目录

二、分数规划问题..... 1
三、多项式相关..... 1
四、集合卷积 fwt && SOS dp 2
五、字符串相关..... 3
六、积性函数与筛法..... 5
七、网络流 && 匹配问题..... 7
八、计数问题&反演容斥&组合数学. 8
九、数论..... 9
十、数据结构..... 9
十一、图论问题..... 13
十二、dp 相关 16
十三、树上问题..... 17
十四、分治、分块、莫队..... 18
十五、高斯消元..... 18
十六、二进制技巧..... 18
十七、计算几何常用函数..... 18
多面体欧拉定理 24
二十、代码查错..... 24
二十一、Python Template..... 24

二、分数规划问题

分数规划问题：（平均值、中位数等问题）

对于解空间 S、**连续的实值函数** $a(\mathbf{x}),b(\mathbf{x})$，满足 $\forall \mathbf{x}\in S, b(\mathbf{x})>0$，求

$$\min_{\mathbf{x}\in S} f(\mathbf{x})=\frac{a(\mathbf{x})}{b(\mathbf{x})}$$

仍然设 λ_0 为答案，则

$$\begin{cases} g(\lambda)=0 \Leftrightarrow \lambda=\lambda_0 \\ g(\lambda)<0 \Leftrightarrow \lambda>\lambda_0 \\ g(\lambda)>0 \Leftrightarrow \lambda<\lambda_0 \end{cases}$$

此时便可以进行二分查找了。

$g(\lambda)=\min[a(\mathbf{x})-\lambda\cdot b(\mathbf{x})]$

三、多项式相关

多项式模板

```
namespace Poly{
ll upd(ll x) { return x + (x >> 63 & mod); }

#define VLL vector<ll>
int limit;
ll omg[N];

void pre_ntt(int len) {
    for (limit = 1; limit < len; limit <<= 1) ;
    static int L = 1;
    for (int &i = L; i < limit; i <<= 1) {
        omg[i] = 1; int w = qpow(3, mod / 2 / i);
        for (int j = 1; j < i; j++) omg[i + j] = omg[i + j - 1] * w % mod;
    }
}
void dft(VLL &p) {
    for (int i = limit >> 1, s = limit; i; i >>= 1, s >>= 1)
        for (int j = 0; j < limit; j += s) for (int k = 0, o = i; k < i; ++k, ++o) {
            int x = p[j + k], y = p[i + j + k];
            p[j + k] = upd(x + y - mod), p[i + j + k] = omg[o] * upd(x - y) % mod;
        }
}
void idft(VLL &p) {
    for (int i = 1, s = 2; i < limit; i <<= 1, s <<= 1)
        for (int j = 0; j < limit; j += s) for (int k = 0, o = i; k < i; ++k, ++o) {
            int x = p[j + k], y = omg[o] * p[i + j + k] % mod;
            p[j + k] = upd(x + y - mod), p[i + j + k] = upd(x - y);
        }
    reverse(p.begin() + 1, p.end());
    for (int i = 0, inv = qpow(limit, mod - 2); i < limit; i++) p[i] = p[i] * inv % mod;
}
void ntt(VLL &p, int op) {
    p.resize(limit);
    if (op == 1) dft(p);
    else idft(p);
}
VLL operator * (VLL a, VLL b) {
    int len = a.size() + b.size();
    pre_ntt(len);
    ntt(a, 1), ntt(b, 1);
    for (int i = 0; i < limit; i++) a[i] = a[i] * b[i] % mod;
    ntt(a, 0), a.resize(len);
    return a;
}
VLL operator + (VLL a, const VLL &b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < a.size(); i++)
        a[i] = upd(a[i] + (i < b.size() ? b[i] : 0) - mod);
    return a;
}
VLL operator - (VLL a, const VLL &b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < a.size(); i++)
        a[i] = upd(a[i] - (i < b.size() ? b[i] : 0));
    return a;
}
VLL inv(VLL a, int n = -1) {
    VLL res(1), t, t2;
    assert(a[0]);
    res[0] = qpow(a[0], mod - 2);
    for (int l = 1; l < a.size(); l <<= 1) {
        t2 = a, t2.resize(l << 1);
        t = t2 * res, t.resize(l << 1), t = t * res, t.resize(l << 1);
        res = res + res - t;
    }
    return res.resize(a.size()), res;
}
VLL diff(VLL a, int n) {
    a.resize(n - 1);
    for (int i = 0; i < n - 1; i++) a[i] = a[i + 1] * (i + 1) % mod;
    return a;
}
VLL integ(VLL a, int n) {
    a.resize(n + 1);
    for (int i = n; i; i--) a[i] = a[i - 1] * ifac[i] % mod * fac[i - 1] % mod; // inv[i] = (ifac[i] * fac[i - 1])
    return a[0] = 0, a;
}
VLL ln(VLL a, int n) {
    VLL b = inv(a, n);
    a = diff(a, n), pre_ntt(n << 1);
    ntt(a, 1), ntt(b, 1);
    for (int i = 0; i < limit; i++) a[i] = a[i] * b[i] % mod;
    ntt(a, 0), a.resize(n - 1);
    return integ(a, n - 1);
}
VLL exp(VLL a, int n) {
    if (n == 1) return {1};
    a.resize(n); VLL b = exp(a, (n + 1) >> 1), c;
    c = ln(b, n + 1), b.resize(n);
    pre_ntt(n + 1);
    for (int i = 0; i < n; i++) a[i] = upd(a[i] - c[i] + (!i));
    c = b, ntt(a, 1), ntt(c, 1);
```

```
    for (int i = 0; i < limit; i++) a[i] = a[i] * c[i] % mod;
    ntt(a, 0), a.resize(n);
    for (int i = (n + 1) >> 1; i < n; i++) b[i] = a[i];
    return b;
}
}
```

任意模数 mtt:

```
const int N = 4e6 + 100;
const int M = 32767;
const db pi = acos(-1);
const ll mod = 998244352;
struct cp{
    db r, i;
    cp(double r = 0, double i = 0) : r(r), i(i){}
    cp operator * (const cp &a) {return cp(r * a.r - i * a.i, r * a.i + i * a.r);}
    cp operator + (const cp &a) {return cp(r + a.r, i + a.i);}
    cp operator - (const cp &a) {return cp(r - a.r, i - a.i);}
}w[N], A[N], B[N], AA[N], BB[N];
int len, cc, wh[N];
cp conj(cp a)
{return cp(a.r, -a.i);}
void fft(cp *a, bool inv) {
    cp tmp;
    for(int i = 0; i < len; i++)
        if(i < wh[i])swap(a[i], a[wh[i]]);
    for(int l = 2; l <= len; l <<= 1) {
        int mid = l >> 1;
        for(int i = 0; i < len; i += l) {
            for(int j = 0; j < mid; j++) {
                tmp = a[i + j + mid] * (inv ? w[len - len / l * j] : w[len / l * j]);
                a[i + j + mid] = a[i + j] - tmp;
                a[i + j] = a[i + j] + tmp;
            }
        }
    }
}
VLL mul(VLL &a, VLL &b) { // mtt with M = 32767
    VLL res;
    len = 1, cc = 0;
    while(len < a.size() + b.size())
        len <<= 1, ++cc;
    for(int i = 1; i <= len; i++)
        wh[i] = (wh[i >> 1] >> 1) | ((i & 1) << (cc - 1));
    for(int i = 0; i <= len; i++)
        w[i] = cp(cos(2.0 * pi * i / len), sin(2.0 * pi * i / len));
    int sz = a.size() + b.size() - 1;
    a.resize(len), b.resize(len);
    for(int i = 0; i < len; i++) {
        A[i] = cp(a[i] / M, a[i] % M);
        B[i] = cp(b[i] / M, b[i] % M);
    }
    fft(A, 0), fft(B, 0);
    for(int i = 0; i < len; i++) {
        cp aa, bb, cc, dd;
        int j = (len - i) % len;
        aa = (A[i] + conj(A[j])) * cp(0.5, 0);
        bb = (A[i] - conj(A[j])) * cp(0, -0.5);
        cc = (B[i] + conj(B[j])) * cp(0.5, 0);
        dd = (B[i] - conj(B[j])) * cp(0, -0.5);
        AA[i] = aa * cc + aa * dd * cp(0, 1);
        BB[i] = bb * dd + bb * cc * cp(0, 1);
    }
    fft(AA, 1), fft(BB, 1);
    res.resize(sz);
    for(int i = 0; i < sz; i++) {
        ll ac, ad, bc, bd;
        ac = (ll)(AA[i].r / len + 0.5) % mod;
        ad = (ll)(AA[i].i / len + 0.5) % mod;
        bd = (ll)(BB[i].r / len + 0.5) % mod;
        bc = (ll)(BB[i].i / len + 0.5) % mod;
        res[i] = (ac * M * M + (ad + bc) * M + bd) % mod;
    }
    return res;
}
```

另类分治 fft:

求 f（x）=sigma f（i）×f（x-i）

```
if(l==r) {
    if(l==1)f[l]=1;
    else Mul(f[l],jc[l-2]);
    return;
}
int mid=(l+r)>>1;
sol(l,mid);
pre_ntt(r-l+2);
for(int i=0;i<len;i++)A[i]=B[i]=0;
for(int i=l;i<=mid;i++)A[i-l]=F(i);//1LL*(jc[i-1]-f[i]+mod)%mod*ijc[i-1]%mod;
for(int i=1;i<=r-l;i++)B[i]=G(i);
ntt(A,0),ntt(B,0);
for(int i=0;i<len;i++)A[i]=1LL*A[i]*B[i]%mod;
ntt(A,1);
for(int i=mid+1;i<=r;i++)Ad(f[i],A[i-l]);
if(l>1)  {
    for(int i=0;i<len;i++)A[i]=B[i]=0;
    for(int i=l;i<=mid;i++)A[i-l]=G(i);//1LL*(jc[i-1]-f[i]+mod)%mod*ijc[i-1]%mod;
    for(int i=1;i<=r-l;i++)B[i]=F(i);
    ntt(A,0),ntt(B,0);
    for(int i=0;i<len;i++)A[i]=1LL*A[i]*B[i]%mod;
    ntt(A,1);
    for(int i=mid+1;i<=r;i++)Ad(f[i],A[i-l]);
}
sol(mid+1,r);
```

一些应用

1.fft 做字符串匹配：一般用来搞例如通配符之类奇奇怪怪的匹配，构造 F[i] =sigma (S[i] - T[i])^2，反转 T 串，再把平方拆开很容易发现是 fft 的形式。如果 S 中有通配符，就令通配符的值为 0，构造 F[i] = sigma(S[i] - T[i])^2 * S[i]，然后类似的反转拆开，fft 算。（warning：用 ntt 时给每个字符分配一个 rand 权值，不要只用 ascii 码，容易被卡！）

2.fft 做可行性 dp，保证所有物品价值的和大小在 1e5 以内时，先对物品价值进行排序，然后对于价值 vi 物品构建生成函数 1 + x^vi，分治 fft 相乘即可，复杂度 nlog^2

四、集合卷积 fwt && SOS dp

集合卷积:

And 正变换相当于将每个位置的值加到这个位置所有的子集位置上去，逆变换相当于每个位置减到（下传减法）这个位置的所有子集位置上去。Or 正变换相当于将每个位置的值加到这个位置的超集位置上去，逆变换相当于将每个位置的值减到这个位置的超集位置上去。Xor: F(i) = Sum of A[j], (j&i is odd in binary) - Sum of A[j], (j&i) is even in binary). 逆变换的唯一区别在于最后除以 len。

```
void And(ll *a,bool inv) {
    for(int l=2,md=1;l<=len;l<<=1,md<<=1)
        for(int i=0;i<len;i+=l)
            for(int j=0;j<md;j++)
                inv?Dw(a[i+j],a[i+j+md]):Ad(a[i+j],a[i+j+md]);
}
```

```
void Or(ll *a,bool inv) {
  for(int l=2,md=1;l<=len;l<<=1,md<<=1)
    for(int i=0;i<len;i+=l)
      for(int j=0;j<md;j++)
        inv?Dw(a[i+j+md],a[i+j]):Ad(a[i+j+md],a[i+j]);
}
void Xor(ll *a,bool inv) {
  ll tp;
  for(int l=2,md=1;l<=len;l<<=1,md<<=1)
    for(int i=0;i<len;i+=l)
      for(int j=0;j<md;j++)
      {
        tp=a[i+j+md];
        a[i+j+md]=(a[i+j]-tp+mod)%mod;
        Ad(a[i+j],tp);
        if(inv)Mul(a[i+j],inv2),Mul(a[i+j+md],inv2);
      }
}
```

SOS dp $F[mask]=\sum_{i\in mask}A[i]$

可以是一些十进制数，对每个数求所有数位都比它小的数的个数。

初始化 dp 数组，dp(val) = 数字 val 出现次数则首先从低位到高位枚举数位 i，然后从小到大枚举 val，若 val 这一位是 0，则 dp 值不变，否则 dp(val)加上把 val 的第 i 位改小 1 的 dp 值，是一个高维前缀和的过程。

```
for(int i = 0, bas = 1; i < 6; i++, bas *= 10) {
  for(int s = 0; s <= lim; s++) {
    if((s / bas) % 10 < 9)
      dp[s + bas] += dp[s];
  }
}
```

五、字符串相关

3.Z 函数（exkmp）：

Z 函数也称为扩展 kmp，假设现在有一个串 s 和 t，通过一个对 t 的 O（n）时间预处理，之后可以线性时间对串 s 的每一个下标 i，求以 s[i]为开头能和串 t 匹配的最大长度。

具体做法是对 t 的每一个下标 i 求其能与 t 开头匹配的最大长度，求的时候利用一下之前已经求出的信息，匹配的时候同理。

```
void sol() {
  string s, t;
  cin >> t >> s;
  int n, m;
  n = s.length();
  m = t.length();
  VI Z(n, 0);
  int lp, rp;
  lp = rp = 0;
  for(int i = 1; i < n; i++) {
    if(i <= rp && i + Z[i - lp] - 1 < rp)
      Z[i] = Z[i - lp];
    else {
      int k = max(0, rp - i + 1);
      while(i + k < n && s[k] == s[i + k])
        ++k;
      Z[i] = k;
    }
    if(i + Z[i] - 1 > rp)
      lp = i, rp = i + Z[i] - 1;
  }
  Z[0] = n;
  ll res = 0;
  for(int i = 0; i < n; i++)
    res ^= 1LL * (i + 1) * (Z[i] + 1);
  cout << res << '\n';
  res = 0;
  lp = rp = -1;
  VI ans(m, 0);
  for(int i = 0; i < m; i++) {
    if(i <= rp && i + Z[i - lp] - 1 < rp)
      ans[i] = Z[i - lp];
    else {
      int k = max(0, rp - i + 1);
      while(i + k < m && k < n && s[k] == t[i + k])
        ++k;
      ans[i] = k;
    }
    if(i + ans[i] - 1 > rp)
      lp = i, rp = i + ans[i] - 1;
    res ^= 1LL * (i + 1) * (ans[i] + 1);
  }
  cout << res << '\n';
}
```

4.ac 自动机

约等于 kmp 进阶版， 可求多个串在其他多个串哪些位置出现。对于所有的查询串先建一棵 trie 树，然后类似 kmp 的失配数组那样求出每个节点失配指针，就是字典树里能和当前这个位置的后缀匹配最长的前缀（和 kmp 类似，当然也不能是它本身），求法就是对于节点 x，沿着父亲的失配指针跳，直到跳到某个节点，它有和 x 节点代表字符相同的字符的孩子（或者是一直跳到根也没有），匹配也就类似 kmp 类比一下了。

一般对出现多个的串 t1,t2, …,ti 统一建一个 ac 自动机，用较长的某一串 s 在上面跑，可以处理诸如 s 的某一前缀的后缀，匹配 ti 的一些前缀这样的问题。

```
struct Aho {
  int nxt[N][26], fail[N], tot = 1, ed[N], vis[N], id_mx;
  vector<int> adj[N];
  void init() {
    for (int i = 0; i <= tot; i++) {
      memset(nxt[i], 0, sizeof nxt[i]);
      fail[i] = 0;
      ed[i] = 0;
      adj[i].clear();
      vis[i] = 0;
    }
    tot = 1;
    id_mx = 0;
  }
  vector<int> ins(string s, int id) {
    int n = s.length(), ps = 0, nw;
    vector<int> cur;
    cur.pb(ps);
    id_mx = max(id_mx, id);
    for (int i = 0; i < n; i++) {
      nw = s[i] - 'a';
      if (!nxt[ps][nw]) nxt[ps][nw] = tot++;
      ps = nxt[ps][nw];
      cur.pb(ps);
    }
    ed[ps] = id;
    return cur;
  }
  void build() {
    int nw, nx, tmp;
    fail[0] = 0;
    vector<int> que;
    que.pb(0);
    for (int q = 0; q < que.size(); q++) {
      nw = que[q];
      for (int i = 0; i < 26; i++) {
        if (nxt[nw][i]) {
          nx = nxt[nw][i];
```

```
                if (!nw) fail[nx] = 0;
                else {
                    tmp = fail[nw];
                    while (tmp && !nxt[tmp][i]) tmp = fail[tmp];
                    fail[nx] = nxt[tmp][i];
                }
                que.pb(nx);
            }
        }
        for (int i = 1; i < tot; i++) {
            adj[fail[i]].pb(i);
        }
    }
    void match(string s) {
        int n = s.length(), ps = 0;
        for (int i = 0; i < n; i++) {
            int nw = s[i] - 'a';
            while (ps && !nxt[ps][nw]) ps = fail[ps];
            ps = nxt[ps][nw];
            ++vis[ps];
        }
    }
    void sol() {
    }
}aho;
// usage: aho.init() aho.ins(), aho.build()
```

5.后缀自动机：后缀自动机维护的每个节点，代表结束位置集合相同的多个子串，即前缀的后缀。做一些题时，可以反过来，变成后缀的前缀，此时 fail 树即后缀树，但是要处理出 fail 边代表的字符。

```
const int SAM_SZ = 2e6 + 100;
struct SAM {
    bool ocr[SAM_SZ][2]; // 多串改 map
    int tot, las, nxt[SAM_SZ][26], fa[SAM_SZ], mx[SAM_SZ], mx_ed[SAM_SZ];
    vector<int> adj[SAM_SZ]; // fail tree (parent tree), edge sorted in lexi order for suffix tree
    void init() {
        for (int i = 0; i <= tot; i++) {
            memset(nxt[i], 0, sizeof nxt[i]);
            fa[i] = mx[i] = 0;
            ocr[i][0] = ocr[i][1] = 0;
            mx_ed[i] = -1;
            adj[i].clear();
        }
        tot = 1, las = 1;
    }
    int mn(int p) {
        return mx[fa[p]] + 1;
    }
    void ins(int x, int op = 0, int ps = -1) {
        int p = las, np = nxt[las][x], t, nt;
        if (np) { // 广义用
            if (mx[p] + 1 == mx[np]) {
                las = np;
                ocr[np][op] = 1;
            } else {
                nt = ++tot;
                mx[nt] = mx[p] + 1;
                memcpy(nxt[nt], nxt[np], sizeof nxt[np]);
                fa[nt] = fa[np], fa[np] = nt;
                while (p && nxt[p][x] == np) {
                    nxt[p][x] = nt, p = fa[p];
                }
                ocr[nt][op] = 1;
                las = nt;
            }
            return;
        }
        np = ++tot, mx[np] = mx[p] + 1;
        las = np, ocr[np][op] = 1, mx_ed[np] = ps;
        while (p && !nxt[p][x]) {
            nxt[p][x] = np;
            p = fa[p];
        }
        if (!p) {
            fa[np] = 1;
            return;
        }
        t = nxt[p][x];
        if (mx[t] == mx[p] + 1) {
            fa[np] = t;
        } else {
            nt = ++tot, mx[nt] = mx[p] + 1;
            memcpy(nxt[nt], nxt[t], sizeof nxt[t]);
            fa[nt] = fa[t], fa[t] = fa[np] = nt;
            while (p && nxt[p][x] == t) {
                nxt[p][x] = nt;
                p = fa[p];
            }
        }
    }
    void build_edge(string &s) { // 出边满足后缀树字典序，需要插入反串
        for (int i = 2; i <= tot; i++) {
            adj[fa[i]].pb(i);
        }
        auto dfs = [&](auto self, int x) -> void{
            for (int y : adj[x]) {
                self(self, y);
                mx_ed[x] = max(mx_ed[x], mx_ed[y]);
            }
        };
        dfs(dfs, 1);
        for (int i = 1; i <= tot; i++) {
            sort(all(adj[i]), [&](int x, int y) {
                int cx = s[mx_ed[x] - mx[i]];
                int cy = s[mx_ed[y] - mx[i]];
                return cx < cy;
            });
        }
    }
    void build(string &s) {
        init();
        for (int i = 0; i < s.length(); i++) {
            ins(s[i] - 'a', 0, i);
        }
        build_edge(s);
    }
    vector<int> match(string t) {
        vector<int> cur;
        int nw = 1, cur_len = 0;
        cur.pb(nw);
        for (int i = 0; i < t.length(); i++) {
            int x = t[i] - 'a';
            while (nw > 1 && !nxt[nw][x]) {
                nw = fa[nw];
                cur_len = mx[nw];
            }
            if (nxt[nw][x]) nw = nxt[nw][x], cur_len++;
            cur.pb(nw);
        }
        return cur;
    }
    void sol(int n) {
    }
}sam;
```

6.后缀数组//rnk[i]表示下标 i 起始后缀排名是多少，sa[i]表示排名第 i 小的起始位置下标，ht[i] = lcp(sa[i], sa[i - 1])

```
const int N = 2e6 + 100; // 2 * string_length
struct SA {
  int n;
  string s;
  int m, x[N], y[N], bin[N], rnk[N], sa[N], ht[N];
  // m number of dif vals, x[i] = suf i's first dimension val
  // y[i] = who is rank i in second dimension
  void rsort() {
    for (int i = 0; i <= m; i++) bin[i] = 0;
    for (int i = 1; i <= n; i++) bin[x[i]]++;
    for (int i = 1; i <= m; i++) bin[i] += bin[i-1];
    for (int i = n; i >= 1; i--) sa[bin[x[y[i]]]--] = y[i];
  }
  void cal(int _n, string _s)
  {
    n = _n, s = _s;
    int num;
    for (int i = 0; i <= 2 * n; i++) { // remember clear!!
      x[i] = y[i] = bin[i] = rnk[i] = sa[i] = ht[i] = 0;
    }
    for(int i = 1; i <= n; i++){
      x[i] = s[i], y[i] = i;
    }
    m = 130, rsort();
    for (int l = 1; l <= n; l <<= 1) {
      num = 0;
      for (int i = n - l + 1; i <= n; i++) y[++num] = i;
      for (int i = 1; i <= n; i++) {
        if(sa[i] > l) {
          y[++num] = sa[i] - l;
        }
      }
      rsort();
      for (int i = 0; i <= n; i++) {
        swap(x[i], y[i]);
      }
      num = x[sa[1]] = 1;
      for(int i = 2; i <= n; i++) {
        x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + l] ==
= y[sa[i - 1] + l])? num: ++num;
      }
      if(num == n)break;
      m = num;
    }
    for (int i = 1; i <= n; i++) {
      rnk[sa[i]] = i;
    }
    int k = 0;
    for (int i = 1, j; i <= n; i++) {
      if(rnk[i] == 1) {
        ht[1] = k = 0;
        continue;
      }
      if (k) --k;
      j = sa[rnk[i] - 1];
      while (i + k <= n && j + k <= n && s[i + k] == s[j +
k]) ++k;
      ht[rnk[i]] = k;
    }
  }
} sa;
```

后缀数组处理本质不同字串的话，可以考虑 sa[i]为开头没有被之前统计的字串，发现之前被统计的数量就是 h[i]，减掉即可。

后缀树，串反转建后缀自动机，parent 树就是后缀树，不过每个边具体是什么需要额外维护。如果要看后缀树上表示的边到底压缩了哪些字母我们可以考虑，在后缀自动机上用我当前的节点能表示的最长串-fa[当前节点]表示的最长串，然后将其反过来就是后缀树上压缩的边了。fft 字符串匹配，通配之类问题，见多项式。

询问一些多个串匹配的最大长度，类似后缀数组的处理，将多个串排序，求 lcp 数组。

7.回文自动机，对奇数长度偶数长度各建树，奇根表示长度-1，偶根表示长度 0，转移边表示同时 xx 跳，fail 直接暴力找。

```
const int N = 1e6 + 100;
struct PAM{
  int tot, len[N], las, nxt[N][26], dis[N], fail[N], ocr[N];
  int dep[N];
  vector<int> ed[N];
  string s;
  int n;
  void ins(int c, int ps)  {
    int nw = las;
    while(ps - len[nw] - 1 < 0 || c != s[ps - len[nw] - 1] - '
a') {
      nw = fail[nw];
    }
    if(nxt[nw][c]) {
      las = nxt[nw][c];
      ed[las].pb(ps);
      ocr[las]++;
      return;
    }
    int np = ++tot;
    len[np] = len[nw] + 2;
    int t = fail[nw];
    while(ps - len[t] - 1 < 0 || c != s[ps - len[t] - 1] - 'a') {
      t = fail[t];
    }
    fail[np] = nxt[t][c];
    nxt[nw][c] = np;
    las = np;
    ed[np].pb(ps);

    dep[las] = dep[fail[las]] + 1;
    ocr[las]++;
  }
  void init(string _s, int _n)  {
    s = _s, n = _n;
    tot = 1;
    len[1] = -1;
    len[0] = 0;
    dis[0] = dis[1] = 0;
    fail[0] = 1;
    dep[0] = dep[1] = 0;
    las = 0;
    for (int i = 0; i < n; i++) {
      ins(s[i] - 'a', i);
    }
  }
  void sol() {
  }
}pam;
```

六、积性函数与筛法

积性函数即 gcd(a, b) = 1 时满足 f(ab) = f(a) * f(b)的函数

迪利克雷卷积：

f*g(n) = sigma f(d) * g(n / d) 其中 d|n

常用积性函数及其迪利克雷卷积

- $\epsilon(n) = [n = 1]$
- $1(n) = 1$
- $\mathrm{Id}(n) = n$
- $\mu(n) = [\max(e_1, e_2, \ldots, e_k) \leq 1](-1)^k$
- $\varphi(n) = n\prod_{d|n}(1 - \frac{1}{d})$
- $\mathrm{d}(n) = \sum_{d|n} 1$
- $\sigma(n) = \sum_{d|n} d$
- $\lambda(n) = (-1)^k$
- $\ldots$

μ∗1=ϵ【莫比乌斯反演】【μ 与 1 互为逆元】ϕ∗1=Id ，ϕ=Id∗μ ， d=1∗1，1=μ∗d

首先对于每个x = [floor(N / i)]， 求$\sum_{i=1}^{x}[i\,is\,prime] * i^k$，先预处理根号N以内的质数k次前缀和，记为sum[i]，令g(N, i)表示埃氏筛第i轮没有被筛去的数的k次幂和，

可以发现所求就是i = cnt，cnt为根号N内质数个数。

可以发现

g(N,i) = g(N,i−1)，当prime[i]^2 > N

g(N,i) = g(N,i−1) − prime[i]^k∗(g(⌊N/prime[i]⌋,i−1)−sum[i−1])

然后设S(N, i) 为1到N中最小质因数大于等于prime[i] 的f(i)之和，所求就是S(N, 1) + f(1)

如何求S(N, i)

1.考虑质数贡献 $\sum_{j=1}^{N}[j\,is\,a\,prime]f(j) - \sum_{j=1}^{i-1} f(prime_j)$

2.考虑合数贡献，通过枚举最小质因子以及出现幂次

$\sum_{j=1}^{prime_j^2<=N}\sum_{k=1}^{prime_j^{k+1}<=N}(S(\lfloor\frac{N}{prime_j^k}\rfloor, j+1) * f(prime_j^k) + f(prime_j^{k+1}))$

(第一个sigma下标是j = i， 打错了)

以上为 min_25 筛做法，有时一些函数不是积性，但满足一些特殊性质，也可以用 min25

```
1.  namespace siever {
2.      const int N = 2e5 + 100;
3.      ll n, B, num;
4.      ll val[N], g[2][N], sum[2][N], f[N], pre[N];
5.      int ps1[N], ps2[N];
6.      int pos(ll x) {
7.          if(x <= B)
8.              return ps1[x];
9.          return ps2[n / x];
10.     }
11.     int pnum;
12.     int pri[N];
13.     bool np[N];
14.     void work(ll _n) {
15.         n = _n;
16.         B = sqrt(n);
17.         /*
18.             sum[k][i] = Prefix Sum of [pri_i ^ k]
19.         */
20.         for (int i = 2; i <= B; i++) {
21.             if(!np[i]) {
22.                 pri[++pnum] = i;
23.                 sum[0][pnum] = pnum;
24.                 sum[1][pnum] = (sum[1][pnum - 1] + i) % mod;
25.             }
26.             for(int j = 1; j <= pnum && i * pri[j] <= B; j++) {
27.                 np[i * pri[j]] = 1;
28.                 if(i % pri[j] == 0)
29.                     break;
30.             }
31.         }
32.         /*
33.             Give index to Each Block of [n / d]
34.             Small Index -> Bigger Original Value
35.
36.             g[k][i] Indicates Prefix Sum of (i ^ k), Notice it Start
    s From Index 2nd
37.         */
38.         for (ll l = 1, r; l <= n; l = r + 1) {
39.             ll x = n / l;
40.             r = n / x;
41.             val[++num] = x;
42.             if(x <= B)
43.                 ps1[x] = num;
44.             else ps2[n / x] = num;
45.             x %= mod;
46.             g[0][num] = (x - 1 + mod) % mod;
47.             g[1][num] = (x - 1 + mod) % mod * (x + 2) / 2 % mo
    d;
48.         }
49.         /*
50.             Calculate Prefix Sum of (i ^ k) with [i is Prime]
51.             g[k][num] = Prefix Sum from (1 to val[num]) of (i ^
    k) with [i is Prime]
52.         */
53.         for(int i = 1; i <= pnum; ++i) {
54.             for(int j = 1; j <= num && 1LL * pri[i] * pri[i] <= val
    [j]; ++j) {
55.                 int bf = pos(val[j] / pri[i]);
56.                 g[0][j] = (g[0][j] - g[0][bf] + sum[0][i - 1] + mod)
     % mod;
57.                 g[1][j] = (g[1][j] - pri[i] * (g[1][bf] - sum[1][i - 1]
    + mod) % mod + mod) % mod;
58.             }
59.         }
60.         /*
61.             Calculate the contribution of Prime Position to Fina
    l answer
62.             f[i] = the contribution of Prime Position to Final ans
    wer in [1 to val[i]]
63.
64.             Calculate Pre[i] = Contribution of the [1st Prime to
    i'th Prime]
65.         */
66.         for(int i = 1; i <= num; i++)
67.             f[i] = (g[1][i] - g[0][i] + mod) % mod;
68.         for(int i = 1; i <= pnum; i++)
69.             pre[i] = (sum[1][i] - sum[0][i] + mod) % mod;
70.         for(int j = pnum; j >= 1; j--) {
71.             for(int i = 1; i <= num; i++) {
72.                 if(1LL * pri[j] * pri[j] > val[i])
73.                     break;
74.                 ll tmp = pri[j];
75.                 for(int e = 1; tmp <= val[i] / pri[j]; e++, tmp *=
    pri[j]) {
76.                     /*
77.                         s = Contribution of [Prime_{j} ^ e]
78.                         t = Contribution of [Prime_{j} ^ (e+1)]
79.                     */
80.                     ll s = (tmp - tmp / pri[j]) % mod;
81.                     ll t = (tmp * pri[j] - tmp) % mod;
82.                     (f[i] += (s * (f[pos(val[i] / tmp)] - pre[j] + mod)
     + t)) %= mod;
83.                 }
84.             }
85.         }
86.         // 1 should be replaced by Contribution of index 1
87.         cout << (f[1] + 1) % mod;
88.     }
89. }
```

最短线性递推式： （bm）

求一个 r 数组，$\forall m + 1 \leq i \leq n, s.t. \sum_{j=1}^{m} R_j * a_{i-j} = a_i$，且 m 最短。

```
1.  int cnt, fail[MAXN];
2.  double val[MAXN], delta[MAXN];
3.  vector <double> ans[MAXN];
4.  int main() {
5.      int n; read(n);
6.      for (int i = 1; i <= n; i++)
7.          scanf("%lf", &val[i]);
8.      for (int i = 1; i <= n; i++) {
9.          double tmp = val[i];
10.         for (unsigned j = 0; j < ans[cnt].size(); j++)
11.             tmp -= ans[cnt][j] * val[i - j - 1];
12.         delta[i] = tmp;
```

```
        if (fabs(tmp) <= eps) continue;
        fail[cnt] = i;
        if (cnt == 0) {
            ans[++cnt].resize(i);
            continue;
        }
        double mul = delta[i] / delta[fail[cnt - 1]];
        cnt++; ans[cnt].resize(i - fail[cnt - 2] - 1);
        ans[cnt].push_back(mul);
        for (unsigned j = 0; j < ans[cnt - 2].size(); j++)
            ans[cnt].push_back(ans[cnt - 2][j] * -mul);
        if (ans[cnt].size() < ans[cnt - 1].size()) ans[cnt].resize(a
    ns[cnt - 1].size());
        for (unsigned j = 0; j < ans[cnt - 1].size(); j++)
            ans[cnt][j] += ans[cnt - 1][j];
    }
    for (unsigned i = 0; i < ans[cnt].size(); i++)
        cout << ans[cnt][i] << ' ';
    return 0;
}
```

七、网络流 && 匹配问题

最大流 Dinic:

```
const int N = 1e6 + 100;
struct Dinic {
    int s, t;  // 汇点 t 要是最大编号点
    int hed[N], cur[N], dep[N], nxt[N], las[N], to[N], cnt;
    bool bfs() {
        memset(dep, 0, (t + 5) * sizeof (int));
        dep[s] = 1;
        vector<int> que;
        que.pb(s);
        for (int q = 0; q < que.size(); q++) {
            int nw = que[q];
            for (int i = hed[nw]; ~i; i = nxt[i]) {
                if (!dep[to[i]] && las[i]) {
                    dep[to[i]] = dep[nw] + 1;
                    que.pb(to[i]);
                }
            }
        }
        return dep[t];
    }
    int dfs(int ps,int flow) {
        if(ps == t) return flow;
        for (int &i = cur[ps]; ~i; i = nxt[i]) {
            if (dep[to[i]] == dep[ps] + 1 && las[i]) {
                int nw = dfs(to[i], min(flow, las[i]));
                if (nw) {
                    las[i] -= nw;
                    las[i ^ 1] += nw;
                    return nw;
                }
            }
        }
        return 0;
    }
    void add_edge(int u,int v,int w)//加边
    {
        nxt[++cnt] = hed[u], las[cnt] = w, to[cnt] = v, hed[u]
    = cnt;
        nxt[++cnt] = hed[v], las[cnt] = 0, to[cnt] = u, hed[v] =
     cnt;
    }
    void init(int _s, int _t)
    {
        memset(nxt,-1,sizeof nxt);
        memset(hed,-1,sizeof hed);
        cnt = -1;
        s = _s, t = _t;
        return;
    }
    ll max_flow() {
        ll flow = 0;
        while (bfs()) {
            for (int i = 0; i <= t; i++) cur[i] = hed[i];
            ll tmp;
            while ((tmp = dfs(s, (ll)1e18)) != 0) {
                flow += tmp;
            }
        }
        return flow;
    }
}dinic;
```

最小费用最大流 mcmf:

```
const int N = 1e6 + 100;
struct Mcmf {
    int s, t;
    bool inq[N];
    int pre[N], dis[N], hed[N], nxt[N], las[N], to[N], cost[N], c
    nt;
    bool spfa()
    {
        memset(inq, 0, (t + 5) * sizeof (int));
        memset(pre, -1, (t + 5) * sizeof (int));
        memset(dis, 127, (t + 5) * sizeof (int));
        vector<int> que;
        dis[s] = 0, inq[s] = 1, que.pb(s);
        for (int q = 0; q < que.size(); q++) {
            int nw = que[q];
            inq[nw] = 0;
            for (int i = hed[nw]; ~i; i = nxt[i]) {
                int v = to[i];
                if (las[i] && dis[v] > dis[nw] + cost[i]) {
                    dis[v] = dis[nw] + cost[i];
                    pre[v] = i;
                    if (!inq[v]) {
                        que.pb(v);
                        inq[v] = 1;
                    }
                }
            }
        }
        return dis[t] <= (int)1e9;
    }
    pair<ll, ll> mcmf()
    {
        ll flow = 0, nw = 0, spd = 0;
        while(spfa())
        {
            nw = 1e9;
            for (int i = pre[t]; ~i; i = pre[to[i ^ 1]]) {
                nw = min(nw, las[i]);
            }
            for (int i = pre[t]; ~i; i = pre[to[i ^ 1]]) {
                las[i] -= nw;
                las[i ^ 1] += nw;
            }
            flow += nw, spd += nw * dis[t];
        }
        return make_pair(flow, spd);
    }
    void init(int _s, int _t)
    {
        memset(nxt,-1,sizeof nxt);
        memset(hed,-1,sizeof hed);
        cnt = -1;
        s = _s, t = _t;
```

```
53.         return;
54.     }
55.     void add_edge(int u,int v,int w, int c)//加边
56.     {
57.         nxt[++cnt] = hed[u], las[cnt] = w, to[cnt] = v, hed[u]
    = cnt, cost[cnt] = c;
58.         nxt[++cnt] = hed[v], las[cnt] = 0, to[cnt] = u, hed[v] =
     cnt, cost[cnt] = -c;
59.     }
60. };
```

数据范围较大时，可能会有优秀的增广策略。

网络流建图技巧:

1. 把有关系的点之类的建边,对于题目条件转化成对于流量或费用的的限制.

2. 搞清楚是最大流还是费用流…

3. 常用模型：

（1）点覆盖、最小点覆盖：点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点” 覆盖了所有“边”。**（2）**最小点覆盖：点最少的点覆盖。**（3）**点覆盖数：最小点覆盖的点数。**（4）**独立集：独立集即一个点集，集合中任两个结点不相邻，则称 V 为独立集。**（5）**最大独立集：点最多的独立集。**（6）**独立数：最大独立集的点。**（7）**若把上面最小点覆盖和最大独立集中的端点数改成点的权值，分别就是最小点权覆盖和最大点权独立集的定义。**（8）**最大点权独立集=总权值-最小点权覆盖集，最小点权覆盖集=图的最小割值=最大流。**（9）**最大权闭合子图：选择某个东西可以得到一定收益，但选了它就必须选择它的一些后继，而不同物品可能有一些相同后继，求能选到的最大收益。做法：建源点 s，向正权点连流量为点权的边，正权点再向它后继连流量 inf 的边，建汇点 t，负权点向汇点连点权绝对值边，跑一波最小割（最大流），然后答案就是正权点和-最小割代价.**（10）**转化的一些技巧,如要求最大费用流可以把所有边费用取负后做,答案再取负,还有一些对于点的限制,可以把点一分为二,在拆出来的两个点间建边作为原来限制…

八、计数问题&反演容斥&组合数学

1.卡特兰数：统计 n 个元素出栈的方案数，h(n)=C(2n,n)/(n+1) =C(2n,n) - C(2n,n-1)

看成走折线图，每次执行一次进栈或出栈操作横坐标加 1，同时若进栈纵坐标加 1、出栈纵坐标-1，问题等价于从(0,0)到(2n,0)且不碰到 y=-1 直线方案数。假设某个方案碰到了-1，我们考察该方案从原点第一次碰到-1 的走法，把每一步都取反(向上向下互换)，发现可以等价于从(0,-2)碰到-1，那么可能会碰到-1 的方案数等价于从(0,-2)走到（2n,0）的方案数，即 C(2n, n-1)，故最终方案数 C(2n,n) - C(2n,n-1)

2.斯特林数:

（1）第一类：

表示方法：S1(n,m)或$\begin{bmatrix} n \\ m \end{bmatrix}$

组合意义：指n个点组成m个圆排列的方案数。

递推求法：S1(n,m)=S1(n-1,m-1)+(n-1)*S1(n-1,m)

快速求法：

$\prod_{i=0}^{n-1}(x+i)$

的第k次项系数就是S1(n,k)，所以可用分治fft做到n'

（2）第二类：

表示方法:S2(n,m)或$\begin{Bmatrix} n \\ m \end{Bmatrix}$

组合意义：指n个点划分成m个非空集合的方案数。

递推求法：S2(n,m)=S2(n-1,m-1)+m*S2(n-1,m)

快速求法：考虑枚举至少i个集合是空的，容斥所求的0个集合非空得:

$$S2(n,m)=\frac{1}{m!}\sum_{i=0}^{m}(-1)^i*\binom{m}{i}*(m-i)^n$$

其中之前乘的阶乘分之一是因为后面式子求出的是在盒子有标号 (有序) 情况下，这样才能变显然搞一搞就是卷积形式可以直接fft来n*log求

（4）常用斯特林数公式：

$$x^k=\sum_{i=0}^{k}\binom{x}{i}*\begin{Bmatrix} k \\ i \end{Bmatrix}*i!=\sum_{i=0}^{k}x^{\underline{i}}*\begin{Bmatrix} k \\ i \end{Bmatrix}$$

理解：x种颜色给k个点染色，直接考虑是等式左边，复杂地考虑用上k种颜色的方案数求和就是等式右边了。会多于点数，而且推式子题目k也一般比x范围小。

$$x^{\overline{k}}=\sum_{i=0}^{k}\begin{bmatrix} k \\ i \end{bmatrix}*x^i$$

理解：k个点组成圆排列，在为每个圆排列里的所有点染一种颜色。等式右边意义为暴枚组成了几个圆排列个加入点，对于当前点可以染x种颜色任意一种或者放到某个点左边并和这个点颜色相同，那么显然就是$x^{\underline{k}}$

而下降幂与上升幂有如下转化：

$$x^{\underline{n}}=(-1)^n(-x)^{\overline{n}}$$
$$x^{\overline{n}}=(-1)^n(-x)^{\underline{n}}$$

所以之前两个东西可以互相带来带去得出一堆等式...(推详见参考的大佬博客)

斯特林反演:

$$f(n)=\sum_{k=0}^{n}\begin{Bmatrix} n \\ k \end{Bmatrix}g(k)\iff g(n)=\sum_{k=0}^{n}(-1)^{n-k}\begin{bmatrix} n \\ k \end{bmatrix}f(k)$$

3.容斥:

(1).最经典的排斥包含原理：

$$\sum|Ai|-\sum|Ai\cap Aj|+\sum|Ai\cap Aj\cap Ak|-\ldots+(-1)^m|A1\cap A2\cap\ldots\cap Am$$

(2).二项式反演：

$$f(n)=\sum_{i=0}^{n}g(i)C_n^i$$

可以推出：

$$g(n)=\sum_{i=0}^{n}(-1)^{n-i}C_n^if(i)$$

(3).莫比乌斯反演：

$$F(n)=\sum_{d|n}f(d)$$

可以推出：

$$f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$$

(4).斯特林反演：

见之前斯特林数。

(5).min-max 容斥：

基本就这俩式子：

max(S)=$\sum_{T\subseteq S}(-1)^{|T|-1}*min(T)$

min的话上面min和max反一反...

还有一个推广的式子

kth-max(S)=$\sum_{T\subseteq S}(-1)^{|T|-k}*\binom{|T|-1}{k-1}*min(T)$

看上去很没有用，但其实它在算期望（或者lcm）的时

lcm:

将每个数a_i分解为$\prod_{p_j=prime[i][]}p_j^{k_j}$。则LCM就是求每个质因数中指数的最大值之积，而C之积。因此由最值反演可得

$$\text{lcm}\{S\}=\prod_{T\subseteq S}(\gcd\{T\})^{(-1)^{|T|+1}}$$

期望:

对原式套一个期望，就有该式子的期望形式：

$$\mathbb{E}[\max\{S\}]=\sum_{T\subseteq S}(-1)^{|T|+1}\mathbb{E}[\min\{T\}]$$

min-max 容斥经典问题，每次（随机）覆盖一个点，问某个集合内元素都被完全覆盖的期望时间（即集合内所有点覆盖时间最大值期望）《=》枚举集合中第一个点被覆盖的时间（最小值期望），套用 min-max 容斥。

比如树上一些点集被完全覆盖，之后还可以转化成（1-这些点划分连通块后只在连通块内取的概率）分之一，上树形 dp（可能需要包括容斥系数-1 的幂次）。

5.组合数前缀和

（1）第一种形式：把 C(n,n)看成 C(n + 1, n + 1),由组合数递推公式从左到右依次合并

C(n, n) + C(n + 1, n) + C(n + 2, n) + ... + C(m, n) = C(m + 1, n + 1)，（m >= n）

（2）第二种形式：

C(n, 0) + C(n, 1) + C(n, 2) + ... + C(n, m) 其中 m <= n

记所求 = S(n, m)

S(n, m + 1) = S(n, m) + C(n, m + 1)

S(n + 1, m) = 2 * S(n, m) - C(n, m)

一些题目里 m 是恒定值，然后需要代入不同的 n 求 S(n, m)，这时候就需要第二个递推式了

如果询问 n, m 都是变量，可以暴力预处理 f(x, y)其中 x = k * sqrt(n), y = k * sqrt(m)，然后询问的时候找最接近的 f(x, y)暴力走，复杂度是 n * sqrt(n)

6.组合数恒等式

(1) $\sum_{k=0}^{n} C(n,k) * C(S,n-k) = C(n+S,n)$ //左式枚举前 n 个被选中几个

(2) $\sum_{x} C(k-x,i) * C(x,j) = C(k+1,i+j+1)$ //左式枚举第 i+1 个球所在位置

7.同构问题，burnside 引理

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|$$

给定一置换集，定义两个方案相同为两方案可经过置换集中某一置换得到（例如环上一些点染色问题，存在 n 个置换，就是逆时针旋转 $1,2,3\cdots n$ 个）。本质不同的方案数 = 每个置换下不同点数 / 不同置换个数（即置换不动点均值）。（环上一些点问题，对于逆时针转 x 个置换，不动点就是逆时针转 x 个仍然和初始状态完全相同的染色方案）。

九、数论

1.质因数分解：

值域不大时，可以直接线性筛出每个数最小质因子（记录线性筛的时候每个数被谁筛的），然后每个值往下 dfs 即可分解。值域大时，暴力枚举素因子分解，dfs 枚举因数：

2.扩展欧几里得

```
1.  int ex_gcd(int a,int b,int &x,int &y)
2.  {  int d=a;
3.     if(b)
4.     {
5.      d=ex_gcd(b,a%b,x,y);
6.      x-=(a/b)*y;swap(x,y);
7.     }
8.     else
9.     x=1,y=0;
10.    return d;
11.  }
```

3 类欧几里得算法

用于求：（要保证系数非负数，如果有负数，处理的时候要注意对负数的下取整，可能需要调整成对正数的上取整）

$$\sum_{i=0}^{n}\lfloor\frac{a*i+b}{c}\rfloor$$

分类讨论计算。

当 $a=0$ 时:

$$f(a,b,c,n) = (n+1)\lfloor\frac{b}{c}\rfloor$$

当 $a\geq c$ 或 $b\geq c$ 时:

$$f(a,b,c,n) = \sum_{i=0}^{n}\lfloor\frac{i(a \bmod c)+(b \bmod c)}{c}\rfloor + i\lfloor\frac{a}{c}\rfloor + \lfloor\frac{b}{c}\rfloor$$

$$= f(a \bmod c, b \bmod c, c, n) + \frac{n(n+1)}{2}\lfloor\frac{a}{c}\rfloor + (n+1)\lfloor\frac{b}{c}\rfloor$$

当 $a<c$ 且 $b<c$ 时:

为方便，设 $M = \lfloor\frac{an+b}{c}\rfloor$。

$$f(a,b,c,n) = \sum_{i=0}^{n}\sum_{j=1}^{M}[j\leq\lfloor\frac{ai+b}{c}\rfloor] = \sum_{i=0}^{n}\sum_{j=0}^{M-1}[jc+c<ai+b+$$

$$= \sum_{j=0}^{M-1}\sum_{i=0}^{n}[jc+c-b-1<ai]$$

$$= \sum_{j=0}^{M-1}\sum_{i=0}^{n}[i>\lfloor\frac{jc+c-b-1}{a}\rfloor]$$

$$= \sum_{j=0}^{M-1} n - \lfloor\frac{jc+c-b-1}{a}\rfloor$$

$$= nM - f(c,c-b-1,a,M-1)$$

```
1.  ll floor_sum(ll a, ll b, ll c, ll n) { // sum i=0...n, floor((a*i + b)
    /c)
2.      if (a == 0) {
3.          return (n + 1) * (b / c) % mod;
4.      }
5.      ll mx = (a * n + b) / c; // 0, mx - 1
6.      if (mx == 0) {
7.          return 0;
8.      }
9.      if (a >= c || b >= c) {
10.         return (n * (n + 1) / 2 % mod * (a / c) % mod + (n + 1)
    * (b / c) % mod + floor_sum(a % c, b % c, c, n)) % mod;
11.     }
12.     return ((mx * (n + 1) - floor_sum(c, a + c - b - 1, a, mx -
    1)) % mod + mod) % mod;
13. }
```

4.卢卡斯定理

Lucas 定理内容如下：对于质数 p，有

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p\rfloor}{\lfloor m/p\rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

观察上述表达式，可知 $n \bmod p$ 和 $m \bmod p$ 一定是小于 p 的数，可以直接求解，$\binom{\lfloor n/p\rfloor}{\lfloor m/p\rfloor}$ 可以继续用 Lucas 定理求解。这也就要求 p 的范围不能够太大，一般在 10^5 左右。边界条件：当 $m=0$ 的时候，返回 1。

时间复杂度为 $O(f(p)+g(n)\log n)$，其中 $f(n)$ 为预处理组合数的复杂度，$g(n)$ 为单次求组合数的复杂度。

5.皮克定理：

点阵中多边形（顶点在格点）面积 S=a+b/2−1，其中 a 是严格在图形内部格点数，b 是图形边界上格点数。应用，对面积和内部格点的要求可以转化为对边界格点要求，然后运用同余知识。

6.欧拉定理&扩展欧拉定理

欧拉定理：$\gcd(a,m)=1 \rightarrow a^{\Phi(m)}=1(\bmod m)$

扩展欧拉定理：

综上，$a^b \equiv a^{b \bmod \varphi(m)+\varphi(m)} \pmod m \quad (b\geq\varphi(m))$

7.扩展中国剩余定理 excrt

构造答案式

构造答案式的过程如下：

对于 $\begin{cases} x_1 \equiv a_1 \pmod{m_1} \\ x_2 \equiv a_2 \pmod{m_2} \end{cases}$

1. 解方程 $m_1 \cdot x + m_2 \cdot y = \gcd(m_1, m_2)$
2. 若 $a_1 \neq a_2 \pmod{\gcd(m_1,m_2)}$ 则无解，否则有解
3. 两个同余方程合并为 $x \equiv \frac{a_1 m_2 y + a_2 m_1 x}{gcd(m_1,m_2)} \pmod{lcm(m_1,m_2)}$

十、数据结构

0.线段树

```
1.  template<class Info, class Lazy>
2.  struct SegTree {
3.      #define ls k << 1
4.      #define rs k << 1 | 1
5.      #define mid ((l + r) >> 1)
6.      int n;
7.      vector<Info> seg;
8.      vector<bool> tag;
9.      vector<Lazy> laz;
10.
11.     void push_up(int k) {
12.         seg[k] = seg[ls] + seg[rs];
13.     }
14.     void change(int k, Lazy val) {
15.         seg[k] = seg[k] + val;
16.         if (tag[k]) {
17.             laz[k] = laz[k] + val;
18.         } else {
19.             laz[k] = val;
20.         }
21.         tag[k] = 1;
22.     }
23.     void push_down(int k) {
24.         if (tag[k]) {
25.             change(ls, laz[k]);
26.             change(rs, laz[k]);
27.             tag[k] = 0;
28.             laz[k] = {};
29.         }
30.     }
31.     void init(int _n) {
32.         n = _n;
33.         seg.clear(), seg.resize(n * 4 + 100);
34.         tag.clear(), tag.resize(n * 4 + 100);
35.         laz.clear(), laz.resize(n * 4 + 100);
36.         return;
```

```
    }
    void init(vector<Info> bg) {
        init(bg.size() - 1);
        function<void(int, int, int)> build = [&](int l, int r, int k
    ) {
            if (l == r) return (void)(seg[k] = bg[l]);
            build(l, mid, ls), build(mid + 1, r, rs);
            push_up(k);
        };
        build(1, n, 1);
    }
    Info ask(int L, int R, int l, int r, int k) {
        if (l > R || r < L) return Info();
        if (L <= l && r <= R) {
            return seg[k];
        }
        push_down(k);
        return ask(L, R, l, mid, ls) + ask(L, R, mid + 1, r, rs);
    }
    Info ask(int L, int R) {
        return ask(L, R, 1, n, 1);
    }
    void upd(int to, Info val, int l, int r, int k) {
        if (l == r) return (void)(seg[k] = val);
        push_down(k);
        if (to <= mid) upd(to, val, l, mid, ls);
        else upd(to, val, mid + 1, r, rs);
        push_up(k);
    }
    void upd(int to, Info val) {
        upd(to, val, 1, n, 1);
    }
    void add(int L, int R, Lazy val, int l, int r, int k) {
        if (l > R || r < L) return;
        if (L <= l && r <= R) {
            change(k, val);
            return;
        }
        push_down(k);
        add(L, R, val, l, mid, ls);
        add(L, R, val, mid + 1, r, rs);
        push_up(k);
    }
    void add(int L, int R, Lazy val) {
        add(L, R, val, 1, n, 1);
    }
    int rnk; // 下面两个查询，如果查区间第 k 大时用，下面两个询问均返回位置
    template<class F>
    int findFirst(int L, int R, int l, int r, int k, F pred) {
        if (l > R || r < L || !pred(seg[k])) {
            if (L <= l && r <= R) rnk -= seg[k].cnt;
            return -1;
        }
        if (l == r) {
            return l;
        }
        push_down(k);
        int res = findFirst(L, R, l, mid, ls, pred);
        if (res == -1) {
            res = findFirst(L, R, mid + 1, r, rs, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred, int need) {
        rnk = need;
        return findFirst(l, r, 1, n, 1, pred);
    }
    template<class F>
    int findLast(int L, int R, int l, int r, int k, F pred) {
        if (l > R || r < L || !pred(seg[k])) {
            return -1;
        }
        if (l == r) {
            return l;
        }
        push_down(k);
        int res = findLast(L, R, mid + 1, r, rs, pred);
        if (res == -1) {
            res = findLast(L, R, l, mid, ls, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(l, r, 1, n, 1, pred);
    }
    #undef ls
    #undef rs
    #undef mid
};
/*
(1) implement struct Info (2) implement struct Lazy (3) implement Info operator +(Info a, Info b) (4) implement Info operator +(Info a, Lazy b) (5) implement Lazy operator +(Lazy a, Lazy b)
.init(vector bg/size_bg) should give vector or size
vector first element should start from 1 (index 0 not use)
*/
```

1.无旋 treap

```
mt19937 rng(time(0));
struct Node {
    int fa;
    int ch[2], w, rnd, laz, sz;
    bool rev;
} node[N];

#define ls node[nw].ch[0]
#define rs node[nw].ch[1]

int tot;
void push_up(int nw)
{
    node[nw].sz = node[ls].sz + node[rs].sz + 1;
    if(ls)node[ls].fa = nw;
    if(rs)node[rs].fa = nw;
}
int new_node(int w)
{
    int nw = ++tot;
    node[nw].rnd = rng();
    node[nw].laz = node[nw].rev = 0;
    ls = rs = 0;
    node[nw].sz = 1;
    node[nw].fa = 0;
    node[nw].w = w;
    return nw;
}
void R(int nw)
{
    if (!nw) {
        return;
    }
    swap(ls, rs);
    node[nw].rev ^= 1;
}
void push_down(int nw)
{
    if (node[nw].rev) {
```

```
        R(ls), R(rs);
        node[nw].rev = 0;
    }
}
int mg(int x, int y)
{
    if (!x || !y) {
        return x + y;
    }
    push_down(x), push_down(y);
    if (node[x].rnd <= node[y].rnd) {
        node[x].ch[1] = mg(node[x].ch[1], y);
        push_up(x);
        return x;
    } else {
        node[y].ch[0] = mg(x, node[y].ch[0]);
        push_up(y);
        return y;
    }
}
void split1(int nw, int k, int &x, int &y)
{
    if (!nw) {
        x = y = 0;
    } else {
        push_down(nw);
        node[nw].fa = 0;
        if (node[ls].sz >= k) {
            y = nw, split1(ls, k, x, ls);
        } else {
            x = nw, split1(rs, k - node[ls].sz - 1, rs, y);
        }
        push_up(nw);
    }
}
void split2(int nw, int k, int &x, int &y)
{
    if (!nw) {
        x = y = 0;
    } else {
        push_down(nw);
        node[nw].fa = 0;
        if (node[nw].w <= k) {
            x = nw, split2(rs, k, rs, y);
        } else {
            y = nw, split2(ls, k, x, ls);
        }
        push_up(nw);
    }
}
void ins(int &nw, int val)
{
    static int x, y;
    split2(nw, val, x, y);
    nw = mg(mg(x, new_node(val)), y);
    return;
}
vector<int> get_all(int nw)
{
    if (nw == 0) {
        return vector<int> {};
    }
    push_down(nw);
    vector<int> l, r;
    l = get_all(ls);
    r = get_all(rs);
    l.pb(node[nw].w);
    for (auto v : r) {
        l.pb(v);
    }
    return l;
}

int find_rt(int nw)
{
    while(node[nw].fa) {
        nw = node[nw].fa;
    }
    return nw;
}

bool is_rs(int nw)
{
    if (!nw || nw == node[node[nw].fa].ch[0]) {
        return 0;
    }
    return 1;
}

int get_rnk(int nw)
{
    int res = node[ls].sz + 1;
    while(node[nw].fa) {
        if (is_rs(nw)) {
            nw = node[nw].fa;
            res += node[ls].sz + 1;
        } else {
            nw = node[nw].fa;
        }
    }
    return res;
}
```

2.可并堆（随机堆）

```
int merge(int x,int y)
{
    if(!x||!y)return x+y;
    if(v[x]<v[y])swap(x,y);
    rand()&1?ls[x]=merge(ls[x],y):rs[x]=merge(rs[x],y);
    return x;
}
```

3.笛卡尔树:

类似 treap，建立一个有两个关键字的二叉树，满足以第一维关键字看是二叉搜索树，即左<父<右，以第二维看是堆，即孩子大于父亲，构建方法：按第一维关键字从小到大插入，通过一个保存二维关键字的单调栈维护正在构建的树的当前最右链，插入一个点时一直 pop 到单调栈中第一个能做当前点父亲的点，然后将之前最后 pop 出的当成当前点左孩子，将当前点记为栈顶的右孩子，加入单调栈中即可，复杂度 O（n）

```
void get_tree()
{
    int s=0;
    for(int i=1;i<=n;i++)
    {
        while(s&&num[i]<=num[st[s]])
            l[i]=st[s--];//pop 单调栈，同时把最后 pop 记为左孩子
        if(st[s]) r[st[s]]=i;//当前点记为栈顶右儿子
        st[++s]=i;
    }
}
```

7.李超线段树

李超树是维护线段的线段树，线段的定义是一些 y=kx + b 的直线限制其横坐标取值在一个区间，李超树可以实现 nlog^2 地维护某个横坐标 x 处 y 最高的线段 y 值（如果每个线段的横坐标定义域都是整个定义域，则复杂度为 nlogn）。

```
struct Line {
    ll k, b;
    ll operator () (ll x)
```

```
    {
        return k * x + b;
    }
    Line (){};
    Line (ll k, ll b):k(k), b(b){};
};

const int m = 2e5 + 100;
struct lc_tree {
    #define mid ((l + r) >> 1)
    #define ls (k << 1)
    #define rs (k << 1 | 1)
    Line seg[m + m + 100];
    bool hav[m + m + 100];
    Vl bin;

    void ins(Line x, int L, int R, int l = 1, int r = m, int k = 1)
    {
        if(l > R || r < L)
            return;
        if(L <= l && r <= R)
        {
            if(!hav[k])
                return (void)(hav[k] = 1, seg[k] = x, bin.pb(k));
            if(seg[k](mid) > x(mid))
                swap(seg[k], x);
            if(x(l) < seg[k](l))
                ins(x, L, R, l, mid, ls);
            if(x(r) < seg[k](r))
                ins(x, L, R, mid + 1, r, rs);
            return;
        }
        ins(x, L, R, l, mid, ls);
        ins(x, L, R, mid + 1, rs);
        return;
    }
    ll ask(ll x, int l = 1, int r = m, int k = 1)
    {
        if(!hav[k])
            return 1e18;
        if(l == r)
            return seg[k](x);
        if(x <= mid)
            return min(seg[k](x), ask(x, l, mid, ls));
        else
            return min(seg[k](x), ask(x, mid + 1, r, rs));
    }
    void init()
    {
        memset(hav, 0, sizeof hav);
        bin.clear();
    }
    void clear()
    {
        trav(v, bin)
            hav[v] = 0;
        bin.clear();
    }
}lc;
```

8.吉司机线段树

做什么区间取 min/max,赋值，然后还有询问的，具体就是维护最值、最值个数、次最值（次大次小值）

```
struct gg{
    ll fi, sc;
    int num, l, r;// first mx  second mx  number of first mx
    ll sum = 0;
}seg[N << 2];

int n, a[N];

void push_up(int k)
{
    int ls = k << 1;
    int rs = k << 1 | 1;
    seg[k].num = 0;
    seg[k].sum = seg[ls].sum + seg[rs].sum;
    seg[k].fi = max(seg[ls].fi, seg[rs].fi);
    seg[k].sc = max(seg[ls].sc, seg[rs].sc);
    if(seg[ls].fi != seg[k].fi)seg[k].sc = max(seg[k].sc, seg[ls].fi
);
    if(seg[rs].fi != seg[k].fi)seg[k].sc = max(seg[k].sc, seg[rs].f
i);
    if(seg[k].fi == seg[ls].fi)seg[k].num += seg[ls].num;
    if(seg[k].fi == seg[rs].fi)seg[k].num += seg[rs].num;
}

void build(int l = 1, int r = n, int k = 1)
{
    seg[k].l = l;
    seg[k].r = r;
    if(l == r)
    {
        seg[k].fi = a[l];
        seg[k].sc = -1;
        seg[k].num = 1;
        seg[k].sum = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, k << 1);
    build(mid + 1, r, k << 1 | 1);
    push_up(k);
}

void push_down(int k)
{
    if(seg[k << 1].fi > seg[k].fi)
    {
        seg[k << 1].sum += 1LL * (seg[k].fi - seg[k << 1].fi) *
seg[k << 1].num;
        seg[k << 1].fi = seg[k].fi;
    }
    if(seg[k << 1 | 1].fi > seg[k].fi)
    {
        seg[k << 1 | 1].sum += 1LL * (seg[k].fi - seg[k << 1 |
1].fi) * seg[k << 1 | 1].num;
        seg[k << 1 | 1].fi = seg[k].fi;
    }
}

void upd(int L, int R, int val, int k = 1)
{
    if(seg[k].fi <= val)return;
    if(seg[k].fi > val && seg[k].sc <= val)
    {
        seg[k].sum += 1LL * (val - seg[k].fi) * seg[k].num;
        seg[k].fi = val;
        return;
    }
    if(seg[k].l < seg[k].r)push_down(k);
    int mid = (seg[k].l + seg[k].r) >> 1;
    if(L <= mid)upd(L, R, val, k << 1);
    if(R > mid)upd(L, R, val, k << 1 | 1);
    push_up(k);
}

ll ask_mx(int L, int R, int k = 1)
{
    if(seg[k].l > R || seg[k].r < L)return -1;
    if(L <= seg[k].l && seg[k].r <= R)return seg[k].fi;
```

```
    if(seg[k].l != seg[k].r)push_down(k);
    return max(ask_mx(L, R, k << 1), ask_mx(L, R, k << 1 | 1)
);
}

ll ask_sum(int L, int R, int k = 1)
{
    if(seg[k].l > R || seg[k].r < L)return 0;
    if(L <= seg[k].l && seg[k].r <= R)return seg[k].sum;
    if(seg[k].l != seg[k].r)push_down(k);
    return ask_sum(L, R, k << 1) + ask_sum(L, R, k << 1 | 1);
}
```

十一、图论问题

spfa 及判环、差分约束系统:

判环：存在负环则会判断是否有点入队大于等于 n 次即可。（n 次后仍然可以松弛）。

差分约束：一堆形如 a-b<=c 的不等式,最后求给定未知数范围，建图跑 spfa。

（1）求取最小值，那么求出最长路，那么将不等式全部化成 xi - xj >= k 的形式，这样建立 j->i 的边，权值为 k 的边。

(2)如果求取的是最大值，那么求取最短路，将不等式全部化成 xi - xj <= k 的形式，这样建立 j->i 的边，权值为 k 的边。

(3)如果要判断差分约束系统是否存在解，一般都是判断环，选择求最短路或者最长路求解都行，只是不等式标准化时候不同，判环地话，用 spfa 即可，n 个点中如果同一个点入队超过 n 次，那么即存在环。建立的图可能不联通，需要加入一个超级源点，比如求最长路时图不联通的话，加入一个点 S，对其他的每个点建立一条权值为 0 的边图就联通了，从 S 点开始进行 spfa 判环。

```
int spfa(int n)
{
    queue<int>q;int id;
    memset(dis,127,sizeof dis);
    q.push(1);dis[1]=0;inq[1]=1;
    while(!q.empty())
    {
        id=q.front();q.pop();inq[id]=0;
        vis[id]++;if(vis[id]>n)return -1;
        for(int i=0;i<mp[id].size();i++)
        {
            if(dis[mp[id][i].first]>dis[id]+mp[id][i].second)
            {
                dis[mp[id][i].first]=dis[id]+mp[id][i].second;
                if(!inq[mp[id][i].first])q.push(mp[id][i].first),inq[m
p[id][i].first]=1;
            }
        }
    }
    return dis[n];
}
```

3.欧拉路:

一张无向连通图是欧拉图，当且仅当所有节点度数是偶数。

一张无向连通图有欧拉路，当且仅当只有两个点（起点终点）度数为奇数。

求欧拉回路：

dfs（x）：

若某条边（x，y）未访问

标记为已访问，dfs（y），把 y 入栈。

倒序输出为答案

竞赛图:

性质

- **性质一：竞赛图缩点后呈链状**

注意，这里的链状并不是直接呈一条链，而是类似于 n 个点链状偏序集，

任意两个强连通分量之间从拓扑序小的连向拓扑序大的

- **性质二：竞赛图每个强连通分量存在哈密顿回路**
- **性质三：竞赛图存在一条哈密顿路径**

- 性质四（性质二拓展）：对于竞赛图的一个大小为 $n(n\geq 3)$ 的强连通分量，其一定存在大小 $\forall l\in[3,n]$ 的简单环

首先这个命题等价于弱化结论：对任意一个强连通竞赛图成立即可。

任然考虑对强连通竞赛图点数归纳，在环点后的链上容易找到。

- 性质五（兰道定理）：存在竞赛图满足每个点出度序列为 s 当且仅当：
- 将 s 从小至大排序，$\forall k\in[1,n],\sum_{i}^{k}s_i\geq\binom{k}{2}$ 且 $k=n$ 时等号成立。
- 性质六（竞赛图强连通分量判定定理）：将所有点按照入度 d_i 从小到大排序为 p_i，则有如下性质成立：该竞赛图上所有极大的 SCC 在 p 中分为若干个互不相交的区间，且 i 为这些区间的右端点当且仅当：$\sum_{j\leq i}d_{p_j}=\binom{i}{2}$.

前者根据性质一显然成立，下面考虑证明后者。

首先，这个条件等价于不存在 $j>i$ 使得 j 能连到 $\leq i$ 的一个点；这等价于 i 这个划分点不能划分在某个 SCC 中间，与假设等价。

4.tarjan 算法总结

（1）无向图 tarjan：

1.求 low（x）：首先将 low（x）初始化为 dfn（x），对于连着 x 的边（x，y），若 x 为 y 父亲，则先 dfs（y），low（x）=min（low（x），low（y））,不然说明为非树边，low（x）=min（low（x），dfn（y））

2.割点和桥

定义桥为删除掉图会分裂成两个及以上子图的边，割点为删除掉图会分裂为两个以上子图的点。若一条 x 到子节点 y 的边是桥，则要满足 dfn（x）<low（y），所以 dfs 判一下即可，要注意与平时 dfs（x）传入 fa（x）不同这里要传入 fa(x)到 x 的边标号以防重边情况。

割点情况类似，若 x 是割点，则要存在一个 y，使得 dfn（x）<=low（y），这里可以不用判断重边情况（最好还是判反正没多大区别），需要注意根是割点的条件较特殊是有两个及以上子节点。

3.点双边双：点双图是一个无向图，其中不存在割点。点双连通分量指原图一个极大点双连通子图（即不存在包含它更大点双子图）。

边双图是一个无向图，其中不存在桥。边双连通分量指原图一个极大边双连通子图（即不存在包含它更大边双子图）。

1.点双图至少满足以下两条件之一：顶点数不超过 2；任意两点至少包含在一个简单环（不自交环）中。

2.一张图是边双图当且仅当任意边都包含在至少一个简单环中。

点双连通分量 BCC code:

```
struct BCC {
    int n;
    vector<vector<int>> adj;
    vector<int> stk;
    vector<int> dfn, low;
    vector<vector<int>> bccs;
    int tim, cnt;

    BCC() {}
    BCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n + 5, {});
        dfn.assign(n + 5, 0);
        low.resize(n + 5);
        bccs.clear();
        stk.clear();
        tim = cnt = 0;
    }

    void add_edge(int u, int v) {
```

```
        adj[u].pb(v);
        adj[v].pb(u);
    }

    void dfs(int x) {
        dfn[x] = low[x] = ++tim;
        stk.pb(x);
        for (auto y : adj[x]) {
            if (!dfn[y]) {
                dfs(y);
                low[x] = min(low[x], low[y]);
                if (low[y] >= dfn[x]) {
                    vector<int> bcc;
                    int del;
                    do {
                        del = stk.back();
                        stk.pop_back();
                        bcc.pb(del);
                    } while (del != y);
                    bcc.pb(x);
                    bccs.pb(bcc);
                }
            } else {
                low[x] = min(low[x], dfn[y]);
            }
        }
    }

    vector<vector<int>> work() {
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                dfs(i);
                if (tim == dfn[i]) { // alone
                    bccs.pb({i});
                }
            }
        }
        return bccs;
    }
};
```

边双连通分量 EBCC code:

```
struct EBCC {
    int n;
    vector<vector<int>> adj;
    vector<int> stk;
    vector<int> dfn, low, bel;
    int tim, cnt;

    EBCC() {}
    EBCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n + 5, {});
        dfn.assign(n + 5, 0);
        low.resize(n + 5);
        bel.assign(n + 5, 0);
        stk.clear();
        tim = cnt = 0;
    }

    void add_edge(int u, int v) {
        adj[u].pb(v);
        adj[v].pb(u);
    }

    void dfs(int x, int fa) {
        dfn[x] = low[x] = ++tim;
        stk.pb(x);
        bool skip = 0;
        for (auto y : adj[x]) {
            if (y == fa && !skip) {
                skip = 1;
                continue;
            }
            if (!dfn[y]) {
                dfs(y, x);
                low[x] = min(low[x], low[y]);
            } else {
                low[x] = min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            ++cnt;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
        }
    }

    vector<int> work() {
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                dfs(i, 0);
            }
        }
        return bel;
    }
    // bel range = [1, cnt]
};
```

（2）有向图 tarjan

一张有向图，若其中任意两个点 x，y 都既存在 x 到 y 路径也存在 y 到 x 路径，则其为强连通图，类比点双边双分量可以定义强连通分量。求强连通分量：维护一个栈记录当前点祖先点集合，然后计算 low（x）时与求割点桥差不多，就是一个点要判断是当前点祖先要看它是否在栈中。最后在 x 点回溯前判断是否有 low（x）==dfn（x），如果有则将栈弹到 x 出栈为止，所有出栈的构成一个强连通分量，缩点则与边双类似。

```
struct SCC {
    int n;
    vector<vector<int>> adj;
    vector<int> stk;
    vector<int> dfn, low, bel;
    int tim, cnt;

    SCC() {}
    SCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n + 5, {});
        dfn.assign(n + 5, 0);
        low.resize(n + 5);
        bel.assign(n + 5, 0);
        stk.clear();
        tim = cnt = 0;
    }

    void add_edge(int u, int v) {
        adj[u].pb(v);
```

```
    }

    void dfs(int x) {
        dfn[x] = low[x] = ++tim;
        stk.pb(x);
        for (auto y : adj[x]) {
            if (!dfn[y]) {
                dfs(y);
                low[x] = min(low[x], low[y]);
            } else if (!bel[y]) {
                low[x] = min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            ++cnt;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
        }
    }

    vector<int> work() {
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                dfs(i);
            }
        }
        return bel;
    }
};
```

(3) 2-SAT 问题:

假设有 $a1, a2$ 和 $b1, b2$ 两对，已知 $a1$ 和 $b2$ 间有矛盾，于是为了方案自洽，由于两者中必须选一个，所以我们就要拉两条有向边 $(a1, b1)$ 和 $(b2, a2)$ 表示选了 $a1$ 则必须选 $b1$，选了 $b2$ 则必须选 $a2$ 才能够自洽。

然后通过这样子建边我们跑一遍 Tarjan SCC 判断是否有一个集合中的两个元素在同一个 SCC 中，若有则输出不可能，否则输出方案。构造方案只需要把几个不矛盾的 SCC 拼起来就好了。

输出方案时可以通过变量在图中的拓扑序确定该变量的取值。如果变量 $\neg x$ 的拓扑序在 x 之后，那么取 x 值为真。应用到 Tarjan 算法的缩点，即 x 所在 SCC 编号在 $\neg x$ 之前时，取 x 为真。因为 Tarjan 算法求强连通分量时使用了栈，所以 Tarjan 求得的 SCC 编号相当于反拓扑序。

显然地，时间复杂度为 $O(n+m)$。

暴搜

就是沿着图上一条路径，如果一个点被选择了，那么这条路径以后的点都将被选择，那么，出现不可行的情况就是，存在一个集合中两者都被选择了。

那么，我们只需要枚举一下就可以了，数据不大，答案总是可以出来的。

如果要输出 2-SAT 问题的一个可行解，只需要在 tarjan 缩点后所得的 DAG 上自底向上地进行选择和删除。

具体实现的时候，可以通过构造 DAG 的反图后在反图上进行拓扑排序实现；也可以根据 tarjan 缩点后，所属连通块编号越小，节点越靠近叶子节点这一性质，优先对所属连通块编号小的节点进行选择。（从小到大，之前没被分配就随便分配一个，依次往后）。

无自己到自己的 2-sat 可用并查集确定可行性。

tarjan 法代码

```
struct SAT {
    int n, tim, num;
    bool valid;
    vector<vector<int>> adj;
    vector<int> dfn, low, stk, bel;
    vector<bool> ins, hav;
    SAT() {}
    SAT(int n) { // i = sel, i + n = non-sel
        this -> n = n;
        tim = num = 0;
        valid = 1;
        stk.clear();
        adj.assign(2 * n + 5, {});
        dfn.assign(2 * n + 5, 0);
        low.assign(2 * n + 5, 0);
        bel.assign(2 * n + 5, 0);
        ins.assign(2 * n + 5, 0);
        hav.assign(2 * n + 5, 0);
    }
    void dfs(int x) {
        dfn[x] = low[x] = ++tim;
        ins[x] = 1;
        stk.pb(x);
        for (auto v : adj[x]) {
            if (!dfn[v]) {
                dfs(v);
                if (!valid) return;
                low[x] = min(low[x], low[v]);
            } else if (ins[v]) {
                low[x] = min(low[x], dfn[v]);
            }
        }
        if (low[x] == dfn[x]) {
            ++num;
            vector<int> bin;
            while (1) {
                int nw = stk.back();
                stk.pop_back();
                if (nw > n && hav[nw - n] || nw <= n && hav[nw
    + n]) {
                    valid = false;
                    return;
                }
                hav[nw] = 1, ins[nw] = 0, bel[nw] = num;
                bin.pb(nw);
                if (nw == x) break;
            }
            for (auto v : bin) hav[v] = 0;
            bin.clear();
        }
    }
    void add_edge(int x, int y) {
        adj[x].pb(y);
    }
    pair<bool, vector<int>> sol() {
        valid = 1;
        for (int i = 1; i <= n * 2; i++) {
            if (!dfn[i]) dfs(i);
        }
        vector<int> sel(n + 1, 0);
        for (int i = 1; i <= n; i++) {
            sel[i] = (bel[i] < bel[i + n]);
        }
        return make_pair(valid, sel);
    }
};
```

暴搜:

```
struct Twosat {
  int n;
  vector<int> g[maxn * 2];
  bool mark[maxn * 2];
  int s[maxn * 2], c;

  bool dfs(int x) {
    if (mark[x ^ 1]) return false;
    if (mark[x]) return true;
    mark[x] = true;
    s[c++] = x;
    for (int i = 0; i < (int)g[x].size(); i++)
      if (!dfs(g[x][i])) return false;
    return true;
  }

```

```
17.     void init(int n) {
18.       this->n = n;
19.       for (int i = 0; i < n * 2; i++) g[i].clear();
20.       memset(mark, 0, sizeof(mark));
21.     }
22.
23.     void add_clause(int x, int y) {  // 这个函数随题意变化
24.       g[x].push_back(y ^ 1);       // 选了 x 就必须选 y^1
25.       g[y].push_back(x ^ 1);
26.     }
27.
28.     bool solve() {
29.       for (int i = 0; i < n * 2; i += 2)
30.         if (!mark[i] && !mark[i + 1]) {
31.           c = 0;
32.           if (!dfs(i)) {
33.             while (c > 0) mark[s[--c]] = false;
34.             if (!dfs(i + 1)) return false;
35.           }
36.         }
37.       return true;
38.     }
39.   };
```

5. 生成树问题：

（1）克鲁斯卡尔重构树

依照克鲁斯卡尔求最小生成树的做法，把边按权值从小到大排序，枚举(x,y,val)加边时，不直接加边，先访问到 x,y 当前并查集内的根 fx,fy，然后新开一个点 np,把 fx 和 fy 都挂到 np 下面当儿子，np 点的权值则是该边边权 val。这样建出来的树两点间 lca 的权值就是原最小生成树两点间路径边权最大值。可以解决将对边的询问转化为对点，如查询最小生成树某两个点上路径边权最大值，就可以用克鲁斯卡尔重构树重构之后，转化为欧拉序 0（1）lca 的问题。

（2）.生成树计数-矩阵树定理

无向图：度数矩阵减去邻接矩阵，之后任意划去一行一列，求行列式的绝对值即为所求生成树个数。

有向图：

定义出度 Laplace 矩阵 L^{out} 为

$$L^{\text{out}}(G) = D^{\text{out}}(G) - A(G).$$

定义入度 Laplace 矩阵 L^{in} 为

$$L^{\text{in}}(G) = D^{\text{in}}(G) - A(G).$$

定理 3（矩阵树定理，有向图根向形式）对于任意的 k，都有

$$t^{\text{root}}(G,k) = \det L^{\text{out}}(G)\begin{pmatrix}1,2,\cdots,k-1,k+1,\cdots,n\\1,2,\cdots,k-1,k+1,\cdots,n\end{pmatrix}$$

因此如果要统计一张图所有的根向树形图，只要枚举所有的根 k 并对 $t^{\text{root}}(G,k)$ 求和即可。

定理 4（矩阵树定理，有向图叶向形式）对于任意的 k，都有

$$t^{\text{leaf}}(G,k) = \det L^{\text{in}}(G)\begin{pmatrix}1,2,\cdots,k-1,k+1,\cdots,n\\1,2,\cdots,k-1,k+1,\cdots,n\end{pmatrix}$$

因此如果要统计一张图所有的叶向树形图，只要枚举所有的根 k 并对 $t^{\text{leaf}}(G,k)$ 求和即可。

定理 5（BEST 定理）设 G 是有向欧拉图，那么 G 的不同欧拉回路总数 ec(G) 是

$$ec(G) = t^{\text{root}}(G,k) \times \prod_{v\in V}(\deg(v)-1)!$$

其中 $t^{\text{root}}(G,k)$ 表示以 root 为根的生成树的数量。

Prufer 序列：

生成方法：把无根树里所有度数为 1 的点定义为叶子，重复执行以下操作直至原树里只剩下两个节点：选取当前标号最小的叶子，把它的父亲接在当前生成的 prufer 序列末尾，并在原树里删除该点。**还原方法：**设一个点集 V={1,2,3,...,n}，然后重复执行以下操作，取出 prufer 序列当前开头元素 x，然后在 V 中从左到右遍历找到第一个没有在当前 prufer 序列中出现的元素 y，并把 x、y 连一条边，然后 prufer 序列删除开头 x，点集 V 删除 y。直到 V 中剩下两个点时，把这两个点连一条边，至此，原树还原完成。**性质 I.** 从上面的生成还原看，prufer 序列内每个点取值都可以在[1,n]内，这样可以证明一个 n 个点的完全图生成的无根树数量为 n^(n-2) **性质 II.** 可以发现每个点在 prufer 序列每出现一次就对应一个与它相连的点的删除，所以 prufer 序列中某点出现次数等于该点在原树上的度数减 1 **性质 III.** 依据上面第二条性质，假设给定树上每个节点度数，可以发现生成树个数就是一个含重复元素的集合的排列数，就是阶乘除以一堆阶乘积的东西

（3）有限制最小生成树：给定边集合黑白两色，要求求出包含恰好 k 条黑色边的最小生成树。用 wqs 二分，因为记 f(k)表示选择 k 条黑边的答案，发现 f(k)是关于 k 的凸函数，故二分-inf 到+inf 的权值，给每条黑边加上这个权值，分别求当前情况最少用多少黑边，最多用多少黑边（第二维颜色关键字两种排序方式），然后找到 k 所在的一段输出即可。

十二、dp 相关

基本思路：用多种方式设 dp 状态，dp 最大最小值、dp 两个变量的和或差值、dp 答案为某一个值的时候最多能满足多少要求……；同时正向 dp 不行，尝试反向（比如构造排列，然后有一些要求，可以尝试 1~n 插入，也可以尝试 n~1 插入）；dp 选择的时候可以用贪心减少转移数。

0.斜率优化

dp(i) = max{a(j) * i + b(j)}

可以直接上李超树，不过线性做法为：

维护凸包，假设j1 < j2，且a(j)

关于 j 单调递增，假如j2更优

$$a(j1)*i+b(j1) < a(j2)*i+b(j2)$$

得到 $i < \frac{b(j2)-b(j1)}{a(j2)-a(j1)}$

右侧为一个二点式确定的斜率，因此把 j 视为一个点$(a(j),b(j))$

然后用栈、队列之类的维护上/下凸包。

wqs 二分优化

一般是让你选一些东西获取最大价值，但是存在一些限制，例如只能选 k 个（选择方案一般不能直接求出贡献，函数较为复杂），然后会发现 f(k)表示选 k 个获得最大价值，这东西满足凸性，即 f（k + 1）- f(k) >= f(k + 2) - f(k + 1)，因此二分权值，给每选一个东西加上这个权值，此时任意选取并记录选择个数，如果选择个数的区间包含 k 即合法了。

决策单调性优化 dp

一般 dp(i)由之前某个 dp(k) 转移过来，每个 dp 选择的最优决策点单调不下降，即用 f(i)表示 dp(i)选择的 k，任取 i < j, f(i) < f(j)。此时有两种情况：

（1）简单情况，最优决策点连续增长

（2）麻烦情况，不满足最优决策点连续增长，即 g(i + 1, k) <= g(i + 1, kk)但是 k 与 kk 之间不一定是单调不降,考虑维护一个存放决策点队列，存决策点位置，以及该位置为决策点管辖的区间（二分更新）。或者**分治优化 dp**：如果要求取 dp(l to r)的 dp 值，而 dp(mid)可以在 O(r-l+1)复杂度内求出，那么得到 dp(mid)的最优决策位置，分治解决 l to mid 和 mid + 1 to r,因为此时左右区间可能决策点位置相加到 O(n)，故复杂度 O(nlogn).注意要保证复杂度与当前处理区间同阶，可能当前区间做完一些信息不能清空，要保留到下一层使用保证复杂度.

3.倍增优化 dp

一般是维护 2^k 能到达的状态，然后查询的时候类似树上倍增的合并。

4.四边形不等式优化 dp

dp[i][j]=min(dp[i][k]+dp[k+1][j]+cost[i][j])

对于一个区间值关系，如果它满足交叉小于包含的话，那么就说它是满足四边形不等式的。如当 i < i' <= j < j'时，对于 cost 的值来说，如果它满足 cost[i][j]+cost[i'][j']<=cost[i][j']+cost[i'][j] 的关系时，那么我们就说它满足四边形不等式。

优化方法：**s[i][j-1]<=s[i][j]<=s[i+1][j]**，其中 s 为最优决策点函数，故 dp 时先枚举区间长度，再枚举起点，此时最优决策点就被限制在一个区间内，可以证明复杂度 O(n^2)

5.动态 dp

Dp 方程满足是矩阵乘积之类的，用线段树之类的维护区间信息即可（树上就是重链剖分或者全局平衡二叉树之类的）

十三、树上问题

1.点分治:

思想就是每次找到当前树的重心结点，然后处理这个重心下挂着各个子树间产生的对答案的贡献（也即所有包含这个重心节点的路径产生的贡献），然后递归处理每棵子树。

```
int n;
int a[N];
vector<int> adj[N];
int sz[N], mx[N], allsz, rt;
bool vis[N];
ll res = 0, pw2[N], ipw2[N];

void get_rt(int x, int fa) {
    sz[x] = 1, mx[x] = 0;
    for (auto u : adj[x]) {
        if (vis[u] || u == fa) {
            continue;
        }
        get_rt(u, x), sz[x] += sz[u];
        mx[x] = max(mx[x], sz[u]);
    }
    mx[x] = max(mx[x], allsz - sz[x]);
    if (mx[x] < mx[rt]) {
        rt = x;
    }
}

int bel[N];
ll sum[N], dep[N];
vector<int> buk;

void dfs(int x, int fa) {
    buk.pb(x);
    for (auto u : adj[x]) {
        if (vis[u] || u == fa) {
            continue;
        }
        bel[u] = bel[x], sum[u] = 0, dep[u] = dep[x] + 1;
        dfs(u, x);
    }
}

void work(int x) {
    vis[x] = 1, buk.clear();
    buk.pb(x), bel[x] = x, sum[x] = 0, dep[x] = 0;
    for (auto u : adj[x]) {
        if (vis[u]) {
            continue;
        }
        bel[u] = u, sum[u] = 0, dep[u] = 1;
        dfs(u, x);
    }
    sort(all(buk), [&](int x, int y) {
        return a[x] < a[y];
    });
    ll ss = 0;
    for (auto nw : buk) {
        int b = bel[nw];
        res = (res + a[nw] * (ss - sum[b] + mod) % mod * ipw2[dep[nw]]) % mod;
        sum[b] = (sum[b] + ipw2[dep[nw]]) % mod;
        ss = (ss + ipw2[dep[nw]]) % mod;
    }
    ss = 0;
    for (auto nw : buk) {
        sum[bel[nw]] = 0;
    }
    reverse(all(buk));
    for (auto nw : buk) {
        int b = bel[nw];
        res = (res - a[nw] * (ss - sum[b] + mod) % mod * ipw2[dep[nw]] % mod + mod) % mod;
        sum[b] = (sum[b] + ipw2[dep[nw]]) % mod;
        ss = (ss + ipw2[dep[nw]]) % mod;
    }
    ll bf = allsz;
    for (auto u : adj[x]) {
        if (vis[u]) {
            continue;
        }
        allsz = (sz[u] < sz[x] ? sz[u] : bf - sz[x]);
        rt = 0, get_rt(u, x);
        work(rt);
    }
}
void sol()
{
    cin >> n;
    for (int i = 1; i < n; i++) {
        int x, y;
        cin >> x >> y;
        adj[x].pb(y);
        adj[y].pb(x);
    }
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    mx[0] = inf, allsz = n;
    get_rt(1, 0);
    work(rt);
    cout << res * pw2[n] % mod << '\n';
}
```

3.树上启发式合并:

树上启发式合并，即每次将轻儿子的信息暴力合并到重儿子的信息上从而得到自己的信息。实现的时候，不需要维护轻儿子的数组，一般我们选择全局只维护一个数组，记录上一个点的信息，然后每次暴力遍历轻儿子的子树将信息合并过来。我们只需要每次先处理了轻儿子，然后清空数组再遍历重儿子，就可以使上一个点的信息成为我们需要的重儿子的信息了。

4.虚树:

把一棵很大的树压缩一波成一棵比较小的树，然后对原树的询问就可以在这棵压缩信息的树上搞，以降低复杂度。（一般就是关键点及它们的 lca）。注意一般都要把根节点先记为关键点。

```
bool cmp(int x,int y)
```

```
2.  {return dfn[x]<dfn[y];}
3.  void build() {
4.    int tt=k,nw,f;
5.    sort(imp+1,imp+k+1,cmp);hd=0;
6.    for(int i=1;i<=k;i++) {
7.      nw=imp[i];
8.      if(!hd){fa[nw]=0,S[++hd]=nw;continue;}
9.      f=lca(nw,S[hd]);
10.     while(dep[S[hd]]>dep[f]) {
11.       if(dep[S[hd-1]]<dep[f])fa[S[hd]]=f; //该点之前点在
    f 上，故父亲选为 f
12.       --hd;
13.     }
14.     if(f!=S[hd]) { //一个新的关键点之间的 lca 加入
15.       imp[++tt]=f;
16.       fa[f]=S[hd];
17.       S[++hd]=f;
18.     }
19.     fa[nw]=f,S[++hd]=nw;
20.   }
21.   k=tt;sort(imp+1,imp+k+1,cmp); //新加了关键点，重新
    sort
22. }
```

十四、分治、分块、莫队

1.整除分块：

```
Floor: x = n / l, r = n / x
Ceil: x = (n + l - 1) / l, r = (x == 1 ? l : (n - 1) / (x - 1))
```

2.莫队：

可以理解成二维平面最小遍历的问题。变种：树上莫队，在括号序上做；回滚莫队，处理一个 block 的所有询问时，都把左端点重新移到块起始做。

```
1.  bool cmp(ques a,ques b)//排序比较函数
2.  {return a.l/block==b.l/block?a.r<b.r:a.l/bloc
    k<b.l/block;}
```

其中 block 取根号 n

带修改莫队：一个查询记录 l,r,tim,tim 表示执行这次询问修改了多少次，此时相当于三个指针移动，排序改为：

```
1.  bool operator < (const query &b) const {
2.      if(bel(l) != bel(b.l)) return l < b.l;
3.      if(bel(r) != bel(b.r)) return r < b.r;
4.      return tim < b.tim;
5.  }
```

其中 bel 就是除以块大小，此时块大小取 (n^(2/3)),复杂度(n^(5/3))次。

3.线段树分治：

对于一类有插入、删除（撤销插入）和整体查询操作的题目，可以考虑按时间分治（也可以叫线段树分治）。对于每一个插入操作处理出它存在的时间，就不用管删除操作了，再将这些插入操作存在区间建立一棵时间线段树，每个节点是一个 vector，然后从线段树 dfs 到叶子经过的点上所有点 vector 的并就是在这个点时会对其产生影响的所有操作了。这类题不会真的要把所有 vector 传到叶子，可能是线性基之类的东西

4.整体二分

处理多个询问的一些问题。以区间 k 小为例，将询问和修改离线下来，然后对所有询问修改，二分值 L、R 以及 Mid 并同时维护哪些修改的值<=Mid,哪些询问答案会在[L,R]内，那么用树状数组跑当前这层的修改（即小于等于 Mid 的在 bit 上位置+1），然后对每个询问查询当前答案是否达到 k，是的话分到[L，Mid]，不然当前 k 减掉 k 然后分到[Mid+1,R]，再把当前这层修改撤销并把修改也判断一下分到左右递归，当到达 L=R 时把这层的询问答案置为 L 即可。

十五、高斯消元

```
1.  void gauss(vector<vector<ll>> &a, int n, int m) // n 个方
    程 m 个变量，可能非满秩
2.  {
3.    int bg = 0;
4.    for(int i = 0; i < m - 1; i++)
5.    {
6.      int ps = bg;
7.      while(ps < n && a[ps][i] == 0)
8.        ++ps;
9.      if(ps >= n)
10.       continue;
11.     if(ps != bg)
12.       for(int j = 0; j < m; j++)
13.         swap(a[ps][j], a[bg][j]);
14.     ll coef = qpow(a[bg][i]);  // coef = 1 / a[bg][i]
15.     for(int j = i; j < m; j++)
16.       a[bg][j] = (a[bg][j] * coef) % mod;
17.     for(int j = 0; j < n; j++)
18.     {
19.       if(j == bg)
20.         continue;
21.       coef = a[j][i];
22.       for(int k = i; k < m; k++)
23.         a[j][k] = (a[j][k] - a[bg][k] * coef % mod + mod) %
    mod;
24.     }
25.     ++bg;
26.   }
27. }
```

十六、二进制技巧

！！！！若 x 为 long long，则使用是要改为 long long 版本，如__builtin_popcountll(x)

__builtin_ffs(x) 返回 x 中最后一个为 1 的位是从后向前的第几位，从 1 开始标号，如 1 返回 1

__builtin_popcount(x)：x 中 1 的个数。

__builtin_ctz(x)：x 末尾 0 的个数。x=0 时结果未定义。

__builtin_clz(x)：x 前导 0 的个数。x=0 时结果未定义。

__builtin_parity(x)：x 中 1 的奇偶性。

Lowbit(x) = x&(-x)

枚举子集的子集，3^n

```
1.  for (int S=1; S<(1<<n); ++S){
2.    for (int S0=S; S0; S0=(S0-1)&S)
3.      //do something.
4.  }
```

bitset<1111> bst

bst.__Find_next(x)，找 x 位置之后（不包括 x）下一个 1 的位置，失败的话返回 bitset 大小

bst.__Find_first()，找第一个 1 的位置

bst.flip() 逐位取反

bst.count() 返回 1 的个数

十七、计算几何常用函数

```
1.  const double eps = 1e-14;
2.  const double pi = acosl(-1);
3.  int sgn(ld x) {
4.    return x < -eps ? -1 : x > eps;
5.  }
6.  int cmp(ld x, ld y) {
7.    return sgn(x - y);
8.  }
```

```
using T = ld;
struct Point { // 点
    T x, y;
    Point(T x = 0, T y = 0) : x(x), y(y) {}
    bool operator<(Point B) const{ // x 第一关键字
        return x == B.x ? y < B.y : x < B.x;
    }
    bool operator==(Point B) const{
        return !sgn(x - B.x) && !sgn(y - B.y);
    }
    bool operator <=(Point B) const {
        return ((*this) < B || (*this) == B);
    }
    Point operator+(Point B) const{
        return Point(x + B.x, y + B.y);
    }
    Point operator-(Point B) const{
        return Point(x - B.x, y - B.y);
    }
    Point operator*(T a) const{ // 标量乘
        return Point(x * a, y * a);
    }
    Point operator/(T a) const{ // 标量除
        return Point(x / a, y / a);
    }
    T operator*(Point B) const{ // 点积
        return x * B.x + y * B.y;
    }
    T operator^(Point B) const{ // 叉积模长
        return x * B.y - y * B.x;
    }
    Point operator-() const{    // 取负,关于原点对称
        return Point(-x, -y);
    }
    ld angle() {    // 反正切,与 x 轴方位角, (-pi, pi]
        return atan2l(this->y, this->x);
    }
    T length2() { // 视为原点到(x,y)向量，模长的平方
        return x * x + y * y;
    }
    ld length() { // 模长
        return sqrtl(length2());
    }
    Point unit() {  // 单位方向向量
        return *this / this -> length();
    }
    Point normal() {  // 单位法向量
        return Point(-y, x) / this -> length();
    }
    Point trunc(ld r) {  // 化为长度为 r 的向量
        ld l = length();
        if (!sgn(l))
            return *this;
        r /= l;
        return Point(x * r, y * r);
    }
    Point conj() { // 共轭向量
        return Point(x, -y);
    }
    Point to_left() { // 绕原点左转 90 度
        return Point(-y, x);
    }
    Point to_right() { // 绕原点右转 90 度
        return Point(y, -x);
    }
    Point rotate(ld rad) {  // 逆时针旋转 rad 弧度
        return Point(x * cos(rad) - y * sin(rad), x * sin(rad) + y * cos(rad));
    }
    friend int relation(Point a, Point b, Point c) { // c 是否在(a,b)的逆时针侧
        return sgn((b - a) ^ (c - a));
    }
    friend ld get_angle(Point a, Point b) { // 向量夹角
        return acosl((a * b) / a.length() / b.length());
    }
    friend T area(Point a, Point b, Point c) { //
        return fabsl((b - a) ^ (c - a)); // (a,b)(a,c)平行四边形面积
    }
    friend T get_dis2(Point a, Point b) { // 两间距离方
        return (a - b).length2();
    }
    friend ld get_dis(Point a, Point b) { // 两点距离
        return sqrtl(get_dis2(a, b));
    }
    friend ld project(Point a, Point b, Point c) {  // 求向量 ac 在向量 ab 上的投影长度
        return ((b - a) * (c - a)) / (b - a).length();
    }
    bool up() const { // 是否在一二象限内,象限的定义均为左闭右开,即第一象限[0, pi/2)
        return y > 0 || (y == 0 && x >= 0);
    }
    friend ostream& operator<<(ostream& os, Point a) {
        return os << "(" << a.x << ',' << a.y << ')';
    }
};

bool argcmp(Point a, Point b) { // 关于原点极角排序,一二三四象限顺序
    if (a.up() != b.up())
        return a.up() > b.up();
    return (a ^ b) == 0 ? a.x < b.x : (a ^ b) > 0;
}
Point rotate(Point a, Point bas, ld theta) { // a 点绕 bas 点逆时针转 theta 弧度
    return (a - bas).rotate(theta) + bas;
}
Point reflect(Point v, Point l) { // 光线 v 照射到平面 l 后反射
    Point res;
    Point E = l / l.length();  // 单位向量
    T d = E * v;
    return (E * 2 * d - v);
}

struct Line { // 直线
    Point p, v; // p 为直线上一点,v 为方向向量
    ld rad; // rad 为方向角 (-pi, pi]
    //亦可表示向量 v 逆时针方向的半平面
    Line(){}
    Line(Point p, Point v):p(p), v(v) {
        rad = atan2l(v.y, v.x);
    }
    Point get_point(ld t) {
        return p + v * t;
    }
    int under(Point a) { // 射线是否在点 a 下方
        return relation(p, p + v, a);
    }
    bool operator <(Line b) { // 比较
        if (!cmp(rad, b.rad)) {
            return under(b.p) < 0; // 靠上侧的排前面
        }
        return rad < b.rad;
```

```
    }
    ld dis2point(Point a) {  // 点 a 到直线的距离
        return fabsl((v ^ (a - p)) / v.length());
    }
    Point foot(Point a) { // 点在直线的投影点(垂足)
        return p + v * (v * (a - p) / v.length2());
    }
    Point symmetry(Point a) { // 点关于直线对称的点(即镜面反射)
        return foot(a) * 2 - a;
    }
    bool parallel(Line b) { // 两直线是否平行(共线也算)
        if (!sgn(v ^ b.v)) {
            return 1;
        }
        return 0;
    }
    bool colinear(Line b) { // 两直线是否共线
        if (parallel(b)) {
            if (!sgn((b.p - p) ^ v)) {
                return 1;
            }
        }
        return 0;
    }
    Point intersect(Line b) {  // 两直线交点(不能平行)
        assert(!parallel(b));
        Point u = p - b.p;
        T t = (b.v ^ u) / (v ^ b.v);
        return get_point(t);
    }
};

struct Segment { // 线段
    Point a, b;
    Segment(){}
    Segment(Point aa, Point bb) {
        if (bb < aa) {
            swap(aa, bb);
        }
        a = aa, b = bb;
    }
    bool isIntersect(Point p) { // 点 p 是否在线段上
        return !sgn((a - p) ^ (b - p)) && sgn((a - p) * (b - p)) <= 0;
    }
    bool parallel(Segment seg) {
        return Line(a, b - a).parallel(Line(seg.a, seg.b - seg.a));
    }
    bool colinear(Segment seg)  { // 两线段是否共线, 有重合部分
        // 只考虑 seg 退化的情况
        if (seg.a == seg.b) {
            if (Line(a, b - a).under(seg.a) == 0) {
                if (a <= seg.a && seg.a <= b) {
                    return 1;
                }
            }
            return 0;
        }
        if (Line(a, b - a).colinear(Line(seg.a, seg.b - seg.a))) {
            if ((a <= seg.a && seg.a <= b) || (seg.a <= a && a <= seg.b)) {
                return 1;
            }
        }
        return 0;
    }
    bool isIntersect(Segment seg) { // 两线段是否相交(含端点含共线)
        if (parallel(seg)) { //线段平行
            return colinear(seg);
        }
        const Point &a1 = a, &a2 = b, &b1 = seg.a, &b2 = seg.b;
        T c1 = (a2 - a1) ^ (b1 - a1), c2 = (a2 - a1) ^ (b2 - a1);
        T c3 = (b2 - b1) ^ (a1 - b1), c4 = (b2 - b1) ^ (a2 - b1);
        return sgn(c1) * sgn(c2) <= 0 && sgn(c3) * sgn(c4) <= 0;
    }
    bool isIntersect(Line l) { // 线段和直线是否相交 (含端点)
        return l.under(a) * l.under(b) <= 0;
    }
    ld dis2point(Point p) { // 点到线段距离
        if (sgn((p - a) * (b - a)) < 0 || sgn((p - b) * (a - b)) < 0) {
            return min(get_dis(b, p), get_dis(a, p));
        }
        return Line(a, b - a).dis2point(p);
    }
    ld dis2seg(Segment seg) {
        if (isIntersect(seg)) {
            return 0;
        }
        return min({dis2point(seg.a), dis2point(seg.b), seg.dis2point(a), seg.dis2point(b)});
    }
};

ld get_area(vector<Point> pts) {
    ld res = 0;
    for (int i = 1; i + 1 < pts.size(); i++) {
        res += (pts[i] - pts[0]) ^ (pts[i + 1] - pts[0]);
    }
    return res / 2;
}

vector<Point> half_plane_intersect(vector<Line> lines) {
    sort(all(lines));
    deque<Point> pts;
    deque<Line> ls;
    for (auto l : lines) {
        if (ls.empty()) {
            ls.pb(l);
            continue;
        }
        while(!pts.empty() && l.under(pts.back()) <= 0) {
            pts.pop_back();
            ls.pop_back();
        }
        while(!pts.empty() && l.under(pts[0]) <= 0) {
            pts.pop_front();
            ls.pop_front();
        }
        if (!sgn(ls.back().v ^ l.v)) {
            if (sgn(ls.back().v * l.v) > 0) {
                continue;
            } else {
                return vector<Point>{};
            }
        }
        pts.pb(l.intersect(ls.back()));
```

```
        ls.pb(l);
    }
    while(!pts.empty() && ls[0].under(pts.back()) <= 0) {
        pts.pop_back();
        ls.pop_back();
    }
    if (ls.size() > 2) {
        pts.push_back(ls[0].intersect(ls.back()));
    }
    return vector<Point>(pts.begin(), pts.end());
}

vector<Line> get_half_plane(Point u, Segment v) {
    if (((v.a - u) ^ (v.b - u)) == 0) {
        return vector<Line>{Line(u, v.a - u), Line(v.a, u - v.a)};
    }
    if (((v.a - u) ^ (v.b - u)) < 0) {
        swap(v.a, v.b);
    }
    return vector<Line> {
        Line(v.a, u - v.a),
        Line(v.a, v.b - v.a),
        Line(u, v.b - u)
    };
}

// (回转数法) 判点在多边形内外；点在多边形内返回 1, 点在多边形外返回 0, 点在多边形上返回-1
int point_in_polygon(const Point& p, vector<Point>& poly) {
    int wn = 0;
    int n = poly.size();
    for (int i = 0; i < n; i++) {
        if (Segment(poly[i], poly[(i + 1) % n]).isIntersect(p))
            return -1;
        int k = sgn((poly[(i + 1) % n] - poly[i]) ^ (p - poly[i]));
        int d1 = sgn(poly[i].y - p.y);
        int d2 = sgn(poly[(i + 1) % n].y - p.y);
        if (k > 0 && d1 <= 0 && d2 > 0)
            wn++;
        if (k < 0 && d2 <= 0 && d1 > 0)
            wn--;
    }
    if (wn != 0)
        return 1;
    return 0;
}

int inConvex(const Point& p, const vector<Point>& a) {  // a 为凸包(按顺序排列), 1 内 0 外 -1 边上
    if (a.empty())
        return false;
    int l = 1, r = a.size() - 1;
    while (l <= r) {
        int mid = l + r >> 1;
        double ls = (a[mid] - a[0]) ^ (p - a[0]);
        double rs = (a[mid + 1] - a[0]) ^ (p - a[0]);
        if (ls >= 0 && rs <= 0) {
            int type = sgn((a[mid + 1] - a[mid]) ^ (p - a[mid]));
            if (type == 0)
                return -1;
            else if (type == 1)
                return 1;
            return 0;
        } else if (ls < 0) {
            r = mid - 1;
        } else {
            l = mid + 1;
        }
    }
    return false;
}

struct Circle {
    Point p;
    double r;
    Circle(Point _p = Point(0, 0), double _r = 0) : p(_p), r(_r) {}
    // 三角形外接圆
    Circle(Point a, Point b, Point c) {
        Line u = Line({(a + b) / 2}, {(b - a).rotate(pi / 2)});
        Line v = Line({(a + c) / 2}, {(c - a).rotate(pi / 2)});
        p = u.intersect(v);
        r = get_dis(p, a);
    }
    // 三角形内切圆(bool t 只是为了与外接圆区别)
    Circle(Point a, Point b, Point c, bool t) {
        Line u, v;
        double m = atan2l(b.y - a.y, b.x - a.x), n = atan2l(c.y - a.y, c.x - a.x);
        u.p = a;
        u.v = u.p + Point(cos((n + m) / 2), sin((n + m) / 2));
        v.p = b;
        m = atan2l(a.y - b.y, a.x - b.x), n = atan2l(c.y - b.y, c.x - b.y);
        v.v = v.p + Point(cos((n + m) / 2), sin((n + m) / 2));
        p = u.intersect(v);
        r = Line(a, b).dis2point(p);
    }
    bool operator==(Circle v) {
        return (p == v.p) && sgn(r - v.r) == 0;
    }
    bool operator<(Circle v) const {
        return ((p < v.p) || ((p == v.p) && sgn(r - v.r) < 0));
    }
    double area() {
        return pi * r * r;
    }
    double length() {
        return 2 * pi * r;
    }
    // 点和圆的关系       -1 圆内   0 圆上   1 圆外
    int relation(Point a) {
        double dist = get_dis(p, a);
        if (sgn(dist - r) < 0)
            return -1;
        else if (sgn(dist - r) == 0)
            return 0;
        return 1;
    }
    // 直线和圆的关系     -1 相交   0 相切   1 相离
    int line_relation(Line v) {
        double dist = v.dis2point(p);
        if (sgn(dist - r) < 0)
            return -1;
        else if (sgn(dist - r) == 0)
            return 0;
        else
            return 1;
    }
    // 两圆的关系  5 相离   4 外切   3 相交   2 内切   1 内含
```

```
    int circle_relation(Circle v) {
        double dist = get_dis(p, v.p);
        if (sgn(dist - r - v.r) > 0)
            return 5;
        if (sgn(dist - r - v.r) == 0)
            return 4;
        double l = fabs(r - v.r);
        if (sgn(dist - r - v.r) < 0 && sgn(dist - l) > 0)
            return 3;
        if (sgn(dist - l) == 0)
            return 2;
        if (sgn(dist - l) < 0)
            return 1;
        return -1;
    }
    // 求两个圆的交点，并返回交点个数
    int cross_circle(Circle v, Point& p1, Point& p2) {
        int rel = circle_relation(v);
        if (rel == 1 || rel == 5)
            return 0;
        double d = get_dis(p, v.p);
        double l = (d * d + r * r - v.r * v.r) / (d * 2);
        double h = sqrtl(r * r - l * l);
        Point tmp = p + (v.p - p).trunc(l);
        p1 = tmp + ((v.p - p).to_left().trunc(h));
        p2 = tmp + ((v.p - p).to_right().trunc(h));
        if (rel == 2 || rel == 4)
            return 1;
        return 2;
    }
    // 求直线和圆的交点，返回交点个数
    int cross_line(Line v, Point& p1, Point& p2) {
        if ((*this).line_relation(v) == 1)
            return 0;
        Point a = v.foot(p);
        double d = v.dis2point(p);
        d = sqrtl(r * r - d * d);
        if (sgn(d) == 0) {
            p1 = a, p2 = a;
            return 1;
        }
        p1 = a + v.v.trunc(d);
        p2 = a - v.v.trunc(d);
        return 2;
    }
    // 过一点作圆的切线(先判断点和圆的关系)
    int tangent_line(Point q, Line& u, Line& v) {
        int x = relation(q);
        if (x == -1)
            return 0;
        if (x == 0) {
            u = Line(q, (p - q).to_left());
            v = u;
            return 1;
        }
        double d = get_dis(p, q);
        double rad = asin(r / d);
        u = Line(q, (p - q).rotate(rad));
        v = Line(q, (p - q).rotate(-rad));
        return 2;
    }
    // 求两圆相交面积
    double circle_cross_area(Circle v) {
        int rel = circle_relation(v);
        if (rel >= 4)
            return 0;
        if (rel <= 2)
            return min(area(), v.area());
        double d = get_dis(p, v.p);
        double hf = (r + v.r + d) / 2;
        double ss = 2 * sqrtl(hf * (hf - r) * (hf - v.r) * (hf -
     d));
        double a1 = acos((r * r + d * d - v.r * v.r) / (2.0 *
    r * d));
        a1 = a1 * r * r;
        double a2 = acos((v.r * v.r + d * d - r * r) / (2.0 *
    v.r * d));
        a2 = a2 * v.r * v.r;
        return a1 + a2 - ss;
    }
    // 得到过 a,b 两点, 半径为 r1 的两个圆
    friend int get_circle(Point a, Point b, double r1, Circl
    e& c1, Circle& c2) {
        Circle x(a, r1), y(b, r1);
        int t = x.cross_circle(y, c1.p, c2.p);
        if (!t)
            return 0;
        c1.r = c2.r = r1;
        return t;
    }
};

template <class T>
struct convex {
    vector<Point> q;
    convex() {}
    convex(vector<Point>& B) : q(B) {}
    convex(const convex& B) : q(B.q) {}
    convex& operator=(const convex& B) {
        q = B.q;
        return *this;
    }
    Point& operator[](int x) noexcept {
        return q[x];
    }
    int size() const {
        return q.size();
    }
    int nxt(int x) const {
        return x == size() - 1 ? 0 : x + 1;
    }
    int pre(int x) const {
        return x == 0 ? size() - 1 : x - 1;
    }
    void init(vector<Point>& v) {
        sort(v.begin(), v.end());
        int n = v.size(), top = 0;
        vector<int> st(n + 10);
        for (int i = 0; i < n; i++) {
            while (top > 1 && sgn((v[st[top]] - v[st[top - 1
    ]]) ^ (v[i] - v[st[top - 1]])) <= 0)
                top--;
            st[++top] = i;
        }
        int k = top;
        for (int i = n - 2; i >= 0; i--) {
            while (top > k && sgn((v[st[top]] - v[st[top - 1]
    ]) ^ (v[i] - v[st[top - 1]])) <= 0)
                top--;
            st[++top] = i;
        }
        for (int i = 1; i < top; i++)
            q.push_back(v[st[i]]);
        return;
    }
    double get_length() {
        double res = 0;
        for (int i = 0; i < size(); i++)
            res += get_dist(q[i], q[nxt(i)]);
        return res;
    }
```

```
    T get_area2() {
        T res = 0;
        for (int i = 0; i < size(); i++)
            res += (q[i] ^ q[nxt(i)]);
        return abs(res);
    }
    Point getBaryCentre() const { // 重心
        Point res(0, 0);
        double are = 0;
        const int sz = size();
        for (int i = 1; i < sz - 1; i++) {
            double tmp = (q[i] - q[0]) ^ (q[i + 1] - q[0]);
            if (!sgn(tmp))
                continue;
            are += tmp;
            res.x += (q[0].x + q[i].x + q[i + 1].x) / 3 * tmp;
            res.y += (q[0].y + q[i].y + q[i + 1].y) / 3 * tmp;
        }
        if (sgn(are))
            res = res / are;
        return res;
    }
    vector<T> sum;
    void get_sum() {
        vector<T> a(q.size());
        for (int i = 0; i < q.size(); i++)
            a[i] = q[pre(i)] ^ q[i];
        sum.resize(q.size());
        partial_sum(a.begin(), a.end(), sum.begin());
    }
    T query_sum(int l, int r) const {
        if (l <= r)
            return sum[r] - sum[l] + (q[r] ^ q[l]);
        return sum[size() - 1] - sum[l] + sum[r] + (q[r] ^ q[l]);
    }

    // 闵可夫斯基和
    convex operator+(const convex& B) const {
        const auto& a = this->q;
        const auto& b = B.q;
        int n = q.size(), m = b.size();
        Point sa = q[0], sb = b[0];
        for (int i = 0; i < n; i++) {
            if (a[i].y < sa.y || (a[i].y == sa.y && a[i].x < sa.x)
                sa = a[i];
        }
        for (int i = 0; i < m; i++) {
            if (b[i].y < sb.y || (b[i].y == sb.y && b[i].x < sb.x)) {
                sb = b[i];
            }
        }
        auto s = sa + sb;
        vector<Point> d(n + m);
        for (int i = 0; i < n; i++)
            d[i] = a[(i + 1) % n] - a[i];
        for (int i = 0; i < m; i++)
            d[n + i] = b[(i + 1) % m] - b[i];
        sort(d.begin(), d.end(), [&](const Point& A, const Point& B) {
            if (A.up() ^ B.up())
                return A.up() > B.up();
            return (A ^ B) > 0;
        });
        vector<Point> c(n + m);
        c[0] = s;
        for (int i = 0; i < n + m - 1; i++)
            c[i + 1] = c[i] + d[i];
        convex res;
        res.init(c);
        return res;
    }

    // 旋转卡壳
    template <class F>
    void rotating_calipres(const F& func) const {
        for (int i = 0, j = 1; i < q.size(); i++) {
            auto d = q[i], e = q[nxt(i)];
            func(d, e, q[j]);
            while (area(d, e, q[j]) <= area(d, e, q[nxt(j)])) {
                j = nxt(j);
                func(d, e, q[j]);
            }
        }
    }

    // 凸包直径(平方)
    T diameter2() const {
        if (q.size() == 1)
            return 0;
        if (q.size() == 2)
            return get_dis2(q[0], q[1]);
        T ans = 0;
        auto func = [&](const Point& a, const Point& b, const Point& c) {
            ans = max({ans, get_dis2(a, c), get_dis2(b, c)});
        };
        rotating_calipres(func);
        return ans;
    }

    // 凸多边形关于某一方向的极点, 复杂度 O(logn)
    template <class F>
    int extreme(const F& dir) const {
        const auto check = [&](const int i) {
            return sgn(dir(q[i]) ^ (q[nxt(i)] - q[i])) >= 0;
        };
        const auto dir0 = dir(q[0]);
        const auto check0 = check(0);
        if (!check0 && check(this->size() - 1))
            return 0;
        const auto cmp = [&](const Point& v) {
            const int vi = &v - q.data();
            if (vi == 0)
                return 1;
            const auto checkv = check(vi);
            const auto t = sgn(dir0 ^ (v - q[0]));
            if (vi == 1 && checkv == check0 && sgn(dir0 ^ (v - q[0])) == 0)
                return 1;
            return checkv ^ (checkv == check0 && t <= 0);
        };
        return partition_point(q.begin(), q.end(), cmp) - q.begin();
    }

    // 过凸多边形外一点求凸多边形的切线, 返回切点下标, 复杂度 O(logn)
    // 必须保证点在多边形外
    pair<int, int> tangent(const Point& a) const {
        const int i = extreme([&](const Point& u) {
            return u - a;
        });
        const int j = extreme([&](const Point& u) {
            return a - u;
        });
```

```
        return {i, j};
    }

    // 求平行于给定直线的凸多边形的切线, 返回切点下标, 复杂度 O(logn)
    pair<int, int> tangent(const Line& a) const {
        const int i = extreme([&](...) {
            return a.v;
        });
        const int j = extreme([&](...) {
            return -a.v;
        });
        return {i, j};
    }

    friend int inConvex(const Point& p, const convex& c) {
        return inConvex(p, c.q);
    }
};
using Convex = convex<ld>;

using _T = ld;
pair<_T, _T> minmax_triangle(const vector<Point>& vec) { //最小最大三角形面积
    if (vec.size() <= 2)
        return {0, 0};
    const _T tmpans = abs((vec[0] - vec[1]) ^ (vec[0] - vec[2]));
    _T maxans = tmpans, minans = tmpans;

    vector<pair<int, int>> evt;
    evt.reserve(vec.size() * vec.size());

    for (signed i = 0; i < vec.size(); i++) {
        for (signed j = 0; j < vec.size(); j++) {
            if (i == j || vec[i] == vec[j])
                continue;
            evt.push_back({i, j});
        }
    }
    sort(evt.begin(), evt.end(), [&](const pair<int, int>& u, const pair<int, int>& v) {
        const Point du = vec[u.second] - vec[u.first], dv = vec[v.second] - vec[v.first];
        return argcmp({du.y, -du.x}, {dv.y, -dv.x});
    });
    vector<signed> vx(vec.size()), pos(vec.size());
    for (signed i = 0; i < vec.size(); i++)
        vx[i] = i;
    sort(vx.begin(), vx.end(), [&](int x, int y) {
        return vec[x] < vec[y];
    });
    for (signed i = 0; i < vx.size(); i++)
        pos[vx[i]] = i;
    for (auto [u, v] : evt) {
        signed i = pos[u], j = pos[v];
        if (i > j)
            swap(u, v), swap(i, j);
        const Point vecu = vec[u], vecv = vec[v];
        if (i > 0)
            minans = min(minans, abs((vec[vx[i - 1]] - vecu) ^ (vec[vx[i - 1]] - vecv)));
        if (j < vx.size() - 1)
            minans = min(minans, abs((vec[vx[j + 1]] - vecu) ^ (vec[vx[j + 1]] - vecv)));
        maxans = max({maxans, abs((vec[vx[0]] - vecu) ^ (vec[vx[0]] - vecv)), abs((vec[vx.back()] - vecu) ^ (vec[vx.back()] - vecv))});
        swap(vx[i], vx[j]);
        pos[u] = j, pos[v] = i;
    }
    return {minans, maxans};
}
Circle min_circle_cover(vector<Point> q) {
    int n = q.size();
    random_shuffle(all(q));
    Circle c = (Circle){q[0], 0};
    auto get_line = [](Point a, Point b) { // 求中垂线
        return Line({(a + b) / 2}, {(b - a).rotate(-pi / 2)});
    };
    auto get_circle = [&](Point a, Point b, Point c) {
        Line l = get_line(a, b), r = get_line(a, c);
        Point p = l.intersect(r);
        return (Circle){p, get_dis(p, a)};
    };
    for (int i = 1; i < n; i++) {    // O(n)
        if (cmp(c.r, get_dis(c.p, q[i])) < 0) {
            c = {q[i], 0};
            for (int j = 0; j < i; j++) {
                if (cmp(c.r, get_dis(c.p, q[j])) < 0) {
                    c = {(q[i] + q[j]) / 2, get_dis(q[i], q[j]) / 2};

                    for (int k = 0; k < j; k++) {
                        if (cmp(c.r, get_dis(c.p, q[k])) < 0)
                            c = get_circle(q[i], q[j], q[k]);
                    }
                }
            }
        }
    }
    return c;
}
```

多面体欧拉定理

多面体欧拉定理

平面图欧拉定理: $V-E+F=k+1$ (V:点数, E:边数, F:面数, k:连通分量)

顶点数 - 棱长数 + 表面数 = 2

点数为n, 则边数最多为$3n-6$, 面数最多为$2n-4$

二十、代码查错

0. 造小数据/大数据+对拍
1. 输入输出的顺序、形式
2. 变量名是否打错
3. 特殊情况，0,1,2 等小范围情况
4. 数据范围（int -> long long -> __int128）
5. 检查算法思路、尝试换更简单实现方式
6. 全文检查，评测错误类型可能不准确！

二十一、Python Template

```
import heapq
import random
from collections import deque, defaultdict
import sys
import copy #use copy.deepcopy to copy data structure!
from functools import cmp_to_key
input = sys.stdin.readline
sys.setrecursionlimit(int(1e6))
##n = int(input())
##a = list(map(int, input().split()))
##自定义排序
##def mycmp(x, y):
##    return -1 if x > y else 1
##
##list.sort(a, key = cmp_to_key(mycmp))
##for i in range(0, len(a)):
##    if i > 0:
##        print("",end = ' ')
```

```
21.     ##      print(a[i], end = '')
22.     ##print("")
24.     ##alt3 注释,alt4 取消
25.     ##F1 看帮助文档
```

Another Python Template:

```
1.  import sys
2.  input = sys.stdin.readline
3.  # 1) inp — For taking integer inputs.
4.  # 2) inlt — For taking List inputs.
5.  # 3) insr — For taking string inputs. Actually it returns
    a List of Characters, instead of a string, which is easier
    to use in Python, because in Python, Strings are
    Immutable.
6.  # 4) invr — For taking space seperated integer
    variable inputs.
7.  ############ ---- Input Functions ----
    ############
8.  def inp():
9.      return(int(input()))
10. def inlt():
11.     return(list(map(int,input().split())))
12. def insr():
13.     s = input()
14.     return(list(s[:len(s) - 1]))
15. def invr():
16.     return(map(int,input().split()))
17. def solve():
18.     pass
19. if __name__ == "__main__":
20.     tc = inp()
21.     for _ in range(tc):
22.         solve()
```