常用模板（包括网络赛）

一、基础算法

1. 二分

1. 整数二分

```
// 1
while (l < r) {
    int mid = (l + r) >> 1;
    if (check(mid)) l = mid + 1;
    else r = mid;
}
// 2
while (l < r) {
    int mid = (l + r + 1) >> 1;
    if (check(mid)) l = mid;
    else r = mid - 1;
}
// 常用
while (l <= r) {
    int mid = (l + r) >> 1;
    if (check(mid)) ans = mid, l = mid + 1;
    else r = mid - 1;
}
```

2. 浮点数二分

```
// 通用版
while (r - l > 1e-5) {
    double mid = (l + r) / 2;
    if (check(mid)) l = mid;
    else r = mid;
}
// 防卡精度
for (int i = 0; i < 100; ++i) {
    double mid = (l + r) / 2;
    if (check(mid)) l = mid;
    else r = mid;
}
```

2. 离散化

```
vector<int> vt;
//二分找离散化值
inline int get_id(int x) { return lower_bound(vt.begin(), vt.end(), x) - vt.begin() + 1; }
inline void erase_vt() { // 离散化后去重
    sort(vt.begin(), vt.end());
    vt.erase(unique(vt.begin(), vt.end()), vt.end());
}
inline void id_table(int n, int *a, vector<int>& res) { // 打表，注意，原数组下标要从1开始，返回离散化后的表
    res.emplace_back(0);
    for (int i = 1; i <= n; ++i) res.emplace_back(get_id(a[i]));
}
```

3. 高精

```
#include <cstdio>
#include <cctype>
#include <cstring>
#include <algorithm>
#include <string>
#include <iostream>

using namespace std;

typedef long long ll;
/*
用前必读：
    本高精采用可调压位式运算对于不同需求记得更改下面的压位代码
    注意笔者只封装了 bign * int 没有封装 int * bign , 用时注意顺序
    若出现 bign * x,中 x 不是常量变量或者不是变量（例如具体数字 3 4 5……）
        记得把封装的与int相关的 运算符重载函数 的const和取地址符&去掉，防止报错
```

```
    若是爆栈请尝试开全局变量或者把数组改小即 M 改小
    本代码暂时只支持
        高精 + 高精
        高精1 - 高精2 （高精1 > 高精2）
        高精 * 低精
        高精 / 低精
        高精 * 高精
        高精 % 低精
        高精 与 高精 的大小对比
        高精 += 高精
        高精1 -= 高精2 （高精1 > 高精2）
        高精 *= 低精
        高精 /= 低精
        高精 *= 高精
        高精 %= 低精
    尚未完成的功能
        高精 / 高精
        高精 % 高精
        高精1 - 高精2 （高精1 < 高精2）
        ……
    其余请读者自己体会
*/

const int w = 1e8, M = 1e4, wsize = 8;//压位8个0
const char pout[] = "%08lld";//记得修改
struct bign{
    ll num[M];
    char str[M * wsize];
    int len;
    void clear() { memset(num, 0, sizeof num); len = 0;}
       /* 初始化 */
    bign() : len(0) { clear(); }
    bign(int n);
    bign(ll n);
    bign(char str[]);
    bign(string str);
    void change();
    void operator= (const int x) { *this = bign(x); }
    void operator= (const ll x) { *this = bign(x); }
    void operator= (char x[]) { *this = bign(x); }
    void operator= (string x) { *this = bign(x); }
    ll &operator[] (int x) { return num[x]; }
    /* 输出 */
    void print();
    void print() const;
    /* 比较(未验证) */
    bool operator< (const bign &b) const;
    bool operator> (const bign &b) const { return b < *this; }
    bool operator<= (const bign &b) const { return !(b < *this); }
    bool operator>= (const bign &b) const { return !(*this < b); }
    bool operator!= (const bign &b) const { return b < *this || *this < b; }
    bool operator== (const bign &b) const { return !(b < *this) && !(*this < b); }
    /* 各种运算 */
    bign operator+ (const bign &b) const; // 高精 + 高精
    bign operator* (const int &b) const; // 高精 * 低精
    bign operator* (const bign &b);      // 高精 * 高精
    bign operator- (const bign &b) const; // 高精 - 高精
    bign operator/ (const int &b) const; // 高精 / 低精
    bign operator% (const int &b);       // 高精 % 低精
    void operator+= (const bign &b);     // 高精 += 高精
    void operator*= (const int &b);      // 高精 *= 低精
    void operator*= (const bign &b);     // 高精 *= 高精
    void operator-= (const bign &b);     // 高精 -= 高精
    void operator/= (const int &b);      // 高精 /= 低精
    void operator%= (const int &b);      // 高精 %= 低精
    /* 输入输出重载 */
    friend istream& operator>> (istream &in, bign &res) ;
    friend ostream& operator<< (ostream &out, const bign &res) ;

};
bign::bign(int n) : len(0) {
    clear();
    while (1) {
        num[++len] = n % w, n /= w;
        if (!n) break;
    }
}
bign::bign(ll n) : len(0) {
    clear();
    while (1) {
        num[++len] = n % w, n /= w;
        if (!n) break;
    }
}
bign::bign(char str[]) : len(0) {
    int l = strlen(str) - 1;
    for (int i = l; i >= 0; i -= wsize) {
```

```
        ll tmp = 0, k = 1;
        for (int j = 0; j < wsize && i - j >= 0; j++, k *= 10) {
            tmp += (str[i - j] - '0') * k;
        }
        num[++len] = tmp;
    }
}
bign::bign(string str) : len(0) {
    int l = str.size() - 1;
    for (int i = l; i >= 0; i -= wsize) {
        ll tmp = 0, k = 1;
        for (int j = 0; j < wsize && i - j >= 0; j++, k *= 10) {
            tmp += (str[i - j] - '0') * k;
        }
        num[++len] = tmp;
    }
}
void bign::change() {
    int l = strlen(str) - 1;
    len = 0;
    for (int i = l; i >= 0; i -= wsize) {
        ll tmp = 0, k = 1;
        for (int j = 0; j < wsize && i - j >= 0; j++, k *= 10) {
            tmp += (str[i - j] - '0') * k;
        }
        num[++len] = tmp;
    }
}
/* 输出 */
void bign::print() {
    printf("%lld", num[len]);
    for (int i = len - 1; i > 0; i--) printf(pout, num[i]);
    puts("");
}

void bign::print() const {
    printf("%lld", num[len]);
    for (int i = len - 1; i > 0; i--) printf(pout, num[i]);
    // puts("");
}

/* 比较(未验证) */
bool bign::operator< (const bign &b) const {
    if (len != b.len) return len < b.len;
    for (int i = len; i > 0; i--) {
        if (num[i] != b.num[i]) return num[i] < b.num[i];
    }
    return false;
}

/************************常用***************************/
/* 各种运算 */
bign bign::operator+ (const bign &b) const {
    bign res = *this;
    if (res.len < b.len) res.len = b.len;
    for (int i = 1; i <= res.len; i++) {
        res.num[i] += b.num[i];
        res.num[i + 1] += res.num[i] / w;
        res.num[i] %= w;
    }
    while (res.num[res.len + 1]) res.len++;
    return res;
}

bign bign::operator* (const int &b) const {
    bign res;
    ll carry = 0;
    for (int i = 1; i <= len; i++) {
        ll tmp = num[i] * b + carry;
        res.num[++res.len] = tmp % w;
        carry = tmp / w;
    }
    while (carry) {
        res.num[++res.len] = carry % w;
        carry /= w;
    }
    return res;
}

bign bign::operator* (const bign &b) {
    bign res;
    for (int i = 1; i <= len; i++) {
        ll up = 0;
        for (int j = 1; j <= b.len; j++) {
            ll tmp = num[i] * b.num[j] + res.num[i + j - 1] + up;
            res.num[i + j - 1] = tmp % w;
            up = tmp / w;
```

```
        }
        if (up) res.num[i + b.len] = up;
    }
    res.len = len + b.len;
    while (res.len > 1 && res.num[res.len] == 0) res.len--;
    return res;
}
/*****************************************************/

bign bign::operator- (const bign &b) const {
    bign res = *this;
    for (int i = 1; i <= len; i++) {
        if (res.num[i] < b.num[i]) res.num[i] += w, res.num[i + 1]--;
        res.num[i] -= b.num[i];
    }
    while (res.len > 1 && res.num[res.len] == 0) res.len--;
    return res;
}

bign bign::operator/ (const int &b) const {
    bign res;
    res.len = len;
    ll r = 0;//余数
    for (int i = len; i > 0; i--) {
        r = r * w + num[i];
        if (r < b) res.num[i] = 0;
        else res.num[i] = r / b, r %= b;
    }
    while (res.len > 1 && res.num[res.len] == 0) res.len--;
    return res;
}

bign bign::operator% (const int &b) {
    ll r = 0;//余数
    for (int i = len; i > 0; i--) {
        r = r * w + num[i];
        if (r >= b) r %= b;
    }
    return bign(r);
}
void bign::operator+= (const bign &b) {
    if (len < b.len) len = b.len;
    for (int i = 1; i <= len; i++) {
        num[i] += b.num[i];
        num[i + 1] += num[i] / w;
        num[i] %= w;
    }
    while (num[len + 1]) len++;
}
void bign::operator*= (const int &b) {
    for (int i = 1; i <= len; i++) num[i] *= b;
    for (int i = 1; i <= len; i++) {
        num[i + 1] += num[i] / w;
        num[i] %= w;
    }
    while(num[len + 1]) {
        len++;
        num[len + 1] = num[len] / w;
        num[len] %= w;
    }
}

void bign::operator*= (const bign &b) {
    *this = *this * b;
}

void bign::operator-= (const bign &b) {
    for (int i = 1; i <= len; i++) {
        if (num[i] < b.num[i]) num[i] += w, num[i + 1]--;
        num[i] -= b.num[i];
    }
    while (len > 1 && num[len] == 0) len--;
}

void bign::operator/= (const int &b) {
    ll r = 0;
    for (int i = len; i > 0; i--) {
        r = r * w + num[i];
        if (r < b) num[i] = 0;
        else num[i] = r / b, r %= b;
    }
    while (len > 1 && num[len] == 0) len--;
}

void bign::operator%= (const int &b) {
    ll r = 0;
    for (int i = len; i > 0; i--) {
```

```
        r = r * w + num[i];
        if (r >= b) r %= b;
    }
    *this = bign(r);
}

istream& operator>> (istream &in, bign &res) {
    in >> res.str;
    res.change();
    return in;
}
ostream& operator<< (ostream &out, const bign &res) {
    res.print();
    return out;
}

int main() {
    return 0;
}
```

4. 差分

```
const int M = 1e5 + 10;
int dif[M], arr[M], res[M];

void change(int l, int r, int v) { // [l, r] + v
    dif[l] += v, dif[r + 1] -= v;
}
void init(int n) {
    dif[1] = arr[1];
    for (int i = 2; i <= n; ++i) dif[i] = arr[i] - arr[i - 1];
}
void get_res(int n) {
    res[1] = dif[1];
    for (int i = 2; i <= n; ++i) {
        res[i] = dif[i] + res[i - 1];
    }
}
```

5.三分

```
double ts(int l, int r) {
    double ans1 = 0x3f3f3f3f, ans2 = 0x3f3f3f3f;
    while (l < r) {
        int lmid = l + (r - l) / 3;
        int rmid = r - (r - l) / 3;
        ans1 = f(lmid), ans2 = f(rmid);
        if (ans1 <= ans2) r = rmid - 1; // 凹函数的最小值， 凸函数<=反过来
        else l = lmid + 1;
    }
    return min(ans1, ans2); //凸函数这里记得改成max
}
```

二、数论

1. gcd与lcm

```
typedef long long ll;
//最大公因数，公约数
ll gcd(ll a, ll b, ll m = 1) { while(b) m = a % b, a = b, b = m; return a; }
//最小公倍数
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
```

2. ex_gcd

```
/*
通解为
x' = x * c / gcd + (b / gcd) * k
y' = y * c / gcd - (a / gcd) * k
*/
typedef long long ll;
// 扩展欧几里得算法核心函数
void exgcd(ll a, ll b, ll &g, ll &x, ll &y) {
    if (!b) {
        g = a, x = 1, y = 0;
```

```
        return;
    }
    exgcd(b, a % b, g, y, x);
    y -= x * (a / b);
}
/**
 * 此函数为 求解 ax + by = c
 * 返回 x的最小正整数解
 * 返回 -1 说明无解
*/
ll minx(ll a, ll b, ll c) {
    ll x, y, g;
    exgcd(a, b, g, x, y);
    if (c % g != 0) return -1;
    ll t = abs(b / g);
    x = (x * c / g % t + t) % t;
    return x == 0 ? x + t : x; // 返回最小正整数解
}
```

3. 素数筛

- 埃式筛

```
const int M = 101000;
int pri[M], cnt = 0;
bool isp[M];
// 复杂度O(nlogn)
// true 为非素数， false 为素数
void table() {
    isp[0] = isp[1] = true;
    for (int i = 2; i < M; i++) {
        if (isp[i]) continue;
        pri[cnt++] = i;
        for (int j = i + i; j < M; j += i) isp[j] = true;
    }
}
```

- 线性筛

```
// 复杂度O(n)
const int M = 1e5 + 10;
int pri[M], cnt = 0;
bool isp[M];
// true 为非素数， false 为素数
void table() {
    isp[0] = isp[1] = 1;
    for (int i = 2; i < M; i++) {
        if (!isp[i]) pri[cnt++] = i;
        for (int j = 0; j < cnt && i * pri[j] < M; j++) {
            isp[i * pri[j]] = 1;
            if (!(i % pri[j])) break;
        }
    }
}
```

4. 逆元

1. 线性法

```
typedef long long ll;
// 时间复杂度O(n)
void fny(const int &n, ll *inv, const ll mod) {
    inv[0] = inv[1] = 1;
    for (ll i = 2; i <= n; i++) {
        inv[i] =((mod - mod / i) * inv[mod % i]) % mod;
    }
}
```

2. 扩欧法

```
typedef long long ll;
void exgcd(ll a, ll b, ll &g, ll &x, ll &y) {
    if (!b) {
        g = a, x = 1, y = 0;
        return;
    }
    exgcd(b, a % b, g, y, x);
    y -= x * (a / b);
```

```
}

// ax=1(mod m)
ll inverse(ll a, ll m) {//扩展欧几里得法求逆元，返回-1代表没有逆元
    ll g, x, y;
    exgcd(a, m, g, x, y);
    return g == 1 ? (x % m + m) % m : -1;
}
```

2. 费马小定理法

```
//a ^ (p - 1) = 1 (mod p), p为素数
//a ^ (p - 2) = a ^ (-1) (mod p)
//a 的逆元为 a ^ (p - 2)
typedef long long ll;
ll pow_f(ll a, ll b, const ll mo) {
    ll ans = 1;
    a %= mo;
    while (b) {
        if (b & 1) ans = (ans * a) % mo;
        a = (a * a) % mo;
        b >>= 1;
    }
    return ans;
}

ll inverse(ll a, ll m) {
    return pow_f(a, m - 2, m);
}
```

5. 快速幂

```
#define ll long long
ll powf(ll a, ll b, const ll mod) {// 返回a^b % mod
    a %= mod;
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod, b >>= 1;
    }
    return res;
}
```

6. 矩阵快速幂

```
const ll MOD = 1e9 + 7;
#define MO(x) ((x) % MOD)

struct Mat {
    ll mat[10][10];
    int n; // n * n 阶矩阵
    Mat(int n = 2) : n(n) { memset(mat, 0, sizeof mat); }//记得修改
    void to_one() {
        for (int i = 0; i < n; i++)
            mat[i][i] = 1;
    }
    Mat operator*(const Mat a) const {
        Mat res;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                ll sum = 0;
                for (int k = 0; k < n; k++) {
                    sum += MO(this->mat[i][k] * a.mat[k][j]);
                }
                res.mat[i][j] = MO(sum);
            }
        }
        return res;
    }
    ll *operator[](int x) { return mat[x]; }
};


Mat pow_f(Mat a, ll b) { //a ^ b次幂
    Mat ans;
    ans.to_one();
    while (b) {
        if (b & 1)
            ans = ans * a;
        a = a * a;
```

```
        b >>= 1;
    }
    return ans;
}
```

7. 高斯消元

1. 普通浮点数高斯消元，洛谷模板题

```
// 洛谷模板题
#include <algorithm>
#include <cstdio>
#include <vector>

using namespace std;
const int N = 110;
const double eps = 1e-6; // 用来控制进度
// 普通的高斯消元是将矩阵转化成上三角的形式，再回带求出答案
double ans[N]; // 用来记录答案
int gauss(int n, int m, vector<vector<double>> &a) { // n行m + 1列的增广矩阵
    int r, c; // 当前行和当前列
    for (r = c = 0; c < m && r < n; ++ c) {
        int maxr = r; // 记录最大
        for (int i = r + 1; i < n; ++i) if (abs(a[i][c]) > abs(a[maxr][c]))
            maxr = i; // 寻找从当前行开始向下走的当前列中的绝对值最大值
        if (r ^ maxr) swap(a[r], a[maxr]); // 如果不是当前行，则交换两行
        if (abs(a[r][c]) < eps) continue; // 如果当前行当前列的最大值为0则不作消元
        for (int i = m; i >= c; --i) a[r][i] /= a[r][c]; // 将当前行的从当前列开始到最后一列
        for (int i = r + 1; i < n; ++i) {
            if (abs(a[i][c]) < eps) continue; // 如果改行的当前列已经是0，则不作消元
            for (int j = m; j >= c; --j) { // 逆向消元，可以少开一个变量
                a[i][j] -= a[r][j] * a[i][c];
            }
        }
        ++r;
    }
    if (r < n) { // 无穷解 或者 无解
        for (int i = r; i < n; ++r) if (abs(a[i][n]) < eps) return 0;
        return -1; // 无穷解
    }
    for (int i = n - 1; ~i; --i) { // 回带
        for (int j = i + 1; j < n; ++j) {
            a[i][n] -= a[i][j] * ans[j];
        }
        ans[i] = a[i][n];
    }
    return 1; // 唯一解
}
vector<vector<double>> a;
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        a.push_back({});
        for (int j = 0; j <= n; ++j) {
            double x;
            scanf("%lf", &x);
            a[i].push_back(x);
        }
    }
    if (gauss(n, n, a) == 1) {
        for (int i = 0; i < n; ++i) printf("%.2f\n", ans[i]);
    } else puts("No Solution");
    return 0;
}
```

2. 浮点数高斯约旦消元法， 洛谷模板题

```
// 洛谷模板题
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;
const double eps = 1e-6;
double ans[110]; // 记录答案
int gauss_j(int n, int m, vector<vector<double>> &a) { // n行m + 1列增广矩阵
    int r, c; // 当前行当前列
    for (r = c = 0; c < m && r < n; ++c) { //
        int maxr = r; // 记录最大值
```

```
        for (int i = r + 1; i < n; ++i) if (abs(a[i][c]) > abs(a[maxr][c]))
            maxr = i; // 寻找从当前行开始向下走的当前列中的绝对值最大值
        if (maxr ^ r) swap(a[r], a[maxr]); // 交换两行
        if (abs(a[r][c]) < eps) continue; // 如果为当前行中的当前列的值为0
        for (int i = 0; i < n; ++i) { // 约旦消元
            if (abs(a[i][c]) < eps || i == r) continue; // 如果是当前行或者改行的当前列已经是0
            for (int j = m; j >= c; --j) a[i][j] -= a[i][c] / a[r][c] * a[r][j];
        }
        ++r;
    }
    if (r < n) return 0; // 无解或者无穷解
    for (int i = 0; i < n; ++i) ans[i] = a[i][n] / a[i][i]; // 回带
    return 1;
}

vector<vector<double>> a;
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        a.push_back({});
        for (int j = 0; j <= n; ++j) {
            double x;
            scanf("%lf", &x);
            a[i].push_back(x);
        }
    }
    if (gauss_j(n, n, a) == 1) {
        for (int i = 0; i < n; ++i) printf("%.2f\n", ans[i]);
    } else puts("No Solution");
    return 0;
}
```

3. 模意义下的高斯消元法，POJ - 2065 SETI

```
// POJ - 2065 SETI
#include <cassert>
#include <vector>
#include <cstdio>
#include <algorithm>
#include <cstring>

using namespace std;

int gcd(int a, int b) { int m; while(b) m = a % b, a = b, b = m; return a; }
int lcm(int a, int b) { return a / gcd(a, b) * b; }
int p, ans[310]; // 记录答案

int powf(int a, int b, const int mod, int ans = 1) {
    a %= mod;
    while (b) {
        if (b & 1) ans = ans * a % mod;
        b >>= 1, a = a * a % mod;
    }
    return ans;
}
int inv(int a, int m) {
    return powf(a, m - 2, m);
}
//模意义下的高斯消元不需要用约旦的方式，因为整数不用考虑精度问题
// n 行 m + 1列的增广矩阵，从0开始
int gauss(int n, int m, vector<vector<int> > &a, const int &p) { // 传入模p
    int r, c; // 当前行和当前列
    for (r = c = 0; c < m && r < n; ++c) {
        int maxr = r; // 记录最大值
        for (int i = r + 1; i < n; ++i) if (abs(a[i][c]) > abs(a[maxr][c]))
            maxr = i; // 寻找当前列中从当前行开始的绝对值的最大值
        if (maxr ^ r) swap(a[r], a[maxr]); // 交换两行
        if (!a[r][c]) continue; // 如果为0
        for (int i = r + 1; i < n; ++i) {
            if (!a[i][c]) continue; // 如果当前列中 改行已经为0
            int LCM = lcm(abs(a[i][c]), abs(a[r][c]));
            int x = LCM / abs(a[i][c]), y = LCM / abs(a[r][c]); // 使该行乘x，当前行乘y使得他们在当前列的数都变成同一个数
            if (a[i][c] * a[r][c] < 0) y = -y; // 如果有一个是负数
            for (int j = c; j <= m; ++j)
                a[i][j] = ((a[i][j] * x - a[r][j] * y) % p + p) % p;
        }
        ++r;
    }
    for (int i = r; i < n; ++i) if (a[i][c]) return 0; // 无解
    if (r < m) return -1; // 无穷解
    for (int i = m - 1; ~i; --i) {
        int tmp = a[i][m];
        for (int j = i + 1; j < m; ++j) {
```

```
            if (!a[i][j]) continue;
            tmp -= ans[j] * a[i][j];
            tmp = ((tmp % p) + p) % p;
        }
        ans[i] = tmp * inv(a[i][i], p) % p; // 求逆元可以换成别的方式求
    }
    return 1;
}

vector<vector<int> > mat;
char str[110];
int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        mat.clear();
        scanf("%d %s\n", &p, str + 1);
        int n = strlen(str + 1);
        for (int i = 1; i <= n; ++i) {
            mat.push_back(vector<int>(n + 1, 0));
            if (str[i] != '*') mat[i - 1][n] = str[i] - 'a' + 1;
            for (int j = 0; j < n; ++j) mat[i - 1][j] = powf(i, j, p);
        }
        int res = gauss(n, n, mat, p);
        for (int i = 0; i < n; ++i) printf("%d ", ans[i]);
        puts("");
    }
    return 0;
}
```

4. 异或的高斯消元法（带解决自由变元的），POJ1681 Painter's Problem

```
//POJ1681 Painter's Problem
#include <vector>
#include <cstdio>
#include <algorithm>

using namespace std;
const int N = 305;
int ans[N], freew[N]; // 记录答案和记录自由变元是哪些
int gauss(int n, int m, vector<vector<int> > &a) { // n行m + 1列的增广矩阵
    int num = 0, r, c; // 自由变元的个数 当前行 当前列
    for (int i = 0; i < n; ++i) ans[i] = freew[i] = 0; // 初始化答案和自由变元
    for (r = c = 0; r < n && c < m; ++c) {
        int maxr = r; // 记录最大值
        for (int i = r + 1; i < n && !a[maxr][c]; ++i) maxr = i; // 查找1
        if (maxr ^ r) swap(a[r], a[maxr]); // 交换两行
        if (!a[r][c]) {// 如果这个数是0， 则说明第c个变元是自由变元
            freew[num++] = c; // 记录自由变元
            continue;
        }
        for (int i = r + 1; i < n; ++i) { // 消元，消成一个上三角
            if (!a[i][c]) continue; //如果该行的当前列是0则该行不作消元
            for (int j = m; j >= c; --j) a[i][j] ^= a[r][j]; // 直接异或，和加法不同
        }
        ++r;
    }
    for (int i = r; i < n; ++i) if (a[i][m]) return -1; // 无解
    int cnt = 1 << (n - r), ret = 1 << 29; // 自由变元的取值方案数， 记录最小操作数
    for (int i = 0; i < cnt; ++i) { // 状态压缩求解
        int res = 0, indx = i; // 当前方案的最小操作数， 操作方案
        for (int j = 0; j < num; ++j, indx >>= 1) {
            ans[freew[j]] = indx & 1;
            res += indx & 1; // 如果是1则说明这个地方需要操作，固操作数+1
        }
        for (int j = r - 1; j >= 0; --j) {
            ans[j] = a[j][m]; // 记录答案
            for (int k = j + 1; k < m; ++k) {
                if (a[j][k]) ans[j] ^= ans[k]; // 如果这个数的系数不为0
            }
            res += ans[j]; // 增加操作数
        }
        ret = min(res, ret); // 取最小值
    }
    return ret;
}

vector<vector<int> > mat;
int n;
const int dx[] = {0, 1, 0, -1, 0};
const int dy[] = {0, 0, 1, 0, -1};
char g[30][30];
bool check(int r, int c) { return r >= 0 && c >= 0 && r < n && c < n; }
int main() {
    int _;
```

```
    scanf("%d", &_);
    for (int t = 1; t <= _; ++t) {
        mat.clear();
        scanf("%d", &n);
        mat.resize(n * n, vector<int>(n * n + 1, 0));
        for (int i = 0, r = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j, ++r) {
                for (int k = 0; k < 5; ++k) {
                    int nr = i + dx[k], nc = j + dy[k];
                    if (check(nr, nc)) mat[r][nr * n + nc] = 1;
                }
            }
        }
        for (int i = 0, r = 0; i < n; ++i) {
            scanf("%s", g[i]);
            for (int j = 0; j < n; ++j, ++r) {
                if (g[i][j] == 'w') mat[r][n * n] = 1;
            }
        }
        int res = gauss(n * n, n * n, mat);
        printf("%d\n", res);
    }
    return 0;
}
```

8. lucas

```
const ll mod = 10007;
ll fac[mod + 10], inv[mod + 10];
void fny(const int &n, ll *inv, const ll mod) {
    fac[0] = fac[1] = inv[0] = inv[1] = 1;
    for (ll i = 2; i <= n; i++) {
        inv[i] =((mod - mod / i) * inv[mod % i]) % mod;
        fac[i] = fac[i - 1] * i % mod;
    }
}

ll comb(ll n, ll m) {
    if (m > n) return 0;
    return fac[n] * inv[fac[n - m] * fac[m] % mod] % mod;
}


ll lucas(ll n, ll m) {
    if (m == 0) return 1;
    if (n < mod) return comb(n, m);
    return comb(n % mod, m % mod) * lucas(n / mod, m / mod) % mod;
}
```

9. 线性基

```
struct LB {
    using ll = long long;
    ll d[65], cnt, num; // cnt 原序列的个数， num 基的个数
    bool re;
    LB () : cnt(0), num(0), re(false) { memset(d, 0, sizeof d); }
    // 添加一个数x
    void add(ll x) {
        ++cnt;
        for (int i = 60; ~i && x; --i) {
            if ((x >> i) & 1) {
                if (d[i]) x ^= d[i];
                else d[i] = x, x = 0, ++num, re = 0;
            }
        }
    }
    // 询问是否能异或出x
    bool check(ll x) {
        for (int i = 60; ~i && x; --i) {
            if ((x >> i) & 1) {
                if (d[i]) x ^= d[i];
                else return true;
            }
        }
        return false;
    }
    ll get_max() {
        ll res = 0;
        for (int i = 60; ~i; --i) {
            if ((d[i] ^ res) > res) res ^= d[i];
        }
        return res;
```

```
    }

    // 求的是线性基的异或最小值，不是原序列，否则要特判是否为0
    ll get_min() {
        for (int i = 0; i <= 60; ++i) {
            if (d[i]) return d[i];
        }
    }
    void rebuild() {
        for (int i = 0; i <= 60; ++i) {
            for (int j = 0; j < i; ++j) {
                if ((d[i] >> j) & 1) {
                    d[i] ^= d[j];
                }
            }
        }
        re = true;
    }
    ll k_th(ll k) {
        if (!re) rebuild();
        if (k == 1 && num < cnt) return 0; // 如果异或得到0
        if (num < cnt) --k;
        ll res = 0;
        for (int i = 0; i <= 60; ++i) {
            if (d[i]) {
                if (k & 1) res ^= d[i];
                k >>= 1;
            }
        }
        return res;
    }
};
```

10. CRT

```
ll crt(ll *a, ll *b, int n) {// x % b[i] = a[i], 返回最小的x,  b[i]中互质
    ll mul = 1, ret = 0;
    for (int i = 0; i < n; ++i) mul *= b[i];
    for (int i = 0; i < n; ++i) {
        ll minlcm = mul / b[i];
        ll inv = inverse(minlcm, b[i]); // 求逆元
        ret = (ret +  minlcm * inv * a[i]) % mul;
    }
    return (ret + mul) % mul;
}
```

11. 欧拉函数

- 1. 线性O(n)

```
void getphi() {
    phi[1] = 1;
    for (int i = 2; i < M; ++i) {
        if (!isp[i]) pri[cnt++] = i, phi[i] = i - 1;
        for (int j = 0; j < cnt && i * pri[j] < M; ++j) {
            isp[i * pri[j]] = 1;
            if (i % pri[j] == 0) {
                phi[i * pri[j]] = pri[j] * phi[i];
                break;
            } else {
                phi[i * pri[j]] = (pri[j] - 1) * phi[i];
            }
        }
    }
}
```

12. 求组合数

- 递推

```
ll comb(ll n, ll m) {
    if (n < m) return 0;
    ll ret = 1;
    for (int i = 1; i <= m; ++i) ret *= (n - i + 1) / i;
    return ret;
}
```

三、数据结构

1. 并查集

1. 简便的路径压缩版

```
const int Max = 1e5 + 10;
int fa[Max];
inline void init() { for (int i = 0; i < Max; i++) fa[i] = i; }

int findfa(int x) {
    return x == fa[x] ? x : fa[x] = findfa(fa[x]);
}

void Un(int a, int b) {
    int fa1 = findfa(a);
    int fa2 = findfa(b);
    if (fa1 != fa2) fa[fa1] = fa2;
}
```

2. 网络赛版

```
class UF {
public:
    vector<int> parent;
    vector<int> size;
    int n;
    // 当前连通分量数目
    int cnt;

public:
    UF(int _n): n(_n), cnt(_n), parent(_n), size(_n, 1) {
        int i = 0;
        for (auto &x : parent) x = i++;
    }

    int findset(int x) {
        return parent[x] == x ? x : parent[x] = findset(parent[x]);
    }

    bool unite(int x, int y) {
        x = findset(x);
        y = findset(y);
        if (x == y) {
            return false;
        }
        if (size[x] < size[y]) {
            swap(x, y);
        }
        parent[y] = x;
        size[x] += size[y];
        --cnt;
        return true;
    }

    bool conn(int x, int y) {
        x = findset(x);
        y = findset(y);
        return x == y;
    }
};
```

2. 树状数组

1. 单点修改与区间查询

```
// 修改复杂度与查询复杂度O(logn)
#define lb(x) ((x) & (-x))
#define ll long long
const int N = 1e6 + 5, M = 2e5 + 5;
ll n, m, a[N], bit[N];
// 初始化
void build(int n) {
    for (int i = 1; i <= n; ++i) {
        bit[i] += a[i];
        int j = lb(i) + i;
        if (j <= n) bit[j] += bit[i];
    }
}
// 单点修改
void update(int index, ll val) {
    a[index] += val;
    while (index <= n) {
```

```
        bit[index] += val;
        index += lb(index);
    }
}
// 前缀查询
ll get(int index) {
    ll res = 0;
    while (index) {
        res += bit[index];
        index -= lb(index);
    }
    return res;
}
// 区间和查询
ll get(int l, int r) { return get(r) - get(l - 1); }
```

2. 区间修改，区间查询

    1. 设$d[i]=a[i]-a[i-1]$
    2. 则$a[x]=\sum_{i=1}^{x}d_i$
    3. 设$sum[x]=\sum_{i=1}^{x}a_i$，即$sum[x]=d[1]+d[1]+d[2]+d[1]+d[2]+d[3]+.....+d[1]+.....+d[n]$
    4. 化简得$sum[x]=\sum_{i=1}^{x}d_i\times(n-i+1)$
    5. 得$sum[x]=(n+1)\times\sum_{i=1}^{x}d_i-\sum_{i=1}^{x}i\times d_i$
    6. 固开两个树状数组，一个维护差分数组d_i，一个维护$i\times d_i$

```
#define lb(x) ((x) & (-x))
#define int long long
const int N = 5e3 + 5, M = 1e6 + 5;
int n, m, d[M], id[M];
// 基础树状数组单点更新
void update(int i, int val, int *bit) {
    while (i <= n) {
        bit[i] += val;
        i += lb(i);
    }
}
// 单点修改
void update(int i, int val) {
    update(i, val, d), update(i, val * i, id);
}
// 区间修改
void update(int l, int r, int val) {
    update(l, val, d), update(r + 1, -val, d);
    update(l, l * val, id), update(r + 1, (-val) * (r + 1), id);
}
// 前缀查询
int get(int i, int *bit) {
    int res = 0;
    while (i) res += bit[i], i -= lb(i);
    return res;
}
// 区间和查询
int get(int l, int r) {
    int res = get(r, d) * (r + 1) - get(r, id);
    res -= get(l - 1, d) * l - get(l - 1, id);
    return res;
}
```

- 网络赛类封装版

```
template<class T>
class BIT{
    #define lb(x) ((x) & (-x))
    vector<T> sum, maxv, arr;
    const int inf = 0x3f3f3f3f;
    T getSum(int i) {
        T ret = 0;
        while (i) {
            ret += sum[i];
            i -= lb(i);
        }
        return ret;
    }
    int n;
public:
    BIT(int n, T *a) : sum(n + 1, 0), maxv(n + 1, -inf), arr(n + 1), n(n) {
        for (int i = 1; i <= n; ++i) arr[i] = a[i];
        for (int i = 1; i <= n; ++i) {
            sum[i] += a[i];                 //求和
            maxv[i] = max(maxv[i], a[i]);   //最大值
            int j = i + lb(i);
            if (j <= n) {
```

```
                sum[j] += sum[i];
                maxv[j] = max(maxv[i], maxv[j]);
            }
        }
    }
    BIT(int n) : sum(n + 1, 0), maxv(n + 1, -inf), arr(n + 1, 0), n(n) {}
    void update(int indx, T val) { // 将indx下标的数加v
        int i = indx;
        T &tmp = arr[indx];
        tmp += val;
        while (i <= n) {
            sum[i] += val;
            maxv[i] = arr[i];
            for (int j = 1; j < lb(i); j <<= 1) {
                maxv[i] = max(maxv[i], maxv[i - j]);
            }
            i += lb(i);
        }
    }

    T getSum(int l, int r) {
        return getSum(r) - getSum(l - 1);
    }

    T getMax(int l, int r) {
        T ret = arr[r];
        while(r >= l) {
            ret = max(arr[r], ret);
            for (r--; r - lb(r) >= l; r -= lb(r)) ret = max(maxv[r], ret);
        }
        return ret;
    }

    T getMax(int i) { return getMax(1, i); }
    void clear() {
        fill(sum.begin(), sum.end(), 0);
        fill(arr.begin(), arr.end(), 0);
        fill(maxv.begin(), maxv.end(), -inf);
    }
};
```

3. 线段树

- 精简版
- 结构体版

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

const int M = 1e6;
typedef long long ll;
template<class T>
struct Tree{
    #define ls(node) (node << 1)
    #define rs(node) ((node << 1) | 1)
    Tree(int len = 10): len(len) {}
    T sum[M << 2], lazy[M << 2], arr[M];
    int len;

private:
    void pushup(const int node) {// 写题目要求维护的代码，如求和，最大最小……
        sum[node] = max(sum[ls(node)], sum[rs(node)]);
    }

    void pushdown(int l, int r, const int node) {//同pushup
        int mid = (l + r) >> 1;
        lazy[ls(node)] += lazy[node], lazy[rs(node)] += lazy[node];
        sum[ls(node)] += (mid - l + 1) * lazy[node];
        sum[rs(node)] += (r - mid) * lazy[node];
        lazy[node] = 0;
    }

    void build(int l, int r, int node) {
        if (l == r) {
            sum[node] = arr[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(l, mid, ls(node)), build(mid + 1, r, rs(node));
        pushup(node);
    }

    void update(int ql, int qr, T v, int l, int r, int node) {
        if (ql <= l && r <= qr) {
```

```
            sum[node] += (r - l + 1) * v;//更新操作根据题目要求更改
            lazy[node] += v;
            return;
        }
        if (lazy[node]) pushdown(l, r, node);
        int mid = (l + r) >> 1;
        if (ql <= mid) update(ql, qr, v, l, mid, ls(node));
        if (qr > mid) update(ql, qr, v, mid + 1, r, rs(node));
        pushup(node);
    }

    T getAsk(int ql, int qr, int l, int r, int node) {
        if (ql <= l && r <= qr) return sum[node];//上同
        if (lazy[node]) pushdown(l, r, node);
        T res = 0;
        int mid = (l + r) >> 1;
        if (ql <= mid) res = getAsk(ql, qr, l, mid, ls(node));
        if (qr > mid) res = max(res, getAsk(ql, qr, mid + 1, r, rs(node)));
        return res;
    }

public:
    //以下为可以，直接调用的函数
    void build() { build(1, len, 1); }
    void update(int ql, int qr, T v) { update(ql, qr, v, 1, len, 1); }
    void update(int index, T v) { update(index, index, v, 1, len, 1); }
    T getAsk(int ql, int qr) { return getAsk(ql, qr, 1, len, 1); }
    void clear() {
        memset(arr, 0, sizeof(T) * (len + 10));
        memset(sum, 0, sizeof(T) * (len << 2));
        memset(lazy, 0, sizeof(T) * (len << 2));
    }
    T& operator[] (int x) { return arr[x]; }
};

int main(){
    return 0;
}
```

- 类版-网络赛

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

using namespace std;


/**
 * 使用说明：
 * 本线段树板子可用于多数 网络赛
 * 需要注意的是此板子在性能方面由于三层封装，固较简洁版的线段树性能慢，但灵活性高（适合网络赛抢时间）
 * 使用时只需要修改 "可修改区的上下界" 中间的代码部分
 * lazy使用结构体封装是应付在某些题目在区间修改时有两种操作，下面给的例题有示例
 * vals是最终题目需要维护的东西（sum，max，min……），针对不同的操作只需修改结构体内的加法（+）操作即可
 * 同时若题目需要区间修改，则直接修改区间修改部分的代码即可，单点修改同理
 * 注意：区间修改针对不同的操作，lazy中的加等于（+=）也要具体根据题意修改
 * 例题1：洛谷
 */
/*****************************可修改区上界*****************************/
template<class T>
struct lazy{
    T lazy1;
    lazy(T lazy1 = 0){
        this->lazy1 = lazy1;
    }
    void operator+= (const lazy<T> &b) {
        this->lazy1 += b.lazy1;//常见操作
    }
    inline void del() {
        this->lazy1 = 0; // 常见操作
    }
};
template<class T>
struct vals{
    T sum;
    //初始化构造，用来方便做运算的初始化-
    vals(int x = 0) {
        sum = x;//常见初始化，方便后面做 + 的操作
    }
    // 操作区
    constexpr vals operator+ (const vals& b){
        vals res;
        res.sum = this->sum + b.sum;//常见操作
        return res;
    }
    //建树初始化区
```

```
    inline void init(const T &v) {
        // this->sum = v; // 常见操作，即和本身一样
    }
    // 区间更新区
    inline void update(const int &l, const int &r, const lazy<T> &v) {
        // this->sum += (r - l + 1) * v.lazy1;// 常见操作
        // printf("%d %d %d %d\n", l, r, v, (r - l + 1) * v); // 常见操作
    }
    // 单点更新区
    inline void update(const lazy<T> &v) {
        // this->sum += v; // 常见操作
    }
    //初始化，通常在多组输入题时才调用
    inline void del() {
        this->sum = 0;
    }
};
/*****************************可修改区下界*****************************/

/*********************************以下区域建议不动（除特殊题目）***********************************/
template<class T>
class segtree{
private:
    const static int maxn = 1e5 + 10;
    const int start = 1;
    struct lazy_tabs{
        lazy<T> v;
        bool flag;
        lazy_tabs (bool flag = false) : flag(flag) {}
        // 懒标记更新
        void update(const lazy<T> &v) { this->v += v, flag = true; }
        // 懒标记删除
        void del() { this->v.del(), flag = false; }
    }lazy_tab[maxn << 2];
    vals<T> val[maxn << 2];
    //计算左右孩子节点下标
    inline static constexpr int ls(const int &indx) { return indx << 1;}
    inline static constexpr int rs(const int &indx) { return indx << 1 | 1;}
    // 上放
    inline void push_up(const int &pos) { val[pos] = val[ls(pos)] + val[rs(pos)]; }
    // 下放
    void push_down(const int &l, const int &r, const int &pos) {
        if (!lazy_tab[pos].flag) return; // 若没有标记则返回
        const auto &v = lazy_tab[pos].v;
        lazy_tab[ls(pos)].update(v), lazy_tab[rs(pos)].update(v);
        const int mid = (l + r) >> 1;
        val[ls(pos)].update(l, mid, v), val[rs(pos)].update(mid + 1, r, v);  // 更新儿子节点
        lazy_tab[pos].del();
    }
    // 建树
    void build(const int &l, const int &r, const int &pos, const T *a) {
        if (l == r) {
            val[pos].init(a[l]);
            return;
        }
        const int mid = (l + r) >> 1;
        build(l, mid, ls(pos), a), build(mid + 1, r, rs(pos), a);
        push_up(pos);
    }
    // 单点修改
    void update(const int &indx, const int &l, const int &r,
                        const lazy<T> &v, const int &pos) {
        if (l == r) {
            val[pos].update(v);
            return;
        }
        const int mid = (l + r) >> 1;
        if (indx <= mid) update(indx, l, mid, ls(pos));
        else update(indx, mid + 1, r, rs(pos));
        push_up(pos);
    }
    //区间修改
    void update(const int &ql, const int &qr, const lazy<T> &v,
                        const int &l, const int &r, const int &pos) {
        if (ql <= l && r <= qr) { // 如果在查询区间内
            lazy_tab[pos].update(v);    //懒更新
            val[pos].update(l, r, v);   // 区间值更新
            return;
        }
        push_down(l, r, pos);
        const int mid = (l + r) >> 1;
        if (ql <= mid) update(ql, qr, v, l, mid, ls(pos));
        if (qr > mid) update(ql, qr, v, mid + 1, r, rs(pos));
        push_up(pos);
    }
    //区间查询
    vals<T> get(const int &ql, const int &qr, const int &l,
```

```
                        const int &r, const int &pos) {
        if (ql <= l && r <= qr) return val[pos];
        push_down(l, r, pos);
        vals<T> res;
        const int mid = (l + r) >> 1;
        if (ql <= mid) res = res + get(ql, qr, l, mid, ls(pos));
        if (qr > mid) res = res + get(ql, qr, mid + 1, r, rs(pos));
        return res;
    }
public:
    /**********************一下均可调用**************************************/
    int n;
    segtree (int n = -1) : n(n) {}
    // 建树
    void build(T *a) { build(start, n, start, a); }
    // 查询
    vals<T> get(int ql, int qr) { return get(ql, qr, start, n, start); }
    //单点修改
    void update(int indx, lazy<T> val) { update(indx, start, n, val, start); }
    //区间修改
    void update(int ql, int qr, lazy<T> val) { update(ql, qr, val, start, n, start); }
    //清空-
    // 切记，先用再清空，因为通常是多组输入时太调用
    void clear() {
        assert(n != -1);
        int len = n << 2;
        assert(n < maxn);
        for (int i = 1; i <= len; ++i) {
            lazy_tab[i].del();
            val[i].del();
        }
    }
};

int main() {
#ifndef ONLINE_JUDGE
    freopen("D:/MYCODE/vsCode-c/test.in", "r", stdin);
    freopen("D:/MYCODE/vsCode-c/test.out", "w", stdout);
#endif
    return 0;
}
```

4. ST表

```
const int M = 1e5 + 5;
int st[M][30], lg[M];
// st表预处理，注意下标从1开始到n结束
void init(int *a, int n) {
    lg[0] = -1;
    for (int i = 1; i <= n; ++i) lg[i] = lg[i >> 1] + 1, st[i][0] = a[i];
    for (int j = 1; j <= lg[n]; ++j) {
        int k = 1 << (j - 1);
        for (int i = 1; i + k - 1 <= n; ++i) {
            st[i][j] = max(st[i][j - 1], st[i + k][j - 1]);
        }
    }
}
// 询问
// 尽可能让l + 2^(len) - 1接近r
int get(int l, int r) {
    int x = lg[r - l + 1];
    return max(st[l][x], st[r - (1 << x) + 1][x]);
}
```

5. 分块

```
const int N = 1e5 + 5, M = 500;
#define ll long long
ll a[N];
int belong[N];
struct blocks {
    int l, r;
    ll lazy;
    blocks() : lazy(0){}
}b[M];
// 以下函数是基本不变的
void build(int n) {
    int siz = sqrt(n), cnt = n / siz;
    if (n % siz) ++cnt;
    for (int i = 1; i <= cnt; ++i) {
        b[i].l = (i - 1) * siz + 1;
        b[i].r = i * siz;
```

```
    }
    b[cnt].r = n;
    for (int i = 1; i <= n; ++i) belong[i] = (i - 1) / siz + 1;
}

```

6. 莫队

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>

using namespace std;

const int M = 1e5 + 10;
int n, m, block, arr[M], pos[M], ans[M], res;
struct MO{
    int l, r, k;
    MO(int l = 0, int r = 0, int k = 0) : l(l), r(r), k(k) {}
}q[M];

bool cmp(MO a, MO b) {
    if (pos[a.l] ^ pos[b.l]) {//不在同一个块
        return  pos[a.l] < pos[b.l];
    }
    if (pos[a.l] & 1) return a.r < b.r;
    return b.r < a.r;
}
void add(int x) {

}
void del(int x) {

}

void solve() {
    int l = 1, r = 0;
    for (int i = 0; i < m; i++) {
        while (l > q[i].l) add(--l);
        while (l < q[i].l) del(l++);
        while (r < q[i].r) add(++r);
        while (r > q[i].r) del(r--);
        ans[q[i].k] = res;//res根据题目意思来
    }
}
void init() {
    scanf("%d %d", &n, &m);
    block = sqrt(n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", arr + i);
        pos[i] = i / block;
    }
    for (int i = 0; i < m; i++) {
        int l, r;
        scanf("%d %d", &l, &r);
        q[i] = MO(l, r, i);
    }
    sort(q, q + m, cmp);
}

int main() {
    init();
    solve();
    return 0;
}
```

7. 平衡树

1. fhq treap

```
// 洛谷板子题
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <random>
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
```

```cpp
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}
using namespace std;
const int N = 4e5 + 10;
mt19937 rnd(233);
struct treap{
    int val, l, r, size, key;
}fhq[N];
int root, cnt;
inline void update(int now) {
    fhq[now].size = fhq[fhq[now].l].size + fhq[fhq[now].r].size + 1;
}

int new_node(int val) {
    fhq[++cnt] = {.val = val, .l = 0, .r = 0, .size = 1, .key = rnd()};
    return cnt;
}
void split(int now, int val, int &x, int &y) {
    if (!now) { x = y = 0; return; }
    if (fhq[now].val <= val) x = now, split(fhq[now].r, val, fhq[now].r, y);
    else y = now, split(fhq[now].l, val, x, fhq[now].l);
    update(now);
}

int merge(int x, int y) {
    if (!x || !y) return x | y;
    // 大根堆
    if (fhq[x].key > fhq[y].key) { //右下角
        fhq[x].r = merge(fhq[x].r, y), update(x);
        return x;
    }
    // 左下角
    fhq[y].l = merge(x, fhq[y].l), update(y);
    return y;
}
// 插入
inline void insert(int val) {
    int x, y;
    split(root, val, x, y);
    root = merge(merge(x, new_node(val)), y);
}
// 按值删除
inline void del(int val) {
    int x, y, z;
    split(root, val, x, z);
    split(x, val - 1, x, y);
    y = merge(fhq[y].l, fhq[y].r);
    root = merge(merge(x, y), z);
}
// 按值获取排名
inline int getrank(int val) {
    int x, y, ans;
    split(root, val - 1, x, y);
    ans = fhq[x].size + 1;
    root = merge(x, y);
    return ans;
}
// 按排名获取值
inline int getval(int rank) {
    int now = root;
    while (now) {
        if (fhq[fhq[now].l].size + 1 == rank) break;
        else if (fhq[fhq[now].l].size >= rank) now = fhq[now].l;
        else rank -= fhq[fhq[now].l].size + 1, now = fhq[now].r;
    }
    return fhq[now].val;
}
// 求前驱，即严格比val小的最大值
inline int pre(int val) {
    int x, y;
    split(root, val - 1, x, y);
    int now = x;
    while (fhq[now].r) now = fhq[now].r;
    root = merge(x, y);
    return fhq[now].val;
}
// 求后继，即严格比val大的最小值
inline int nxt(int val) {
    int x, y;
    split(root, val, x, y);
```

```
    int now = y;
    while (fhq[now].l) now = fhq[now].l;
    root = merge(x, y);
    return fhq[now].val;
}

int main() {
    int t = IO();
    while (t--) {
        int q = IO(), val = IO();
        if (q == 1) insert(val);
        else if (q == 2) del(val);
        else if (q == 3) printf("%d\n", getrank(val));
        else if (q == 4) printf("%d\n", getval(val));
        else if (q == 5) printf("%d\n", pre(val));
        else printf("%d\n", nxt(val));
    }
    return 0;
}
```

2. spaly
3. 替罪羊

8. 左偏树

9. 主席树

1. 主席树（静态）洛谷模板题

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}
using namespace std;

/*************************************离散化*********************************************/
// vt存放可用于查询原本的数（用离散化值），打表后用于查询离散化表（用下标）
vector<int> vt;
inline int get_id(const int &x) { return lower_bound(vt.begin(), vt.end(), x) - vt.begin() + 1; }
inline void erase_vt() {
    sort(vt.begin(), vt.end());
    vt.erase(unique(vt.begin(), vt.end()), vt.end());
}
// 打表，注意，原数组下标要从1开始，返回离散化后的表大小
inline int id_table(int n, int *a, vector<int> &res) {
    res.emplace_back(0);
    for (int i = 1; i <= n; ++i) res.emplace_back(get_id(a[i]));
    return vt.size();
}

/*************************************主席树*********************************************/
const int N = 2e5 + 5;
struct nodes{
    int l, r, sum;
    nodes() : sum(0) {}
}hjt[N << 5];
int root[N], cnt; // 记录每个根结点的内存池编号， 内存池
int build(int l, int r) {
    int now = ++cnt; // 内存申请
    if (l < r) {
        int mid = (l + r) >> 1;
        hjt[now].l = build(l, mid);
        hjt[now].r = build(mid + 1, r);
    }
    return now;
}
// 插入新节点的操作
int update(int pre, int l, int r, int x) {
    int now = ++cnt; // 内存申请
```

```
    hjt[now] = hjt[pre], ++hjt[now].sum; // 继承
    if (l < r) { // 寻找拼接点
        int mid = (l + r) >> 1;
        if (x <= mid) hjt[now].l = update(hjt[now].l, l, mid, x); // 如果x在左边，则让当前新节点的左孩子接继承后的左孩子
        else hjt[now].r = update(hjt[now].r, mid + 1, r, x); // 否则同理
    }
    return now;
}
// 返回第qr版本的主席树 - 第ql版本的主席树， 注意返回的是离散化后的值
int get(int ql, int qr, int l, int r, int k) {
    if (l == r) return l;
    int mid = (l + r) >> 1;
    int dif = hjt[hjt[qr].l].sum - hjt[hjt[ql].l].sum;
    if (k <= dif) return get(hjt[ql].l, hjt[qr].l, l, mid, k); // 左孩子上
    return get(hjt[ql].r, hjt[qr].r, mid + 1, r, k - dif); // 右孩子上
}

/************************************主函数*****************************************/
int a[N];
int main() {
    int n = IO(), m = IO();
    for (int i = 1; i <= n; ++i) a[i] = IO(), vt.emplace_back(a[i]);
    erase_vt();
    vector<int> id;
    int siz = id_table(n, a, id);
    root[0] = build(1, siz);
    for (int i = 1; i <= n; ++i) root[i] = update(root[i - 1], 1, siz, id[i]);
    while (m--) {
        int l = IO(), r = IO(), k = IO();
        printf("%d\n", vt[get(root[l - 1], root[r], 1, siz, k) - 1]);
    }
    return 0;
}
```

10. LCA

```
// 洛谷板子题
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cctype>
#define ll long long
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}
using namespace std;
const int maxn = 5e5 + 5, maxm = 5e5 + 5;
const int INF = 0x3f3f3f3f;

int head[maxn], cnt;

struct edges {
    int to, next;
    void add(int t, int n) {
        to = t, next = n;
    }
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v) {
    edge[++cnt].add(v, head[u]);
    head[u] = cnt;
}

int fa[maxn][35], dep[maxn], lg[maxn];

void dfs(int u, int f) {
    fa[u][0] = f;
    dep[u] = dep[f] + 1;
    for (int i = 1; i <= lg[dep[u]]; ++i) fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v ^ f) dfs(v, u);
```

```
    }
}
void init(int root, int n) {
    dep[root] = lg[0] = -1;
    memset(head, -1, sizeof head); cnt = 0;
    for (int i = 1; i <= n; ++i) lg[i] = lg[i >> 1] + 1;
}
int lca(int a, int b) {
    if (dep[a] < dep[b]) swap(a, b);
    while (dep[a] > dep[b]) a = fa[a][lg[dep[a] - dep[b]]];
    if (a == b) return a;
    for (int i = lg[dep[a]]; ~i; --i) {
        if (fa[a][i] != fa[b][i]) a = fa[a][i], b = fa[b][i];
    }
    return fa[a][0];
}
int main() {

    int n = IO(), m = IO(), root = IO();
    init(root, n);
    for (int i = 1; i < n; ++i) {
        int u = IO(), v = IO();
        add(u, v), add(v, u);
    }
    dfs(root, 0);
    while (m--) {
        int a = IO(), b = IO();
        printf("%d\n", lca(a, b));
    }
    return 0;
}
```

11. 树链剖分

```
// 洛谷板子题
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <iostream>
#include <cmath>
#include <bitset>
using namespace std;
#define ll long long

const int N = 1e5 + 5, M = 2e5 + 5;
const int maxn = 1e5 + 5, maxm = 2e5 + 5;
const int INF = 0x3f3f3f3f;

int head[maxn], cnt;

//初始化
void init() { memset(head, -1, sizeof head); cnt = -1; }

struct edges {
    int to, next;
    void add(int t, int n) {
        to = t, next = n;
    }
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v) {
    edge[++cnt].add(v, head[u]);
    head[u] = cnt;
}

/*******************************树链剖分*********************************/
int fa[N], dep[N], siz[N], son[N];
void dfs1(int u, int f) {
    fa[u] = f, siz[u] = 1;
    dep[u] = dep[f] + 1;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == f) continue;
        dfs1(v, u);
        siz[u] += siz[v];
        if (siz[v] > siz[son[u]]) son[u] = v; // 找重儿子
    }
}
int v[N]; // 点上的权值
int tim, dfn[N], top[N], w[N]; // w的下标是时间戳，对应的是相应时间戳上的点的点权
void dfs2(int u, int t) {
    dfn[u] = ++tim, top[u] = t;
    w[tim] = v[u];
```

```cpp
    if (!son[u]) return;
    dfs2(son[u], t);
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa[u] || v == son[u]) continue;
        dfs2(v, v);
    }
}
/*****************************线段树*****************************/
inline int ls(const int& x) { return x << 1;}
inline int rs(const int& x) { return x << 1 | 1;}
ll seg[N << 2], lazy[N << 2], p;
int n, m;
inline ll op(const ll& a, const ll& b) {
    // seg[x] = max(seg[ls(x)], seg[rs(x)]);
    return (a + b) % p;
}
inline void push_down(const int& l, const int& r, const int& node) {
    if (!lazy[node]) return;
    lazy[ls(node)] += lazy[node], lazy[rs(node)] += lazy[node];
    lazy[ls(node)] %= p, lazy[rs(node)] %= p;
    int mid = (l + r) >> 1;
    seg[ls(node)] = (lazy[node] * (mid - l + 1) + seg[ls(node)]) % p;
    seg[rs(node)] = (lazy[node] * (r - mid) + seg[rs(node)]) % p;
    lazy[node] = 0;
}

void build(int l, int r, int node = 1) {
    if (l == r) {
        seg[node] = w[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, ls(node)), build(mid + 1, r, rs(node));
    seg[node] = op(seg[ls(node)], seg[rs(node)]);
}

void update(int ql, int qr, ll x, int l = 1, int r = n, int node = 1) {
    if (ql <= l && r <= qr) {
        lazy[node] = (lazy[node] + x) % p;
        seg[node] = (seg[node] + (r - l + 1) * x) % p;
        return;
    }
    push_down(l, r, node);
    int mid = (l + r) >> 1;
    if (ql <= mid) update(ql, qr, x, l, mid, ls(node));
    if (qr > mid) update(ql, qr, x, mid + 1, r, rs(node));
    seg[node] = op(seg[ls(node)], seg[rs(node)]);
}

int get(int ql, int qr, int l = 1, int r = n, int node = 1) {
    if (ql <= l && r <= qr) return seg[node];
    push_down(l, r, node);
    int mid = (l + r) >> 1, res = 0;
    if (ql <= mid) res = get(ql, qr, l, mid, ls(node));
    if (qr > mid) res = op(res, get(ql, qr, mid + 1, r, rs(node)));
    return res;
}
/*****************************树上操作*****************************/
void update_chain(int x, int y, ll z) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        update(dfn[top[x]], dfn[x], z);
        x = fa[top[x]];
    }
    if (dep[x] > dep[y]) swap(x, y);
    update(dfn[x], dfn[y], z);
}

ll get_chain(int x, int y) {
    int res = 0;
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        res = op(res, get(dfn[top[x]], dfn[x]));
        x = fa[top[x]];
    }
    if (dep[x] > dep[y]) swap(x, y);
    return op(res, get(dfn[x], dfn[y]));
}

void update_son(int x, ll z) {
    update(dfn[x], dfn[x] + siz[x] - 1, z);
}
ll get_son(int x) {
    return get(dfn[x], dfn[x] + siz[x] - 1);
}
/*****************************主函数*****************************/
```

```cpp

int main() {
    std::ios::sync_with_stdio(false);
    cout.tie(0), cin.tie(0);
    init();
    int root;
    cin >> n >> m >> root >> p;
    for (int i = 1; i <= n; ++i) cin >> v[i];
    for (int i = 1; i < n; ++i) {
        int u, v;
        cin >> u >> v;
        add(u, v), add(v, u);
    }
    dfs1(root, root), dfs2(root, root);
    build(1, n);
    while (m--) {
        int q, x, y, z;
        cin >> q >> x;
        if (q == 1) {
            cin >> y >> z;
            update_chain(x, y, z % p);
        } else if (q == 2) {
            cin >> y;
            cout << get_chain(x, y) << endl;
        } else if (q == 3) {
            cin >> z;
            update_son(x, z);
        } else {
            cout << get_son(x) << endl;
        }
    }
    return 0;
}
```

四、图论

前置存图

```cpp
const int maxn = 1e5, maxm = 2e5;
const int INF = 0x3f3f3f3f;

int head[maxn], cnt;

//初始化
void init() { memset(head, -1, sizeof head); cnt = -1; }

struct edges {
    int to, next;
    int w;
    void add(int t, int n, int w) {
        to = t, next = n, this->w = w;
    }
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v, int w) {
    edge[++cnt].add(v, head[u], w);
    head[u] = cnt;
}

```

1. 最短路

1. dijkstra

```cpp
//顶点数和边数
const int maxn = 1e5, maxm = 2e5;
const int INF = 0x3f3f3f3f;

int head[maxn], cnt, dis[maxn];
bool vis[maxn];
//初始化
void init() {
    memset(head, -1, sizeof head);
    memset(vis, false, sizeof vis);
    cnt = 0;
}

struct edges {
    int to, next;
    int w;
    edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
}edge[maxm << 1]; //无向图则需要乘2
```

```cpp

inline void add_edges(int u, int v, int w) {
    edge[++cnt] = edges(v, head[u], w);
    head[u] = cnt;
}

struct qnode{
    int v;
    int w;
    qnode(int v = 0, int w = 0) : v(v), w(w) {}
    bool operator< (const qnode &t) const { return w > t.w; }
};

void dijkstra(int n, int s) {//n 为顶点数， m 为边数
    for (int i = 0; i <= n; ++i) dis[i] = INF;
    dis[s] = 0;
    priority_queue<qnode> heap;
    heap.push(qnode(s, dis[s]));
    while (heap.size()) {
        int u = heap.top().v;
        heap.pop();
        if (vis[u]) continue;
        vis[u] = true;
        for (int i = head[u]; ~i; i = edge[i].next) {
            int v = edge[i].to;
            int w = edge[i].w;
            if (!vis[v] && dis[u] + w < dis[v]) { //松弛
                dis[v] = dis[u] + w;
                heap.push(qnode(v, dis[v]));
            }
        }
    }
}
```

2. bellman-ford

```cpp
const int maxn = 1e5, maxm = 2e5;
int n, m, s; // n 为顶点数， m 为边数
int dis[maxn];

struct edges {
    int u, v, w;
    edges(int u = 0, int v = 0, int w = 0) : u(u), v(v), w(w) {}
} edge[maxm];

bool bf() {
    for (int k = 1; k < n; k++) {
        for (int i = 1; i < n; i++) {
            if (dis[edge[i].v] > dis[edge[i].u] + edge[i].w) { //松弛
                dis[edge[i].v] = dis[edge[i].u] + edge[i].w;
            }
        }
    }
    for (int i = 1; i < n; i++) {
        if (dis[edge[i].v] > dis[edge[i].u] + edge[i].w) {
            return false;
        }
    }

    return true;
}
```

3. spfa

```cpp
const int maxn = 1e5, maxm = 2e5;
int n, m, s, dis[maxn], num[maxn], head[maxn], cnt; //num 数组时判断是否有负环
bool inq[maxn];

void init() {
    memset(inq, false, sizeof inq);
    memset(dis, 0x3f, sizeof dis);
    memset(num, 0, sizeof num);
    memset(head, -1, sizeof head);
    cnt = 0;
}

struct edges {
    int to, w, next;
    edges(int to = 0, int w = 0, int next = -1) : to(to), w(w), next(next) {}
} edge[maxm];

inline void add_edges(int u, int v, int w) {
    edge[++cnt] = edges(v, w, head[u]);
```

```
    head[u] = cnt;
}

bool spfa() {
    queue<int> q;
    q.push(s);
    inq[s] = true;
    num[s]++;
    while (q.size()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int i = head[u]; ~i; i = edge[i].next) {
            int v = edge[i].to, w = edge[i].next;
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                if (!inq[v]) {
                    q.push(v);
                    inq[v] = true;
                    num[v]++;
                    if (num[v] >= n) return false;
                    //如果从1号点到x的最短路中包含至少n个点（不包括自己），则存在环
                }
            }
        }
    }
    return true;
}
```

4. floyd

```
const int M = 2e2;
int n, m; //顶点数和边数
int dis[M][M];

void floyd() {
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dis[i][j] > dis[i][k] + dis[k][j]) {
                    dis[i][j] = dis[i][k] + dis[k][j];
                }
            }
        }
    }
}

void init() {
    memset(dis, 0x3f, sizeof dis);
    for (int i = 0; i < M; ++i) dis[i][i] = 0;
}
```

2. 生成树

1. kruskal 适合稀疏图

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#define ll long long
using namespace std;
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}
const int M = 2e5 + 10, N = 5e5 + 5;
int fa[M];
struct edges { int u, v; ll w; } e[N];

bool cmp(edges& i, edges& j) { return i.w < j.w; }
int findset(int x) { return x == fa[x] ? x : fa[x] = findset(fa[x]); }
```

```cpp

bool un(int a, int b) {
    int fa1 = findset(a), fa2 = findset(b);
    if (fa1 == fa2) return false;
    fa[fa1] = fa2;
    return true;
}

ll kruskal(int n, int m) {
    sort(e, e + m, cmp);
    for (int i = 0; i <= n; ++i) fa[i] = i;
    int cnt = 0;
    ll ans = 0;
    for (int i = 0; i < m; ++i) {
        if (un(e[i].u, e[i].v)) {
            ans += e[i].w;
            if (++cnt == n - 1) break;
        }
    }
    return n - 1 == cnt ? ans : -1;
}

int main() {
    int n = IO(), m = IO();
    for (int i = 0; i < m; ++i) e[i].u = IO(), e[i].v = IO(), e[i].w = IO();
    printf("%lld\n", kruskal(n, m));
    return 0;
}
```

3. tarjan

4. 网络流

1. Edmonds-Karp算法，速度较慢

```cpp
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;
#define ll long long
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}

const int maxn = 1e5, maxm = 2e5;
const int INF = 0x3f3f3f3f;
const ll inf = 0xfffffffffff;

int head[maxn], cnt;

//初始化
void init() { memset(head, -1, sizeof head); cnt = -1; }

struct edges {
    int to, next;
    ll c;
    edges(int to = 0, int next = -1, int c = 0) : to(to), next(next), c(c) {}
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v, ll c, bool f = 1) {
    edge[++cnt] = edges(v, head[u], c);
    head[u] = cnt;
    if (f) add(v, u, 0, 0); // 建立反向弧
}
#include <queue>
bool vis[maxn]; // 记录是否在队内
ll minc[maxn]; // 记录增广路的最小流
struct pairs { int u, i; } pre[maxn];
bool bfs(int s, int t, int n) {
    queue<int> q;
```

```
    for (int i = 0; i <= n; ++i) vis[i] = false;
    q.push(s), vis[s] = true, minc[s] = inf;
    while (q.size()) {
        int u = q.front(); q.pop();
        for (int i = head[u]; ~i; i = edge[i].next) {
            int v = edge[i].to;
            if (vis[v] || !edge[i].c) continue;
            vis[v] = true, pre[v] = {.u = u, .i = i}; //记录当前点的前驱点和当前点的内存池编号
            minc[v] = min(minc[u], edge[i].c);
            if (v == t) return true;
            q.push(v);
        }
    }
    return false;
}


ll EK(int s, int t, int n) {
    ll ans = 0, &dif = minc[t];
    while (bfs(s, t, n)) {
        ans += dif;
        for (int i = t; i != s; i = pre[i].u) {
            edge[pre[i].i].c -= dif;
            edge[pre[i].i ^ 1].c += dif;
        }
    }
    return ans;
}

int main() {
    int n = IO(), m = IO(), s = IO(), t = IO();
    init();
    for (int i = 0; i < m; ++i) {
        int u = IO(), v = IO();
        ll c = IO();
        add(u, v, c);
    }
    printf("%lld", EK(s, t, n));
    return 0;
}

```

2. dinic, 当前弧优化+多路增广优化+炸点优化(模板题),复杂度$O(n^2m)$

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}
#include <iostream>
#include <string>
#include <queue>
#include <cmath>
#define ll long long
const int N = 1e5 + 5, M = 1e6 + 5;
const ll inf = 0xfffffffffff;

const int maxn = 1500, maxm = 1e5 + 2e4 + 5;

int head[maxn], cnt;

//初始化
inline void init() {
    memset(head, -1, sizeof head);
    cnt = -1;
}

struct edges {
    int to, next;
```

```
    ll c; //容量
    edges(int to = 0, int next = -1, ll c = 0) : to(to), next(next), c(c) {}
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v, ll c, bool f = 1) {
    edge[++cnt] = edges(v, head[u], c), head[u] = cnt;
    if (f) add(v, u, 0, 0); // 添加反向弧
}
#include <queue>
int deep[maxn], cur[maxn];
// bfs求增广路，一次求出多条增广路
bool bfs(int s, int t, int n) {
    queue<int> q;
    for (int i = 1; i <= n; ++i) deep[i] = 0;
    deep[s] = 1;
    q.push(s);
    while (q.size()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; ~i; i = edge[i].next) {
            const int &v = edge[i].to;
            const ll &c = edge[i].c;
            if (deep[v] || !c) continue;
            deep[v] = deep[u] + 1;
            q.push(v);
        }
    }
    return deep[t] != 0;
}

ll dfs(int u, int t, ll f) {
    if (u == t) return f;
    ll nowflow = 0;
    for (int i = cur[u]; ~i; i = edge[i].next) {
        cur[u] = i; // 当前弧优化
        int &v = edge[i].to;
        ll &c = edge[i].c;
        if (deep[v] != deep[u] + 1 || !c) continue;
        if (ll low = dfs(v, t, min(f - nowflow, c))) {
            c -= low, edge[i ^ 1].c += low;
            nowflow += low;// 多路增广优化
            if (nowflow == f) break;
        }
    }
    if (!nowflow) deep[u] = -2; // 炸点优化
    return nowflow;
}

ll dinic(int s, int t, int n) {
    ll ans = 0;
    while (bfs(s, t, n)) {
        for (int i = 1; i <= n; ++i) cur[i] = head[i]; // 预处理，方便当前弧优化
        ans += dfs(s, t, inf); // 进过多路增广优化可不用循环
    }
    return ans;
}


int main() {
    int n = IO(), m = IO(), s = IO(), t = IO();
    init();
    for (int i = 0; i < m; ++i) {
        int u = IO(), v = IO();
        ll c = IO();
        add(u, v, c);
    }
    printf("%lld", dinic(s, t, n));
    return 0;
}
```

3. 最小费用最大流，将ek算法中的bfs换成spfa

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;
#define ll long long
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
```

```
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}

const int maxn = 1e5, maxm = 2e5;
const int INF = 0x3f3f3f3f;
const ll inf = 0xfffffffffff;

int head[maxn], cnt;

//初始化
void init() { memset(head, -1, sizeof head); cnt = -1; }

struct edges {
    int to, next;
    ll c, w;
    edges(int to = 0, int next = -1, ll c = 0, ll w = 0) :
        to(to), next(next), c(c), w(w) {}
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v, ll c, ll w, bool f = 1) {
    edge[++cnt] = edges(v, head[u], c, w);
    head[u] = cnt;
    if (f) add(v, u, 0, -w, 0); // 建立反向弧，费用相反
}
#include <queue>
bool inq[maxn];
ll dist[maxn];
struct pairs { int u, i; } pre[maxn];
// 利用spaf找最小费用的路，即最短路
bool spfa(int s, int t, int n) {
    for (int i = 0; i <= n; ++i) inq[i] = false, dist[i] = inf;
    queue<int> q;
    dist[s] = 0, inq[s] = true, q.push(s), pre[t].i = -1;
    while (q.size()) {
        int u = q.front(); q.pop();
        inq[u] = false;
        for (int i = head[u]; ~i; i = edge[i].next) {
            int v = edge[i].to;
            if (dist[v] > dist[u] + edge[i].w && edge[i].c) {
                pre[v] = {.u = u, .i = i}; //记录当前点的前驱点和当前点的内存池编号
                dist[v] = dist[u] + edge[i].w;
                if (inq[v]) continue;
                inq[v] = true, q.push(v);
            }
        }
    }
    return pre[t].i != -1; // 如果说t没有前驱则说明找不到增广路了
}

void mcmf(int s, int t, int n, ll &maxf, ll &minv) {
    maxf = minv = 0;
    while (spfa(s, t, n)) {
        ll low = inf;
        for (int i = t; i != s; i = pre[i].u) low = min(low, edge[pre[i].i].c);//寻找最小流
        for (int i = t; i != s; i = pre[i].u) {
            edge[pre[i].i].c -= low;
            edge[pre[i].i ^ 1].c += low;
            minv += low * edge[pre[i].i].w;
        }
        maxf += low;
    }
}

int main() {
    int n = IO(), m = IO(), s = IO(), t = IO();
    init();
    for (int i = 0; i < m; ++i) {
        int u = IO(), v = IO();
        ll c = IO(), w = IO();
        add(u, v, c, w);
    }
    ll maxflow, mincost;
    mcmf(s, t, n, maxflow, mincost);
    printf("%lld %lld", maxflow, mincost);
    return 0;
}
```

5. 二分图

1. 匈牙利算法，时间复杂度$O(ev)$

```
int match[M];
bool vis[M];
bool dfs(int u) {
    for (int &v : gp[u]) {
        if (vis[v]) continue;
        vis[v] = true;
        if (!match[v] || dfs(match[v])) {
            match[u] = v, match[v] = u;
            return true;
        }
    }
    return false;
}
// 主函数里
fill_n(match, n + 1, 0); // n是点的个数
for (int i = 1; i <= n; ++i) {
    if (match[i]) continue;
    fill_n(vis, n + 1, false);
    dfs(i);
}
```

1.

五、字符串

1. KMP

```
const int M = 1e6 + 5;
//普通版本
void getnext(char *x, int len, int *nxt) {
    int i = 0, j;
    j = nxt[0] = -1;
    while (i < len) {
        while(j != -1 && x[i] != x[j]) j = nxt[j];
        nxt[++i] = ++j;
    }
}
//略微优化版本
void getNext(char *x, int len, int *nxt) {
    int i = 0, j;
    j = nxt[0] = -1;
    while (i < len) {
        while (j != -1 && x[i] != x[j]) j = nxt[j];
        if (x[++i] == x[++j]) nxt[i] = nxt[j];
        else nxt[i] = j;
    }
}
// y是主串
int nxt[M];
int kmpCount(char *y, int n, char *x, int m) {
    int i = 0, j = 0, ans = 0;
    getnext(x, m, nxt);
    while (i < n) {
        while (j != -1 && y[i] != x[j]) j = nxt[j];
        ++i, ++j;
        if (j >= m) ++ans, j = nxt[j];
    }
    return ans;
}
```

2. 字符串Hash

```
unsigned int DJBHash(const char *str) {
    unsigned int hash = 5381;
    while (*str) hash += (hash << 5) + (*str++);
    return (hash & 0x7fffffff); //7个f
}

unsigned int BKDRHash(const char *str) {
    unsigned int seed = 131; // 31 131 1313 13131 131313...
    unsigned int hash = 0;
    while (*str) hash = hash * seed + (*str++);
    return (hash & 0x7fffffff);
}
```

```
#define ull unsigned long long
ull strhash(const char *s) {
    ull seed = 1313, res = 0; // 31 131 1313 13131
    while (*s) res = res * seed + (*s++);
    return res;
}

```

3. 马拉车

4. exkmp

```
void pre_exkmp(char x[], int m, int next[]) {
    next[0] = m;
    int j = 0;
    while (j + 1 < m && x[j] == x[j + 1]) j++;
    next[1] = j;
    int k = 1;
    for (int i = 2; i < m; i++) {
        int p = next[k] + k - 1;
        int L = next[i - k];
        if (i + L < p + 1) next[i] = L;
        else {
            j = max(0, p - i + 1);
            while (i + j < m && x[i + j] == x[j]) j++;
            next[i] = j;
            k = i;
        }
    }
}
void exkmp(char x[], int m, char y[], int n, int next[], int extend[]) {
    pre_exkmp(x, m, next);
    int j = 0;
    while (j < n && j < m && x[j] == y[j]) j++;
    extend[0] = j;
    int k = 0;
    for (int i = 1; i < n; i++) {
        int p = extend[k] + k - 1;
        int L = next[i - k];
        if (i + L < p + 1) extend[i] = L;
        else {
            j = max(0, p - i + 1);
            while (i + j < n && j < m && y[i + j] == x[j]) j++;
            extend[i] = j;
            k = i;
        }
    }
}
```

六、 计算几何

- 未完善

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <vector>

/**
 * 本板子属于半成品，有些功能并没有验证
 * 函数说明：
 *          关于点的函数
 * 点的Point(double, double) 构造函数
 * + 向量加法
 * - 向量减法
 * == 判断两个点是否相等
 * *(Point) 向量点乘
 * *(double) 向量伸长(没有除法，要用除法直接乘倒数)
 * ^ 向量叉乘
 * < 点对点的比较
 * double len() 向量的长度，也可以用来求两个点的距离
 * Point rotate(double angle)  向量逆时针旋转angle弧度
 * Point rotate(Point, double)  点让点p逆时针旋转angle弧度
 * void print() 将点输出
 * int init() 输入点的坐标 返回值和scanf相同
 *          其他非结构体函数
 * angle(Point&, Point&) 计算两个向量的夹角
 *
 *          关于线的函数
 * Line(Point, Point) 构造函数
 * Line(Point, double) 根据一个点和一个倾斜角 0 <= angle < PI确定直线 （未验证）
```

```
 * double len() 返回线段的长度
 * double point(double t)  返回距离点p向前t倍向量的点
 * double angle() 返回直线的倾斜角 范围[0, PI) (未验证)
 * double disPointToLine(const Point&) 点到这条直线的距离
 * double disPointToSeg(const Point&) 点到这条线段的距离（未验证）
 * Point getPro(const Point&) 点在这条线上的投影 （未验证）
 * Point getSym(const Point&) 点关于这条线的对称点 （未验证）
 * bool isOnLine(const Point&) 验证该点是否在这条直线上（未验证）
 * bool isOnSeg(const Point&) 验证该点是否在这条线段上（未验证）
 * Point cross(Line&)  直线和这条直线的交点，前提是相交
 * void print()  输出这条线段
 *
 *          其他非结构体
 * int LineAndLine(Line&, Line&)  直线和直线的关系 0平行 1重合 2相交 （未验证）
 * Point getLineInter(const Line&, const Line&) 求两直线的交点,必须相交才能调用
 * int SegAndSeg(const Line& l1, const Line& l2) 两个线段的关系 0不相交 1非规范相交(其中一个线段的端点和另一个线段相交)
2规范相交 （未验证）
 * int LineAndSeg(const Line& line, const Line& seg) 直线和线段的关系，0不相交 1非规范相交 2规范相交 （未验证）
 *
 *           关于圆的函数
 * Circle(Point, double) 构造函数
 * Circle(Point, Point, Point) 过三点的圆
 * double area() const; 返回圆的面积
 * double circum() const  返回圆的周长
 * int PointAndCircle(Point&) 点和圆的关系 返回 0圆外 1圆上 2圆内
 * int LineAndCircle(Point&)  点和圆的关系 返回 0不相交 1相交 2相交两个点（未验证）
 * int CircleAndCircle(Circle&) 圆和圆的关系 返回 0内含 1内切 2相交两点 3外切 4外离 （未验证）
 *
 *         关于三角形的函数
 * Triangle(Point, Point, Point) 构造函数
 * double area() const 返回三角形函数
 * Circle outerCircle() 获取三角形的外接圆
 *
 *         关于多边形的函数
 * Polygon(vector<Point>&) 构造函数
 * double circum();  求凸包的周长
 * void graham(Polygon&) 求凸包 传入值为需要求出的凸包的点集
 * int PointAndPolgon(Point&) 判断点与多边形的关系，0外 1内 2边上 3点上 （未实现）
 * double minRectCover() 点集的最小矩形覆盖，自己必须是（逆时针）凸包才能调用 （未实现）
 * Circle minCircleCover() 点集的最小圆覆盖
*/

using namespace std;
const double eps = 1e-8, PI = acos(-1.0);
int dcmp(double x) {
    if (fabs(x) < eps) return 0;
    return x > 0 ? 1 : -1;
}
/*****************************点*****************************/
/*
除了结构体内部函数还有
angle(Point& a, Point& b) // 两个向量的夹角
*/
struct Point {
    double x, y;
    Point(double x = 0, double y = 0) : x(x), y(y) {}
    Point operator + (const Point&) const;
    Point operator - (const Point&) const;
    double operator * (const Point&) const;    // 点乘
    double operator ^ (const Point&) const;    // 叉乘
    bool operator == (const Point&) const;
    bool operator < (const Point&) const;  // 排序需要
    Point operator * (double);    //向量伸长b倍
    double len() const; // 向量的长度
    Point rotate(double);    // 向量逆时针旋转a弧度后
    Point rotate(Point&, double);    // 点绕p点顺时针旋转a弧度后
    void print() { printf("%.2f %.2f", x, y); }
    int init() { return scanf("%lf%lf", &x, &y); }
};
Point Point::operator + (const Point& b) const {
    return Point(x + b.x, y + b.y);
}
Point Point::operator - (const Point& b) const {
    return Point(x - b.x, y - b.y);
}
// 点乘
double Point::operator * (const Point& b) const {
    return x * b.x + y * b.y;
}
// 叉乘
double Point::operator ^ (const Point& b) const {
    return x * b.y - y * b.x;
}
bool Point::operator == (const Point& b) const {
    return !dcmp(x - b.x) && !dcmp(y - b.y);
}
bool Point::operator < (const Point& b) const {
```

```
    return (!dcmp(x - b.x)) ? dcmp(y - b.y) < 0 : x < b.x;
}
// 向量的长度
double Point::len() const { return sqrt(x * x + y * y); }
//向量伸长b倍
Point Point::operator * (double b) {
    return Point(x * b, y * b);
}
// 向量逆时针旋转a弧度后
// cosx -sinx
// sinx cosx
Point Point::rotate(double a) {
    return Point(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
}
// 点绕p点顺时针旋转a弧度后
Point Point::rotate(Point &p, double a) {
    Point vec = (*this) - p;
    return vec.rotate(a) + p;
}
// 两个向量的夹角
double angle(Point& a, Point& b) {
    return acos(a * b / a.len() / b.len());
}
/*******************************线*******************************/
/*
除了结构体内部函数还有
int LineAndLine(Line& l1, Line&)  直线和直线的关系 0平行 1重合 2相交
Point getLineInter(const Line& l1, const Line& l2) 求两直线的交点,必须相交才能调用
int SegAndSeg(const Line& l1, const Line& l2) 两个线段的关系 0不相交 1非规范相交(其中一个线段的端点和另一个线段相交) 2规范相交
int LineAndSeg(const Line& line, const Line& seg) 直线和线段的关系，0不相交 1非规范相交 2规范相交
*/
struct Line {
    Point p1, p2, v;
    Line() {}
    Line(Point p1, Point p2) : p1(p1), p2(p2), v(p2 - p1) {}
    Line(Point, double); // 根据一个点和一个倾斜角0<= angle < PI确定直线
    double len(); // 线段的长度
    Point point(double);  // P = p1 + vt
    double angle(); // 直线的倾斜角[0, PI)
    double disPointToLine(const Point&); // 点到这条直线的距离
    double disPointToSeg(const Point&); // 点到这条线段的距离
    Point getPro(const Point&); // 点到这条线的投影
    Point getSym(const Point&); // 点关于这条线的对称点
    bool isOnLine(const Point&); // 点是否在这条直线上
    bool isOnSeg(const Point&); // 点是否在这条线段上
    Point cross(Line&); // 直线和这条直线的交点，前提是相交才能调用
    void print(); // 输出线段
};
Line::Line(Point p, double angle) : p1(p) {
    if (!dcmp(angle - PI / 2)) p2 = p1 + Point(0, 1);
    else p2 = p1 + Point(1, tan(angle));
}
double Line::len() { return v.len(); }
Point Line::point(const double t){ return (p1 + (v * t)); }
double Line::angle() {
    double ret = atan2(p2.y - p1.y, p2.x - p1.x);
    if (dcmp(ret) < 0) ret += PI;
    if (!dcmp(ret - PI)) ret -= PI;
    return ret;
}
void Line::print() {
    printf("(%f,%f)->(%f,%f)", p1.x, p1.y, p2.x, p2.y);
}
double Line::disPointToLine(const Point& p) {
    Point vec = p - p1;
    return fabs(v ^ vec) / v.len();
}
double Line::disPointToSeg(const Point& p) {
    if (p1 == p2) return (p1 - p).len();
    Point v1 = p - p1, v2 = p - p2;
    if (dcmp(v1 * v) < 0) return v1.len();
    if (dcmp(v2 * v) < 0) return v2.len();
    return disPointToLine(p);
}
Point Line::getPro(const Point& p) {
    return p1 + v * (v * (p - p1) / v.len());
}
Point Line::getSym(const Point& p) {
    Point q = getPro(p);
    return Point(2 * q.x - p.x, 2 * q.y - p.y);
}
bool Line::isOnLine(const Point& p) {
    return !dcmp((p - p1) ^ (p - p2));
}
bool Line::isOnSeg(const Point& p) {
    return isOnLine(p) && (dcmp((p - p1) * (p - p2)) <= 0);
```

```
}
Point Line::cross(Line& l) {
    double a1 = l.v ^ (p1 - l.p1);
    double a2 = l.v ^ (p2 - l.p1);
    return Point((p1.x * a2 - p2.x * a1) / (a2 - a1),
    (p1.y *a2 - p2.y * a1) / (a2 - a1));
}
// 直线和直线的关系 0平行 1重合 2相交
int LineAndLine(Line& l1, Line& l2) {
    if (!dcmp(l1.v ^ l2.v)) return l2.isOnLine(l1.p1);
    return 2;
}
// 求两个直线的交点，必须相交才能调用
Point getLineInter(Line& l1, Line& l2) {
    Point vec = l1.p1 - l2.p1;
    double t = (l2.v ^ vec) / (l1.v ^ l2.v);
    return l1.point(t);
}
// 判断两个线段的关系
int SegAndSeg(const Line& l1, const Line& l2) {
    int d1 = dcmp(l1.v ^ (l2.p1 - l1.p1));
    int d2 = dcmp(l1.v ^ (l2.p2 - l1.p1));
    int d3 = dcmp(l2.v ^ (l1.p1 - l2.p1));
    int d4 = dcmp(l2.v ^ (l1.p2 - l2.p1));
    if((d1 ^ d2) == -2 && (d3 ^ d4) == -2) return 2;
    return (!d1 && dcmp((l2.p1 - l1.p1) * (l2.p1 - l1.p2)) <= 0) ||
        (!d2 && dcmp((l2.p2 - l1.p1) * (l2.p2 - l1.p2)) <= 0) ||
        (!d3 && dcmp((l1.p1 - l2.p1) * (l1.p1 - l2.p2)) <= 0) ||
        (!d4 && dcmp((l1.p2 - l2.p1) * (l1.p2 - l2.p2)) <= 0);
}
// 直线和线段的关系，0不相交 1非规范相交 2规范相交
int LineAndSeg(const Line& line, const Line& seg) {
    int d1 = dcmp(line.v ^ (seg.p1 - line.p1));
    int d2 = dcmp(line.v ^ (seg.p2 - line.p1));
    if ((d1 ^ d2) == -2) return 2;
    return d1 == 0 || d2 == 0;
}
/*****************************圆*****************************/
struct Circle {
    Point p;
    double r;
    Circle() {}
    Circle(Point p, double r) : p(p), r(r) {}
    Circle(Point, Point , Point);  // 过三点一个圆
    double area() const; // 面积
    double circum() const; // 周长
    int PointAndCircle(Point&); // 点和圆的关系 0圆外 1圆上 2圆内
    int LineAndCircle(Line&);  // 直线和圆的关系 0不相交 1相切 2相交两点
    int CircleAndCircle(Circle&); // 圆和圆的关系 0内含 1内切 2相交两点 3外切 4外离
};
double Circle::area() const { return PI * r * r; }
double Circle::circum() const { return 2 * PI *r; }
Circle::Circle(Point a, Point b, Point c) {
    Point v1 = b - a, v2 = c - b;
    Line l1((a + b) * 0.5, ((a + b) * 0.5) + Point(-v1.y, v1.x));
    Line l2((b + c) * 0.5, ((b + c) * 0.5) + Point(-v2.y, v2.x));
    p = getLineInter(l1, l2);
    r = (p - a).len();
}
int Circle::PointAndCircle(Point& a) {
    int d = dcmp((a - p).len() - r);
    if (d > 0) return 0;
    if (d < 0) return 2;
    return 1;
}
int Circle::LineAndCircle(Line& l) {
    int d = dcmp(l.disPointToLine(p) - r);
    if (d > 0) return 0;
    if (d < 0) return 2;
    return 1;
}
int Circle::CircleAndCircle(Circle& c) {
    double dist = (c.p - p).len();
    if (dcmp(dist - r - c.r) > 0) return 4;
    if (!dcmp(dist - r - c.r)) return 3;
    double l = fabs(r - c.r);
    if (dcmp(dist - r - c.r) < 0 && dcmp(dist - l) > 0) return 2;
    return dcmp(dist - l) == 0;
}
/*****************************三角形*****************************/
struct Triangle{
    Point p[3];
    Triangle() {}
    Triangle(Point A, Point B, Point C);
    double area() const ;  // 三角形面积
    Circle outerCircle(); // 外接圆
};
```

```cpp
Triangle::Triangle(Point A, Point B, Point C) {
    p[0] = A, p[1] = B, p[2] = C;
}
double Triangle::area() const {
    double ret = 0;
    for (int i = 0; i < 3; ++i) {
        ret += p[i] ^ p[(i + 1) % 3];
    }
    return fabs(ret) / 2;
}
// r = a * b * c / 4S
// 两条边的中垂线的交点，也可用kuangbin的方法，但此方法不需要再次调用Line的函数
Circle Triangle::outerCircle() {
    double A1 = 2.0 * (p[1].x - p[0].x), B1 = 2.0 * (p[1].y - p[0].y);
    double A2 = 2.0 * (p[2].x - p[1].x), B2 = 2.0 * (p[2].y - p[1].y);
    double C1 = 0, C2 = 0;
    for (int i = 0, j = 1, k = -1; i < 2; ++i, ++j, k = 1) {
        C1 += (p[i].x * p[i].x + p[i].y * p[i].y) * k;
        C2 += (p[j].x * p[j].x + p[j].y * p[j].y) * k;
    }
    double x = ((C1 * B2) - C2 * B1) / ((A1 * B2) - A2 * B1);
    double y = ((A1 * C2) - A2 * C1) / ((A1 * B2) - A2 * B1);
    return Circle(Point(x, y), (Point(x, y) - p[0]).len());
}

/***************************多边形/点集*****************************/

struct Polygon {
    vector<Point> p;
    // vector<Line> l;
    Polygon(){}
    Polygon(vector<Point>& p) : p(p) {};
    double circum(); // 求凸包的周长
    void graham(Polygon&); // 凸包
    int PointAndPolygon(Point&); // 判断点与多边形的关系，0外 1内 2边上 3点上
    double minRectCover(); // 最小矩形覆盖，自己必须是（逆时针）凸包才能调用
    Circle minCircleCover(); // 点集的最小圆覆盖
};
double Polygon::circum() {
    int n = p.size();
    if (n < 2) return 0.0;
    if (n == 2) return (p[0] - p[1]).len();
    double ret = 0;
    for (int i = 0; i < n; ++i) {
        ret += (p[(i + 1) % n] - p[i]).len();
    }
    return ret;
}
void Polygon::graham(Polygon& res) {
    int indx = 0, n = p.size();
    for (int i = 1; i < n; ++i) if (p[i] < p[indx]) indx = i;
    swap(p[0], p[indx]);
    sort(p.begin() + 1, p.end(), [&](Point& i, Point& j) {
        int d = dcmp(atan2(i.y - p[0].y, i.x - p[0].x) - atan2(j.y - p[0].y, j.x - p[0].x));
        if (d) return d < 0;
        return i.x < j.x;
    });
    res.p.emplace_back(p[0]);
    if (n == 1) return;
    res.p.emplace_back(p[1]);
    if (n == 2) {
        if (p[0] == p[1]) res.p.pop_back();
        return;
    }
    int x = res.p.size();
    for (int i = 2; i < n; ++i, ++x) {
        while (x >= 2 && dcmp((res.p[x - 2] - res.p[x - 1]) ^ (res.p[x - 2] - p[i])) <= 0) {
            --x;
            res.p.pop_back();
        }
        res.p.emplace_back(p[i]);
    }
    if (res.p.size() == 2 && (res.p[0] == res.p[1])) res.p.pop_back();
}
Circle Polygon::minCircleCover() {
    random_shuffle(p.begin(), p.end());
    Circle res(p[0], 0);
    int n = p.size();
    for (int i = 1; i < n; ++i) {
        if (res.PointAndCircle(p[i])) continue; // 在圆内
        res = Circle(p[i], 0);
        for (int j = 0; j < i; ++j) {
            if (res.PointAndCircle(p[j])) continue; // 在圆内
            res = Circle((p[i] + p[j]) * 0.5, (p[i] - p[j]).len() * 0.5);
            for (int k = 0; k < j; ++k) {
                if (res.PointAndCircle(p[k])) continue; // 在圆内
                res = Circle(p[i], p[j], p[k]);
```

```
            }
        }
    }
    return res;
}
vector<Point> vt;
int main() {
    int t;
    while(scanf("%d", &t), t) {
        Point p;
        for (int i = 0; i < t; ++i) {
            p.init();
            vt.emplace_back(p);
        }
        Polygon pol(vt), ans;
        pol.graham(ans);
        printf("%.2f\n", ans.circum());
        vt.clear();
    }
    return 0;
}
```

七、动态规划

1. 树形dp

1. 树的最大独立集

```
/*
Loj 10160
每个点都有一个快乐值，子结点和父节点不能同时被选，问你最大的快乐值
dp[i][0]表示第i号结点不选时最大的快乐值
dp[i][1]表示第i号结点选时的最大的快乐值
*/
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cctype>
#include <cstring>
using namespace std;
const int M = 6e3 + 5;
inline int read() {
    int x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - 48);
        c = getchar();
    }
    return f ? -x : x;
}

struct es{
    int to, nxt;
}e[M << 1];
int head[M], cnt;
inline void init() { memset(head, -1, sizeof head); cnt = 0; }
inline void add(int u, int v) {
    e[++cnt] = {.to = v, .nxt = head[u]};
    head[u] = cnt;
}

int n, h[M], vis[M], dp[M][2];

void dfs(int u) {
    dp[u][1] = h[u];
    for (int i = head[u]; ~i; i = e[i].nxt) {
        int v = e[i].to;
        dfs(v);
        dp[u][0] += max(dp[v][0], dp[v][1]);
        dp[u][1] += dp[v][0];
    }

}

int main() {
    int n = read();
    for (int i = 1; i <= n; ++i) h[i] = read();
```

```
    int u = read(), v = read();
    init();
    while (u | v) {
        add(v, u);
        vis[u] = true;
        u = read(), v = read();
    }
    int root = 0;
    for (int i = 1; i <= n; ++i) {
        if (!vis[i]) root = i;
    }
    dfs(root);
    printf("%d\n", max(dp[root][0], dp[root][1]));
    return 0;
}
```

2. 树的最小支配集

```
/*
Loj 10157
每个点都有点权，一个点可以看守连着他的边上的点，选一些点出来，使他们能够看守整颗树上所有的点
问你最小选出来的权值
dp[0][i]表示i点被选上，则其 += min({dp[0][son], dp[2][son], dp[1][son]})
dp[1][i]表示i点没被选上，但是其父亲被选上了，则其 += min(dp[0][son], dp[2][son])
dp[2][i]表示i点没被选上，但是去其中某几个儿子被选上了，注意这个比较难转移，转移方式如下
先求出所有儿子min(dp[0][son], dp[1][son])的总和，然后在递归完后选出最小是那个儿子的dp[0][son]
即dp[2][u] = min(dp[2][u], sum - min(dp[2][v], dp[0][v]) + dp[0][v]);这行
*/
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;
#define ll long long
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}

const int maxn = 1e5, maxm = 1e5;
const int INF = 0x3f3f3f3f;

int head[maxn], cnt, dis[maxn];

//初始化
void init() {
    memset(head, -1, sizeof head);
    // memset(vis, false, sizeof vis);
    cnt = 0;
}

struct edges {
    int to, next;
    int w;
    edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v, int w = 0) {
    if (cnt == 0) init();
    edge[++cnt] = edges(v, head[u], w);
    head[u] = cnt;
}
const int M = 1e4, inf = 0x3f3f3f3f;
int n, m, dp[3][M], vis[M], c[M];

void dfs(int u, int fa) {
    dp[0][u] = c[u], dp[2][u] = inf;
    int sum = 0;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        dfs (v, u);
        dp[0][u] += min({dp[0][v], dp[2][v], dp[1][v]});
```

```
        if (fa != -1) dp[1][u] += min(dp[2][v], dp[0][v]);
        sum += min(dp[2][v], dp[0][v]);
    }
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        dp[2][u] = min(dp[2][u], sum - min(dp[2][v], dp[0][v]) + dp[0][v]);
    }
}


int main() {
    n = IO();
    init();
    for (int i = 0; i < n; ++i) {
        int u = IO(), w = IO(), k = IO();
        c[u] = w;
        while (k--) {
            int v = IO();
            add(u, v), add(v, u);
        }
    }
    dfs(1, -1);
    printf("%d", min(dp[0][1], dp[2][1]));
    return 0;
}
```

3. 树的最小点覆盖

```
/*
Loj10156
每个点都能看到他所连着的边，问你选出最少的点使树上所有的边都能被看到
dp[0][i]表示不选i点的最小选择数 则其 += dp[1][son]
dp[1][i]表示选i点的最小选择数，则其 += min(dp[0][son], dp[1][son])
*/
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;
#define ll long long
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}

const int maxn = 1e5, maxm = 1e5;
const int INF = 0x3f3f3f3f;

int head[maxn], cnt, dis[maxn];

//初始化
void init() {
    memset(head, -1, sizeof head);
    // memset(vis, false, sizeof vis);
    cnt = 0;
}

struct edges {
    int to, next;
    int w;
    edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v, int w = 0) {
    if (cnt == 0) init();
    edge[++cnt] = edges(v, head[u], w);
    head[u] = cnt;
}
const int M = 1e4;
int n, m, dp[2][M], vis[M];

void dfs(int u, int fa) {
```

```
    dp[1][u] = 1;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        dfs(v, u);
        dp[1][u] += min(dp[0][v], dp[1][v]);
        dp[0][u] += dp[1][v];
    }
}

int main() {
    n = IO();
    for (int i = 0; i < n; ++i) {
        int u = IO(), k = IO();
        while (k--) {
            int v = IO();
            add(u, v), add(v, u);
        }
    }
    dfs(0, -1);
    // printf("%d\n", ans);
    printf("%d", min(dp[0][0], dp[1][0]));
    return 0;
}
```

4. 树的直径

```
/*
Loj 10159
树的直径：树上最长路径
本题需要求出所有直径（直径可能不唯一）上的所有点
解决方法：每次递归算出结点到其儿子中的最长路径和次长路径，然后相加
维护好全局变量ans，最终答案就是ans
*/
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;
#define ll long long
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}

const int maxn = 2e5 + 5, maxm = 2e5 + 5;
const int INF = 0x3f3f3f3f;

int head[maxn], cnt, dis[maxn];

//初始化
void init() {
    memset(head, -1, sizeof head);
    // memset(vis, false, sizeof vis);
    cnt = 0;
}

struct edges {
    int to, next;
    int w;
    edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v, int w = 0) {
    if (cnt == 0) init();
    edge[++cnt] = edges(v, head[u], w);
    head[u] = cnt;
}
const int M = 2e5, inf = 0x3f3f3f3f;
int dp[M], t[M], ans = 0, d1[M], d2[M];
vector<int> res;

int dfs(int u, int fa) {
```

```
    d1[u] = 0, d2[u] = 0;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        int len = dfs(v, u) + 1;
        if (len >= d1[u]) {
            d2[u] = d1[u], d1[u] = len;
        } else if (len > d2[u]) {
            d2[u] = len;
        }
    }
    ans = max(ans, d1[u] + d2[u]);
    return d1[u];
}

void dfs(int u, int fa, int d) {
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        if (d1[v] == d) dfs(v, u, d - 1), res.push_back(v);
    }
}

void solve(int u, int fa) {
    if (d1[u] + d2[u] == ans) {
        res.push_back(u);
        if (d1[u] != d2[u]) dfs(u, fa, d2[u] - 1);
        dfs(u, fa, d1[u] - 1);
    }
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        solve(v, u);
    }
}

int main() {
    int n = IO();
    init();
    for (int i = 1; i < n; ++i) {
        int u = IO(), v = IO();
        add(u, v), add(v, u);
    }
    dfs(0, -1);
    solve(0, -1);
    sort(res.begin(), res.end());
    auto x = unique(res.begin(), res.end());
    auto i = res.begin();
    while (i != x) {
        printf("%d\n", *i);
        i++;
    }
    return 0;
}
```

5. 树的重心

- 树的重心的一些重要性质：
- 一棵树最少有一个重心，最多有两个重心，若有两个重心，则他们相邻（即连有直接边）
- 树上所有点到某个点的距离和里，到重心的距离和最小；若有两个重心，则其距离和相同
- 若以重心为根，则所有子树的大小都不超过整棵树的一半
- 在一棵树上添加或删除一个叶子节点，其重心最多平移一条边的距离
- 两棵树通过连一条边组合成新树，则新树重心在原来两棵树的重心的连线上

```
/*
Poj 1655
树的重心：重心是指树种的一个结点，如果将这个结点删除后剩余的各个连通块中结点数的最大值最小，则称为树的重心
本题需要求出重心，如果有多个输出最小编号的结点，并输出重心被删除后连通块结点数的最大值
*/
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
```

```
        c = getchar();
    }
    return f ? -x : x;
}
using namespace std;

const int maxn = 1e5 + 5, maxm = 2e5 + 5, inf = 0x3f3f3f3f;

int head[maxn], cnt;

//初始化
void init() { memset(head, -1, sizeof head); cnt = 0; }

struct edges {
    int to, next;
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v) {
    edge[++cnt].to = v, edge[cnt].next = head[u];
    head[u] = cnt;
}
int ans, siz;
int dfs(int u, int fa, const int &n) {
    int tot = 1, num = 0;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        int tmp = dfs(v, u, n);
        tot += tmp, num = max(num, tmp);
    }
    int res = max(n - tot, num);
    if (siz >= res) {
        if (siz == res) ans = min(u, ans);
        else ans = u, siz = res;
    }
    return tot;
}
void solve() {
    int n = IO();
    init(), siz = inf;
    for (int i = 1; i < n; ++i) {
        int u = IO(), v = IO();
        add(u, v), add(v, u);
    }
    dfs(1, -1, n);
    printf("%d %d\n", ans, siz);
}

int main() {
    int t = IO();
    while (t--) solve();
    return 0;
}
```

6. 树的中心

```
/*
例题：无
树的中心：找出一个点，使该点到其他点的最远距离最小，则这个点就是树的中心
解题思路：
从u点到其他点的最远距离分为两类
1.从u点向下走的最远距离，用d1[u]表示
2.从u点向上走的最远距离，用up[u]表示
则从u点到其他点的最远距离就是 max(d1[u], up[u]);
则中心到其他点的最远距离就是ans = min{dp[i]}
其中d1[u]可用求树的直径的方法求出
记得同时维护d2[u]即次长距离，和维护最长的路是哪个儿子
关于up的计算方式要用父亲节点来更新儿子节点，与求d1是相反的
如果u的儿子结点son在最长的路径上则
    up[son] = w[son] + max(up[u], d2[u])
否则
    up[son] = w[son] + max(up[u], d1[u])
输入
5
2 1 1
3 2 1
4 3 1
5 1 1
输出中心到其他节点的最长长度
2
*/
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
```

```cpp
using namespace std;

const int maxn = 1e5, maxm = 2e5, inf = 0x3f3f3f3f;

int head[maxn], cnt;

//初始化
void init() { memset(head, -1, sizeof head); cnt = 0; }

struct edges {
    int to, next;
    int w;
}edge[maxm << 1]; //无向图则需要乘2

inline void add(int u, int v, int w) {
    edge[++cnt].to = v, edge[cnt].next = head[u];
    edge[cnt].w = w, head[u] = cnt;
}

int d1[maxn], d2[maxn], maxv[maxn], up[maxn];
int dfs1(int u, int fa) {
    d1[u] = d2[u] = 0;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        int d = dfs1(v, u) + edge[i].w;
        if (d >= d1[u]) {
            d2[u] = d1[u], d1[u] = d;
            maxv[u] = v;
        } else if (d > d2[u]) {
            d2[u] = d;
        }
    }
    return d1[u];
}
void dfs2(int u, int fa) {
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        if (maxv[u] == v) {
            up[v] = max(up[u], d2[u]) + edge[i].w;
        } else {
            up[v] = max(up[u], d1[u]) + edge[i].w;
        }
        dfs2(v, u);
    }
}

int main() {
    int n;
    init();
    scanf("%d\n", &n);
    for (int i = 1; i < n; ++i) {
        int u, v, w;
        scanf("%d %d %d\n", &u, &v, &w);
        add(u, v, w), add(v, u, w);
    }
    dfs1(1, -1);
    dfs2(1, -1);
    int res = 0x3f3f3f3f;
    for (int i = 1; i <= n; ++i) {
        res = min(res, max(up[i], d1[i]));
    }
    printf("%d", res);
    return 0;
}

```

7. 依赖背包问题

```cpp
/*
Loj 10154选课
学生不可能学完大学开设的所有课程，因此必须在入学时选定自己要学的课程。
每个学生可选课程的总数是给定的。请找出一种选课方案使得你能得到的学分最多，
并满足先修课优先的原则。假定课程间不存在时间上的冲突。

输入的第一行包括两个正整数 ，分别表示待选课程数和可选课程数。
接下来  行每行描述一门课，课号依次为 。每行两个数，依次表示这门课先修课课号（若不存在，则该项值为 ）和该门课的学分。
输出一行，表示实际所选课程学分之和。

dp[i][j]代表第i门课程选j个课的最大学分和
题中的课程号是从1开始的，并不是一颗树，而是森林，我们假设有一个课程0，连接所有森林的根结点
*/
#include <cstdio>
#include <cstring>
```

```cpp
#include <algorithm>
#include <vector>

using namespace std;
#define ll long long
#include <cctype>
inline long long IO() {
    long long x = 0;
    bool f = false;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') f = true;
        c = getchar();
    }
    while (isdigit(c)) {
        x = (x << 1) + (x << 3) + (c - '0');
        c = getchar();
    }
    return f ? -x : x;
}

const int maxn = 1e4, maxm = 1e4;
const int INF = 0x3f3f3f3f;

int head[maxn], cnt, dis[maxn];

//初始化
void init() {
    memset(head, -1, sizeof head);
    // memset(vis, false, sizeof vis);
    cnt = 0;
}

struct edges {
    int to, next;
    int w;
    edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
}edge[maxm << 1]; //无向图则需要乘2

inline void add_edges(int u, int v, int w) {
    edge[++cnt] = edges(v, head[u], w);
    head[u] = cnt;
}
const int M = 310;
int n, m, dp[M][M];

int dfs(int u) {
    int num = 1;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to, w = edge[i].w;
        int m = dfs(v);
        for (int j = num + m; j; --j) {//01背包，反向循环
            for (int k = max(0, j - num); k < j && k <= m; ++k) {
                dp[u][j] = max(dp[u][j], dp[v][k] + dp[u][j - k - 1] + w);
            }
        }
        num += m;
    }
    return num; //返回包括自己加上子树有多少个节点
}

int main() {
    n = IO(), m = IO();
    init();
    for (int i = 1; i <= n; ++i) {
        int u = IO(), w = IO();
        add_edges(u, i, w);
    }
    dfs(0);
    printf("%d", dp[0][m]);
    return 0;
}
```

8. 基环树dp
9. 换根