha_1+1)*(\alpha_2+1)*\cdots*(\alpha_k+1)$

对于 D 题，要求 $(\alpha_1+1)*(\alpha_2+1)*\cdots*(\alpha_k+1)=9$

那么可以有 $(0+1)*(8+1)=9$ 和 $(2+1)*(2+1)=9$，也就是说 N 的质因数分解形式为 $N = p_1^8$ 和 $N = p_1^2 * p_2^2$ 时，它的约数有 9 个，那么我们可以预处理题目给定范围内的质数个数，然后一个一个去$O(1)$ 的验证 8 次方是否小于等于题目给的范围，或者用一个双重带剪枝的循环去验证 $p_1^2 * p_2^2 \leq$ 题目的 N 即可，统计答案。那么这里的质数个体，3 个 p_i，取个 max，也就是说我们至少要预处理出 $\sqrt{N}$ 以内的所有质数然后才能去枚举这些情况。

这里时间复杂度，素数筛，可以近似看成 $O(log\ target)$ 我记得乘的常数都是 $log(log\ target)$ 这种了，调和级数算出来的，其实很小。

八次方枚举，其实很快就会 $break$，从第一个质数开始，到某个质数的八次方大于 N 就停了。就算最坏当成 $O(\text{质数个数} * 8)$ 也是能接受的范围，最坏是$O(\frac{\sqrt{N}}{\ln \sqrt{N}})$

双重那个也是一个道理其实很快，最坏看起来和 $O(N)$ 似的超时了，其实不会。。。

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

vector<int> f(int target) {
    vector<int> res;
    vector<bool> is_prime(target + 1, true);
    is_prime[0] = false; is_prime[1] = false;
    for (int i = 2; i * i <= target; i ++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= target; j += i) {
                is_prime[j] = false;
            }
        }
    }
    for (int i = 2; i <= target; i ++) {
        if (is_prime[i]) res.push_back(i);
    }
    return res;
}

int main() {
    long long n;
    cin >> n;
    int m = (int)(sqrtl(n)) + 1;
    vector<int> p = f(m);
    long long ans = 0;
    for (int i = 0; i < p.size(); i++) {
        long long t = 1LL;
        bool ok = true;
        for (int cnt = 0; cnt < 8; cnt++) {
            if (t > n / p[i]) {
                ok = false;
                break;
            }
            t *= p[i];
        }
        if (ok) ans ++;
```

```
    }

    for (int i = 0; i < p.size(); i ++) {
        long long sum1 = (long long)p[i] * p[i];
        if (sum1 > n) break;
        for (int j = i + 1; j < p.size(); j ++) {
            long long sum2 = (long long)p[j] * p[j];
            if (sum2 > n / sum1) break;
            long long res = sum1 * sum2;
            if (res <= n) ans++;
            else break;
        }
    }

    cout << ans << endl;
    return 0;
}
```

E - Sum of Max Matching

这题是说，给你一个无向图，边有权值，定义一条路径的 $weight$ 是这条路径上最大的边权，定义是 $f(x, y)$ 表示从 x 点到 y 点的所有路径中最小的路径的 $weight$，给定 A 和 B 分别是一组点的集合，集合大小为 k，且保证 $A_i \neq B_j (1 \le i, j \le k)$。让 B 内的元素任意排列后，使得 $\sum_{i=1}^{k} f(A_i,\ B_i)$ 最小。

看别人题解学的。

- 将所有边按照边权排序。
- 然后遍历边，做并查集，在并查集的过程中一边联通一边计算答案。

解释说明：

按照边权排序之后，遍历边，做并查集，可以达到什么效果？因为是按边权从小到大遍历边，那么对于当前遍历到的这条边的两个端点：

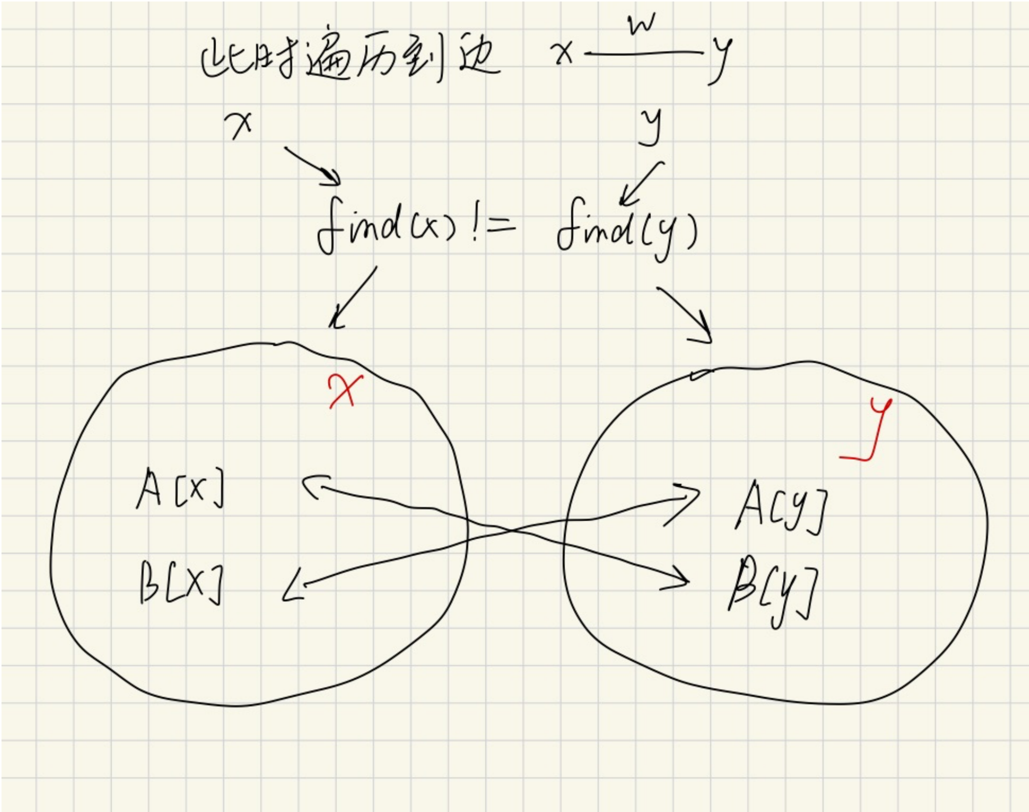
- 如果他们不在一个连通块中，那么这条边的 $weight$ 是能够连通他们的最小 $weight$，我们可以在这个时候去做一些答案贡献的计算。
- 如果他们原本已经在一个连通块中，那么其实相关对答案的贡献在之前的遍历中已经计算过了。

对于答案的贡献具体如何计算呢：

这个我觉得比较抽象，没法从因到果的讲述，功力不够吧，只能直接描述做法。尽量讲清楚怎么做的为什么对。

维护两个数组，A 和 B，其中 $A[i]$ 表示 A 集合中以 i 为代表的连通块中的点的数量是多少。B 同理。

在按权值从小到大遍历的边的时候，连接这条边的两个点 x 和 y 如果已经在一个连通块中了，就跳过，如果不在一个连通块中，此时的状态类似下图所示。



对于上图的解释：

- 发现遍历到的这条边，连接的两点，此时属于两个不同的连通块 x 和 y 。
- A 序列中，有些点会属于 x ，这些点的个数为 $A[x]$ ，有些点会属于 y ，这些点的个数为 $A[y]$ 。
- B 序列中，有些点会属于 x ，这些点的个数为 $B[x]$ ，有些点会属于 y ，这些点的个数为 $B[y]$ 。
- 在此次维护之后，x 和 y 会连通，后续遇到同一个连通块的点会直接跳过。
- 在此次维护之中，x 和 y 尚未连通：
  - x 中属于序列 A 的点会和 y 中属于序列 A 的点连通，同属于一个序列的点对答案没有贡献，但是我们要维护好新的 $A[?]$ 个数。
  - x 中属于序列 B 的点会和 y 中属于序列 B 的点连通，同属于一个序列的点对答案没有贡献，但是我们要维护好新的 $B[?]$ 个数。
  - x 中属于序列 A 的点会和 y 中属于序列 B 的点连通，此时不同序列的而且不在一个连通块的点被连通了，而且当前这条边的 $wight$ 是被连通的点对的 $path$ 的最小的 $weight$ ，我们此时要计算这个 $weight$ 对答案的贡献。
  - x 中属于序列 B 的点会和 y 中属于序列 A 的点连通，此时不同序列的而且不在一个连通块的点被连通了，而且当前这条边的 $weight$ 是被连通的点对的 $path$ 的最小的 $weight$ ，我们此时要计算这个 $weight$ 对答案的贡献。
  - 综上所述，我们要计算答案贡献，并且维护好 $A[x]$，$A[y]$，$B[x]$，$B[y]$ 在此次维护后的值。具体怎么维护呢？
    - $A[x]$ 和 $B[y]$ 被连通，此时的 $weight$ 是这些点对被连通成有 $path$ 时最小的 $weight$ 了，我们要尽可能的连通它们，而根据 $f()$ 的定义，我们知道，此时有效被连通的点对数量为 $c = min(A[x], B[y])$ ，所以答案更新为：$ans = ans + c * weight$ ，那么 $A[x]$ 和 $B[y]$ 中各自有 c 个点已经被连通了，后续不用再计算了，所以维护好新的 $A[x] = A[x] - c$ 且 $B[x] = B[x] - c$ 。

- $A[y]$ 和 $B[x]$ 同理。

所以整个做法就解释完了。。。自己对于这个解释很不满意，因为是直接给出做法。这个大概就是一个贪心的过程，利用上述并查集和 $weight$ 的性质，贪过去，把连通点对对答案的贡献计算起来。。而不是有了理论证明之后理解这个做法。。。看以后进步了，能不能重新写一下理解吧。

时间复杂度：排序边，遍历边，并查集路径压缩，最坏$O(mlogm + m * logn)$

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

struct edge {
    int u, v, w;
};

vector<int> p;

int find(int x) {
    if(x != p[x]) p[x] = find(p[x]);
    return p[x];
}

int main() {
    int n, m, k;
    cin >> n >> m >> k;
    vector<edge> es;
    p.clear();
    p.resize(n + 1, 0);
    for(int i = 1; i <= n; i ++) p[i] = i;
    for(int i = 0; i < m; i ++) {
        int u, v, w;
        cin >> u >> v >> w;
        es.push_back({u, v, w});
    }

    vector<int> a(n + 1, 0), b(n + 1, 0);
    for(int i = 0; i < k; i ++) {
        int node;
        cin >> node;
        a[node] ++;
    }

    for(int i = 0; i < k; i ++) {
        int node;
        cin >> node;
        b[node] ++;
    }

    sort(es.begin(), es.end(), [&](const edge& a, const edge& b) -> bool{
        return a.w < b.w;
    });
```

```
    long long ans = 0;

    for(auto e : es) {
        int u = e.u, v = e.v, w = e.w;
        int x = find(u), y = find(v);
        if(x != y) {
            int c = min(a[x], b[y]);
            a[x] -= c;
            b[y] -= c;
            ans += 1LL * c * w;

            c = min(a[y], b[x]);
            a[y] -= c;
            b[x] -= c;
            ans += 1LL * c * w;

            p[x] = y;
            a[y] += a[x];
            b[y] += b[x];
        }
    }

    cout << ans << endl;
    return 0;
}
```

F - Diversity

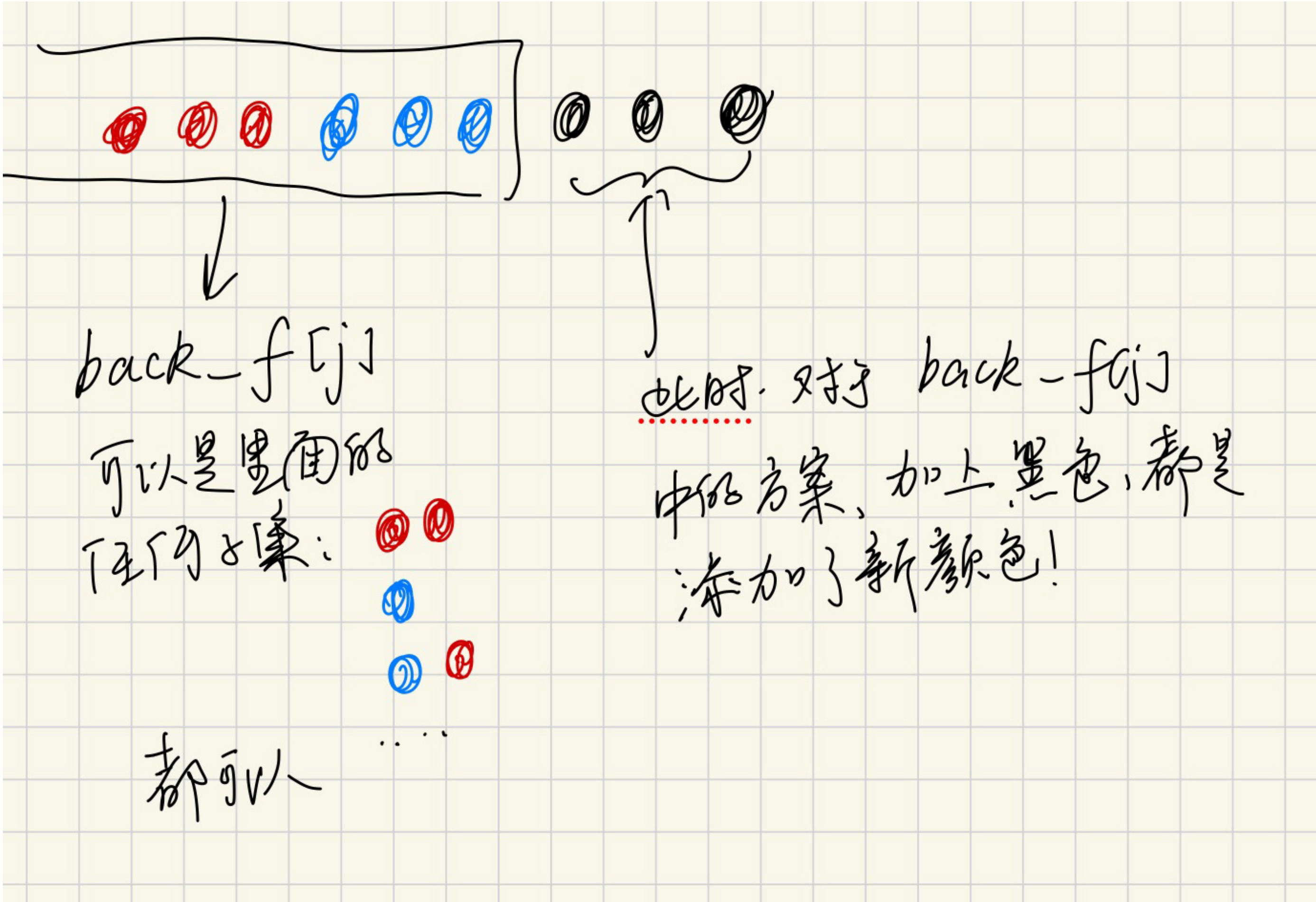
这题是说有 N 个商品，每个商品有三个属性：P 价格，U 使用价值，C 颜色，你需要购买一些商品 （N 的子集）在总价格不超过 X 日元的情况下，使得 $S+T*K$ 最大，其中 S 是你选择的这些商品的 U 的总和，K 是一个常数，T 是你选择的商品的不同颜色的个数。

其实一看就是个背包的感觉，就是有这个颜色的进去了。

假设没有颜色就是常规的，$f[i][j]$ 表示我从前 i 件商品中购买总价格为 j 的商品时，S 最大是多少，但是这里要计算的是 $S+T*K$ ，也就是说我们要考虑当前商品买了之后，颜色的情况。

我们可以将商品按照颜色排序，按照颜色的顺序去 dp 。

我们可以用一个 $back\_f[j]$ 表示，当我处理完某段同种颜色的最后一个颜色之后，当前遇到了某段新颜色时，$back\_f[j]$ 表示遇到新颜色之前的那些所有的颜色集合中，总价格不超过 j 的方案。比如下图：



图中“任何子集”应该是“某个/某些子集能达到这个 j 且答案最大”。

所以，当遇到第一个新颜色的时候，我们要更新 $back_f[j] = f[i-1][j]$，$j = 1, 2, \ldots, x$ 。

状态转移就是：

$f[i][j] = max(f[i-1][j],\ f[i-1][j-p]+u,\ back\_f[j-p]+u+k)$ 。

$f[i-1][j]$ 表示不选该商品。

$f[i-1][j-p]+u$ 表示该商品不是同颜色的第一个商品的话，选出来颜色种类没变化。

$back\_f[j-p]+u+k$ 表示不管该商品是不是同颜色的第一个商品，我是在前 $i-1$ 个商品中选择了颜色种类少一个的子集来转移，此时颜色种类多一个。

注意到对于某段同颜色的第一个商品，代码中其实都会走相当于 $f[i][j] = f[i-1][j-p]+u+k$ 。

时间复杂度 $O(nlogn + n*x)$

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

struct products{
    int p, u, c;
};

int main() {
    int n, x, k;
```

```
    cin >> n >> x >> k;
    vector<products> a(n + 1);
    for(int i = 1; i <= n; i ++) {
        int p, u, c;
        cin >> p >> u >> c;
        a[i] = {p, u ,c};
    }

    vector<vector<long long>> f(n + 1, vector<long long>(x + 1, 0));
    vector<long long> b_f(x + 1, 0);
    f[0][0] = 0;
    sort(a.begin(), a.end(), [](auto& A, auto& B) {
        return A.c < B.c;
    });
    int last = -1;
    long long ans = 0;
    for(int i = 1; i <= n; i ++) {
        int p = a[i].p, u = a[i].u, c = a[i].c;
        bool ok = (i == 1 || c != a[i - 1].c);
        if(ok) {
            for(int j = 0; j <= x; j ++) b_f[j] = f[i - 1][j];
        }
        for(int j = 0; j <= x; j ++) {
            f[i][j] = f[i - 1][j];
            if(j >= p) {
                f[i][j] = max(f[i][j], f[i - 1][j - p] + u);
                f[i][j] = max(f[i][j], b_f[j - p] + u + k);
            }
            ans = max(ans, f[i][j]);
        }
    }
    cout << ans << endl;
    return 0;
}
```

G 题解都没找到。。。