2022ACM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~ 。°·(//>/ ︿/</)·°。

Frontier Tripper

题意翻译：求 $\sum_{i=1}^{n}\varphi(i^k)\cdot\sigma(i^k)\bmod (10^9+7)$ ，$1\le n,k\le 10^6$。

考点：素数筛 + 欧拉函数 + 积性函数 + 快速幂

积性函数是满足 $\forall x,y\in N_+,(x,y)=1$ 则 $f(xy)=f(x)\cdot f(y)$ 的函数

对欧拉函数 $\varphi(n)$ ，根据数论知识，可知它是积性函数，且对素数 p 和正整数 k ，有 $\varphi(p^k)=p^k-p^{k-1}$ 。证明：素数 p 的任意倍数(包括 $1,k$ 倍数)都不与 p^k 互质，那么所有倍数 $t\in[1,p^{k-1}]$ 组成的 pt 都不与它互质，有 p^{k-1} 个这样的数，所以其他数都与它互质，故为 p^k-p^{k-1} 。

那么根据积性函数的性质，$\forall n_1,n_2\in Z_+,n_1+n_2=n,\varphi(n)=\varphi(n_1)\times\varphi(n_2)$ 。所以对每个 p^k 用上述方法计算；对其他数，用质因数分解将其拆解为两个已求出欧拉函数的乘积即可。

对倍数函数 $\sigma(n)$ ，根据数论知识，可知它也是积性函数，且对素数 p 和正整数 k ，有 $\sigma(p^k)=\dfrac{p^k-1}{p-1}$ 。证明：p^k 的因数一定能构成等比数列 $1,p,p^2,\cdots,p^k$ ，对此数列求前 k 项和即可。

那么根据积性函数的性质，对 p^k 直接用上述方法计算；对其他数，用质因数分解将其拆解为两个已求出倍数函数的乘积即可。

使用素数筛(以欧式筛为例)，在筛的过程可以线性求出每个数能够被拆分成的幂指数、底数及其幂数。设 e_i 是 i 质因数分解能得到的一个幂的指数，p_i 是对应的底数，pe_i 是对应的幂数 $p_i^{e_i}$ (具体计算方法见代码)。那么若 $pe_i=i$ ，直接计算函数值，否则用积性函数进行拆分。

根据积性函数性质，因为 φ,σ 是积性函数，所以复合函数 $\varphi\cdot\sigma$ 也是积性函数。可以用素数筛求出 $[1,n]$ 内的 φ,σ 值。那么对 $\varphi(i^k),\sigma(i^k)$ ，当 i 是素数时根据上文结论直接求，否则，把 i^k 拆分为两个数互质的数(其中一个是素数，那么一定跟任意数互质) i_1,i_2 ，用已知值 i_1^k,i_2^k 相乘即可。

题目需要对 10^9+7 (质数)取模，根据取模公式和逆元含义，由费马小定理可知 $\dfrac{x}{y}\bmod (10^9+7)=x\times y^{10^9+5}\bmod (10^9+7)$ ，其中 y 的幂使用快速幂进行计算。

素数筛复杂度为 $O(n)$ ，求快速幂和逆元复杂度为 $O(\log P),P=10^9+7$ ，故总复杂度为 $O(n\log(10^9+7))$ 。

也可以不使用素数筛，用其他数论方法，或别的数论公式求解本题，可自行尝试。

参考代码：

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

```cpp
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 1000002
ll n, k, p[mn], pri[mn], e[mn], pe[mn], g[mn], cnt, ans = 1, mod = 1e9 + 7;
void euler(ll n)
{ // e[i]是i质因数分解得到的最大的幂a_i,pe[i]是对应最大的(p^e[i])
    for (ll i = 2; i <= n; ++i)
    {
        if (!p[i])
        {
            p[i] = i, pri[++cnt] = i, pe[i] = i, e[i] = 1;
        }
        for (ll j = 1; i * pri[j] <= n; ++j)
        {
            p[i * pri[j]] = pri[j];
            if (pri[j] == p[i])
            {
                e[i * pri[j]] = e[i] + 1;
                pe[i * pri[j]] = pe[i] * pri[j];
                break;
            }
            e[i * pri[j]] = 1;
            pe[i * pri[j]] = pri[j];
        }
    }
}
ll qpow(ll a, ll b)
{
    ll res = 1;
    for (; b > 0; b >>= 1)
    {
        if (b & 1)
        {
            res = res * a % mod;
        }
        a = a * a % mod;
    }
    return res;
}
signed main()
{
    sc(n), sc(k);
    g[1] = 1;
    euler(n);
    for (ll i = 2; i <= n; ++i)
    {
        if (pe[i] == i)
        {
            g[i] = (qpow(p[i], e[i] * k + 1) - 1 + mod) % mod * qpow(p[i] - 1, mod - 2) % mod;
            g[i] = g[i] * (qpow(p[i], e[i] * k) - qpow(p[i], e[i] * k - 1) + mod) % mod;
        }
        else
        {
            g[i] = g[i / pe[i]] * g[pe[i]] % mod;
        }
```

```
        ans = (ans + g[i]) % mod;
    }
    printf("%lld", ans);
    return 0;
}
```

Abyss Cycle

题意翻译：一开始有 n 个更新，一开始都是标记状态，想要选出一个更新。若只有一个更新是标记状态，直接选出它；否则，所有在标记状态的更新等概率从 $[1, n]$ 的整数中抽一个，设抽到最小值为 x ，那么所有抽到 x 的更新保持标记状态，其他更新取消标记状态。不断执行该过程直到选出一个更新为止。求期望执行多少次更新才能选出，输出对 10^9+7 取模结果。$1 \le n \le 10^3, 2 \le m \le 10^3$。

考点：组合数学 + 动态规划 + 快速幂

设 dp_i 是 i 个更新和常数 m 时的期望值。显然 $dp_1 = 0$

设共有 i 个更新时，有 t 个更新同时取得最小的概率是 p_t ，根据组合数学公式：

$$p_t = \frac{C_n^t \sum_{j=1}^{m-1} j^{i-t}}{m^i} (t \neq i)$$

意思是选出 t 个更新取得最小，设最小位置是第 $m-j$ 位，那么比它大的 j 个位都可以任意选，共有 $i-t$ 个剩下的更新，都可以任选，所以是幂。

特别地，$t=i$ 时，每个更新都在同一个位置，概率为 $p_i = \dfrac{m}{m^n}$ 。

根据概率 DP 公式，有：

$$dp_i = 1 + \sum_{i=j}^{i} p_j dp_j$$

意思为从 dp_i 状态执行一次过程，过程数 $+1$ ，之后得到状态 j ，概率是 p_j ，这个状态还需要执行 dp_j 次过程，所以除去这次执行外，还期望要执行 $\sum_{i=j}^{i} p_j dp_j$ 次才能选出来。

代入 p_j ，移项(dp_i 都移到左边)，同除 dp_i 的系数，化简，得：

$$dp_i = \frac{m^i + \sum_{j=1}^{n-1} C_i^j dp_j \sum_{k=1}^{m-1} k^{i-j}}{m^i - m}$$

使用 $O(nm)$ 预处理，可以去掉内层求和。所以计算复杂度为 $O(n^2)$ 。加上求逆元的复杂度，总复杂度为 $O(nm + n^2 \log p)$ ，其中 $p = 10^9+7$ 。

注意取模细节，**不能**化简 $\dfrac{a+b}{c} \bmod p$ 为 $(\dfrac{a}{c} \bmod p + \dfrac{b}{c} \bmod p) \bmod p$ 。

题目需要对 10^9+7 (质数)取模，根据取模公式和逆元含义，由费马小定理可知 $\dfrac{x}{y} \bmod (10^9+7) = x \times y^{10^9+5} \bmod (10^9+7)$ ，其中 y 的幂使用快速幂进行计算。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
```

```
typedef long long ll;
ll mod = 1e9 + 7, n, m;
ll qpow(ll a, ll b = mod - 2)
{
    ll r = 1;
    for (; b; b >>= 1)
    {
        if (b & 1)
        {
            r = r * a % mod;
        }
        a = a * a % mod;
    }
    return r;
}
#define mn 1024
ll fac[mn], inv[mn], dp[mn];
ll c(ll d, ll u) // C _d(own) ^u(p)
{
    return fac[d] * inv[u] % mod * inv[d - u] % mod;
}
#define mm 1024
ll s[mm], pw[mm];
signed main()
{
    sc(n), sc(m);
    fac[0] = inv[0] = 1;
    for (ll i = 1; i <= n; ++i)
    {
        fac[i] = fac[i - 1] * i % mod;
        inv[i] = qpow(fac[i]);
    }
    for (ll i = 1; i <= m - 1; ++i)
    {
        pw[i] = 1;
    }
    for (ll i = 1; i <= n - 1; ++i)
    {
        ll cnt = 0;
        for (ll i = 1; i <= m - 1; ++i)
        {
            pw[i] = pw[i] * i % mod;
            cnt = (cnt + pw[i]) % mod;
        }
        s[i] = cnt; // s[i]=1^i + 2^i + ... + (m-1)^i
    }
    dp[1] = 0;
    for (ll i = 2; i <= n; ++i)
    {
        ll c0 = qpow((qpow(m, i) - m + mod) % mod); // 1/(m^i-m)
        dp[i] = qpow(m, i);                         //(m^i)
        for (ll j = 1; j <= i - 1; ++j)
        { // dp[i]+=C(i,j)*(1^j + 2^j + ... + (m-1)^j)*dp[j]
            dp[i] = (dp[i] + c(i, j) * s[i - j] % mod * dp[j] % mod) % mod;
        }
        dp[i] = (dp[i] * c0 % mod);
    }
    printf("%lld", dp[n]);
```

```
    return 0;
}
```

In Another Time

题意翻译：给定奇数 x 和整数 m ，构造取值为 ± 1 的长为 m 的序列 $a_0, a_1, \cdots, a_{m-1}$ ，使得 $x = a_0 \cdot 2^0 + a_1 \cdot 2^1 + \cdots + a_{m-1} \cdot 2^{m-1}$ ，若无解输出 0，否则输出唯一解

考点：思维(二进制)签到题

这题不能暴力搜索，其复杂度为 $O(2^m)$，会超时。但可以折半搜索(见下文)。更推荐使用数学解法：

如果没有解题思路，不妨在小数据 m 下枚举所有可能的序列 a 得到的值，例如令 $a = (-1,-1,-1),(1,-1,-1),(-1,1,-1),(1,1,-1),\cdots,(1,1,1)$ ，可以发现得到的右式分别是 $-7,-5,-3,-1,\cdots,7$ 。如果把 a 看成二进制(a_0 是最小进制位，每个位只能取 -1 或 1)，那么可以发现 a 的二进制数值每增大 1，右式就增大 2。即取值是首项为 -2^m+1，末项为 2^m-1，公差为 2 的 2^m 项等差数列。那么显然不在 $[-2^m+1, 2^m-1]$ 的就输出无解。

而 m 二进制数的取值范围是 $[0, 2^m-1]$ ，即是首项为 0，末项为 2^m-1，公差为 1 的 2^m 项等差数列。我们不妨设想能否将这两个等差数列形成映射关系。设二进制等差数列为 $b_n = n-1$ ，而上文等差数列为 $a'_n = -2^m - 1 + 2n$ ，将 $n = b_n + 1$ 代入得 $a'_n = -2^m + 1 + 2b_n$ 即 $b_n = \dfrac{2^m - 1 + a'_n}{2}$ 。那么当 $a'_n = x$ 时，代入得 $b_n = \dfrac{2^m - 1 + x}{2}$ ，也就是说已知 $x = a'_n$ ，那么求得是二进制数 b_n ，其项序号 n 就是 a_n 的 n 。

因为 a', b 两数列一一对应，所以有解时必然有唯一解。只需要输出 $\dfrac{2^m-1+x}{2}$ 的二进制形式即可(0 代表 $-1, 1$ 代表 1)。

复杂度为 $O(\log x)$ 。也可以用倍增来做本题，可自行尝试。

参考代码：（数学解法）

```
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%d", &x)
typedef int ll;
signed main()
{
    ll x, m;
    sc(x), sc(m);
    if (abs(x) > (1 << m) - 1)
    {
        printf("0");
        return 0;
    }
    ll v = ((1 << m) + x - 1) / 2;
    for (ll i = 0; i < m; ++i, v >>= 1)
    {
        printf("%d ", v & 1 ? 1 : -1);
    }
    return 0;
}
```

若折半搜索，可以先将前一半(即搜索 2^0 到 $2^{\lceil \frac{m}{2} \rceil}$)的全部结果存起来(比如用 map 或结构体等)，后一半(即搜索 $2^{\lceil \frac{m}{2} \rceil}$ 到 2^{m-1})的也存起来。记二进制状态 s 表示每个位选 -1 还是 1。然后遍历前半结果，对每个搜到的结果值 u，还需要 $x-u$ 就能凑出 x，在右半部分找是否存在 $x-u$，若有，就输出答案。时间复杂度为 $O(2^{\frac{m}{2}}+2^{\frac{m}{2}}\log 2^{\frac{m}{2}})=O(2^{\frac{m}{2}}+\dfrac{m}{2}2^{\frac{m}{2}})=O(\dfrac{m}{2}2^{\frac{m}{2}})$，空间复杂度为 $O(2^{\frac{m}{2}})$。

参考程序：（折半搜索）

```cpp
#include <bits/stdc++.h> //meet in the middle 折半搜索
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
#define mn (1 << 15) + 10
ll x, m, mid;
map<ll, ll> a, b;
void dfs(ll s, ll v, ll i, ll ed, map<ll, ll> &h)
{ //二进制状态s,当前是2^i,值是v,最大项是2^ed,h[v]=s
    if (i > ed)
    {
        h[v] = s;
        return;
    }
    dfs(s, v - (1 << i), i + 1, ed, h);            //第i位设为(-1)
    dfs(s + (1 << i), v + (1 << i), i + 1, ed, h); //第i位设为1
}
signed main()
{
    sc(x), sc(m), mid = (m + 1) / 2;
    dfs(0, 0, 0, mid - 1, a);
    dfs(0, 0, mid, m - 1, b);
    for (auto u : a)
    {
        ll d = x - u.first;
        if (b.find(d) != b.end())
        {
            ll r = u.second + b.find(d)->second;
            for (ll i = 0; i < m; ++i)
            {
                printf("%d ", r & (1 << i) ? 1 : -1);
            }
            return 0;
        }
    }
    printf("0");
    return 0;
}
```

Chronosphere Hacker

题意翻译：有 $n\times m$ 矩阵 a，定义集合运算 mex 返回集合里最小的没出现过的非负整数。令 $b_i=mex(a_{i,1},a_{i,2},\cdots,a_{i,m}),e_0=mex(b)$，可以删掉 a 开头和结尾的若干行，使得删后 e_0 不变，求最少剩多少行。$1\le n,m,n\times m\le 10^6,0\le a_{ij}\le 10^9$。

考点：滑动窗口 + 离散化

一种快速求 mex 的方法是先对序列快排然后再离散化去重(`std::unique`)，然后顺次遍历，发现首个下标不等于值时就返回下标作为结果。没发现就返回去重后长度。另一种方法是开一个 set ，思路类似。设序列长度为 s ，复杂度均为 $O(s\log s)$ ，前者空间常数更优。因此对矩阵，可以用 $O(nm\log m+n\log n)$ 的复杂度求出 e_0 。不难发现 $e_0\in[0,n]$ 。

剩下的行一定是连续的一段下标 $[l,r]$。并且 $b_l,b_{l+1},\cdots,b_r$ 里一定要能取遍 $[0,e_0)$ 的所有值，即出现次数不少于 1 次。可以发现，使得长度最小时，应该在这一段下标 $[l,r]$ 和值域 $[0,e_0)$ 里，b_l,b_r 都只出现一次是最优的。因为如果出现了多次，那么可以一直删头/删尾，直到只出现一次为止。

到这一步，不难看出可以使用单调队列维护滑动窗口，用一个数组 bin 记录当前窗口 $[l,r]$ 内值 v 出现的次数为 bin_v ，那么当满足 $i\in[0,e_0),bin_i\ge 1$ 时，若发现 $bin_{b_r}>1$ ，就可以不断缩减左端 l ，直到把重复的 b_r 删掉。在整个过程中出现的最小 $[l,r]$ 长度即为答案。

注意特判，如 $e_0=0$ 时直接输出 0 ，因为 $b=\varnothing$ 时，$mex(b)=0$ 。并且每次遇到 $b_i\ge e_0$ 时直接 continue 掉，不予判断滑窗。

滑窗复杂度为 $O(n)$ ，故总复杂度为 $O(nm\log m+n\log n+n)$ 。

> 其他解法：可以发现，b 的取值最大不超过 m。那么 e_0 的取值最大不超过 $\min(n,m+1)$。也就是说大致有 $ne_0\le 10^6$。那么二分答案，二分 $O(\log n)$ 次，每次枚举所有等长子段，加以滑窗优化，即使每个子段都用桶排 $O(e_0)$ 求 mex，也能保证 $O(ne_0\log e_0)$ 的复杂度理论上可以过题。
>
> 求解静态区间 mex 问题也可以用可持久化权值线段树或回滚莫队。但是本题并未发现可解方法。

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
#define mn 1000010
ll n, m, b[mn], a[mn], t[mn], e, bin[mn], ans, cnt;
ll mex(ll *s, ll len)
{
    memcpy(t, s, sizeof(ll) * (len + 2));
    sort(t, t + len);
    ll ts = unique(t, t + len) - t;
    for (ll i = 0; i < ts; ++i)
    {
        if (t[i] != i)
        {
            return i;
        }
    }
    return ts;
}
signed main()
{
    sc(n), sc(m);
    for (ll i = 1; i <= n; ++i)
    {
        for (ll j = 0; j < m; ++j)
        {
            assert(a[j] >= 0 && a[j] <= 1e9);
            sc(a[j]);
```

```
        }
        b[i] = mex(a, m);
    }
    e = mex(b + 1, n);
    ans = e == 0 ? 0 : n;
    for (ll lf = 1, rf = 1; rf <= n; ++rf)
    {
        if (b[rf] >= e)
        {
            continue;
        }
        cnt += bin[b[rf]]++ == 0;
        while (cnt == e && (bin[b[lf]] > 1 || b[lf] >= e))
        {
            cnt -= --bin[b[lf]] == 0;
            ++lf;
        }
        if (cnt == e)
        {
            ans = min(rf - lf + 1, ans);
        }
    }
    printf("%lld\n", ans);
    return 0;
}
```

Hefeng's Plan

题意翻译：有 n 个流，第 i 个流需要放置 a_i 种不同元素。且任意相邻流不能拥有相同元素。有 m 次操作：①修改 a_i 为 j；②查询为 $[l,r]$ 的所有流放置元素至少要几种元素。
$1 \le n,m \le 10^5, 1 \le a_i \le 10^9$。

考点：线段树

对每次查询，可以发现，最少所需元素数为 $k=\max_{i=l}^{r-1}(a_i+a_{i+1})$，也就是相邻和的最大值。证明：假设在 i 取得最大，那么首先为流 i 分配第 $[1,a_i]$ 种元素，为流 $i+1$ 分配第 $[a_i+1,a_i+a_{i+1}]$ 种元素。那么在 i 向左拓展到 l 的过程中，设 $j\in[l,i)$，每次流 j 可选的元素有 $k-a_{j+1}$ 种，在可选的里面任意选择均可，而 $k-a_{j+1}$ 一定不为 0，所以一定能分配。在 $i+1$ 向右拓展到 r 的过程也同理。

因此，可设数组 $a'_i(1\le i<n)$ 为 a_i+a_{i+1}。那么只需要维护 a'_i 的最大值即可。这是经典的单点修改+区间最值查询，可以用线段树 / 树状数组来实现(树状数组能做但比较复杂，这里不介绍)。每次询问 $[l,r]$ 就等于查 $\max_{i\in[l,r)} a'_i$ (注意特判 $l=r$ 输出 a_l)。每次更新 a_i 就同时修改 a'_i, a'_{i-1} 的值(注意特判边界)。

> 分块也能做，但实现起来细节更麻烦，可自行尝试。

复杂度为 $O(n\log n+m\log n)$。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
```

```
typedef long long ll;
#define mn 100010
ll n, m, a[mn], t[mn * 4];
#define mkcf ll cf = (lf + rf) >> 1
#define lfs p << 1
#define rfs p << 1 | 1
void build(ll p, ll lf, ll rf)
{
    if (lf == rf)
    {
        t[p] = a[lf] + a[lf + 1];
        return;
    }
    mkcf;
    build(lfs, lf, cf);
    build(rfs, cf + 1, rf);
    t[p] = max(t[lfs], t[rfs]);
}
ll query(ll p, ll lf, ll rf, ll lc, ll rc)
{
    if (lc <= lf && rf <= rc)
    {
        return t[p];
    }
    ll res = 0;
    mkcf;
    if (lc <= cf)
    {
        res = max(res, query(lfs, lf, cf, lc, rc));
    }
    if (cf + 1 <= rc)
    {
        res = max(res, query(rfs, cf + 1, rf, lc, rc));
    }
    return res;
}
void update(ll p, ll lf, ll rf, ll pos, ll v)
{
    if (lf == rf)
    {
        t[p] = v;
        return;
    }
    mkcf;
    if (pos <= cf)
    {
        update(lfs, lf, cf, pos, v);
    }
    else
    {
        update(rfs, cf + 1, rf, pos, v);
    }
    t[p] = max(t[lfs], t[rfs]);
}
signed main()
{
    sc(n), sc(m);
    for (ll i = 1; i <= n; ++i)
```

```
    {
        sc(a[i]);
    }
    if (n > 1)
    {
        build(1, 1, n - 1);
    }
    for (ll c, l, r; m--;)
    {
        sc(c), sc(l), sc(r);
        if (c == 1)
        {
            a[l] = r;
            if (l != n)
            {
                update(1, 1, n - 1, l, a[l] + a[l + 1]);
            }
            if (l != 1)
            {
                update(1, 1, n - 1, l - 1, a[l - 1] + a[l]);
            }
        }
        else
        {
            if (l == r)
            {
                printf("%lld\n", a[l]);
                continue;
            }
            printf("%lld\n", query(1, 1, n - 1, l, r - 1));
        }
    }
    return 0;
}
```

The End of the Blue Planet's Duel

题意翻译：有长为 n 的小写字母字符串 S，下标从 1 开始。初始每个 $\pi(i)>0$ 的下标未锁。两玩家轮流操作，每回合可以选择一个未锁的下标，并把 $(i-\pi(i),i]$ 范围的下标全部上锁。无法操作的玩家负。双方用最优策略，先手必胜输出 `580`，后手必胜输出 `1437`。

考点：KMP + Nim游戏

π 就是 KMP 算法的前缀函数。可以用 KMP 算法直接 $O(n)$ 求出 π 数组每个值。

可以发现，将 π 以 0 为分割符拆分为若干非空子段后，找出每个子段最大值，得到新数组 a，例如 $\pi=(0,0,1,2,3,0,1,2,3,4,5)$ 得到 $a=(3,5)$。那么原题转化为：

每回合一个玩家可以从 a 的任一个元素 a_i 内取走 $[1,a_i]$。最后无法取的输。那么这个转化后的题意就是 Nim 模板题了(Nim 游戏为有 n 堆数，每堆有s_i个，每次可以从任意堆里取 1 到任意多个数，最后取完者胜，求先手是否必胜)。求 a 的异或和即可。根据 Nim 游戏结论若异或和为 0 先手必败，否则先手必胜。

复杂度 $O(n)$。

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
#define mn 1000010
char s[mn];
ll kmp[mn], n, x;
signed main()
{
    scanf("%lld%s", &n, s + 1);
    for (ll i = 2, j = 0; i <= n; ++i)
    {
        while (j > 0 && s[j + 1] != s[i])
        {
            j = kmp[j];
        }
        if (s[j + 1] == s[i])
        {
            ++j;
        }
        kmp[i] = j;
    }
    for (ll i = 1; i <= n; ++i)
    {
        if (kmp[i] != 0 && kmp[i + 1] == 0)
        {
            x ^= kmp[i];
        }
    }
    printf("%s", x ? "580" : "1437");
    return 0;
}
```

Living in Peace with Errors

题意翻译：定义无向图每个点的价值为点权的度数次方，图的价值为点权价值和。给定价值和 $t(1 \le t \le 10^9)$，构造出 $n(1 \le n \le 580)$ 点，点权在 $[2, 58]$，总边数不超过 $\dfrac{5800}{2}$ 的图。若无解输出 `-1`，否则输出任意一个方案。

考点：构造(倍增)

本题解法多样。一种较简单的思路是构造菊花图(所有点都只与特定一个点连边的简单图)。若构造恒有 $p = 2$，那么对点数为 i 的菊花图，有 $i - 1$ 个点度数为 1，1 个点度数为 $i - 1$，其价值为 $2^{i-1} + 2(i - 1)$。对当前的 t，可以先不断构造价值最大不超过 t 的 i 点菊花子图。

当剩下 t 较小时(如 $t \le 58$)，可以进行特判分类讨论，若 $t \ge 4$，构造两个点权为 $2, t - 2$ 的点连成点数为 2 的子图。否则，构造 t 个孤立点，孤立点价值恒为 1。

当 $t > 58$ 时，我们对 $i+1$ 个点的菊花图价值与 i 个点的菊花图价值作差，得价值差为 $2^i + 2i - (2^{i-1} + 2(i-1)) = 2^{i-1} + 2$ 。该价值差恒小于 $2^{i-1} + 2(i-1)$，这意味 $i+1$ 个点的菊花图拼不成但 i 个点能拼成时，需要 i 个点的菊花图只需要至多 1 张。当用上 1 到 30 个点的菊花图时，总价值为：$\sum_{i=1}^{30} 2^{i-1} + 2(i-1) \approx 1.07 \times 10^9 > 7$ ，而 $\sum_{i=1}^{30} i = 465$ ，这意味着最坏情况大约需要 465 个点。而菊花图的边数不大于点数，构造的点权全为 2 ，所以三个条件都能满足。因此该解法是可行的。即只要有解必然能构造出。

下面证明无解是不可能的。当 $1 \le t \le 58$，根据上文解法可知恒有解。当 $58 < t \le 10^9$ ，设 $p_i = 2^{i-1} + 2(i-1)$ ，则 t 一定可以由若干个不重的 p_i 加一个 58 以内的常数表示出来。设首个超过 t 的为 p_i，则令 $t' = t - \sum_{j=1}^{i-1} 2(j-1)$，那么 t' 可以由不超过 i 位的二进制数表示。剩余部分 $\sum_{j=1}^{i-1} 2(j-1)$ 的最大值是 465，将它递归拆分(那么原 t' 加上一个二进制数，作二进制进位以保证每个 p_i 只被用一次)。由此，每个值都能被取到。

也可以构造其他子图，如价值为 $2^0, 2^1, \cdots, 2^{29}$ 的子图，不过这种子图更难构造，可自行尝试。

复杂度 $O(\log^3 t)$ 。

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
ll t, part[63], p = 58, n, c[581][581], w[581];
void solve(ll x)
{
    if (x <= p)
    {
        if (x >= 4)
        {
            ll w1 = 2, w2 = x - 2;
            w[n + 1] = w1, w[n + 2] = w2;
            c[n + 1][n + 2] = c[n + 2][n + 1] = 1;
            n += 2;
            x = 0;
        }
        for (ll i = 1; i <= x; ++i)
        {
            w[++n] = 2;
        }
        return;
    }
    ll k;
    for (k = 62; k >= 1; --k)
    {
        if (x >= part[k])
        {
            break;
        }
    }
    x -= part[k];
    w[n + 1] = 2;
    for (ll i = 1; i < k; ++i)
    {
        ll u = n + 1, v = n + 1 + i;
        c[u][v] = c[v][u] = 1;
        w[v] = 2;
```

```
    }
    n += k;
    solve(x);
}
signed main()
{
    part[1] = 1;
    for (ll i = 2; i <= 62; ++i)
    {
        part[i] = (1LL << (i - 1)) + (i - 1) * 2;
    }
    sc(t);
    solve(t);
    printf("%lld\n", n);
    for (ll i = 1; i <= n; ++i)
    {
        printf("%lld ", w[i]);
    }
    printf("\n");
    for (ll i = 1; i <= n; ++i)
    {
        for (ll j = 1; j <= n; ++j)
        {
            printf("%lld ", c[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Happy Ending

题意翻译：给定 n 点 m 边有向无环图，点权是 a_i，问从任意无入度的点出发到达点 n 的，路径点权和不超过 t 的路径有多少条(对 10^9+7 取模)。$2 \le n \le 10^4, 1 \le m \le 3\times 10^4, 1 \le t \le 10^3$。

考点：拓扑排序 + 动态规划

记 $dp_{i,j}$ 表示从任意无入度点出发到达 i，点权和为 j 时的路径数目。初始时对所有无入度点，有 $dp_{i,a_i}=1$，即从自己到自己一条。所求为 $\sum_{i=1}^{t} dp_{n,i} \bmod (10^9+7)$。递推关系为对每个点 v 和它的所有前驱 u，满足 $\forall j\in[0,t-a_v], dp_{v,i+a_v}=\sum_u dp_{u,j}$。可以用 $O(mt)$ 的时间复杂度求出解。空间复杂度为 $O(nt)$，假设用 long long 为 76 MB。

> 不能用 DFS 搜索，因为可以构造出一些图，使得不同的路径数达到指数级。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
#define mn 10010
#define mm 30010
#define mt 1010
ll n, m, t, dp[mn][mt], hd[mn], cnt, a[mn], ru[mn], ans, mod = 1e9 + 7;
```

```
struct edge
{
    ll to, nx;
} e[mm * 2];
signed main()
{
    sc(n), sc(m), sc(t);
    for (ll i = 1; i <= n; ++i)
    {
        sc(a[i]);
    }
    for (ll i = 1, u, v; i <= m; ++i)
    {
        sc(u), sc(v);
        ++ru[v], e[++cnt] = {v, hd[u]}, hd[u] = cnt;
    }
    queue<ll> q;
    for (ll i = 1; i <= n; ++i)
    {
        if (ru[i] == 0)
        {
            dp[i][a[i]] = 1, q.push(i);
        }
    }
    while (!q.empty())
    {
        ll u = q.front();
        q.pop();
        for (ll i = hd[u], v; i; i = e[i].nx)
        {
            v = e[i].to;
            for (ll j = 0; j + a[v] <= t; ++j)
            {
                (dp[v][j + a[v]] += dp[u][j]) %= mod;
            }
            if (--ru[v] == 0)
            {
                q.push(v);
            }
        }
    }
    for (ll i = 1; i <= t; ++i)
    {
        ans = (ans + dp[n][i]) % mod;
    }
    printf("%lld", ans);
    return 0;
}
```

Diary

> 本题为真·签到题，是选修课期末考核题目，未在筛选赛出现。

题意翻译：每篇文章由 n 件事件组成，每件事件篇幅 1 页纸。初始无文章。有 m 阶段，第 i 阶段初新增 a_i 篇文章，设 $j=(i-1) \bmod n+1$，并对所有前 $j-1$ 件事件都写完且第 j 件事件未写完的文章，写上第 j 件事件。问全过程用多少页纸。$1 \le n,m,a_i \le 10^3$。

考点：模拟

设第 i 阶段加入的文章在第 f_i 天开始写第一件事件。第一天加入的没有前置事件要写，所以 $f_1=1$ 。在第 $[2,n]$ 天加入的文章刚加入的时候都至少有一件前置事件没写，所以只能等待，并在第 $n+1$ 阶段重新开始写第一件事件时开始写这些文章，即 $f_i=n+1(2\le i\le n)$ 。以此类推......不难发现，第 x 阶段加入的文章总是在不小于 x 的第一个 $ny+1$ 阶段开始写 $(y\in N)$ 。总结为：$f_x=n\times\lfloor\dfrac{x-1}{n}\rfloor+1$ 。由于变量是 x ，且 $\lfloor\dfrac{x-1}{n}\rfloor$ 每隔 n 变化一次，所以总是有 n 阶段的文章一起开始写，且在同一天写完。事实上在解题的时候我们甚至不需要用到上面的 f 公式。

用两个整型变量记录当前有多少文章正在被写和多少文章正在等待。然后逐阶段遍历，把每阶段新来的文章数累加到等待变量里，然后判断第 i 阶段是否是满足 $ny+1(y\in N)$ 的形式，若 $i-1\equiv 0 \mod n$ ，那么说明今天在写第一件事件，同时也说明之前的文章都写完了。所以此时清零正在被写的文章数目，并将等待的文章全部放到正在被写的文章里，然后等待的文章数目清零。处理完后，每阶段增加使用正在被写的文章数目张纸张。

时间复杂度为 $O(m)$ ，空间复杂度为 $O(1)$。

> 作为签到题，数据范围故意放过了一些时空复杂度更高的解法，可自行尝试。

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
ll n, m, wait, writing, a, ans;
signed main()
{
    sc(n), sc(m);
    for (ll i = 1; i <= m; ++i)
    {
        sc(a);
        wait += a;
        if ((i - 1) % n == 0)
        {
            writing = wait;
            wait = 0;
        }
        ans += writing;
    }
    printf("%lld", ans);
    return 0;
}
```