All-in at the River

Standard Code Library

Shanghai Jiao Tong University

Desprado2
fstqwq
AntiLeaf

“

不必恐惧黑夜，它只是黎明的前奏；
待尘埃落定时，你的光芒必将盖过满天繁星。

”

Contents

1 数学 1
1.1 多项式 1
1.1.1 FFT 1
1.1.2 NTT 1
1.1.3 任意模数卷积(MTT, 毛梯梯) 2
1.1.4 多项式操作 3
1.1.5 更优秀的多项式多点求值 5
1.1.6 多项式快速插值 6
1.1.7 拉格朗日反演(多项式复合逆) 7
1.1.8 分治FFT 8
1.1.9 半在线卷积 8
1.1.10 常系数齐次线性递推$O(k \log k \log n)$ 8
1.1.11 应用: $O(\sqrt{n} \log^2 n)$快速求阶乘 9
1.2 插值 9
1.2.1 牛顿插值 9
1.2.2 拉格朗日(Lagrange)插值 9
1.2.3 连续点值平移(待完成) 10
1.3 FWT快速沃尔什变换 10
1.3.1 三行FWT 10
1.4 单纯形 10
1.4.1 线性规划对偶原理 11
1.5 线性代数 11
1.5.1 矩阵乘法 11
1.5.2 高斯消元 11
1.5.3 行列式取模 11
1.5.4 线性基(消成对角) 12
1.5.5 线性代数知识 12
1.5.6 矩阵树定理, BEST定理 12
1.6 博弈论 12
1.6.1 SG定理 12
1.6.2 纳什均衡 12
1.6.3 经典博弈 12
1.6.4 例题(待完成) 13
1.7 自适应Simpson积分 13
1.8 常见数列 13
1.8.1 斐波那契数 卢卡斯数 13
1.8.2 伯努利数, 自然数幂次和 13
1.8.3 分拆数 13
1.8.4 斯特林数 14
1.8.5 贝尔数 14
1.8.6 卡特兰数, 施罗德数, 默慈金数 14
1.9 常用公式及结论 15
1.9.1 方差 15
1.9.2 单位根反演(展开整除条件$[n|k]$) 15
1.9.3 康托展开(排列的排名) 15
1.9.4 连通图计数 15
1.9.5 线性齐次线性常系数递推求通项 15
1.10 常用生成函数变换 15

2 数论 16
2.1 $O(n)$预处理逆元 16
2.2 线性筛 16
2.3 杜教筛 16
2.4 Powerful Number筛 16
2.5 洲阁筛 17
2.6 离散对数(BSGS)(待完成) 18
2.7 Miller-Rabin 18
2.8 Pollard's Rho 18
2.9 快速阶乘算法 19
2.10 扩展欧几里德 19
2.10.1 求通解的方法 19
2.10.2 类欧几里德算法(直线下整点个数) 19
2.11 中国剩余定理(待完成) 19
2.11.1 ex-CRT(待完成) 19
2.12 组合数取模(ex-Lucas)(待完成) 19
2.13 原根 阶 19
2.14 常用公式 20
2.14.1 莫比乌斯反演 20
2.14.2 其他常用公式 20

3 图论 21
3.1 最小生成树 21
3.1.1 Boruvka算法 21
3.1.2 动态最小生成树 21
3.1.3 最小树形图 22
3.1.4 Steiner Tree 斯坦纳树 23
3.1.5 最小直径生成树 24
3.2 最短路 24
3.2.1 Dijkstra 24
3.2.2 Johnson算法(负权图多源最短路) 24
3.2.3 k短路 24
3.3 Tarjan算法 25
3.3.1 强连通分量 25
3.3.2 割点 点双 25
3.3.3 桥 边双 26
3.4 仙人掌 26
3.4.1 仙人掌DP 26
3.5 二分图 27
3.5.1 匈牙利 27
3.5.2 Hopcroft-Karp二分图匹配 27
3.5.3 KM二分图最大权匹配 28
3.5.4 二分图原理 29
3.6 一般图匹配 29
3.6.1 高斯消元 29
3.6.2 带花树 30
3.6.3 带权带花树 31
3.6.4 原理 33
3.7 支配树 33
3.8 2-SAT 33
3.9 最大流 34
3.9.1 Dinic 34
3.9.2 ISAP 34
3.9.3 HLPP 最高标号预流推进 35
3.10 费用流 36
3.10.1 SPFA费用流 36
3.10.2 Dijkstra费用流 37
3.11 网络流原理 37
3.11.1 最大流 37
3.11.2 最小割 37
3.11.3 费用流 37
3.11.4 上下界网络流 37
3.11.5 常见建图方法 38
3.11.6 例题 38
3.12 Prufer序列 38
3.13 弦图相关 38

4 数据结构 39
4.1 线段树 39
4.1.1 非递归线段树 39
4.1.2 线段树维护矩形并 39
4.1.3 主席树离线求区间mex(待完成) 40
4.2 陈丹琦分治 40
4.2.1 动态图连通性(分治并查集) 40
4.2.2 四维偏序 41
4.3 整体二分 41
4.4 平衡树 42
4.4.1 Treap 42
4.4.2 无旋Treap/可持久化Treap 43

4.4.3 Splay 43
4.5 树链剖分 44
4.5.1 动态树形DP(最大权独立集) 44
4.6 树分治 46
4.6.1 动态树分治 46
4.6.2 紫荆花之恋 47
4.7 LCT动态树 49
4.7.1 不换根(弹飞绵羊) 49
4.7.2 换根/维护生成树 49
4.7.3 维护子树信息 51
4.7.4 模板题:动态QTREE4(询问树上相距最远点) 52
4.8 K-D树 54
4.8.1 动态K-D树(定期重构) 54
4.9 LCA最近公共祖先 55
4.9.1 Tarjan LCA $O(n+m)$ 55
4.10 虚树 55
4.11 长链剖分 57
4.11.1 梯子剖分 57
4.12 左偏树 58
4.13 莫队 58
4.13.1 回滚莫队(无删除莫队)(待完成) 58
4.13.2 莫队二次离线 58
4.13.3 带修莫队在线化 $O(n^{\frac{5}{3}})$ 59
4.13.4 莫队二次离线 在线化 $O((n+m)\sqrt{n})$ 59
4.14 常见根号思路 61

5 字符串 62
5.1 KMP 62
5.1.1 ex-KMP 62
5.2 AC自动机 62
5.3 后缀数组 63
5.3.1 倍增 63
5.3.2 SA-IS 63
5.3.3 SAMSA 64
5.4 后缀平衡树 65
5.5 后缀自动机 65
5.5.1 广义后缀自动机 65
5.5.2 区间本质不同子串计数(后缀自动机+LCT+线段树) 66
5.6 回文树 66
5.6.1 广义回文树 67
5.7 Manacher马拉车 68
5.8 字符串原理 69

6 动态规划 70
6.1 决策单调性$O(n\log n)$ 70
6.2 例题(待完成) 70

7 计算几何 71
7.1 Delaunay三角剖分 Voronoi图(待完成) 71

8 Miscellaneous 72
8.1 $O(1)$快速乘 72
8.2 Kahan求和算法(减少浮点数累加的误差) 72
8.3 Python Decimal 72
8.4 $O(n^2)$高精度 72
8.5 笛卡尔树 75
8.6 GarsiaWachs算法($O(n\log n)$合并石子) 75
8.7 常用NTT素数及原根 75
8.8 xorshift 75
8.9 枚举子集 76
8.10 STL 76
8.10.1 vector 76
8.10.2 list 76
8.10.3 unordered_set / unordered_map 76
8.11 Public Based DataStructure(PB_DS) 76
8.11.1 哈希表 76
8.11.2 堆 76
8.11.3 平衡树 76
8.12 rope 77
8.13 其他C++相关 77
8.13.1 <cmath> 77
8.13.2 <algorithm> 77
8.13.3 std::tuple 77
8.13.4 <complex> 77
8.14 一些游戏 77
8.14.1 炉石传说 77
8.15 OEIS 77
8.15.1 计数相关 77
8.15.2 线性递推数列 78
8.15.3 数论相关 78
8.15.4 其他 78
8.16 编译选项 78
8.17 注意事项 79
8.17.1 常见下毒手法 79
8.17.2 场外相关 79
8.17.3 做题策略与心态调节 79
8.18 附录: vscode相关 79
8.18.1 插件 79
8.18.2 设置选项 79
8.18.3 快捷键 79
8.19 附录: 骂人的艺术 —梁实秋 79
8.20 附录: Cheat Sheet 80

1. 数学

1.1 多项式

1.1.1 FFT

```
// 使用时一定要注意double的精度是否足够(极限大概是10 ^ 14)

const double pi = acos((double)-1.0);

// 手写复数类
// 支持加减乘三种运算
// += 运算符如果用的不多可以不重载
struct Complex {
    double a, b; // 由于long double精度和double几乎相同，通常没有必要用long double

    Complex(double a = 0.0, double b = 0.0) : a(a), b(b) {}

    Complex operator + (const Complex &x) const {
        return Complex(a + x.a, b + x.b);
    }

    Complex operator - (const Complex &x) const {
        return Complex(a - x.a, b - x.b);
    }

    Complex operator * (const Complex &x) const {
        return Complex(a * x.a - b * x.b, a * x.b + b * x.a);
    }

    Complex operator * (double x) const {
        return Complex(a * x, b * x);
    }

    Complex &operator += (const Complex &x) {
        return *this = *this + x;
    }

    Complex conj() const { // 共轭，一般只有MTT需要用
        return Complex(a, -b);
    }
} omega[maxn], omega_inv[maxn];
const Complex ima = Complex(0, 1);

int fft_n; // 要在主函数里初始化

// FFT初始化
void FFT_init(int n) {
    fft_n = n;

    for (int i = 0; i < n; i++) // 根据单位根的旋转性质可以节省计算单位根逆元的时间
        omega[i] = Complex(cos(2 * pi / n * i), sin(2 * pi / n * i));

    omega_inv[0] = omega[0];
    for (int i = 1; i < n; i++)
        omega_inv[i] = omega[n - i];
    // 当然不存单位根也可以，只不过在FFT次数较多时很可能会增大常数
}

// FFT主过程
void FFT(Complex *a, int n, int tp) {
    for (int i = 1, j = 0, k; i < n - 1; i++) {
        k = n;
        do
            j ^= (k >>= 1);
        while (j < k);

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int k = 2, m = fft_n / 2; k <= n; k *= 2, m /= 2)
        for (int i = 0; i < n; i += k)
            for (int j = 0; j < k / 2; j++) {
                Complex u = a[i + j], v = (tp > 0 ? omega : omega_inv)[m * j] * a[i + j + k / 2];

                a[i + j] = u + v;
                a[i + j + k / 2] = u - v;
            }

    if (tp < 0)
        for (int i = 0; i < n; i++) {
            a[i].a /= n;
            a[i].b /= n; // 一般情况下是不需要的，只有MTT时才需要
        }
}
```

1.1.2 NTT

```
constexpr int p = 998244353; // p为模数

int ntt_n, omega[maxn], omega_inv[maxn]; // ntt_n要在主函数里初始化

void NTT_init(int n) {
    ntt_n = n;

    int wn = qpow(3, (p - 1) / n); // 这里的3代表模数的任意一个原根

    omega[0] = omega_inv[0] = 1;

    for (int i = 1; i < n; i++)
        omega_inv[n - i] = omega[i] = (long long)omega[i - 1] * wn % p;
}

void NTT(int *a, int n, int tp) { // n为变换长度，tp为1或-1，表示正/逆变换
    for (int i = 1, j = 0, k; i < n - 1; i++) { // O(n)旋转算法，原理是模拟加1
        k = n;
        do
            j ^= (k >>= 1);
        while (j < k);

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int k = 2, m = ntt_n / 2; k <= n; k *= 2, m /= 2)
        for (int i = 0; i < n; i += k)
            for (int j = 0; j < k / 2; j++) {
                int w = (tp > 0 ? omega : omega_inv)[m * j];

                int u = a[i + j], v = (long long)w * a[i + j + k / 2] % p;
                a[i + j] = u + v;
                if (a[i + j] >= p)
```

```
                    a[i + j] -= p;

                a[i + j + k / 2] = u - v;
                if (a[i + j + k / 2] < 0)
                    a[i + j + k / 2] += p;
            }

    if (tp < 0) {
        int inv = qpow(n, p - 2);
        for (int i = 0; i < n; i++)
            a[i] = (long long)a[i] * inv % p;
    }
}
```

1.1.3 任意模数卷积(MTT，毛梯梯)

三模数NTT和直接拆系数FFT都太慢了，不要用.
MTT的原理就是拆系数FFT，只不过优化了做变换的次数.
考虑要对$A(x)$，$B(x)$两个多项式做DFT，可以构造两个复多项式

$$P(x) = A(x) + iB(x) \quad Q(x) = A(x) - iB(x)$$

只需要DFT一个，另一个DFT实际上就是前者反转再取共轭，再利用

$$A(x) = \frac{P(x) + Q(x)}{2} \quad B(x) = \frac{P(x) - Q(x)}{2i}$$

即可还原出$A(x)$，$B(x)$.
IDFT的道理更简单，如果要对$A(x)$和$B(x)$做IDFT，只需要对$A(x) + iB(x)$做IDFT即可，因为IDFT的结果必定为实数，所以结果的实部和虚部就分别是$A(x)$和$B(x)$.
实际上任何同时对两个实序列进行DFT，或者同时对结果为实序列的DFT进行逆变换时都可以按照上面的方法优化，可以减少一半的DFT次数.

```
// 常量和复数类略

const Complex ima = Complex(0, 1);

int p, base;

// FFT略

void DFT(Complex *a, Complex *b, int n) {
    static Complex c[maxn];

    for (int i = 0; i < n; i++)
        c[i] = Complex(a[i].a, b[i].a);

    FFT(c, n, 1);

    for (int i = 0; i < n; i++) {
        int j = (n - i) & (n - 1);

        a[i] = (c[i] + c[j].conj()) * 0.5;
        b[i] = (c[i] - c[j].conj()) * -0.5 * ima;
    }
}

void IDFT(Complex *a, Complex *b, int n) {
    static Complex c[maxn];

    for (int i = 0; i < n; i++)
        c[i] = a[i] + ima * b[i];

    FFT(c, n, -1);

    for (int i = 0; i < n; i++) {
        a[i].a = c[i].a;
        b[i].a = c[i].b;
    }
}

Complex a[2][maxn], b[2][maxn], c[3][maxn];
int ans[maxn];

int main() {
    int n, m;
    scanf("%d%d%d", &n, &m, &p);
    n++;
    m++;

    base = (int)(sqrt(p) + 0.5);

    for (int i = 0; i < n; i++) {
        int x;
        scanf("%d", &x);
        x %= p;

        a[1][i].a = x / base;
        a[0][i].a = x % base;
    }

    for (int i = 0; i < m; i++) {
        int x;
        scanf("%d", &x);
        x %= p;

        b[1][i].a = x / base;
        b[0][i].a = x % base;
    }

    int N = 1;
    while (N < n + m - 1)
        N <<= 1;

    FFT_init(N);

    DFT(a[0], a[1], N);
    DFT(b[0], b[1], N);

    for (int i = 0; i < N; i++)
        c[0][i] = a[0][i] * b[0][i];

    for (int i = 0; i < N; i++)
        c[1][i] = a[0][i] * b[1][i] + a[1][i] * b[0][i];

    for (int i = 0; i < N; i++)
        c[2][i] = a[1][i] * b[1][i];

    FFT(c[1], N, -1);
    IDFT(c[0], c[2], N);

    for (int j = 2; ~j; j--)
        for (int i = 0; i < n + m - 1; i++)
            ans[i] = ((long long)ans[i] * base + (long
              ↪ long)(c[j][i].a + 0.5)) % p;
    // 实际上就是c[2] * base ^ 2 + c[1] * base + c[0], 这样
      ↪ 写可以改善地址访问连续性

    for (int i = 0; i < n + m - 1; i++) {
        if (i)
            printf(" ");

        printf("%d", ans[i]);
    }

    return 0;
}
```

1.1.4 多项式操作

```
// A为输入，C为输出，n为所需长度且必须是2^k
// 多项式求逆，要求A常数项不为0
void get_inv(int *A, int *C, int n) {
    static int B[maxn];

    memset(C, 0, sizeof(int) * (n * 2));
    C[0] = qpow(A[0],p - 2); // 一般常数项都是1，直接赋值为1就可以

    for (int k = 2; k <= n; k <<= 1) {
        memcpy(B, A, sizeof(int) * k);
        memset(B + k, 0, sizeof(int) * k);

        NTT(B, k * 2, 1);
        NTT(C,k * 2, 1);

        for (int i = 0; i < k * 2; i++) {
            C[i] = (2 - (long long)B[i] * C[i]) % p * C[i] % p;
            if (C[i] < 0)
                C[i] += p;
        }

        NTT(C, k * 2, -1);

        memset(C + k, 0, sizeof(int) * k);
    }
}

// 开根
void get_sqrt(int *A, int *C, int n) {
    static int B[maxn], D[maxn];

    memset(C, 0, sizeof(int) * (n * 2));
    C[0] = 1; // 如果不是1就要考虑二次剩余

    for (int k = 2; k <= n; k *= 2) {
        memcpy(B, A, sizeof(int) * k);
        memset(B + k, 0, sizeof(int) * k);

        get_inv(C, D, k);

        NTT(B, k * 2, 1);
        NTT(D, k * 2, 1);

        for (int i = 0; i < k * 2; i++)
            B[i] = (long long)B[i] * D[i]%p;

        NTT(B, k * 2, -1);

        for (int i = 0; i < k; i++)
            C[i] = (long long)(C[i] + B[i]) * inv_2 % p;//inv_2是2的逆元
    }
}

// 求导
void get_derivative(int *A, int *C, int n) {
    for (int i = 1; i < n; i++)
        C[i - 1] = (long long)A[i] * i % p;

    C[n - 1] = 0;
}

// 不定积分，最好预处理逆元
void get_integrate(int *A, int *C, int n) {
    for (int i = 1; i < n; i++)
        C[i] = (long long)A[i - 1] * qpow(i, p - 2) % p;

    C[0] = 0; // 不定积分没有常数项
}

// 多项式ln，要求A常数项不为0
void get_ln(int *A, int *C, int n) { // 通常情况下A常数项都是1
    static int B[maxn];

    get_derivative(A, B, n);
    memset(B + n, 0, sizeof(int) * n);

    get_inv(A, C, n);

    NTT(B, n * 2, 1);
    NTT(C, n * 2, 1);

    for (int i = 0;i < n * 2; i++)
        B[i] = (long long)B[i] * C[i] % p;

    NTT(B, n * 2, -1);

    get_integrate(B, C, n);

    memset(C+n,0,sizeof(int)*n);
}

// 多项式exp，要求A没有常数项
// 常数很大且总代码较长，一般来说最好替换为分治FFT
// 分治FFT依据：设G(x) = exp F(x)，则有 g_i = \sum_{k=1}^{i-1} f_{i-k} * k * g_k
void get_exp(int *A, int *C, int n) {
    static int B[maxn];

    memset(C, 0, sizeof(int) * (n * 2));
    C[0] = 1;

    for (int k = 2; k <= n; k <<= 1) {
        get_ln(C, B, k);

        for (int i = 0; i < k; i++) {
            B[i] = A[i] - B[i];
            if (B[i] < 0)
                B[i] += p;
        }
        (++B[0]) %= p;

        NTT(B, k * 2, 1);
        NTT(C, k * 2, 1);

        for (int i = 0; i < k * 2; i++)
            C[i] = (long long)C[i] * B[i] % p;

        NTT(C, k * 2, -1);

        memset(C + k, 0, sizeof(int) * k);
    }
}

// 多项式k次幂，在A常数项不为1时需要转化
// 常数较大且总代码较长，在时间要求不高时最好替换为暴力快速幂
void get_pow(int *A, int *C, int n, int k) {
    static int B[maxn];

    get_ln(A, B, n);

    for (int i = 0; i < n; i++)
        B[i] = (long long)B[i] * k % p;

    get_exp(B, C, n);
}

// 多项式除法，A / B，结果输出在C
// A的次数为n，B的次数为m
void get_div(int *A, int *B, int *C, int n, int m) {
    static int f[maxn], g[maxn], gi[maxn];

    if (n < m) {
        memset(C, 0, sizeof(int) * m);
        return;
    }

    int N = 1;
    while (N < (n - m + 1))
        N <<= 1;

    memset(f, 0, sizeof(int) * N * 2);
    memset(g, 0, sizeof(int) * N * 2);
    // memset(gi, 0, sizeof(int) * N);

    for (int i = 0; i < n - m + 1; i++)
        f[i] = A[n - i - 1];
    for (int i = 0; i < m && i < n - m + 1; i++)
        g[i] = B[m - i - 1];

    get_inv(g, gi, N);

    for (int i = n - m + 1; i < N; i++)
        gi[i] = 0;

    NTT(f, N * 2, 1);
    NTT(gi, N * 2, 1);

    for (int i = 0; i < N * 2; i++)
        f[i] = (long long)f[i] * gi[i] % p;

    NTT(f, N * 2, -1);

    for (int i = 0; i < n - m + 1; i++)
        C[i] = f[n - m - i];
}

// 多项式取模，余数输出到C，商输出到D
void get_mod(int *A, int *B, int *C, int *D, int n, int m) {
    static int b[maxn], d[maxn];

    if (n < m) {
        memcpy(C, A, sizeof(int) * n);

        if (D)
            memset(D, 0, sizeof(int) * m);

        return;
    }

    get_div(A, B, d, n, m);

    if (D) { // D是商，可以选择不要
        for (int i = 0; i < n - m + 1; i++)
            D[i] = d[i];
    }

    int N = 1;
    while (N < n)
        N *= 2;

    memcpy(b, B, sizeof(int) * m);

    NTT(b, N, 1);
    NTT(d, N, 1);

    for (int i = 0; i < N; i++)
        b[i] = (long long)d[i] * b[i] % p;

    NTT(b, N, -1);

    for (int i = 0; i < m - 1; i++)
        C[i] = (A[i] - b[i] + p) % p;

    memset(b, 0, sizeof(int) * N);
    memset(d, 0, sizeof(int) * N);
}

// 多点求值要用的数组
int q[maxn], ans[maxn]; // q是要代入的各个系数，ans是求出的值
int tg[25][maxn * 2], tf[25][maxn]; // 辅助数组，tg是预处理乘积，
// tf是项数越来越少的f，tf[0]就是原来的函数

void pretreat(int l, int r, int k) { // 多点求值预处理
    static int A[maxn], B[maxn];

    int *g = tg[k] + l * 2;

    if (r - l + 1 <= 200) {
        g[0] = 1;

        for (int i = l; i <= r; i++) {
            for (int j = i - l + 1; j; j--) {
                g[j] = (g[j - 1] - (long long)g[j] * q[i]) % p;
                if (g[j] < 0)
                    g[j] += p;
            }
            g[0] = (long long)g[0] * (p - q[i]) % p;
        }

        return;
    }

    int mid = (l + r) / 2;

    pretreat(l, mid, k + 1);
    pretreat(mid + 1, r, k + 1);

    if (!k)
        return;

    int N = 1;
    while (N <= r - l + 1)
        N *= 2;

    int *gl = tg[k + 1] + l * 2, *gr = tg[k + 1] + (mid + 1) * 2;

    memset(A, 0, sizeof(int) * N);
    memset(B, 0, sizeof(int) * N);

    memcpy(A, gl, sizeof(int) * (mid - l + 2));
    memcpy(B, gr, sizeof(int) * (r - mid + 1));

    NTT(A, N, 1);
    NTT(B, N, 1);

    for (int i = 0; i < N; i++)
        A[i] = (long long)A[i] * B[i] % p;

    NTT(A, N, -1);

    for (int i = 0; i <= r - l + 1; i++)
        g[i] = A[i];
}

void solve(int l, int r, int k) { // 多项式多点求值主过程
    int *f = tf[k];

    if (r - l + 1 <= 200) {
        for (int i = l; i <= r; i++) {
            int x = q[i];

            for (int j = r - l; ~j; j--)
                ans[i] = ((long long)ans[i] * x + f[j]) % p;
        }

        return;
    }

    int mid = (l + r) / 2;
    int *ff = tf[k + 1], *gl = tg[k + 1] + l * 2, *gr = tg[k + 1] + (mid + 1) * 2;

    get_mod(f, gl, ff, NULL, r - l + 1, mid - l + 2);
    solve(l, mid, k + 1);

    memset(gl, 0, sizeof(int) * (mid - l + 2));
    memset(ff, 0, sizeof(int) * (mid - l + 1));

    get_mod(f, gr, ff, NULL, r - l + 1, r - mid + 1);
    solve(mid + 1, r, k + 1);

    memset(gr, 0, sizeof(int) * (r - mid + 1));
    memset(ff, 0, sizeof(int) * (r - mid));
}

// f < x^n, m个询问，询问是0-based，当然改成1-based也很简单
void get_value(int *f, int *x, int *a, int n, int m) {
    if (m <= n)
        m = n + 1;
    if (n < m - 1)
        n = m - 1; // 补零方便处理

    memcpy(tf[0], f, sizeof(int) * n);
    memcpy(q, x, sizeof(int) * m);

    pretreat(0, m - 1, 0);
    solve(0, m - 1, 0);

    if (a) // 如果a是NULL，代表不复制答案，直接用ans数组
        memcpy(a, ans, sizeof(int) * m);
}
```

1.1.5 更优秀的多项式多点求值

这个做法不需要写取模，求逆也只有一次，但是神乎其技，完全搞不懂原理

清空和复制之类的地方容易抄错，抄的时候要注意

```
int q[maxn], ans[maxn]; // q是要代入的各个系数，ans是求出的值
int tg[25][maxn * 2], tf[25][maxn]; // 辅助数组，tg是预处理乘积，
// tf是项数越来越少的f，tf[0]就是原来的函数

void pretreat(int l, int r, int k) { // 预处理
    static int A[maxn], B[maxn];

    int *g = tg[k] + l * 2;

    if (r - l + 1 <= 1) {
        g[0] = 1;

        for (int i = l; i <= r; i++) {
            for (int j = i - l + 1; j; j--) {
                g[j] = (g[j - 1] - (long long)g[j] * q[i]) % p;
                if (g[j] < 0)
                    g[j] += p;
            }
            g[0] = (long long)g[0] * (p - q[i]) % p;
        }

        reverse(g, g + r - l + 2);

        return;
    }

    int mid = (l + r) / 2;

    pretreat(l, mid, k + 1);
    pretreat(mid + 1, r, k + 1);

    int N = 1;
    while (N <= r - l + 1)
        N *= 2;

    int *gl = tg[k + 1] + l * 2, *gr = tg[k + 1] + (mid + 1) * 2;

    memset(A, 0, sizeof(int) * N);
    memset(B, 0, sizeof(int) * N);

    memcpy(A, gl, sizeof(int) * (mid - l + 2));
    memcpy(B, gr, sizeof(int) * (r - mid + 1));

    NTT(A, N, 1);
    NTT(B, N, 1);

    for (int i = 0; i < N; i++)
        A[i] = (long long)A[i] * B[i] % p;

    NTT(A, N, -1);

    for (int i = 0; i <= r - l + 1; i++)
        g[i] = A[i];
}

void solve(int l, int r, int k) { // 主过程
    static int a[maxn], b[maxn];

    int *f = tf[k];

    if (l == r) {
        ans[l] = f[0];
        return;
    }

    int mid = (l + r) / 2;
    int *ff = tf[k + 1], *gl = tg[k + 1] + l * 2, *gr = tg[k + 1] + (mid + 1) * 2;

    int N = 1;
    while (N < r - l + 2)
        N *= 2;

    memcpy(a, f, sizeof(int) * (r - l + 2));
    memcpy(b, gr, sizeof(int) * (r - mid + 1));
    reverse(b, b + r - mid + 1);

    NTT(a, N, 1);
    NTT(b, N, 1);
    for (int i = 0; i < N; i++)
        b[i] = (long long)a[i] * b[i] % p;

    reverse(b + 1, b + N);
    NTT(b, N, 1);
    int n_inv = qpow(N, p - 2);
    for (int i = 0; i < N; i++)
        b[i] = (long long)b[i] * n_inv % p;

    for (int i = 0; i < mid - l + 2; i++)
        ff[i] = b[i + r - mid];

    memset(a, 0, sizeof(int) * N);
    memset(b, 0, sizeof(int) * N);

    solve(l, mid, k + 1);

    memset(ff, 0, sizeof(int) * (mid - l + 2));

    memcpy(a, f, sizeof(int) * (r - l + 2));
    memcpy(b, gl, sizeof(int) * (mid - l + 2));
    reverse(b, b + mid - l + 2);

    NTT(a, N, 1);
    NTT(b, N, 1);
    for (int i = 0; i < N; i++)
        b[i] = (long long)a[i] * b[i] % p;

    reverse(b + 1, b + N);
    NTT(b, N, 1);
    for (int i = 0; i < N; i++)
        b[i] = (long long)b[i] * n_inv % p;

    for (int i = 0; i < r - mid + 1; i++)
        ff[i] = b[i + mid - l + 1];

    memset(a, 0, sizeof(int) * N);
    memset(b, 0, sizeof(int) * N);

    solve(mid + 1, r, k + 1);

    memset(gl, 0, sizeof(int) * (mid - l + 2));
    memset(gr, 0, sizeof(int) * (r - mid + 1));
    memset(ff, 0, sizeof(int) * (r - mid + 1));
}

// f < x^n, m个询问, 0-based
void get_value(int *f, int *x, int *a, int n, int m) {
    static int c[maxn], d[maxn];

    if (m <= n)
        m = n + 1;
    if (n < m - 1)
        n = m - 1; // 补零

    memcpy(q, x, sizeof(int) * m);

    pretreat(0, m - 1, 0);

    int N = 1;
    while (N < m)
        N *= 2;

    get_inv(tg[0], c, N);

    fill(c + m, c + N, 0);
    reverse(c, c + m);

    memcpy(d, f, sizeof(int) * m);

    NTT(c, N * 2, 1);
    NTT(d, N * 2, 1);
    for (int i = 0; i < N * 2; i++)
        c[i] = (long long)c[i] * d[i] % p;
    NTT(c, N * 2, -1);

    for (int i = 0; i < m; i++)
        tf[0][i] = c[i + n];

    solve(0, m - 1, 0);

    if (a) // 如果a是NULL，代表不复制答案，直接用ans数组
        memcpy(a, ans, sizeof(int) * m);
}
```

1.1.6 多项式快速插值

快速插值: 给出n个x_i与y_i, 求一个$n-1$次多项式满足$F(x_i)=y_i$. 考虑拉格朗日插值:

$$F(x)=\sum_{i=1}^{n}\frac{\prod_{i\neq j}(x-x_j)}{\prod_{i\neq j}(x_i-x_j)}y_i$$

第一步要先对每个i求出

$$\prod_{i\neq j}(x_i-x_j)$$

设

$$M(x)=\prod_{i=1}^{n}(x-x_i)$$

那么想要的是

$$\frac{M(x)}{x-x_i}$$

取$x=x_i$时, 上下都为0, 使用洛必达法则, 则原式化为$M'(x)$. 使用分治算出$M(x)$, 使用多点求值即可算出每个

$$\prod_{i\neq j}(x_i-x_j)=M'(x_i)$$

设

$$v_i=\frac{y_i}{\prod_{i\neq j}(x_i-x_j)}$$

第二步要求出

$$\sum_{i=1}^{n}v_i\prod_{i\neq j}(x-x_j)$$

使用分治. 设

$$L(x)=\prod_{i=1}^{\lfloor n/2\rfloor}(x-x_i),\ R(x)=\prod_{i=\lfloor n/2\rfloor+1}^{n}(x-x_i)$$

则原式化为

$$\left(\sum_{i=1}^{\lfloor n/2\rfloor}v_i\prod_{i\neq j,j\leq\lfloor n/2\rfloor}(x-x_j)\right)R(x)+$$

$$\left(\sum_{i=\lfloor n/2\rfloor+1}^{n} v_i \prod_{i\neq j, j>\lfloor n/2\rfloor} (x-x_j)\right) L(x)$$

递归计算, 复杂度$O(n\log^2 n)$.
注意由于整体和局部的$M(x)$都要用到, 要预处理一下.

```
int qx[maxn], qy[maxn];
int th[25][maxn * 2], ansf[maxn]; // th存的是各阶段的M(x)

void pretreat2(int l, int r, int k) { // 预处理
    static int A[maxn], B[maxn];
    int *h = th[k] + l * 2;

    if (l == r) {
        h[0] = p - qx[l];
        h[1] = 1;
        return;
    }

    int mid = (l + r) / 2;

    pretreat2(l, mid, k + 1);
    pretreat2(mid + 1, r, k + 1);

    int N = 1;
    while (N <= r - l + 1)
        N *= 2;

    int *hl = th[k + 1] + l * 2, *hr = th[k + 1] + (mid + 1) * 2;

    memset(A, 0, sizeof(int) * N);
    memset(B, 0, sizeof(int) * N);

    memcpy(A, hl, sizeof(int) * (mid - l + 2));
    memcpy(B, hr, sizeof(int) * (r - mid + 1));

    NTT(A, N, 1);
    NTT(B, N, 1);

    for (int i = 0; i < N; i++)
        A[i] = (long long)A[i] * B[i] % p;

    NTT(A, N, -1);

    for (int i = 0; i <= r - l + 1; i++)
        h[i] = A[i];
}

void solve2(int l, int r, int k) { // 分治
    static int A[maxn], B[maxn], t[maxn];

    if (l == r)
        return;

    int mid = (l + r) / 2;

    solve2(l, mid, k + 1);
    solve2(mid + 1, r, k + 1);

    int *hl = th[k + 1] + l * 2, *hr = th[k + 1] + (mid + 1) * 2;

    int N = 1;

    while (N < r - l + 1)
        N *= 2;

    memset(A, 0, sizeof(int) * N);
    memset(B, 0, sizeof(int) * N);

    memcpy(A, ansf + l, sizeof(int) * (mid - l + 1));
    memcpy(B, hr, sizeof(int) * (r - mid + 1));

    NTT(A, N, 1);
    NTT(B, N, 1);

    for (int i = 0; i < N; i++)
        t[i] = (long long)A[i] * B[i] % p;

    memset(A, 0, sizeof(int) * N);
    memset(B, 0, sizeof(int) * N);

    memcpy(A, ansf + mid + 1, sizeof(int) * (r - mid));
    memcpy(B, hl, sizeof(int) * (mid - l + 2));

    NTT(A, N, 1);
    NTT(B, N, 1);

    for (int i = 0; i < N; i++)
        t[i] = (t[i] + (long long)A[i] * B[i]) % p;

    NTT(t, N, -1);

    memcpy(ansf + l, t, sizeof(int) * (r - l + 1));
}

// 主过程
// 如果x, y传NULL表示询问已经存在了qx, qy里
void interpolation(int *x, int *y, int n, int *f = NULL) {
    static int d[maxn];

    if (x)
        memcpy(qx, x, sizeof(int) * n);
    if (y)
        memcpy(qy, y, sizeof(int) * n);

    pretreat2(0, n - 1, 0);

    get_derivative(th[0], d, n + 1);

    multipoint_eval(d, qx, NULL, n, n);

    for (int i = 0; i < n; i++)
        ansf[i] = (long long)qy[i] * qpow(ans[i], p - 2) % p;

    solve2(0, n - 1, 0);

    if (f)
        memcpy(f, ansf, sizeof(int) * n);
}
```

1.1.7 拉格朗日反演(多项式复合逆)

如果$f(x)$与$g(x)$互为复合逆, 则有

$$[x^n]\,g(x) = \frac{1}{n}\,[x^{n-1}]\left(\frac{x}{f(x)}\right)^n$$

$$[x^n]\,h(g(x)) = \frac{1}{n}\,[x^{n-1}]\,h'(x)\left(\frac{x}{f(x)}\right)^n$$

1.1.8 分治FFT

```
void solve(int l,int r) {
    if (l == r)
        return;

    int mid = (l + r) / 2;
    solve(l, mid);

    int N = 1;
    while (N <= r - l + 1)
        N *= 2;

    for (int i = l; i <= mid; i++)
        B[i - l] = (long long)A[i] * fac_inv[i] % p;
    fill(B + mid - l + 1, B + N, 0);
    for (int i = 0; i < N; i++)
        C[i] = fac_inv[i];

    NTT(B, N, 1);
    NTT(C, N, 1);

    for (int i = 0; i < N; i++)
        B[i] = (long long)B[i] * C[i] % p;

    NTT(B, N, -1);

    for (int i = mid + 1; i <= r; i++)
        A[i] = (A[i] + B[i - l] * 2 % p * (long long)fac[i] % p) % p;

    solve(mid + 1, r);
}
```

1.1.9 半在线卷积

```
void solve(int l, int r) {
    if (r <= m)
        return;

    if (r - l == 1) {
        if (l == m)
            f[l] = a[m];
        else
            f[l] = (long long)f[l] * inv[l - m] % p;

        for (int i = l, t = (long long)l * f[l] % p; i <= n; i += l)
            g[i] = (g[i] + t) % p;

        return;
    }

    int mid = (l + r) / 2;

    solve(l, mid);

    if (l == 0) {
        for (int i = 1; i < mid; i++) {
            A[i] = f[i];
            B[i] = (c[i] + g[i]) % p;
        }
        NTT(A, r, 1);
        NTT(B, r, 1);
        for (int i = 0; i < r; i++)
            A[i] = (long long)A[i] * B[i] % p;
        NTT(A, r, -1);

        for (int i = mid; i < r; i++)
            f[i] = (f[i] + A[i]) % p;
    }
    else {
        for (int i = 0; i < r - l; i++)
            A[i] = f[i];
        for (int i = l; i < mid; i++)
            B[i - l] = (c[i] + g[i]) % p;
        NTT(A, r - l, 1);
        NTT(B, r - l, 1);
        for (int i = 0; i < r - l; i++)
            A[i] = (long long)A[i] * B[i] %p;
        NTT(A, r - l, -1);

        for (int i = mid; i < r; i++)
            f[i] = (f[i] + A[i - l]) % p;

        memset(A, 0, sizeof(int) * (r - l));
        memset(B, 0, sizeof(int) * (r - l));

        for (int i = l; i < mid; i++)
            A[i - l] = f[i];
        for (int i = 0; i < r - l; i++)
            B[i] = (c[i] + g[i]) % p;
        NTT(A, r - l, 1);
        NTT(B, r - l, 1);
        for (int i = 0; i < r - l; i++)
            A[i] = (long long)A[i] * B[i] % p;
        NTT(A, r - l, -1);

        for (int i = mid; i < r; i++)
            f[i] = (f[i] + A[i - l]) % p;
    }

    memset(A, 0, sizeof(int) * (r - l));
    memset(B, 0, sizeof(int) * (r - l));

    solve(mid, r);
}
```

1.1.10 常系数齐次线性递推$O(k \log k \log n)$

如果只有一次这个操作可以照抄，否则就开一个全局flag.

```
// 多项式取模，余数输出到C，商输出到D
void get_mod(int *A, int *B, int *C, int *D, int n, int m) {
    static int b[maxn], d[maxn];
    static bool flag = false;

    if (n < m) {
        memcpy(C, A, sizeof(int) * n);

        if (D)
            memset(D, 0, sizeof(int) * m);

        return;
    }

    get_div(A, B, d, n, m);

    if (D) { // D是商，可以选择不要
        for (int i = 0; i < n - m + 1; i++)
            D[i] = d[i];
    }

    int N = 1;
    while (N < n)
        N *= 2;

    if (!flag) {
        memcpy(b, B, sizeof(int) * m);
        NTT(b, N, 1);

        flag = true;
```

```cpp
    }

    NTT(d, N, 1);

    for (int i = 0; i < N; i++)
        d[i] = (long long)d[i] * b[i] % p;

    NTT(d, N, -1);

    for (int i = 0; i < m - 1; i++)
        C[i] = (A[i] - d[i] + p) % p;

    // memset(b, 0, sizeof(int) * N);
    memset(d, 0, sizeof(int) * N);
}

// g < x^n, f是输出答案的数组
void pow_mod(long long k, int *g, int n, int *f) {
    static int a[maxn], t[maxn];

    memset(f, 0, sizeof(int) * (n * 2));

    f[0] = a[1] = 1;

    int N = 1;
    while (N < n * 2 - 1)
        N *= 2;

    while (k) {
        NTT(a, N, 1);

        if (k & 1) {
            memcpy(t, f, sizeof(int) * N);

            NTT(t, N, 1);
            for (int i = 0; i < N; i++)
                t[i] = (long long)t[i] * a[i] % p;
            NTT(t, N, -1);

            get_mod(t, g, f, NULL, n * 2 - 1, n);
        }

        for (int i = 0; i < N; i++)
            a[i] = (long long)a[i] * a[i] % p;
        NTT(a, N, -1);

        memcpy(t, a, sizeof(int) * (n * 2 - 1));
        get_mod(t, g, a, NULL, n * 2 - 1, n);
        fill(a + n - 1, a + N, 0);

        k >>= 1;
    }

    memset(a, 0, sizeof(int) * (n * 2));
}

// f_n = \sum_{i = 1} ^ m  f_{n - i} a_i
// f是0~m-1项的初值
int linear_recurrence(long long n, int m, int *f, int *a) {
    static int g[maxn], c[maxn];

    memset(g, 0, sizeof(int) * (m * 2 + 1));

    for (int i = 0; i < m; i++)
        g[i] = (p - a[m - i]) % p;
    g[m] = 1;

    pow_mod(n, g, m + 1, c);

    int ans = 0;
    for (int i = 0; i < m; i++)
        ans = (ans + (long long)c[i] * f[i]) % p;

    return ans;
}
```

1.1.11 应用: $O(\sqrt{n}\log^2 n)$快速求阶乘

问题: 求$n! \pmod p$, $n < p$, p是NTT模数.
考虑令$m = \lfloor\sqrt{n}\rfloor$, 那么我们可以写出连续$m$个数相乘的多项式:

$$f(x) = \prod_{i=1}^{m}(x+i)$$

那么显然就有

$$n! = \left(\prod_{k=0}^{m-1} f(km)\right) \prod_{i=m^2+1}^{n} i$$

$f(x)$的系数可以用倍增求(或者懒一点直接分治FFT), 然后$f(km)$可以用多项式多点求值求出, 所以总复杂度就是$O(\sqrt{n}\log^2 n)$.
当然如果p不变并且多次询问的话我们只需要取一个m, 也就是预处理$O(\sqrt{p}\log^2 p)$, 询问$O(\sqrt{p})$.

1.2 插值

1.2.1 牛顿插值

牛顿插值的原理是**二项式反演**.
二项式反演:

$$f(n) = \sum_{k=0}^{n}\binom{n}{k}g(k) \Leftrightarrow g(n) = \sum_{k=0}^{n}(-1)^{n-k}\binom{n}{k}f(k)$$

可以用e^x和e^{-x} 的麦克劳林展开式证明.
套用二项式反演的结论即可得到牛顿插值:

$$f(n) = \sum_{i=0}^{k}\binom{n}{i}r_i$$

$$r_i = \sum_{j=0}^{i}(-1)^{i-j}\binom{i}{j}f(j)$$

其中k表示$f(n)$的最高次项系数.
实现时可以用k次差分替代右边的式子:

```cpp
for (int i = 0; i <= k; i++)
    r[i] = f(i);
for (int j = 0; j < k; j++)
    for (int i = k; i > j; i--)
        r[i] -= r[i - 1];
```

注意到预处理r_i 的式子满足卷积形式,必要时可以用FFT优化至$O(k\log k)$ 预处理.

1.2.2 拉格朗日(Lagrange)插值

$$f(x) = \sum_i f(x_i)\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}$$

1.2.3 连续点值平移(待完成)

1.3 FWT快速沃尔什变换

```cpp
// 注意FWT常数比较小, 这点与FFT/NTT不同
// 以下代码均以模质数情况为例, 其中n为变换长度, tp表示正/逆变换

// 按位或版本
void FWT_or(int *A, int n, int tp) {
    for (int k = 2; k <= n; k *= 2)
        for (int i = 0; i < n; i += k)
            for (int j = 0; j < k / 2; j++) {
                if (tp > 0)
                    A[i + j + k / 2] = (A[i + j + k / 2] + A[i + j]) % p;
                else
                    A[i + j + k / 2] = (A[i + j + k / 2] - A[i + j] + p) % p;
            }
}

// 按位与版本
void FWT_and(int *A, int n, int tp) {
    for (int k = 2; k <= n; k *= 2)
        for (int i = 0; i < n; i += k)
            for (int j = 0; j < k / 2; j++) {
                if (tp > 0)
                    A[i + j] = (A[i + j] + A[i + j + k / 2]) % p;
                else
                    A[i + j] = (A[i + j] - A[i + j + k / 2] + p) % p;
            }
}

// 按位异或版本
void FWT_xor(int *A, int n, int tp) {
    for (int k = 2; k <= n; k *= 2)
        for (int i = 0; i < n; i += k)
            for (int j = 0; j < k / 2; j++) {
                int a = A[i + j], b = A[i + j + k / 2];
                A[i + j] = (a + b) % p;
                A[i + j + k / 2] = (a - b + p) % p;
            }

    if (tp < 0) {
        int inv = qpow(n % p, p - 2); // n的逆元，在不取模时需要用每层除以2代替
        for (int i = 0; i < n; i++)
            A[i] = A[i] * inv % p;
    }
}
```

1.3.1 三行FWT

```
void fwt_or(int *a, int n, int tp) {
    for (int j = 0; (1 << j) < n; j++)
        for (int i = 0; i < n; i++)
            if (i >> j & 1) {
                if (tp > 0)
                    a[i] += a[i ^ (1 << j)];
                else
                    a[i] -= a[i ^ (1 << j)];
            }
}

// and自然就是or反过来
void fwt_and(int *a, int n, int tp) {
    for (int j = 0; (1 << j) < n; j++)
        for (int i = 0; i < n; i++)
            if (!(i >> j & 1)) {
                if (tp > 0)
                    a[i] += a[i | (1 << j)];
                else
                    a[i] -= a[i | (1 << j)];
            }
}

// xor同理
```

1.4 单纯形

```
const double eps = 1e-10;

double A[maxn][maxn], x[maxn];
int n, m, t, id[maxn * 2];

// 方便起见,这里附上主函数
int main() {
    scanf("%d%d%d", &n, &m, &t);

    for (int i = 1; i <= n; i++) {
        scanf("%lf", &A[0][i]);
        id[i] = i;
    }

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++)
            scanf("%lf", &A[i][j]);

        scanf("%lf", &A[i][0]);
    }

    if (!initalize())
        printf("Infeasible"); // 无解
    else if (!simplex())
        printf("Unbounded"); // 最优解无限大

    else {
        printf("%.15lf\n", -A[0][0]);
        if (t) {
            for (int i = 1; i <= m; i++)
                x[id[i + n]] = A[i][0];
            for (int i = 1; i <= n; i++)
                printf("%.15lf ",x[i]);
        }
    }
    return 0;
}

//初始化
//对于初始解可行的问题,可以把初始化省略掉
bool initalize() {
    while (true) {
        double t = 0.0;
        int l = 0, e = 0;

        for (int i = 1; i <= m; i++)
            if (A[i][0] + eps < t) {
                t = A[i][0];
                l = i;
            }

        if (!l)
            return true;

        for (int i = 1; i <= n; i++)
            if (A[l][i] < -eps && (!e || id[i] < id[e]))
                e = i;

        if (!e)
            return false;

        pivot(l, e);
    }
}

//求解
bool simplex() {
    while (true) {
        int l = 0, e = 0;
        for (int i = 1; i <= n; i++)
            if (A[0][i] > eps && (!e || id[i] < id[e]))
                e = i;

        if (!e)
            return true;

        double t = 1e50;
        for (int i = 1; i <= m; i++)
            if (A[i][e] > eps && A[i][0] / A[i][e] < t) {
                l = i;
                t = A[i][0]/A[i][e];
            }

        if (!l)
            return false;

        pivot(l, e);
    }
}

//转轴操作,本质是在凸包上沿着一条棱移动
void pivot(int l, int e) {
    swap(id[e], id[n + l]);
    double t = A[l][e];
    A[l][e] = 1.0;

    for (int i = 0; i <= n; i++)
        A[l][i] /= t;

    for (int i = 0; i <= m; i++)
        if (i != l) {
            t = A[i][e];
            A[i][e] = 0.0;
            for (int j = 0; j <= n; j++)
                A[i][j] -= t * A[l][j];
        }
}
```

1.4.1 线性规划对偶原理

给定一个原始线性规划:

$$\begin{array}{rr} \text{Minimize} & \displaystyle\sum_{j=1}^{n} c_j x_j \\ \text{Subject to} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \ge b_i, \\ & x_j \ge 0 \end{array}$$

定义它的对偶线性规划为:

$$\begin{array}{rr} \text{Maximize} & \displaystyle\sum_{i=1}^{m} b_i y_i \\ \text{Subject to} & \displaystyle\sum_{i=1}^{m} a_{ij} y_i \le c_j, \\ & y_i \ge 0 \end{array}$$

用矩阵可以更形象地表示为:

$$\begin{array}{rr c rr} \text{Minimize} & \mathbf{c}^T\mathbf{x} & & \text{Maximize} & \mathbf{b}^T\mathbf{y} \\ \text{Subject to} & A\mathbf{x} \ge \mathbf{b}, & \Longleftrightarrow & \text{Subject to} & A^T\mathbf{y} \le \mathbf{c}, \\ & \mathbf{x} \ge 0 & & & \mathbf{y} \ge 0 \end{array}$$

1.5 线性代数

1.5.1 矩阵乘法

```
for (int i = 1; i <= n; i++)
    for (int k = 1; k <= n; k++)
        for (int j = 1; j <= n; j++)
            a[i][j] += b[i][k] * c[k][j];
// 通过改善内存访问连续性，显著提升速度
```

1.5.2 高斯消元

高斯-约当消元法 Gauss-Jordan

每次选取当前行绝对值最大的数作为代表元，在做浮点数消元时可以很好地保证精度.

```
void Gauss_Jordan(int A[][maxn], int n) {
    for (int i = 1; i <= n; i++) {
        int ii = i;
        for (int j = i + 1; j <= n; j++)
            if (fabs(A[j][i]) > fabs(A[ii][i]))
                ii = j;

        if (ii != i) // 这里没有判是否无解，如果有可能无解的话要判一下
            for (int j = i; j <= n + 1; j++)
                swap(A[i][j], A[ii][j]);

        for (int j = 1; j <= n; j++)
            if (j != i) // 消成对角
                for (int k = n + 1; k >= i; k--)
                    A[j][k] -= A[j][i] / A[i][i] * A[i][k];
    }
}
```

解线性方程组

在矩阵的右边加上一列表示系数即可，如果消成上三角的话最后要倒序回代.

求逆矩阵

维护一个矩阵B，初始设为n阶单位矩阵，在消元的同时对B进行一样的操作，当把A消成单位矩阵时B就是逆矩阵.

行列式

消成对角之后把代表元乘起来. 如果是任意模数，要注意消元时每交换一次行列要取反一次.

1.5.3 行列式取模

```
int p;

int Gauss(int A[maxn][maxn], int n) {
    int det = 1;

    for (int i = 1; i <= n; i++) {
        for (int j = i + 1; j <= n; j++)
            while (A[j][i]) {
                int t = (p - A[i][i] / A[j][i]) % p;
                for (int k = i; k <= n; k++)
                    A[i][k] = (A[i][k] + (long long)A[j][k] * t) % p;

                swap(A[i], A[j]);
                det = (p - det) % p; // 交换一次之后行列式取负
            }

            if (!A[i][i])
                return 0;

            det = (long long)det * A[i][i] % p;
    }

    return det;
}
```

1.5.4 线性基(消成对角)

```
void add(unsigned long long x) {
    for (int i = 63; i >= 0; i--)
        if (x >> i & 1) {
            if (b[i])
                x ^= b[i];
            else {
                b[i] = x;

                for (int j = i - 1; j >= 0; j--)
                    if (b[j] && (b[i] >> j & 1))
                        b[i] ^= b[j];

                for (int j = i + 1; j < 64; j++)
                    if (b[j] >> i & 1)
                        b[j] ^= b[i];

                break;
            }
        }
}
```

1.5.5 线性代数知识

行列式:

$$\det A = \sum_{\sigma} \operatorname{sgn}(\sigma) \prod_{i} a_{i,\sigma_i}$$

逆矩阵:

$$B = A^{-1} \iff AB = 1$$

代数余子式:

$$C_{i,j} = (-1)^{i+j} M_{i,j} = (-1)^{i+j} \left| A^{i,j} \right|$$

也就是A去掉一行一列之后的行列式
伴随矩阵:

$$A^* = C^T$$

即代数余子式矩阵的转置
同时我们有

$$A^* = |A| A^{-1}$$

特征多项式:

$$P_A(x) = \det(Ix - A)$$

特征根: 特征多项式的所有n个根(可能有重根).

1.5.6 矩阵树定理, BEST定理

无向图: 设图G的基尔霍夫矩阵$L(G)$等于度数矩阵减去邻接矩阵, 则G的生成树个数等于$L(G)$的任意一个代数余子式的值.
有向图: 类似地定义$L_{in}(G)$等于**入度**矩阵减去邻接矩阵(i指向j有边, 则$A_{i,j} = 1$), $L_{out}(G)$等于**出度**矩阵减去邻接矩阵.
则以i为根的内向树个数即为L_{out}的第i个主子式(即关于第i行第i列的余子式), 外向树个数即为L_{in}的第i个主子式.
(可以看出, 只有无向图才满足$L(G)$的所有代数余子式都相等.)
BEST定理(有向图欧拉回路计数): 如果G是有向欧拉图, 则G的欧拉回路的个数等于以一个任意点为根的内/外向树个数乘以$\prod_v (\deg(v) - 1)!$.
并且在欧拉图里, 无论以哪个结点为根, 也无论内向树还是外向树, 个数都是一样的.
另外无向图欧拉回路计数是NP问题.

1.6 博弈论

1.6.1 SG定理

对于一个**平等**游戏, 可以为每个状态定义一个SG函数.
一个状态的SG函数等于所有它能一步到达的状态的SG函数的mex, 也就是最小的没有出现过的自然数.
那么所有先手必败态的SG函数为0, 先手必胜态的SG函数非0.
如果有一个游戏, 它由若干个独立的子游戏组成, 且每次行动时**只能选一个**子游戏进行操作, 则这个游戏的SG函数就是所有子游戏的SG函数的异或和. (比如最经典的Nim游戏, 每次只能选一堆取若干个石子.)
同时操作多个子游戏的结论参见“经典博弈”部分.

1.6.2 纳什均衡

首先定义纯策略和混合策略: 纯策略是指你一定会选择某个选项, 混合策略是指你对每个选项都有一个概率分布p_i, 你会以相应的概率选择这个选项.
考虑这样的游戏: 有几个人(当然也可以是两个)各自独立地做决定, 然后同时公布每个人的决定, 而每个人的收益和所有人的选择有关.
那么纳什均衡就是每个人都决定一个混合策略, 使得在其他人都是纯策略的情况下, 这个人最坏情况下(也就是说其他人的纯策略最针对他的时候)的收益是最大的. 也就是说, 收益函数对这个人的混合策略求一个偏导, 结果是0(因为是极大值).
纳什均衡点可能存在多个, 不过在一个双人**零和**游戏中, 纳什均衡点一定唯一存在.

1.6.3 经典博弈

1. **阶梯博弈**
   台阶的每层都有一些石子, 每次可以选一层(但不能是第0层), 把任意个石子移到低一层.
   结论: 奇数层的石子数量进行异或和即可.
   实际上只要路径唯一就可以, 比如在树上博弈, 然后石子向根节点方向移动, 那么就是奇数深度的石子数量进行异或和.
2. **可以同时操作多个子游戏**
   如果某个游戏由若干个独立的子游戏组成, 并且每次可以**任意选几个**(当然至少一个)子游戏进行操作, 那么结论是: 所有子游戏都必败时先手才会必败, 否则先手必胜.
3. **每次最多操作k个子游戏(Nim-K)**
   如果每次最多操作k个子游戏, 结论是: 把所有子游戏的SG函数写成二进制表示, 如果每一位上的1个数都是$(k+1)$的倍数, 则先手必败, 否则先手必胜.
   (实际上上面一条可以看做$k = \infty$的情况, 也就是所有SG值都是0时才会先手必败.)
   如果要求整个游戏的SG函数, 就按照上面的方法每个二进制位相加后$\bmod (k+1)$, 视为$(k+1)$进制数后求值即可. (**未验证**)
4. **反Nim游戏(Anti-Nim)**
   和Nim游戏差不多, 唯一的不同是取走最后一个石子的输.
   分两种情况:

   - 所有堆石子个数都是1: 有偶数堆时先手必胜, 否则先手必败.
   - 存在某个堆石子数多于1: 异或和不为0则先手必胜, 否则先手必败.

   当然石子个数实际上就是SG函数, 所以判别条件全都改成SG函数也是一样的.

5. **威佐夫博弈**

有两堆石子，每次要么从一堆中取任意个，要么从两堆中都取走相同数量.

结论：设两堆石子分别有a个和b个，且$a < b$，则先手必败当且仅当$a = \left\lfloor (b-a)\frac{1+\sqrt{5}}{2} \right\rfloor$.

6. **删子树博弈**

有一棵有根树，两个人轮流操作，每次可以选一个点(除了根节点)然后把它的子树都删掉，不能操作的输.

结论：

$$SG(u) = \mathrm{XOR}_{v\in son_u}\left(SG(v)+1\right)$$

1.6.4 例题(待完成)

1.7 自适应Simpson积分

Forked from fstqwq's template.

```cpp
// Adaptive Simpson's method : double simpson::solve (double (*f) (double), double l, double r, double eps) : integrates f over (l, r) with error eps.

double area (double (*f)(double), double l, double r) {
    double m = l + (r - l) / 2;
    return (f(l) + 4 * f(m) + f(r)) * (r - l) / 6;
}

double solve (double (*f) (double), double l, double r, double eps, double a) {
    double m = l + (r - l) / 2;
    double left = area(f, l, m), right = area(f, m, r);
    if (fabs(left + right - a) <= 15 * eps)
        return left + right + (left + right - a) / 15.0;
    return solve(f, l, m, eps / 2, left) + solve(f, m, r, eps / 2, right);
}

double solve (double (*f) (double), double l, double r, double eps) {
    return solve(f, l, r, eps, area (f, l, r));
}
```

1.8 常见数列

查表参见“Miscallous/OEIS”部分.

1.8.1 斐波那契数 卢卡斯数

斐波那契数：$F_0 = 0,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2}$
$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ \ldots$
卢卡斯数：$L_0 = 2,\ L_1 = 1$
$2,\ 1,\ 3,\ 4,\ 7,\ 11,\ 18,\ 29,\ 47,\ 76,\ 123,\ 199,\ \ldots$

通项公式

$\phi = \frac{1+\sqrt{5}}{2},\ \hat{\phi} = \frac{1-\sqrt{5}}{2}$

$F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}},\ L_n = \phi^n + \hat{\phi}^n$

实际上有$\frac{L_n + F_n\sqrt{5}}{2} = \left(\frac{1+\sqrt{5}}{2}\right)^n$，所以求通项的话写一个类然后快速幂就可以同时得到两者.

快速倍增法

$F_{2k} = F_k\left(2F_{k+1} - F_k\right),\ F_{2k+1} = F_{k+1}^2 + F_k^2$

```cpp
pair<int, int> fib(int n) { // 返回F(n)和F(n + 1)
    if (n == 0) return {0, 1};
    auto p = fib(n >> 1);
    int c = p.first * (2 * p.second - p.first);
    int d = p.first * p.first + p.second * p.second;
    if (n & 1)
        return {d, c + d};
    else
        return {c, d};
}
```

1.8.2 伯努利数，自然数幂次和

指数生成函数:$B(x) = \sum_{i\ge 0}\frac{B_i x^i}{i!} = \frac{x}{e^x - 1}$

$$B_n = [n=0] - \sum_{i=0}^{n-1}\binom{n}{i}\frac{B_i}{n-k+1}$$

$$\sum_{i=0}^{n}\binom{n+1}{i}B_i = 0$$

$$S_n(m) = \sum_{i=0}^{m-1} i^n = \sum_{i=0}^{n}\binom{n}{i}B_{n-i}\frac{m^{i+1}}{i+1}$$

$B_0 = 1,\ B_1 = -\frac{1}{2},\ B_4 = -\frac{1}{30},\ B_6 = \frac{1}{42},\ B_8 = -\frac{1}{30},\ \ldots.$
(除了$B_1 = -\frac{1}{2}$以外，伯努利数的奇数项都是0.)
自然数幂次和关于次数的EGF:

$$\begin{aligned} F(x) &= \sum_{k=0}^{\infty}\frac{\sum_{i=0}^{n} i^k}{k!}x^k \\ &= \sum_{i=0}^{n} e^{ix} \\ &= \frac{e^{(n+1)x-1}}{e^x - 1} \end{aligned}$$

1.8.3 分拆数

```cpp
int b = sqrt(n);
ans[0] = tmp[0] = 1;

for (int i = 1; i <= b; ++i) {
    for (int rep = 0; rep < 2; ++rep)
        for (int j = i; j <= n - i * i; ++j)
            add(tmp[j], tmp[j - i]);

    for (int j = i * i; j <= n; ++j)
        add(ans[j], tmp[j - i * i]);
}

// -----

long long a[100010];
long long p[50005]; // 欧拉五边形数定理

int main() {
  p[0] = 1;
  p[1] = 1;
  p[2] = 2;
  int i;
  for (i = 1; i < 50005;
       i++) /*递推式系数1,2,5,7,12,15,22,26...i*(3*i-1)/2,i*(3*i+1)/2*/
  {
    a[2 * i] = i * (i * 3 - 1) / 2; /*五边形数为1,5,12,22...i*(3*i-1)/2*/
    a[2 * i + 1] = i * (i * 3 + 1) / 2;
  }
  for (
      i = 3; i < 50005;
      i++) /*p[n]=p[n-1]+p[n-2]-p[n-5]-p[n-7]+p[12]+p[15]-...+p[n-i*[3i-1]/2]+p[n-i*[3i+1]/2]*/
  {
    p[i] = 0;
    int j;
    for (j = 2; a[j] <= i; j++) /*有可能为负数,式中加1000007*/
    {
      if (j & 2) {
        p[i] = (p[i] + p[i - a[j]] + 1000007) % 1000007;
      } else {
        p[i] = (p[i] - p[i - a[j]] + 1000007) % 1000007;
      }
    }
  }
  int n;
  while (~scanf("%d", &n))
    printf("%lld\n", p[n]);
}
```

1.8.4 斯特林数

第一类斯特林数

$\begin{bmatrix}n\\k\end{bmatrix}$表示$n$个元素划分成$k$个轮换的方案数.
递推式: $\begin{bmatrix}n\\k\end{bmatrix} = \begin{bmatrix}n-1\\k-1\end{bmatrix} + (n-1)\begin{bmatrix}n-1\\k\end{bmatrix}$.
求同一行: 分治FFT $O(n\log^2 n)$, 或者倍增$O(n\log n)$(每次都是$f(x) = g(x)g(x+d)$的形式, 可以用$g(x)$反转之后做一个卷积求出后者).

$$\sum_{k=0}^{n}\begin{bmatrix}n\\k\end{bmatrix}x^k = \prod_{i=0}^{n-1}(x+i)$$

求同一列: 用一个轮换的指数生成函数做k次幂

$$\sum_{n=0}^{\infty}\begin{bmatrix}n\\k\end{bmatrix}\frac{x^n}{n!} = \frac{(\ln(1-x))^k}{k!} = \frac{x^k}{k!}\left(\frac{\ln(1-x)}{x}\right)^k$$

第二类斯特林数

$\begin{Bmatrix}n\\k\end{Bmatrix}$表示$n$个元素划分成$k$个子集的方案数.
递推式: $\begin{Bmatrix}n\\k\end{Bmatrix} = \begin{Bmatrix}n-1\\k-1\end{Bmatrix} + k\begin{Bmatrix}n-1\\k\end{Bmatrix}$.
求一个: 容斥, 狗都会做

$$\begin{Bmatrix}n\\k\end{Bmatrix} = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n = \sum_{i=0}^{k}\frac{(-1)^i}{i!}\frac{(k-i)^n}{(k-i)!}$$

求同一行: FFT, 狗都会做
求同一列: 指数生成函数

$$\sum_{n=0}^{\infty}\begin{Bmatrix}n\\k\end{Bmatrix}\frac{x^n}{n!} = \frac{(e^x-1)^k}{k!} = \frac{x^k}{k!}\left(\frac{e^x-1}{x}\right)^k$$

普通生成函数

$$\sum_{n=0}^{\infty}\begin{Bmatrix}n\\k\end{Bmatrix}x^n = x^k\left(\prod_{i=1}^{k}(1-ix)\right)^{-1}$$

上升幂与普通幂的转换

$$x^{\overline{n}} = \sum_k\begin{bmatrix}n\\k\end{bmatrix}x^k$$

$$x^n = \sum_k\begin{Bmatrix}n\\k\end{Bmatrix}(-1)^{n-k}x^{\overline{k}}$$

下降幂与普通幂的转换

$$x^n = \sum_k\begin{Bmatrix}n\\k\end{Bmatrix}x^{\underline{k}} = \sum_k\binom{x}{k}\begin{Bmatrix}n\\k\end{Bmatrix}k!$$

$$x^{\underline{n}} = \sum_k\begin{bmatrix}n\\k\end{bmatrix}(-1)^{n-k}x^k$$

另外, 多项式的**点值**表示的每项除以阶乘之后卷上e^{-x}乘上阶乘之后是牛顿插值表示, 或者不乘阶乘就是**下降幂**系数表示. 反过来的转换当然卷上e^x就行了. 原理是每次差分等价于乘以$(1-x)$, 展开之后用一次卷积取代多次差分.

1.8.5 贝尔数

$$B_0 = 1,\ B_1 = 1,\ B_2 = 2,\ B_3 = 5,$$
$$B_4 = 15,\ B_5 = 52,\ B_6 = 203, \ldots$$

$$B_n = \sum_{k=0}^{n}\begin{Bmatrix}n\\k\end{Bmatrix}$$

递推式:

$$B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k$$

指数生成函数:

$$B(x) = e^{e^x-1}$$

Touchard同余:

$$B_{n+p} \equiv (B_n + B_{n+1}) \pmod p,\ p \text{ is a prime}$$

1.8.6 卡特兰数, 施罗德数, 默慈金数

卡特兰数

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1}$$

- n个元素按顺序入栈, 出栈序列方案数
- 长为$2n$的合法括号序列数
- $n+1$个叶子的满二叉树个数

递推式:

$$C_n = \sum_{i=0}^{n-1}C_iC_{n-i-1}$$

$$C_n = C_{n-1}\frac{4n-2}{n+1}$$

普通生成函数:

$$C(x) = \frac{1-\sqrt{1-4x}}{2x}$$

扩展: 如果有n个左括号和m个右括号, 方案数为

$$\binom{n+m}{n} - \binom{n+m}{m-1}$$

施罗德数

$$S_n = S_{n-1} + \sum_{i=0}^{n-1}S_iS_{n-i-1}$$

$$(n+1)s_n = (6n-3)s_{n-1} - (n-2)s_{n-2}$$

其中S_n是(大)施罗德数, s_n是小施罗德数(也叫超级卡特兰数).
除了$S_0 = s_0 = 1$以外, 都有$S_i = 2s_i$.
施罗德数的组合意义:

- 从$(0,0)$走到(n,n), 每次可以走右, 上, 或者右上一步, 并且不能超过$y = x$这条线的方案数
- 长为n的括号序列, 每个位置也可以为空, 并且括号对数和空位置数加起来等于n的方案数
- 凸n边形的**任意**剖分方案数

(有些人会把大(而不是小)施罗德数叫做超级卡特兰数.)

默慈金数

$$M_{n+1} = M_n + \sum_{i=0}^{n-1} M_i M_{n-1-i} = \frac{(2n+3)M_n + 3nM_{n-1}}{n+3}$$

$$M_n = \sum_{i=0}^{\frac{n}{2}} \binom{n}{2i} C_i$$

在圆上的n个**不同的**点之间画任意条不相交(包括端点)的弦的方案数.

也等价于在网格图上, 每次可以走右上, 右下, 正右方一步, 且不能走到$y < 0$的位置, 在此前提下从$(0, 0)$走到$(n, 0)$的方案数.

扩展: 默慈金数画的弦不可以共享端点. 如果可以共享端点的话是A054726, 后面的表里可以查到.

1.9 常用公式及结论

1.9.1 方差

m个数的方差:

$$s^2 = \frac{\sum_{i=1}^m x_i^2}{m} - \overline{x}^2$$

随机变量的方差: $D^2(x) = E(x^2) - E^2(x)$

1.9.2 单位根反演(展开整除条件$[n|k]$)

$$[n|k] = \frac{1}{n} \sum_{i=0}^{n-1} \omega_n^{ik}$$

$$\sum_{i\ge 0} [x^{ik}] f(x) = \frac{1}{k} \sum_{j=0}^{k-1} f(\omega_k^j)$$

1.9.3 康托展开(排列的排名)

求排列的排名: 先对每个数都求出它后面有几个数比它小(可以用树状数组预处理), 记为c_i, 则排列的排名就是

$$\sum_{i=1}^n c_i (n-i)!$$

已知排名构造排列: 从前到后先分别求出c_i, 有了c_i之后再用一个平衡树(需要维护排名)倒序处理即可.

1.9.4 连通图计数

设大小为n的满足一个限制P的简单无向图数量为g_n, 满足限制P且连通的简单无向图数量为f_n, 如果已知$g_{1\dots n}$求f_n, 可以得到递推式

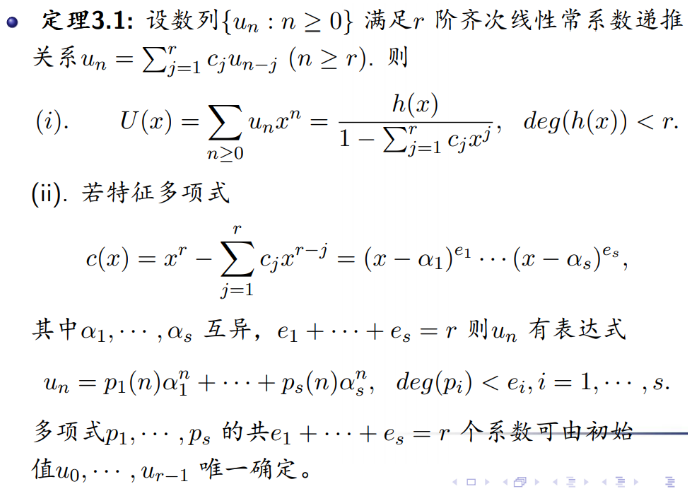
$$f_n = g_n - \sum_{k=1}^{n-1} \binom{n-1}{k-1} f_k g_{n-k}$$

这个递推式的意义就是用任意图的数量减掉不连通的数量, 而不连通的数量可以通过枚举1号点所在连通块大小来计算.

注意, 由于$f_0 = 0$, 因此递推式的枚举下界取0和1都是可以的.

推一推式子会发现得到一个多项式求逆, 再仔细看看, 其实就是一个多项式ln.

1.9.5 线性齐次线性常系数递推求通项



- **定理3.1:** 设数列$\{u_n : n \ge 0\}$ 满足r 阶齐次线性常系数递推关系$u_n = \sum_{j=1}^r c_j u_{n-j}$ $(n \ge r)$. 则

(i). $$U(x) = \sum_{n\ge 0} u_n x^n = \frac{h(x)}{1 - \sum_{j=1}^r c_j x^j}, \quad deg(h(x)) < r.$$

(ii). 若特征多项式

$$c(x) = x^r - \sum_{j=1}^r c_j x^{r-j} = (x-\alpha_1)^{e_1} \cdots (x-\alpha_s)^{e_s},$$

其中$\alpha_1, \cdots, \alpha_s$ 互异, $e_1 + \cdots + e_s = r$ 则u_n 有表达式

$$u_n = p_1(n)\alpha_1^n + \cdots + p_s(n)\alpha_s^n, \quad deg(p_i) < e_i, i = 1, \cdots, s.$$

多项式$p_1, \cdots, p_s$ 的共$e_1 + \cdots + e_s = r$ 个系数可由初始值$u_0, \cdots, u_{r-1}$ 唯一确定。

1.10 常用生成函数变换

$$\frac{x}{(1-x)^2} = \sum_{i\ge 0} i x^i$$

$$\frac{1}{(1-x)^k} = \sum_{i\ge 0} \binom{i+k-1}{i} x^i = \sum_{i\ge 0} \binom{i+k-1}{k-1} x^i, \ k > 0$$

$$\begin{aligned} \sum_{i=0}^{\infty} i^n x^i &= \sum_{k=0}^n \left\{ n \atop k \right\} k! \frac{x^k}{(1-x)^{k+1}} = \sum_{k=0}^n \left\{ n \atop k \right\} k! \frac{x^k (1-x)^{n-k}}{(1-x)^{n+1}} \\ &= \frac{1}{(1-x)^{n+1}} \sum_{i=0}^n \frac{x^i}{(n-i)!} \sum_{k=0}^i \left\{ n \atop k \right\} k! (n-k)! \frac{(-1)^{i-k}}{(i-k)!} \end{aligned}$$

(用上面的方法可以把分子化成一个n次以内的多项式, 并且可以用一次卷积求出来.)

如果把i^n换成任意的一个n次多项式, 那么我们可以求出它的下降幂表示形式(或者说是牛顿插值)的系数r_i, 发现用r_k替换掉上面的$\left\{ n \atop k \right\} k!$之后其余过程完全相同.

2. 数论

2.1 $O(n)$预处理逆元

```cpp
// 要求p为质数

inv[0] = inv[1] = 1;
for (int i = 2; i <= n; i++)
    inv[i] = (long long)(p - (p / i)) * inv[p % i] % p; // p为模数
// i ^ -1 = -(p / i) * (p % i) ^ -1
```

2.2 线性筛

```cpp
// 此代码以计算约数之和函数\sigma_1(对10^9+7取模)为例
// 适用于任何f(p^k)便于计算的积性函数
constexpr int p = 1000000007;

int prime[maxn / 10], sigma_one[maxn], f[maxn], g[maxn];
// f: 除掉最小质因子后剩下的部分
// g: 最小质因子的幂次, 在f(p^k)比较复杂时很有用, 但f(p^k)可以递推时就可以省略了
// 这里没有记录最小质因子, 但根据线性筛的性质, 每个合数只会被它最小的质因子筛掉
bool notp[maxn]; // 顾名思义

void get_table(int n) {
    sigma_one[1] = 1; // 积性函数必有f(1) = 1
    for (int i = 2; i <= n; i++) {
        if (!notp[i]) { // 质数情况
            prime[++prime[0]] = i;
            sigma_one[i] = i + 1;
            f[i] = g[i] = 1;
        }

        for (int j = 1; j <= prime[0] && i * prime[j] <= n; j++) {
            notp[i * prime[j]] = true;

            if (i % prime[j]) { // 加入一个新的质因子, 这种情况很简单
                sigma_one[i * prime[j]] = (long long)sigma_one[i] * (prime[j] + 1) % p;
                f[i * prime[j]] = i;
                g[i * prime[j]] = 1;
            }
            else { // 再加入一次最小质因子,需要再进行分类讨论
                f[i * prime[j]] = f[i];
                g[i * prime[j]] = g[i] + 1;
                // 对于f(p^k)可以直接递推的函数, 这里的判断可以改成
                // i / prime[j] % prime[j] != 0, 这样可以省下f[]的空间,
                // 但常数很可能会稍大一些

                if (f[i] == 1) // 质数的幂次, 这里\sigma_1可以递推
                    sigma_one[i * prime[j]] = (sigma_one[i] + i * prime[j]) % p;
                    // 对于更一般的情况,可以借助g[]计算f(p^k)
                else sigma_one[i * prime[j]] = // 否则直接利用积性, 两半乘起来
                    (long long)sigma_one[i * prime[j] / f[i]] * sigma_one[f[i]] % p;
                break;
            }
        }
    }
}
```

2.3 杜教筛

```cpp
// 用于求可以用狄利克雷卷积构造出好求和的东西的函数的前缀和(有点绕)
// 有些题只要求n <= 10 ^ 9, 这时就没必要开long long了, 但记得乘法时强转

//常量/全局变量/数组定义
const int maxn = 5000005, table_size = 5000000, p = 1000000007, inv_2 = (p + 1) / 2;
bool notp[maxn];
int prime[maxn / 20], phi[maxn], tbl[100005];
// tbl用来顶替哈希表, 其实开到n ^ {1 / 3}就够了, 不过保险起见开成\sqrt n比较好
long long N;

// 主函数前面加上这么一句
memset(tbl, -1, sizeof(tbl));

// 线性筛预处理部分略去

// 杜教筛主过程 总计O(n ^ {2 / 3})
// 递归调用自身
// 递推式还需具体情况具体分析,这里以求欧拉函数前缀和(mod 10 ^ 9 + 7)为例
int S(long long n) {
    if (n <= table_size)
        return phi[n];
    else if (~tbl[N / n])
        return tbl[N / n];
    // 原理: n除以所有可能的数的结果一定互不相同

    int ans = 0;
    for (long long i = 2, last; i <= n; i = last + 1) {
        last = n / (n / i);
        ans = (ans + (last - i + 1) % p * S(n / i)) % p; // 如果n是int范围的话记得强转
    }

    ans = (n % p * ((n + 1) % p) % p * inv_2 - ans + p) % p; // 同上
    return tbl[N / n] = ans;
}
```

2.4 Powerful Number筛

注意Powerful Number筛只能求积性函数的前缀和.

本质上就是构造一个方便求前缀和的函数, 然后做类似杜教筛的操作.

定义Powerful Number表示每个质因子幂次都大于1的数, 显然最多有$\sqrt{n}$个.

设我们要求和的函数是$f(n)$, 构造一个方便求前缀和的**积性**函数$g(n)$使得$g(p) = f(p)$.

那么就存在一个积性函数$h = f * g^{-1}$, 也就是$f = g * h$. 可以证明$h(p) = 0$, 所以只有Powerful Number的h值不为0.

$$S_f(i) = \sum_{d=1}^{n} h(d) S_g\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$$

只需要枚举每个Powerful Number作为d, 然后用杜教筛计算g的前缀和.

求$h(d)$时要先预处理$h(p^k)$, 显然有

$$h\left(p^k\right) = f\left(p^k\right) - \sum_{i=1}^{k} g\left(p^i\right) h\left(p^{k-i}\right)$$

处理完之后DFS就行了.（显然只需要筛$\sqrt{n}$以内的质数.）

复杂度取决于杜教筛的复杂度，特殊题目构造的好也可以做到$O\left(\sqrt{n}\right)$.

例题:

- $f\left(p^k\right) = p^k\left(p^k - 1\right) : g(n) = \mathrm{id}(n)\varphi(n)$.
- $f\left(p^k\right) = p \operatorname{xor} k : n$为偶数时$g(n) = 3\varphi(n)$, 否则$g(n) = \varphi(n)$.

2.5 洲阁筛

计算积性函数$f(n)$的前n项之和时，我们可以把所有项按照是否有$> \sqrt{n}$的质因子分两类讨论，最后将两部分的贡献加起来即可.

1. **有$> \sqrt{n}$的质因子**

显然$> \sqrt{n}$的质因子幂次最多为1, 所以这一部分的贡献就是

$$\sum_{i=1}^{\sqrt{n}} f(i) \sum_{d=\sqrt{n}+1}^{\left\lfloor \frac{n}{i} \right\rfloor} [d \in \mathbb{P}]\, f(d)$$

我们可以DP后面的和式. 由于$f(p)$是一个关于p的低次多项式，我们可以对每个次幂分别DP: 设$g_{i,j}$表示$[1, j]$中和前i个质数都互质的数的k次方之和. 设$\sqrt{n}$以内的质数总共有m个, 显然贡献就转换成了

$$\sum_{i=1}^{\sqrt{n}} i^k g_{m, \left\lfloor \frac{n}{i} \right\rfloor}$$

边界显然就是自然数幂次和, 转移是

$$g_{i,j} = g_{i-1,j} - p_i^k g_{i-1, \left\lfloor \frac{j}{p_i} \right\rfloor}$$

也就是减掉和第i个质数不互质的贡献.

在滚动数组的基础上再优化一下：首先如果$j < p_i$那肯定就只有1一个数；如果$p_i \le j < p_i^2$，显然就有$g_{i,j} = g_{i-1,j} - p_i^k$，那么对每个$j$记下最大的$i$使得$p_i^2 \le j$, 比这个还大的情况就不需要递推了，用到的时候再加上一个前缀和解决.

2. **所有质因子都$\le \sqrt{n}$**

类似的道理, 我们继续DP: $h_{i,j}$表示只含有第i到m个质数作为质因子的所有数的$f(i)$之和.（这里不需要对每个次幂单独DP了；另外倒着DP是为了方便卡上限.）

边界显然是$h_{m+1,j} = 1$, 转移是

$$h_{i,j} = h_{i+1,j} + \sum_{c} f(p_i^c) h_{i+1, \left\lfloor \frac{j}{p_i^c} \right\rfloor}$$

跟上面一样的道理优化，分成三段：$j < p_i$时$h_{i,j} = 1$，$j < p_i^2$时$h_{i,j} = h_{i+1,j} + f(p_i)$(同样用前缀和解决)，再小的部分就老实递推.

预处理$\sqrt{n}$以内的部分之后跑两次DP, 最后把两部分的贡献加起来就行了.

两部分的复杂度都是$\Theta\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$的.

以下代码以洛谷P5325$(f(p^k) = p^k(p^k - 1))$为例.

```cpp
constexpr int maxn = 200005, p = 1000000007;

long long N, val[maxn]; // 询问的n和存储所有整除结果的表
int sqrtn;

inline int getid(long long x) {
    if (x <= sqrtn)
        return x;

    return val[0] - N / x + 1;
}

bool notp[maxn];
int prime[maxn], prime_cnt, rem[maxn]; // 线性筛用数组

int f[maxn], pr[maxn], g[2][maxn], dp[maxn];
int l[maxn], r[maxn];

// 线性筛省略

inline int get_sum(long long n, int k) {
    n %= p;

    if (k == 1)
        return n * (n + 1) % p * ((p + 1) / 2) % p;

    else
        return n * (n + 1) % p * (2 * n + 1) % p * ((p + 1) / 6) % p;
}

void get_dp(long long n, int k, int *dp) {
    for (int j = 1; j <= val[0]; j++)
        dp[j] = get_sum(val[j], k);

    for (int i = 1; i <= prime_cnt; i++) {
        long long lb = (long long)prime[i] * prime[i];
        int pw = (k == 1 ? prime[i] : (int)(lb % p));

        pr[i] = (pr[i - 1] + pw) % p;

        for (int j = val[0]; j && val[j] >= lb; j--) {
            int t = getid(val[j] / prime[i]);

            int tmp = dp[t];
            if (l[t] < i)
                tmp = (tmp - pr[min(i - 1, r[t])] + pr[l[t]]) % p;

            dp[j] = (dp[j] - (long long)pw * tmp) % p;
            if (dp[j] < 0)
                dp[j] += p;
        }
    }

    for (int j = 1; j <= val[0]; j++) {
        dp[j] = (dp[j] - pr[r[j]] + pr[l[j]]) % p;

        dp[j] = (dp[j] + p - 1) % p; // 因为DP数组是有1的，但后面计算不应该有1
    }
}

int calc1(long long n) {
    get_dp(n, 1, g[0]);
    get_dp(n, 2, g[1]);

    int ans = 0;

    for (int i = 1; i <= sqrtn; i++)
        ans = (ans + (long long)f[i] * (g[1][getid(N / i)] - g[0][getid(N / i)])) % p;

    if (ans < 0)
        ans += p;

    return ans;
}

int calc2(long long n) {
    for (int j = 1; j <= val[0]; j++)
        dp[j] = 1;

    for (int i = 1; i <= prime_cnt; i++)
```

```
        pr[i] = (pr[i - 1] + f[prime[i]]) % p;

    for (int i = prime_cnt; i; i--) {
        long long lb = (long long)prime[i] * prime[i];

        for (int j = val[0]; j && val[j] >= lb; j--)
            for (long long pc = prime[i]; pc <= val[j]; pc *= prime[i]) {
                int t = getid(val[j] / pc);

                int tmp = dp[t];
                if (r[t] > i)
                    tmp = (tmp + pr[r[t]] - pr[max(i, l[t])]) % p;

                dp[j] = (dp[j] + pc % p * ((pc - 1) % p) % p * tmp) % p;
            }
    }

    return (long long)(dp[val[0]] + pr[r[val[0]]] - pr[l[val[0]]] + p) % p;
}

int main() {

    // ios::sync_with_stdio(false);

    cin >> N;

    sqrtn = (int)sqrt(N);

    get_table(sqrtn);

    for (int i = 1; i <= sqrtn; i++)
        val[++val[0]] = i;

    for (int i = 1; i <= sqrtn; i++)
        val[++val[0]] = N / i;

    sort(val + 1, val + val[0] + 1);

    val[0] = unique(val + 1, val + val[0] + 1) - val - 1;

    int li = 0, ri = 0;
    for (int j = 1; j <= val[0]; j++) {
        while (ri < prime_cnt && prime[ri + 1] <= val[j])
            ri++;

        while (li <= prime_cnt && (long long)prime[li] * prime[li] <= val[j])
            li++;

        l[j] = li - 1;
        r[j] = ri;
    }

    cout << (calc1(N) + calc2(N)) % p << endl;

    return 0;
}
```

2.6 离散对数(BSGS)(待完成)

2.7 Miller-Rabin

```
// 复杂度可以认为是常数

// 封装好的函数体
// 需要调用check
bool Miller_Rabin(long long n) {
    if (n == 1)
        return false;
    if (n == 2)
        return true;
    if (n % 2 == 0)
        return false;

    for (int i : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (i >= n)
            break;
        if (!check(n, i))
            return false;
    }

    return true;
}

// 用一个数检测
// 需要调用long long快速幂和O(1)快速乘
bool check(long long n, long long b) { // b: base
    long long a = n - 1;
    int k = 0;

    while (a % 2 == 0) {
        a /= 2;
        k++;
    }

    long long t = qpow(b, a, n); // 这里的快速幂函数需要写O(1)快速乘
    if (t == 1 || t == n - 1)
        return true;

    while (k--) {
        t = mul(t, t, n); // mul是O(1)快速乘函数
        if(t == n - 1)
            return true;
    }

    return false;
}
```

2.8 Pollard's Rho

```
// 注意,虽然Pollard's Rho的理论复杂度是O(n ^ {1 / 4})的,
// 但实际跑起来比较慢，一般用于做long long范围内的质因数分解



// 封装好的函数体
// 需要调用solve
void factorize(long long n, vector<long long> &v) { // v用于存分解出来的质因子，重复的会放多个
    for (int i : {2, 3, 5, 7, 11, 13, 17, 19})
        while (n % i == 0) {
            v.push_back(i);
            n /= i;
        }

    solve(n, v);
    sort(v.begin(), v.end()); // 从小到大排序后返回
}

// 递归过程
// 需要调用Pollard's Rho主过程，同时递归调用自身
void solve(long long n, vector<long long> &v) {
    if (n == 1)
        return;

    long long p;
    do
        p = Pollards_Rho(n);
    while (!p); // p是任意一个非平凡因子

    if (p == n) {
        v.push_back(p); // 说明n本身就是质数
        return;
    }

    solve(p, v); // 递归分解两半
    solve(n / p, v);
}

// Pollard's Rho主过程
// 需要使用Miller-Rabin作为子算法
// 同时需要调用O(1)快速乘和gcd函数
long long Pollards_Rho(long long n) {
    // assert(n > 1);

    if (Miller_Rabin(n))
        return n;

    long long c = rand() % (n - 2) + 1, i = 1, k = 2, x = rand() % (n - 3) + 2, u = 2; // 注意这里rand函数需要重定义一下
    while (true) {
        i++;
        x = (mul(x, x, n) + c) % n; // mul是O(1)快速乘函数

        long long g = gcd((u - x + n) % n, n);
        if (g > 1 && g < n)
            return g;

        if (u == x)
            return 0; // 失败, 需要重新调用

        if (i == k) {
            u = x;
            k *= 2;
        }
    }
}
```

2.9 快速阶乘算法

参见“数学/多项式”部分.

2.10 扩展欧几里德

```
void exgcd(LL a, LL b, LL &c, LL &x, LL &y) {
    if (b == 0) {
        c = a;
        x = 1;
        y = 0;
        return;
    }

    exgcd(b, a % b, c, x, y);

    LL tmp = x;
    x = y;
    y = tmp - (a / b) * y;
```

2.10.1 求通解的方法

假设我们已经找到了一组解(p_0, q_0)满足$ap_0 + bq_0 = \gcd(a, b)$, 那么其他的解都满足

$$p = p_0 + \frac{b}{\gcd(p, q)} \times t \qquad q = q_0 - \frac{a}{\gcd(p, q)} \times t$$

其中t为任意整数.

2.10.2 类欧几里德算法(直线下整点个数)

$a, b \ge 0, m > 0$, 计算$\sum_{i=0}^{n-1} \left\lfloor \frac{a+bi}{m} \right\rfloor$.

```
int solve(int n, int a, int b, int m) {
    if (!b)
        return n * (a / m);
    if (a >= m)
        return n * (a / m) + solve(n, a % m, b, m);
    if (b >= m)
        return (n - 1) * n / 2 * (b / m) + solve(n, a, b % m, m);

    return solve((a + b * n) / m, (a + b * n) % m, m, b);
}
```

2.11 中国剩余定理(待完成)

2.11.1 ex-CRT(待完成)

2.12 组合数取模(ex-Lucas)(待完成)

2.13 原根 阶

阶: 最小的整数k使得$a^k \equiv 1 \pmod p$, 记为$\delta_p(a)$.
显然a在原根以下的幂次是两两不同的.
一个性质: 如果a, b均与p互质, 则 $\delta_p(ab) = \delta_p(a)\delta_p(b)$ 的充分必要条件是$\gcd\left(\delta_p(a), \delta_p(b)\right) = 1$.
另外, 如果a与p互质, 则有$\delta_p(a^k) = \dfrac{\delta_p(a)}{\gcd\left(\delta_p(a), k\right)}$. (也就是环上一次跳$k$步的周期.)
原根: 阶等于$\varphi(p)$的数.
只有形如$2, 4, p^k, 2p^k$(p是奇素数)的数才有原根, 并且如果一个数n有原根, 那么原根的个数是$\varphi(\varphi(n))$个.
暴力找原根代码:

```
def split(n): # 分解质因数
    i = 2
    a = []
    while i * i <= n:
        if n % i == 0:
            a.append(i)

            while n % i == 0:
                n /= i

        i += 1

    if n > 1:
        a.append(n)

    return a

def getg(p): # 找原根
    def judge(g):
        for i in d:
            if pow(g, (p - 1) / i, p) == 1:
                return False
        return True

    d = split(p - 1)
    g = 2

    while not judge(g):
        g += 1

    return g

print(getg(int(input())))
```

2.14 常用公式

2.14.1 莫比乌斯反演

$$f(n)=\sum_{d|n}g(d) \Leftrightarrow g(n)=\sum_{d|n}\mu\left(\frac{n}{d}\right)f(d)$$

$$f(d)=\sum_{d|k}g(k) \Leftrightarrow g(d)=\sum_{d|k}\mu\left(\frac{k}{d}\right)f(k)$$

2.14.2 其他常用公式

$$\mu * I = e \quad (e(n)=[n=1])$$

$$\varphi * I = id$$

$$\mu * id = \varphi$$

$$\sigma_0 = I * I,\, \sigma_1 = id * I,\, \sigma_k = id^{k-1} * I$$

$$\sum_{i=1}^{n}[(i,n)=1]\, i = n\frac{\varphi(n)+e(n)}{2}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{i}[(i,j)=d] = S_{\varphi}\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}[(i,j)=d] = \sum_{d|k}\mu\left(\frac{k}{d}\right)\left\lfloor\frac{n}{k}\right\rfloor\left\lfloor\frac{m}{k}\right\rfloor$$

$$\sum_{i=1}^{n}f(i)\sum_{j=1}^{\left\lfloor\frac{n}{i}\right\rfloor}g(j) = \sum_{i=1}^{n}g(i)\sum_{j=1}^{\left\lfloor\frac{n}{i}\right\rfloor}f(j)$$

3. 图论

3.1 最小生成树

3.1.1 Boruvka算法

思想：每次选择连接每个连通块的最小边，把连通块缩起来.
每次连通块个数至少减半，所以迭代$O(\log n)$次即可得到最小生成树.
一种比较简单的实现方法：每次迭代遍历所有边，用并查集维护连通性和每个连通块的最小边权.
应用：最小异或生成树

3.1.2 动态最小生成树

```cpp
// 动态最小生成树的离线算法比较容易,而在线算法通常极为复杂
// 一个跑得比较快的离线做法是对时间分治,在每层分治时找出一定在/不在MST上的边,只带着不确定边继续递归
// 简单起见,找确定边的过程用Kruskal算法实现,过程中的两种重要操作如下:
// - Reduction:待修改边标为+INF,跑MST后把非树边删掉,减少无用边
// - Contraction:待修改边标为-INF,跑MST后缩除待修改边之外的所有MST边,计算必须边
// 每轮分治需要Reduction-Contraction,借此减少不确定边,从而保证复杂度
// 复杂度证明:假设当前区间有k条待修改边,n和m表示点数和边数,那么最坏情况下R-C的效果为(n, m) -> (n, n + k - 1) -> (k + 1, 2k)


// 全局结构体与数组定义
struct edge { //边的定义
    int u, v, w, id; // id表示边在原图中的编号
    bool vis; // 在Kruskal时用,记录这条边是否是树边
    bool operator < (const edge &e) const { return w < e.w; }
} e[20][maxn], t[maxn]; // 为了便于回滚,在每层分治存一个副本


// 用于存储修改的结构体,表示第id条边的权值从u修改为v
struct A {
    int id, u, v;
} a[maxn];


int id[20][maxn]; // 每条边在当前图中的编号
int p[maxn], size[maxn], stk[maxn], top; // p和size是并查集数组,stk是用来撤销的栈
int n, m, q; // 点数,边数,修改数


// 方便起见,附上可能需要用到的预处理代码
for (int i = 1; i <= n; i++) { // 并查集初始化
    p[i] = i;
    size[i] = 1;
}

for (int i = 1; i <= m; i++) { // 读入与预标号
    scanf("%d%d%d", &e[0][i].u, &e[0][i].v, &e[0][i].w);
    e[0][i].id = i;
    id[0][i] = i;
}

for (int i = 1; i <= q; i++) { // 预处理出调用数组
    scanf("%d%d", &a[i].id, &a[i].v);
    a[i].u = e[0][a[i].id].w;
    e[0][a[i].id].w = a[i].v;
}

for(int i = q; i; i--)
    e[0][a[i].id].w = a[i].u;

CDQ(1, q, 0, m, 0); // 这是调用方法


// 分治主过程 O(nlog^2n)
// 需要调用Reduction和Contraction
void CDQ(int l, int r, int d, int m, long long ans) { // CDQ分治
    if (l == r) { // 区间长度已减小到1,输出答案,退出
        e[d][id[d][a[l].id]].w = a[l].v;
        printf("%lld\n", ans + Kruskal(m, e[d]));
        e[d][id[d][a[l].id]].w=a[l].u;
        return;
    }

    int tmp = top;

    Reduction(l, r, d, m);
    ans += Contraction(l, r, d, m); // R-C

    int mid = (l + r) / 2;

    copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
    for (int i = 1; i <= m; i++)
        id[d + 1][e[d][i].id] = i; // 准备好下一层要用的数组

    CDQ(l, mid, d + 1, m, ans);

    for (int i = l; i <= mid; i++)
        e[d][id[d][a[i].id]].w = a[i].v; // 进行左边的修改

    copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
    for (int i = 1; i <= m; i++)
        id[d + 1][e[d][i].id] = i; // 重新准备下一层要用的数组

    CDQ(mid + 1, r, d + 1, m, ans);

    for (int i = top; i > tmp; i--)
        cut(stk[i]);//撤销所有操作
    top = tmp;
}


// Reduction(减少无用边):待修改边标为+INF,跑MST后把非树边删掉,减少无用边
// 需要调用Kruskal
void Reduction(int l, int r, int d, int &m) {
    for (int i = l; i <= r; i++)
        e[d][id[d][a[i].id]].w = INF;//待修改的边标为INF

    Kruskal(m, e[d]);

    copy(e[d] + 1, e[d] + m + 1, t + 1);

    int cnt = 0;
    for (int i = 1; i <= m; i++)
        if (t[i].w == INF || t[i].vis){ // 非树边扔掉
            id[d][t[i].id] = ++cnt; // 给边重新编号
            e[d][cnt] = t[i];
        }

    for (int i = r; i >= l; i--)
        e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改回去

    m=cnt;
}


// Contraction(缩必须边):待修改边标为-INF,跑MST后缩除待修改边之外的所有树边
// 返回缩掉的边的总权值
// 需要调用Kruskal
long long Contraction(int l, int r, int d, int &m) {
    long long ans = 0;

    for (int i = l; i <= r; i++)
        e[d][id[d][a[i].id]].w = -INF; // 待修改边标为-INF

    Kruskal(m, e[d]);
    copy(e[d] + 1, e[d] + m + 1, t + 1);

    int cnt = 0;
    for (int i = 1; i <= m ; i++) {

        if (t[i].w != -INF && t[i].vis) { // 必须边
            ans += t[i].w;
            mergeset(t[i].u, t[i].v);
        }
        else { // 不确定边
            id[d][t[i].id]=++cnt;
            e[d][cnt]=t[i];
        }
    }

    for (int i = r ; i >= l; i--) {
        e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改回去
        e[d][id[d][a[i].id]].vis = false;
    }

    m = cnt;

    return ans;
}


// Kruskal算法 O(mlogn)
// 方便起见,这里直接沿用进行过缩点的并查集,在过程结束后撤销即可
long long Kruskal(int m, edge *e) {
    int tmp = top;
    long long ans = 0;

    sort(e + 1, e + m + 1); // 比较函数在结构体中定义过了

    for (int i = 1; i <= m; i++) {
        if (findroot(e[i].u) != findroot(e[i].v)) {
            e[i].vis = true;
            ans += e[i].w;
            mergeset(e[i].u, e[i].v);
        }
        else
            e[i].vis = false;
    }

    for(int i = top; i > tmp; i--)
        cut(stk[i]); // 撤销所有操作
    top = tmp;

    return ans;
}


// 以下是并查集相关函数
int findroot(int x) { // 因为需要撤销,不写路径压缩
    while (p[x] != x)
        x = p[x];

    return x;
}

void mergeset(int x, int y) { // 按size合并,如果想跑得更快就写一个按秩合并
    x = findroot(x); // 但是按秩合并要再开一个栈记录合并之前的秩
    y = findroot(y);

    if (x == y)
        return;

    if (size[x] > size[y])
        swap(x, y);

    p[x] = y;
    size[y] += size[x];
    stk[++top] = x;
}

void cut(int x) { // 并查集撤销
    int y = x;

    do
        size[y = p[y]] -= size[x];
    while (p[y]! = y);

    p[x] = x;
}
```

3.1.3 最小树形图

对每个点找出最小的入边，如果是一个DAG那么就已经结束了.
否则把环都缩起来，每个点的边权减去环上的边权之后再跑一遍，直到没有环为止.
可以用可并堆优化到$O(m\log n)$，需要写一个带懒标记的左偏树.

$O(nm)$版本

```
constexpr int maxn = 105, maxe = 10005, inf = 0x3f3f3f3f;

struct edge {
    int u, v, w;
} e[maxe];

int mn[maxn], pr[maxn], ufs[maxn], vis[maxn];
bool alive[maxn];

int edmonds(int n, int m, int rt) {
    for (int i = 1; i <= n; i++)
        alive[i] = true;

    int ans = 0;

    while (true) {
        memset(mn, 63, sizeof(int) * (n + 1));
        memset(pr, 0, sizeof(int) * (n + 1));
        memset(ufs, 0, sizeof(int) * (n + 1));
        memset(vis, 0, sizeof(int) * (n + 1));

        mn[rt] = 0;

        for (int i = 1; i <= m; i++)
            if (e[i].u != e[i].v && e[i].w < mn[e[i].v]) {
                mn[e[i].v] = e[i].w;
                pr[e[i].v] = e[i].u;
            }

        for (int i = 1; i <= n; i++)
            if (alive[i]) {
                if (mn[i] >= inf)
                    return -1; // 不存在最小树形图

                ans += mn[i];
            }

        bool flag = false;

        for (int i = 1; i <= n; i++) {
            if (!alive[i])
                continue;

            int x = i;
            while (x && !vis[x]) {
                vis[x] = i;
                x = pr[x];
            }

            if (x && vis[x] == i) {
                flag = true;
                for (int u = x; !ufs[u]; u = pr[u])
                    ufs[u] = x;
            }
        }

        for (int i = 1; i <= m; i++) {
            e[i].w -= mn[e[i].v];

            if (ufs[e[i].u])
                e[i].u = ufs[e[i].u];
            if (ufs[e[i].v])
                e[i].v = ufs[e[i].v];
        }

        if (!flag)
            return ans;

        for (int i = 1; i <= n; i++)
            if (ufs[i] && i != ufs[i])
                alive[i] = false;
    }
}
```

$O(m\log n)$版本

(堆优化版本可以参考fstqwq的模板，在最后没有目录的部分.)

3.1.4 Steiner Tree 斯坦纳树

问题：一张图上有k个关键点，求让关键点两两连通的最小生成树

做法：状压DP，$f_{i,S}$表示以i号点为树根，i与S中的点连通的最小边权和

转移有两种：

1. 枚举子集：

$$f_{i,S}=\min_{T\subset S}\left\{f_{i,T}+f_{i,S\setminus T}\right\}$$

2. 新加一条边：

$$f_{i,S}=\min_{(i,j)\in E}\left\{f_{j,S}+w_{i,j}\right\}$$

第一种直接枚举子集DP就行了，第二种可以用SPFA或者Dijkstra松弛(显然负边一开始全选就行了，所以只需要处理非负边).

复杂度$O(n3^k+2^k\text{SSSP}(n,m))$，其中$\text{SSSP}(n,m)$可以是$nm$或者$n^2+m$或者$m\log n$.

```
constexpr int maxn = 105, inf = 0x3f3f3f3f;

int dp[maxn][(1 << 10) + 1];
int g[maxn][maxn], a[15];
bool inq[maxn];

int main() {

    int n, m, k;
    scanf("%d%d%d", &n, &m, &k);

    memset(g, 63, sizeof(g));

    while (m--) {
        int u, v, c;
        scanf("%d%d%d", &u, &v, &c);

        g[u][v] = g[v][u] = min(g[u][v], c); // 不要忘了是双向边
    }

    memset(dp, 63, sizeof(dp));

    for (int i = 0; i < k; i++) {
        scanf("%d", &a[i]);

        dp[a[i]][1 << i] = 0;
    }

    for (int s = 1; s < (1 << k); s++) {
        for (int i = 1; i <= n; i++)
            for (int t = (s - 1) & s; t; (--t) &= s)
                dp[i][s] = min(dp[i][s], dp[i][t] + dp[i][s ^ t]);

        // SPFA
        queue<int> q;
        for (int i = 1; i <= n; i++)
            if (dp[i][s] < inf) {
                q.push(i);
                inq[i] = true;
            }

        while (!q.empty()) {
            int i = q.front();
            q.pop();
            inq[i] = false; // 最终结束时inq一定全0，所以不用清空

            for (int j = 1; j <= n; j++)
                if (dp[i][s] + g[i][j] < dp[j][s]) {
                    dp[j][s] = dp[i][s] + g[i][j];
                    if (!inq[j]) {
                        q.push(j);
                        inq[j] = true;
                    }
                }
        }
    }

    int ans = inf;
    for (int i = 1; i <= n; i++)
        ans = min(ans, dp[i][(1 << k) - 1]);

    printf("%d\n", ans);

    return 0;
}
```

3.1.5 最小直径生成树

首先要找到图的绝对中心(可能在点上，也可能在某条边上)，然后以绝对中心为起点建最短路树就是最小直径生成树.

3.2 最短路

3.2.1 Dijkstra

见k短路(注意那边是求到t的最短路)

3.2.2 Johnson算法(负权图多源最短路)

首先前提是图没有负环.

先任选一个起点s，跑一边SPFA，计算每个点的势$h_u = d_{s,u}$，然后将每条边$u \to v$的权值w修改为$w + h[u] - h[v]$即可，由最短路的性质显然修改后边权非负.

然后对每个起点跑Dijkstra，再修正距离$d_{u,v} = d'_{u,v} - h_u + h_v$即可，复杂度$O(nm \log n)$，在稀疏图上是要优于Floyd的.

3.2.3 k短路

```
// 注意这是个多项式算法，在k比较大时很有优势，但k比较小时最好还是用A*
// DAG和有环的情况都可以，有重边或环也无所谓，但不能有零环
// 以下代码以Dijkstra + 可持久化左偏树为例

constexpr int maxn = 1005, maxe = 10005, maxm = maxe * 30; //点数,边数,左偏树结点数

// 结构体定义
struct A { // 用来求最短路
    int x, d;

    A(int x, int d) : x(x), d(d) {}

    bool operator < (const A &a) const {
        return d > a.d;
    }
};

struct node { // 左偏树结点
    int w, i, d; // i: 最后一条边的编号 d: 左偏树附加信息
    node *lc, *rc;

    node() {}

    node(int w, int i) : w(w), i(i), d(0) {}

    void refresh(){
        d = rc -> d + 1;
    }
} null[maxm], *ptr = null, *root[maxn];

struct B { // 维护答案用
    int x, w; // x是结点编号，w表示之前已经产生的权值
    node *rt; // 这个答案对应的堆顶,注意可能不等于任何一个结点的堆

    B(int x, node *rt, int w) : x(x), w(w), rt(rt) {}

    bool operator < (const B &a) const {
        return w + rt -> w > a.w + a.rt -> w;
    }
};

// 全局变量和数组定义
vector<int> G[maxn], W[maxn], id[maxn]; // 最开始要存反向图，然后把G清空作为儿子列表
bool vis[maxn], used[maxe]; // used表示边是否在最短路树上
int u[maxe], v[maxe], w[maxe]; // 存下每条边,注意是有向边
int d[maxn], p[maxn]; // p表示最短路树上每个点的父边
int n, m, k, s, t; // s, t分别表示起点和终点


// 以下是主函数中较关键的部分
for (int i = 0 ; i <= n; i++)
    root[i] = null; // 一定要加上!!!

// (读入&建反向图)

Dijkstra();

// (清空G, W, id)

for (int i = 1; i <= n; i++)
    if (p[i]) {
        used[p[i]] = true; // 在最短路树上
        G[v[p[i]]].push_back(i);
    }

for (int i = 1; i <= m; i++) {
    w[i] -= d[u[i]] - d[v[i]]; // 现在的w[i]表示这条边能使路径长度增加多少
    if (!used[i])
        root[u[i]] = merge(root[u[i]], newnode(w[i], i));
}

dfs(t);

priority_queue<B> heap;
heap.push(B(s, root[s], 0)); // 初始状态是找贡献最小的边加进去

printf("%d\n",d[s]); // 第1短路需要特判
while (--k) { // 其余k - 1短路径用二叉堆维护
    if (heap.empty())
        printf("-1\n");
    else {
        int x = heap.top().x, w = heap.top().w;
        node *rt = heap.top().rt;
        heap.pop();

        printf("%d\n", d[s] + w + rt -> w);

        if (rt -> lc != null || rt -> rc != null)
            heap.push(B(x, merge(rt -> lc, rt -> rc), w)); // pop掉当前边，换成另一条贡献大一点的边
        if (root[v[rt -> i]] != null)
            heap.push(B(v[rt -> i], root[v[rt -> i]], w + rt -> w)); // 保留当前边，往后面再接上另一条边
    }
}
// 主函数到此结束


// Dijkstra预处理最短路 O(m\log n)
void Dijkstra() {
    memset(d, 63, sizeof(d));
    d[t] = 0;
    priority_queue<A> heap;
    heap.push(A(t, 0));

    while (!heap.empty()) {
        int x = heap.top().x;
        heap.pop();

        if(vis[x])
            continue;

        vis[x] = true;
        for (int i = 0; i < (int)G[x].size(); i++)
            if (!vis[G[x][i]] && d[G[x][i]] > d[x] + W[x][i]) {
                d[G[x][i]] = d[x] + W[x][i];
                p[G[x][i]] = id[x][i];

                heap.push(A(G[x][i], d[G[x][i]]));
            }
    }
}

// dfs求出每个点的堆 总计O(m\log n)
// 需要调用merge, 同时递归调用自身
void dfs(int x) {
    root[x] = merge(root[x], root[v[p[x]]]);

    for (int i = 0; i < (int)G[x].size(); i++)
        dfs(G[x][i]);
}

// 包装过的new node() O(1)
node *newnode(int w, int i) {
    *++ptr = node(w, i);
    ptr -> lc = ptr -> rc = null;
    return ptr;
}

// 带可持久化的左偏树合并 总计O(\log n)
// 递归调用自身
node *merge(node *x, node *y) {
    if (x == null)
        return y;
    if (y == null)
        return x;

    if (x -> w > y -> w)
        swap(x, y);

    node *z = newnode(x -> w, x -> i);
    z -> lc = x -> lc;
    z -> rc = merge(x -> rc, y);

    if (z -> lc -> d > z -> rc -> d)
        swap(z -> lc, z -> rc);
    z -> refresh();

    return z;
}
```

3.3 Tarjan算法

3.3.1 强连通分量

```
int dfn[maxn], low[maxn], tim = 0;
vector<int> G[maxn], scc[maxn];
int sccid[maxn], scc_cnt = 0, stk[maxn];
bool instk[maxn];

void dfs(int x) {
    dfn[x] = low[x] = ++tim;

    stk[++stk[0]] = x;
    instk[x] = true;

    for (int y : G[x]) {
        if (!dfn[y]) {
            dfs(y);
            low[x] = min(low[x], low[y]);
        }
        else if (instk[y])
            low[x] = min(low[x], dfn[y]);
    }

    if (dfn[x] == low[x]) {
        scc_cnt++;

        int u;
        do {
            u = stk[stk[0]--];
            instk[u] = false;
            sccid[u] = scc_cnt;
            scc[scc_cnt].push_back(u);
        } while (u != x);
    }
}

void tarjan(int n) {
    for (int i = 1; i <= n; i++)
        if (!dfn[i])
            dfs(i);
}
```

3.3.2 割点 点双

```
vector<int> G[maxn], bcc[maxn];
int dfn[maxn], low[maxn], tim = 0, bccid[maxn], bcc_cnt = 0;
bool iscut[maxn];

pair<int, int> stk[maxn];
int stk_cnt = 0;

void dfs(int x, int pr) {
    int child = 0;
    dfn[x] = low[x] = ++tim;

    for (int y : G[x]) {
        if (!dfn[y]) {
            stk[++stk_cnt] = make_pair(x, y);
            child++;
            dfs(y, x);
            low[x] = min(low[x], low[y]);

            if (low[y] >= dfn[x]) {
                iscut[x] = true;
                bcc_cnt++;

                while (true) {
                    auto pi = stk[stk_cnt--];

                    if (bccid[pi.first] != bcc_cnt) {
                        bcc[bcc_cnt].push_back(pi.first);
                        bccid[pi.first] = bcc_cnt;
                    }
                    if (bccid[pi.second] != bcc_cnt) {
                        bcc[bcc_cnt].push_back(pi.second);
                        bccid[pi.second] = bcc_cnt;
                    }

                    if (pi.first == x && pi.second == y)
                        break;
                }
            }
        }
        else if (dfn[y] < dfn[x] && y != pr) {
            stk[++stk_cnt] = make_pair(x, y);
            low[x] = min(low[x], dfn[y]);
        }
    }

    if (!pr && child == 1)
        iscut[x] = false;
}

void Tarjan(int n) {
    for (int i = 1; i <= n; i++)
        if (!dfn[i])
            dfs(i, 0);
}
```

3.3.3 桥 边双

```
int u[maxe], v[maxe];
vector<int> G[maxn]; // 存的是边的编号

int stk[maxn], top, dfn[maxn], low[maxn], tim, bcc_cnt;
vector<int> bcc[maxn];

bool isbridge[maxe];

void dfs(int x, int pr) { // 这里pr是入边的编号
    dfn[x] = low[x] = ++tim;
    stk[++top] = x;

    for (int i : G[x]) {
        int y = (u[i] == x ? v[i] : u[i]);

        if (!dfn[y]) {
            dfs(y, i);
            low[x] = min(low[x], low[y]);

            if (low[y] > dfn[x])
                bridge[i] = true;
        }
        else if (i != pr)
            low[x] = min(low[x], dfn[y]);
    }

    if (dfn[x] == low[x]) {
        bcc_cnt++;
        int y;
        do {
            y = stk[top--];
            bcc[bcc_cnt].push_back(y);
        } while (y != x);
    }
}
```

3.4 仙人掌

一般来说仙人掌问题都可以通过圆方树转成有两种点的树上问题来做.

3.4.1 仙人掌DP

```
struct edge{
    int to, w, prev;
}e[maxn * 2];

vector<pair<int, int> > v[maxn];

vector<long long> d[maxn];

stack<int> stk;

int p[maxn];

bool vis[maxn], vise[maxn * 2];

int last[maxn], cnte;

long long f[maxn], g[maxn], sum[maxn];

int n, m, cnt;

void addedge(int x, int y, int w) {
    v[x].push_back(make_pair(y, w));
}

void dfs(int x) {

    vis[x] = true;

    for (int i = last[x]; ~i; i = e[i].prev) {
        if (vise[i ^ 1])
            continue;

        int y = e[i].to, w = e[i].w;

        vise[i] = true;

        if (!vis[y]) {
            stk.push(i);
            p[y] = x;
            dfs(y);

            if (!stk.empty() && stk.top() == i) {
                stk.pop();
                addedge(x, y, w);
            }
        }

        else {
            cnt++;

            long long tmp = w;
            while (!stk.empty()) {
                int i = stk.top();
                stk.pop();

                int yy = e[i].to, ww = e[i].w;

                addedge(cnt, yy, 0);

                d[cnt].push_back(tmp);

                tmp += ww;

                if (e[i ^ 1].to == y)
                    break;
            }

            addedge(y, cnt, 0);

            sum[cnt] = tmp;
        }
    }
}

void dp(int x) {

    for (auto o : v[x]) {
        int y = o.first, w = o.second;
        dp(y);
    }

    if (x <= n) {
        for (auto o : v[x]) {
            int y = o.first, w = o.second;

            f[x] += 2 * w + f[y];
        }

        g[x] = f[x];

        for (auto o : v[x]) {
            int y = o.first, w = o.second;

            g[x] = min(g[x], f[x] - f[y] - 2 * w + g[y] + w);
        }
    }
    else {
        f[x] = sum[x];
        for (auto o : v[x]) {
            int y = o.first;

            f[x] += f[y];
        }

        g[x] = f[x];

        for (int i = 0; i < (int)v[x].size(); i++) {
            int y = v[x][i].first;

            g[x] = min(g[x], f[x] - f[y] + g[y] + min(d[x][i], sum[x] - d[x][i]));
        }
    }
}
```

3.5 二分图

3.5.1 匈牙利

```
vector<int> G[maxn];

int girl[maxn], boy[maxn]; // 男孩在左边，女孩在右边
bool vis[maxn];

bool dfs(int x) {
    for (int y : G[x])
        if (!vis[y]) {
            vis[y] = true;

            if (!boy[y] || dfs(boy[y])) {
                girl[x] = y;
                boy[y] = x;

                return true;
            }
        }

    return false;
}

int hungary() {
    int ans = 0;

    for (int i = 1; i <= n; i++)
        if (!girl[i]) {
            memset(vis, 0, sizeof(vis));
            ans += dfs(i);
        }

    return ans;
}
```

3.5.2 Hopcroft-Karp二分图匹配

其实长得和Dinic差不太多，或者说像匈牙利和Dinic的缝合怪.

```
vector<int> G[maxn];

int girl[maxn], boy[maxn]; // girl: 左边匹配右边  boy: 右边匹配左边

bool vis[maxn]; // 右半的点是否已被访问
int dx[maxn], dy[maxn];
int q[maxn];

bool bfs(int n) {
    memset(dx, -1, sizeof(int) * (n + 1));
    memset(dy, -1, sizeof(int) * (n + 1));

    int head = 0, tail = 0;
    for (int i = 1; i <= n; i++)
        if (!girl[i]) {
            q[tail++] = i;
            dx[i] = 0;
        }

    bool flag = false;

    while (head != tail) {
        int x = q[head++];

        for (auto y : G[x])
            if (dy[y] == -1) {
                dy[y] = dx[x] + 1;

                if (boy[y]) {
                    if (dx[boy[y]] == -1) {
                        dx[boy[y]] = dy[y] + 1;
                        q[tail++] = boy[y];
                    }
                }
                else
                    flag = true;
            }
    }

    return flag;
}

bool dfs(int x) {
    for (int y : G[x])
        if (!vis[y] && dy[y] == dx[x] + 1) {
            vis[y] = true;

            if (boy[y] && !dfs(boy[y]))
                continue;

            girl[x] = y;
            boy[y] = x;
            return true;
        }

    return false;
}

int hopcroft_karp(int n) {
    int ans = 0;

    for (int x = 1; x <= n; x++) // 先贪心求出一组初始匹配, 当然不写贪心也行
        for (int y : G[x])
            if (!boy[y]) {
                girl[x] = y;
                boy[y] = x;
                ans++;
                break;
            }

    while (bfs(n)) {
        memset(vis, 0, sizeof(bool) * (n + 1));

        for (int x = 1; x <= n; x++)
            if (!girl[x])
                ans += dfs(x);
    }

    return ans;
}
```

3.5.3 KM二分图最大权匹配

```
const long long INF = 0x3f3f3f3f3f3f3f3f;

long long w[maxn][maxn], lx[maxn], ly[maxn], slack[maxn];
// 边权 顶标 slack
// 如果要求最大权完美匹配就把不存在的边设为-INF,否则所有边对0取max

bool visx[maxn], visy[maxn];

int boy[maxn], girl[maxn], p[maxn], q[maxn], head, tail; // p : pre

int n, m, N, e;

// 增广
bool check(int y) {
    visy[y] = true;

    if (boy[y]) {
        visx[boy[y]] = true;
        q[tail++] = boy[y];
        return false;
    }

    while (y) {
        boy[y] = p[y];
        swap(y, girl[p[y]]);
    }

    return true;
}

// bfs每个点
void bfs(int x) {
    memset(q, 0, sizeof(q));
    head = tail = 0;

    q[tail++] = x;
    visx[x] = true;

    while (true) {
        while (head != tail) {
            int x = q[head++];

            for (int y = 1; y <= N; y++)
                if (!visy[y]) {
                    long long d = lx[x] + ly[y] - w[x][y];

                    if (d < slack[y]) {
                        p[y] = x;
                        slack[y] = d;

                        if (!slack[y] && check(y))
                            return;
                    }
                }
        }

        long long d = INF;
        for (int i = 1; i <= N; i++)
            if (!visy[i])
                d = min(d, slack[i]);

        for (int i = 1; i <= N; i++) {
            if (visx[i])
                lx[i] -= d;

            if (visy[i])
                ly[i] += d;
            else
                slack[i] -= d;
        }

        for (int i = 1; i <= N; i++)
            if (!visy[i] && !slack[i] && check(i))
                return;
    }
}

// 主过程
long long KM() {
    for (int i = 1; i <= N; i++) {
        // lx[i] = 0;
        ly[i] = -INF;
        // boy[i] = girl[i] = -1;

        for (int j = 1; j <= N; j++)
            ly[i] = max(ly[i], w[j][i]);
    }

    for (int i = 1; i <= N; i++) {
        memset(slack, 0x3f, sizeof(slack));
        memset(visx, 0, sizeof(visx));
        memset(visy, 0, sizeof(visy));
        bfs(i);
    }

    long long ans = 0;
    for (int i = 1; i <= N; i++)
        ans += w[i][girl[i]];
    return ans;
}

// 为了方便贴上主函数
int main() {

    scanf("%d%d%d", &n, &m, &e);
    N = max(n, m);

    while (e--) {
        int x, y, c;
        scanf("%d%d%d", &x, &y, &c);
        w[x][y] = max(c, 0);
    }

    printf("%lld\n", KM());

    for (int i = 1; i <= n; i++) {
        if (i > 1)
            printf(" ");
        printf("%d", w[i][girl[i]] > 0 ? girl[i] : 0);
    }
    printf("\n");

    return 0;
}
```

3.5.4 二分图原理

最大匹配的可行边与必须边，关键点

以下的“残量网络”指网络流图的残量网络.

- 可行边：一条边的两个端点在残量网络中处于同一个SCC，不论是正向边还是反向边.
- 必须边：一条属于当前最大匹配的边，且残量网络中两个端点不在同一个SCC中.
- 关键点(必须点)：这里不考虑网络流图而只考虑原始的图，将匹配边改成从右到左之后从左边的每个未匹配点进行floodfill，左边没有被标记的点即为关键点. 右边同理.

独立集

二分图独立集可以看成最小割问题，割掉最少的点使得S和T不连通，则剩下的点自然都在独立集中.

所以独立集输出方案就是求出不在最小割中的点，独立集的必须点/可行点就是最小割的不可行点/非必须点.

割点等价于割掉它与源点或汇点相连的边，可以通过设置中间的边权为无穷以保证不能割掉中间的边，然后按照上面的方法判断即可.

(由于一个点最多流出一个流量，所以中间的边权其实是可以任取的.)

二分图最大权匹配

二分图最大权匹配的对偶问题是最小顶标和问题，即：为图中的每个顶点赋予一个非负顶标，使得对于任意一条边，两端点的顶标和都要不小于边权，最小化顶标之和.

显然KM算法的原理实际上就是求最小顶标和.

3.6 一般图匹配

3.6.1 高斯消元

```
// 这个算法基于Tutte定理和高斯消元，思维难度相对小一些，也更方便进行可行边的判定
// 注意这个算法复杂度是满的，并且常数有点大，而带花树通常是跑不满的
// 以及，根据Tutte定理，如果求最大匹配的大小的话直接输出Tutte矩阵的秩/2即可
// 需要输出方案时才需要再写后面那些乱七八糟的东西


// 复杂度和常数所限，1s之内500已经是这个算法的极限了
const int maxn = 505, p = 1000000007; // p可以是任意10^9以内的质数

// 全局数组和变量定义
int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn], id[maxn], a[maxn];
bool row[maxn] = {false}, col[maxn] = {false};
int n, m, girl[maxn]; // girl是匹配点，用来输出方案

// 为了方便使用，贴上主函数
// 需要调用高斯消元和eliminate
int main() {
    srand(19260817);

    scanf("%d%d", &n, &m); // 点数和边数
    while (m--) {
        int x, y;
        scanf("%d%d", &x, &y);
        A[x][y] = rand() % p;
        A[y][x] = -A[x][y]; // Tutte矩阵是反对称矩阵
    }

    for (int i = 1; i <= n; i++)
        id[i] = i; // 输出方案用的，因为高斯消元的时候会交换列
    memcpy(t, A, sizeof(t));
    Gauss(A, NULL, n);

    m = n;
    n = 0; // 这里变量复用纯属个人习惯

    for (int i = 1; i <= m; i++)
        if (A[id[i]][id[i]])
            a[++n] = i; // 找出一个极大满秩子矩阵

    for (int i = 1;i <= n; i++)
        for (int j = 1; j <= n; j++)
            A[i][j] = t[a[i]][a[j]];

    Gauss(A, B, n);

    for (int i = 1; i <= n; i++)
        if (!girl[a[i]])
            for (int j = i + 1; j <= n; j++)
                if (!girl[a[j]] && t[a[i]][a[j]] && B[j][i]) {
                    // 注意上面那句if的写法，现在t是邻接矩阵的备份，
                    // 逆矩阵j行i列不为0当且仅当这条边可行
                    girl[a[i]] = a[j];
                    girl[a[j]] = a[i];

                    eliminate(i, j);
                    eliminate(j, i);
                    break;
                }

    printf("%d\n", n / 2);
    for (int i = 1; i <= m; i++)
        printf("%d ", girl[i]);

    return 0;
}

// 高斯消元 O(n^3)
// 在传入B时表示计算逆矩阵，传入NULL则只需计算矩阵的秩
void Gauss(int A[][maxn], int B[][maxn], int n) {
    if(B) {
        memset(B, 0, sizeof(t));
        for (int i = 1; i <= n; i++)
            B[i][i] = 1;
    }

    for (int i = 1; i <= n; i++) {
        if (!A[i][i]) {
            for (int j = i + 1; j <= n; j++)
                if (A[j][i]) {
                    swap(id[i], id[j]);
                    for (int k = i; k <= n; k++)
                        swap(A[i][k], A[j][k]);

                    if (B)
                        for (int k = 1; k <= n; k++)
```

```
                            swap(B[i][k], B[j][k]);
                    break;
                }

            if (!A[i][i])
                continue;
        }

        int inv = qpow(A[i][i], p - 2);

        for (int j = 1; j <= n; j++)
            if (i != j && A[j][i]){
                int t = (long long)A[j][i] * inv % p;

                for (int k = i; k <= n; k++)
                    if (A[i][k])
                        A[j][k] = (A[j][k] - (long long)t * A[i][k]) % p;

                if (B)
                    for (int k = 1; k <= n; k++)
                        if (B[i][k])
                            B[j][k] = (B[j][k] - (long long)t * B[i][k])%p;
            }
    }

    if (B)
        for (int i = 1; i <= n; i++) {
            int inv = qpow(A[i][i], p - 2);

            for (int j = 1; j <= n; j++)
                if (B[i][j])
                    B[i][j] = (long long)B[i][j] * inv % p;
        }
}

// 消去一行一列 O(n^2)
void eliminate(int r, int c) {
    row[r] = col[c] = true; // 已经被消掉

    int inv = qpow(B[r][c], p - 2);

    for (int i = 1; i <= n; i++)
        if (!row[i] && B[i][c]) {
            int t = (long long)B[i][c] * inv % p;

            for (int j = 1; j <= n; j++)
                if (!col[j] && B[r][j])
                    B[i][j] = (B[i][j] - (long long)t * B[r][j]) % p;
        }
}
```

3.6.2 带花树

```
// 带花树通常比高斯消元快很多，但在只需要求最大匹配大小的时候并没有高斯消元好写
// 当然输出方案要方便很多

// 全局数组与变量定义
vector<int> G[maxn];
int girl[maxn], f[maxn], t[maxn], p[maxn], vis[maxn], tim, q[maxn], head, tail;
int n, m;


// 封装好的主过程 O(nm)
int blossom() {
    int ans = 0;

    for (int i = 1; i <= n; i++)
        if (!girl[i])
            ans += bfs(i);

    return ans;
}


// bfs找增广路 O(m)
bool bfs(int s) {
    memset(t, 0, sizeof(t));
    memset(p, 0, sizeof(p));

    for (int i = 1; i <= n; i++)
        f[i] = i; // 并查集

    head = tail = 0;
    q[tail++] = s;
    t[s] = 1;

    while (head != tail) {
        int x = q[head++];
        for (int y : G[x]) {
            if (findroot(y) == findroot(x) || t[y] == 2)
                continue;

            if (!t[y]) {
                t[y] = 2;
                p[y] = x;

                if (!girl[y]) {
                    for (int u = y, t; u; u = t) {
                        t = girl[p[u]];
                        girl[p[u]] = u;
                        girl[u] = p[u];
                    }
                    return true;
                }

                t[girl[y]] = 1;
                q[tail++] = girl[y];
            }
            else if (t[y] == 1) {
                int z = LCA(x, y);

                shrink(x, y, z);
                shrink(y, x, z);
            }
        }
    }

    return false;
}

//缩奇环 O(n)
void shrink(int x, int y, int z) {
    while (findroot(x) != z) {
        p[x] = y;
        y = girl[x];

        if (t[y] == 2) {
            t[y] = 1;
            q[tail++] = y;
        }

        if (findroot(x) == x)
            f[x] = z;
        if (findroot(y) == y)
            f[y] = z;

        x = p[y];
    }
}

//暴力找LCA O(n)
int LCA(int x, int y) {
    tim++;
    while (true) {
        if (x) {
            x = findroot(x);

            if (vis[x] == tim)
                return x;
            else {
                vis[x] = tim;
                x = p[girl[x]];
            }
        }
        swap(x, y);
    }
}

//并查集的查找 O(1)
int findroot(int x) {
    return x == f[x] ? x : (f[x] = findroot(f[x]));
}
```

3.6.3 带权带花树

Forked from templates of Imperisible Night.（有一说一这玩意实在太难写了，抄之前建议先想想算法是不是假的或者有SB做法）

```
//maximum weight blossom,  change g[u][v].w to INF - g[u][v].w when minimum weight blossom is needed
//type of ans is long long
//replace all int to long long if weight of edge is long long

struct WeightGraph {
    static const int INF = INT_MAX;
    static const int MAXN = 400;
    struct edge{
        int u, v, w;
        edge() {}
        edge(int u, int v, int w): u(u), v(v), w(w) {}
    };
    int n, n_x;
    edge g[MAXN * 2 + 1][MAXN * 2 + 1];
    int lab[MAXN * 2 + 1];
    int match[MAXN * 2 + 1], slack[MAXN * 2 + 1], st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
    int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
    vector<int> flower[MAXN * 2 + 1];
    queue<int> q;
    inline int e_delta(const edge &e){ // does not work inside blossoms
        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
    }
    inline void update_slack(int u, int x){
        if(!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x]))
            slack[x] = u;
    }
    inline void set_slack(int x){
        slack[x] = 0;
        for(int u = 1;u <= n; ++u)
            if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
                update_slack(u, x);
    }
    void q_push(int x){
        if(x <= n)q.push(x);
        else for(size_t i = 0;i < flower[x].size(); i++)
            q_push(flower[x][i]);
    }
    inline void set_st(int x, int b){
        st[x]=b;
        if(x > n) for(size_t i = 0;i < flower[x].size(); ++i)
                    set_st(flower[x][i], b);
    }
    inline int get_pr(int b, int xr){
        int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
        if(pr % 2 == 1){
            reverse(flower[b].begin() + 1, flower[b].end());
            return (int)flower[b].size() - pr;
        } else return pr;
    }
    inline void set_match(int u, int v){
        match[u]=g[u][v].v;
        if(u > n){
            edge e=g[u][v];
            int xr = flower_from[u][e.u], pr=get_pr(u, xr);
            for(int i = 0;i < pr; ++i)
                set_match(flower[u][i], flower[u][i ^ 1]);
            set_match(xr, v);
            rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
        }
    }
    inline void augment(int u, int v){
        for(; ; ){
            int xnv=st[match[u]];
            set_match(u, v);
            if(!xnv)return;
            set_match(xnv, st[pa[xnv]]);
            u=st[pa[xnv]], v=xnv;
        }
    }
    inline int get_lca(int u, int v){
        static int t=0;
        for(++t; u || v; swap(u, v)){
            if(u == 0)continue;
            if(vis[u] == t)return u;
            vis[u] = t;
            u = st[match[u]];
            if(u) u = st[pa[u]];
        }
        return 0;
    }
    inline void add_blossom(int u, int lca, int v){
        int b = n + 1;
        while(b <= n_x && st[b]) ++b;
        if(b > n_x) ++n_x;
        lab[b] = 0, S[b] = 0;
        match[b] = match[lca];
        flower[b].clear();
        flower[b].push_back(lca);
        for(int x = u, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        reverse(flower[b].begin() + 1, flower[b].end());
        for(int x = v, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        set_st(b, b);
        for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
        for(int x = 1; x <= n; ++x) flower_from[b][x] = 0;
        for(size_t i = 0 ; i < flower[b].size(); ++i){
            int xs = flower[b][i];
            for(int x = 1; x <= n_x; ++x)
                if(g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
            for(int x = 1;x <= n; ++x)
                if(flower_from[xs][x]) flower_from[b][x] = xs;
        }
        set_slack(b);
    }
    inline void expand_blossom(int b){ // S[b]  ==  1
        for(size_t i = 0; i < flower[b].size(); ++i)
            set_st(flower[b][i], flower[b][i]);
        int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
        for(int i = 0; i < pr; i += 2){
            int xs = flower[b][i], xns = flower[b][i + 1];
            pa[xs] = g[xns][xs].u;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for(size_t i = pr + 1;i < flower[b].size(); ++i){
            int xs = flower[b][i];
            S[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    inline bool on_found_edge(const edge &e){
        int u = st[e.u], v = st[e.v];
        if(S[v] == -1){
            pa[v] = e.u, S[v] = 1;
            int nu = st[match[v]];
            slack[v] = slack[nu] = 0;
            S[nu] = 0, q_push(nu);
        }else if(S[v] == 0){
            int lca = get_lca(u, v);
            if(!lca) return augment(u, v), augment(v, u), true;
            else add_blossom(u, lca, v);
        }
        return false;
    }
    inline bool matching(){
        memset(S + 1, -1, sizeof(int) * n_x);
        memset(slack + 1, 0, sizeof(int) * n_x);
        q = queue<int>();
        for(int x = 1;x <= n_x; ++x)
            if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
        if(q.empty())return false;
        for(;;){
            while(q.size()){
                int u = q.front();q.pop();
                if(S[st[u]] == 1)continue;
                for(int v = 1;v <= n; ++v)
                    if(g[u][v].w > 0 && st[u] != st[v]){
                        if(e_delta(g[u][v]) == 0){
                            if(on_found_edge(g[u][v]))return true;
                        }else update_slack(u, st[v]);
                    }
            }
            int d = INF;
            for(int b = n + 1; b <= n_x;++b)
                if(st[b] == b && S[b] == 1)d = min(d, lab[b]/2);
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x]){
                    if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
                    else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
                }
            for(int u = 1; u <= n; ++u){
                if(S[st[u]] == 0){
                    if(lab[u] <= d)return 0;
                    lab[u] -= d;
                }else if(S[st[u]] == 1)lab[u] += d;
            }
            for(int b = n+1; b <= n_x; ++b)
                if(st[b] == b){
                    if(S[st[b]] == 0) lab[b] += d * 2;
                    else if(S[st[b]] == 1) lab[b] -= d * 2;
                }
            q=queue<int>();
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) == 0)
                    if(on_found_edge(g[slack[x]][x]))return true;
            for(int b = n + 1; b <= n_x; ++b)
                if(st[b] == b && S[b] == 1 && lab[b] == 0)expand_blossom(b);
        }
        return false;
    }
    inline pair<long long, int> solve(){
        memset(match + 1, 0, sizeof(int) * n);
        n_x = n;
        int n_matches = 0;
        long long tot_weight = 0;
        for(int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();
        int w_max = 0;
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v){
                flower_from[u][v] = (u == v ? u : 0);
                w_max = max(w_max, g[u][v].w);
            }
        for(int u = 1; u <= n; ++u) lab[u] = w_max;
        while(matching()) ++n_matches;
        for(int u = 1; u <= n; ++u)
            if(match[u] && match[u] < u)
                tot_weight += g[u][match[u]].w;
        return make_pair(tot_weight, n_matches);
    }
    inline void init(){
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v)
                g[u][v]=edge(u, v, 0);
    }
```

```
};
```

3.6.4 原理

设图G的Tutte矩阵是$\tilde{A}$, 首先是最基础的引理:

- G的最大匹配大小是$\frac{1}{2}\mathrm{rank}\tilde{A}$.
- $(\tilde{A}^{-1})_{i,j} \neq 0$当且仅当$G - \{v_i, v_j\}$有完美匹配.
  (考虑到逆矩阵与伴随矩阵的关系, 这是显然的.)

构造最大匹配的方法见板子. 对于更一般的问题, 可以借助构造方法转化为完美匹配问题.
设最大匹配的大小为k, 新建$n - 2k$个辅助点, 让它们和其他所有点连边, 那么如果一个点匹配了一个辅助点, 就说明它在原图的匹配中不匹配任何点.

- 最大匹配的可行边: 对原图中的任意一条边(u, v), 如果删掉u, v后新图仍然有完美匹配(也就是$\tilde{A}^{-1}_{u,v} \neq 0$), 则它是一条可行边.
- 最大匹配的必须边: **待补充**
- 最大匹配的必须点: 可以删掉这个点和一个辅助点, 然后判断剩下的图是否还有完美匹配, 如果有则说明它不是必须的, 否则是必须的. 只需要用到逆矩阵即可.
- 最大匹配的可行点: 显然对于任意一个点, 只要它不是孤立点, 就是可行点.

3.7 支配树

记得建反图!

```
vector<int> G[maxn], R[maxn], son[maxn]; // R是反图, son存的是sdom树上的儿子

int ufs[maxn];

int idom[maxn], sdom[maxn], anc[maxn]; // anc: sdom的dfn最小的祖先

int p[maxn], dfn[maxn], id[maxn], tim;

int findufs(int x) {
    if (ufs[x] == x)
        return x;

    int t = ufs[x];
    ufs[x] = findufs(ufs[x]);

    if (dfn[sdom[anc[x]]] > dfn[sdom[anc[t]]])
        anc[x] = anc[t];

    return ufs[x];
}

void dfs(int x) {
    dfn[x] = ++tim;
    id[tim] = x;
    sdom[x] = x;

    for (int y : G[x])
        if (!dfn[y]) {
            p[y] = x;
            dfs(y);
        }
}

void get_dominator(int n) {
    for (int i = 1; i <= n; i++)
        ufs[i] = anc[i] = i;

    dfs(1);

    for (int i = n; i > 1; i--) {
        int x = id[i];

        for (int y : R[x])
            if (dfn[y]) {
                findufs(y);
                if (dfn[sdom[x]] > dfn[sdom[anc[y]]])
                    sdom[x] = sdom[anc[y]];
            }

        son[sdom[x]].push_back(x);
        ufs[x] = p[x];

        for (int u : son[p[x]]) {
            findufs(u);
            idom[u] = (sdom[u] == sdom[anc[u]] ? p[x] : anc[u]);
        }

        son[p[x]].clear();
    }

    for (int i = 2; i <= n; i++) {
        int x = id[i];

        if (idom[x] != sdom[x])
            idom[x] = idom[idom[x]];

        son[idom[x]].push_back(x);
    }
}
```

3.8 2-SAT

如果限制满足对称性(每个命题的逆否命题对应的边也存在), 那么可以使用Tarjan算法求SCC搞定.
具体来说就是, 如果某个变量的两个点在同一SCC中则显然无解, 否则按拓扑序倒序尝试选择每个SCC即可.
由于Tarjan算法的特性, 找到SCC的顺序就是拓扑序**倒序**, 所以判断完是否有解之后, 每个变量只需要取SCC编号**较小**的那个.

```
if (!ok)
    printf("IMPOSSIBLE\n");
else {
    printf("POSSIBLE\n");

    for (int i = 1; i <= n; i++)
        printf("%d%c", sccid[i * 2 - 1] > sccid[i * 2], i < n ? ' ' : '\n');
}
```

如果要字典序最小就用DFS, 注意可以压位优化. 另外代码是0-base的.

```
bool vis[maxn];
int stk[maxn], top;

// 主函数
for (int i = 0; i < n; i += 2)
    if (!vis[i] && !vis[i ^ 1]) {
        top = 0;
        if (!dfs(i)) {
            while (top)
                vis[stk[top--]] = false;

            if (!dfs(i + 1)) {
                bad = true;
```

```
                    break;
                }
            }
        }
// 最后stk中的所有元素就是选中的值

// dfs
bool dfs(int x) {
    if (vis[x ^ 1])
        return false;

    if (vis[x])
        return true;

    vis[x] = true;
    stk[++top] = x;

    for (int i = 0; i < (int)G[x].size(); i++)
        if (!dfs(G[x][i]))
            return false;

    return true;
}
```

3.9 最大流

3.9.1 Dinic

```
// 注意Dinic适用于二分图或分层图,对于一般稀疏图ISAP更优,稠密图则HLPP更优

struct edge{
    int to, cap, prev;
} e[maxe * 2];

int last[maxn], len, d[maxn], cur[maxn], q[maxn];

// main函数里要初始化
memset(last, -1, sizeof(last));

void AddEdge(int x, int y, int z) {
    e[len].to = y;
    e[len].cap = z;
    e[len].prev = last[x];
    last[x] = len++;
}

void addedge(int x, int y, int z) {
    AddEdge(x, y, z);
    AddEdge(y, x, 0);
}

void bfs() {
    int head = 0, tail = 0;
    memset(d, -1, sizeof(int) * (t + 5));
    q[tail++] = s;
    d[s] = 0;

    while (head != tail){
        int x = q[head++];
        for (int i = last[x]; ~i; i = e[i].prev)
            if (e[i].cap > 0 && d[e[i].to] == -1) {
                d[e[i].to] = d[x] + 1;
                q[tail++] = e[i].to;
            }
    }
}

int dfs(int x, int a) {
    if (x == t || !a)
        return a;

    int flow = 0, f;
    for (int &i = cur[x]; ~i; i = e[i].prev)
        if (e[i].cap > 0 && d[e[i].to] == d[x] + 1 && (f = dfs(e[i].to, min(e[i].cap,a)))) {

            e[i].cap -= f;
            e[i^1].cap += f;
            flow += f;
            a -= f;

            if (!a)
                break;
        }

    return flow;
}

int Dinic() {
    int flow = 0;
    while (bfs(), ~d[t]) {
        memcpy(cur, last, sizeof(int) * (t + 5));
        flow += dfs(s, inf);
    }
    return flow;
}
```

3.9.2 ISAP

可能有毒，慎用.

```
// 注意ISAP适用于一般稀疏图，对于二分图或分层图情况Dinic比较优，稠密图则HLPP更优

// 边的定义
// 这里没有记录起点和反向边，因为反向边即为正向边xor 1，起点即为反向边的终点
struct edge{
    int to, cap, prev;
} e[maxe * 2];

// 全局变量和数组定义
int last[maxn], cnte = 0, d[maxn], p[maxn], c[maxn], cur[maxn], q[maxn];
int n, m, s, t; // s, t一定要开成全局变量

void AddEdge(int x, int y, int z) {
    e[cnte].to = y;
    e[cnte].cap = z;
    e[cnte].prev = last[x];
    last[x] = cnte++;
}

void addedge(int x, int y, int z) {
    AddEdge(x, y, z);
    AddEdge(y, x, 0);
}

// 预处理到t的距离标号
// 在测试数据组数较少时可以省略,把所有距离标号初始化为0
void bfs() {
    memset(d, -1, sizeof(d));

    int head = 0, tail = 0;
    d[t] = 0;
    q[tail++] = t;

    while (head != tail) {
        int x = q[head++];
        c[d[x]]++;

        for (int i = last[x]; ~i; i = e[i].prev)
            if (e[i ^ 1].cap && d[e[i].to] == -1) {
                d[e[i].to] = d[x] + 1;
                q[tail++] = e[i].to;
            }
    }
}

// augment函数 O(n) 沿增广路增广一次，返回增广的流量
int augment() {
    int a = (~0u) >> 1; // INT_MAX

    for (int x = t; x != s; x = e[p[x] ^ 1].to)
        a = min(a, e[p[x]].cap);

    for (int x = t; x != s; x = e[p[x] ^ 1].to) {
        e[p[x]].cap -= a;
        e[p[x] ^ 1].cap += a;
    }

    return a;
}

// 主过程 O(n^2 m)，返回最大流的流量
// 注意这里的n是编号最大值,在这个值不为n的时候一定要开个变量记录下来并修改代码
int ISAP() {
    bfs();

    memcpy(cur, last, sizeof(cur));

    int x = s, flow = 0;

    while (d[s] < n) {
        if (x == t) { // 如果走到了t就增广一次，并返回s重新找增广路
            flow += augment();
            x = s;
        }

        bool ok = false;
        for (int &i = cur[x]; ~i; i = e[i].prev)
            if (e[i].cap && d[x] == d[e[i].to] + 1) {
                p[e[i].to] = i;
                x = e[i].to;

                ok = true;
                break;
            }

        if (!ok) { // 修改距离标号
            int tmp = n - 1;
            for (int i = last[x]; ~i; i = e[i].prev)
                if (e[i].cap)
                    tmp = min(tmp, d[e[i].to] + 1);

            if (!--c[d[x]])
                break; // gap优化，一定要加上

            c[d[x] = tmp]++;
            cur[x] = last[x];

            if(x != s)
                x = e[p[x] ^ 1].to;
        }
    }
    return flow;
}

// 重要! main函数最前面一定要加上如下初始化
memset(last, -1, sizeof(last));
```

3.9.3 HLPP 最高标号预流推进

```
constexpr int maxn = 1205, maxe = 120005;

struct edge {
    int to, cap, prev;
} e[maxe * 2];

int n, m, s, t;
int last[maxn], cnte;
int h[maxn], gap[maxn * 2];
long long ex[maxn]; // 多余流量
bool inq[maxn];

struct cmp {
    bool operator() (int x, int y) const {
        return h[x] < h[y];
    }
};

priority_queue<int, vector<int>, cmp> heap;

void adde(int x, int y, int z) {
    e[cnte].to = y;
    e[cnte].cap = z;
    e[cnte].prev = last[x];
    last[x] = cnte++;
}

void addedge(int x, int y, int z) {
    adde(x, y, z);
    adde(y, x, 0);
}

bool bfs() {
    static int q[maxn];

    fill(h, h + n + 1, 2 * n); // 如果没有全局的n，记得改这里
    int head = 0, tail = 0;
    q[tail++] = t;
    h[t] = 0;

    while (head < tail) {
        int x = q[head++];
        for (int i = last[x]; ~i; i = e[i].prev)
            if (e[i ^ 1].cap && h[e[i].to] > h[x] + 1) {
                h[e[i].to] = h[x] + 1;
                q[tail++] = e[i].to;
            }
    }

    return h[s] < 2 * n;
}

void push(int x) {
    for (int i = last[x]; ~i; i = e[i].prev)
        if (e[i].cap && h[x] == h[e[i].to] + 1) {
            int d = min(ex[x], (long long)e[i].cap);

            e[i].cap -= d;
            e[i ^ 1].cap += d;
            ex[x] -= d;
            ex[e[i].to] += d;

            if (e[i].to != s && e[i].to != t && !inq[e[i].to]) {
                heap.push(e[i].to);
                inq[e[i].to] = true;
            }

            if (!ex[x])
                break;
        }
}

void relabel(int x) {
    h[x] = 2 * n;

    for (int i = last[x]; ~i; i = e[i].prev)
        if (e[i].cap)
            h[x] = min(h[x], h[e[i].to] + 1);
}

long long hlpp() {
    if (!bfs())
        return 0;

    // memset(gap, 0, sizeof(int) * 2 * n);
    h[s] = n;

    for (int i = 1; i <= n; i++)
        gap[h[i]]++;

    for (int i = last[s]; ~i; i = e[i].prev)
        if (e[i].cap) {
            int d = e[i].cap;

            e[i].cap -= d;
            e[i ^ 1].cap += d;
            ex[s] -= d;
            ex[e[i].to] += d;

            if (e[i].to != s && e[i].to != t && !inq[e[i].to]) {
                    heap.push(e[i].to);
                    inq[e[i].to] = true;
            }
        }

    while (!heap.empty()) {
        int x = heap.top();
        heap.pop();
        inq[x] = false;

        push(x);
        if (ex[x]) {
            if (!--gap[h[x]]) { // gap
                for (int i = 1; i <= n; i++)
                    if (i != s && i != t && h[i] > h[x])
                        h[i] = n + 1;
            }

            relabel(x);
            ++gap[h[x]];
            heap.push(x);
            inq[x] = true;
        }
    }

    return ex[t];
}

//记得初始化
memset(last, -1, sizeof(last));
```

3.10 费用流

3.10.1 SPFA费用流

```
constexpr int maxn = 20005, maxm = 200005;

struct edge {
    int to, prev, cap, w;
} e[maxm * 2];

int last[maxn], cnte, d[maxn], p[maxn]; // 记得把last初始化成-1, 不然会死循环
bool inq[maxn];

void spfa(int s) {

    memset(d, -63, sizeof(d));
    memset(p, -1, sizeof(p));

    queue<int> q;

    q.push(s);
    d[s] = 0;

    while (!q.empty()) {
        int x = q.front();
        q.pop();
        inq[x] = false;

        for (int i = last[x]; ~i; i = e[i].prev)
            if (e[i].cap) {
                int y = e[i].to;

                if (d[x] + e[i].w > d[y]) {
                    p[y] = i;
                    d[y] = d[x] + e[i].w;
                    if (!inq[y]) {
                        q.push(y);
                        inq[y] = true;
                    }
                }
            }
    }
}

int mcmf(int s, int t) {
    int ans = 0;

    while (spfa(s), d[t] > 0) {
        int flow = 0x3f3f3f3f;
        for (int x = t; x != s; x = e[p[x] ^ 1].to)
            flow = min(flow, e[p[x]].cap);

        ans += flow * d[t];

        for (int x = t; x != s; x = e[p[x] ^ 1].to) {
            e[p[x]].cap -= flow;
            e[p[x] ^ 1].cap += flow;
        }
    }

    return ans;
}

void add(int x, int y, int c, int w) {
    e[cnte].to = y;
    e[cnte].cap = c;
    e[cnte].w = w;

    e[cnte].prev = last[x];
    last[x] = cnte++;
}

void addedge(int x, int y, int c, int w) {
    add(x, y, c, w);
    add(y, x, 0, -w);
}
```

3.10.2 Dijkstra费用流

有的地方也叫原始-对偶费用流.

原理和求多源最短路的Johnson算法是一样的，都是给每个点维护一个势h_u，使得对任何有向边$u \to v$都满足$w + h_u - h_v \geq 0$.

如果有负费用则从s开始跑一遍SPFA初始化，否则可以直接初始化$h_u = 0$.

每次增广时得到的路径长度就是$d_{s,t} + h_t$，增广之后让所有$h_u = h'_u + d'_{s,u}$，直到$d_{s,t} = \infty$(最小费用最大流)或$d_{s,t} \geq 0$(最小费用流)为止.

注意最大费用流要转成取负之后的最小费用流，因为Dijkstra求的是最短路.

```
struct edge {
    int to, cap, prev, w;
} e[maxe * 2];

int last[maxn], cnte;

long long d[maxn], h[maxn];
int p[maxn];

bool vis[maxn];
int s, t;

void Adde(int x, int y, int z, int w) {
    e[cnte].to = y;
    e[cnte].cap = z;
    e[cnte].w = w;
    e[cnte].prev = last[x];
    last[x] = cnte++;
}

void addedge(int x, int y, int z, int w) {
    Adde(x, y, z, w);
    Adde(y, x, 0, -w);
}

void dijkstra() {
    memset(d, 63, sizeof(d));
    memset(vis, 0, sizeof(vis));

    priority_queue<pair<long long, int> > heap;

    d[s] = 0;
    heap.push(make_pair(0ll, s));

    while (!heap.empty()) {
        int x = heap.top().second;
        heap.pop();

        if (vis[x])
            continue;

        vis[x] = true;
        for (int i = last[x]; ~i; i = e[i].prev)
            if (e[i].cap > 0 && d[e[i].to] > d[x] + e[i].w + h[x] - h[e[i].to]) {
                d[e[i].to] = d[x] + e[i].w + h[x] - h[e[i].to];
                p[e[i].to] = i;
                heap.push(make_pair(-d[e[i].to], e[i].to));
            }
    }
}

pair<long long, long long> mcmf() {
    /*
    spfa();
    for (int i = 1; i <= t; i++)
        h[i] = d[i];
    // 如果初始有负权就像这样跑一遍SPFA预处理
    */

    long long flow = 0, cost = 0;

    while (dijkstra(), d[t] < 0x3f3f3f3f) {
        for (int i = 1; i <= t; i++)
            h[i] += d[i];

        int a = 0x3f3f3f3f;

        for (int x = t; x != s; x = e[p[x] ^ 1].to)
            a = min(a, e[p[x]].cap);

        flow += a;
        cost += (long long)a * h[t];

        for (int x = t; x != s; x = e[p[x] ^ 1].to) {
            e[p[x]].cap -= a;
            e[p[x] ^ 1].cap += a;
        }

    }

    return make_pair(flow, cost);
}

// 记得初始化
memset(last, -1, sizeof(last));
```

3.11 网络流原理

3.11.1 最大流

判断一条边是否必定满流

在残量网络中跑一遍Tarjan，如果某条满流边的两端处于同一SCC中则说明它不一定满流．(因为可以找出包含反向边的环，增广之后就不满流了.)

3.11.2 最小割

最小割输出一种方案

在残量网络上从S开始floodfill，源点可达的记为S集，不可达的记为T，如果一条边的起点在S集而终点在T集，就将其加入最小割中.

最小割的可行边与必须边

- 可行边：满流，且残量网络上不存在u到v的路径，也就是u和v不在同一SCC中．(实际上也就是最大流必定满流的边.)
- 必须边：满流，且残量网络上S可达u，v可达T.

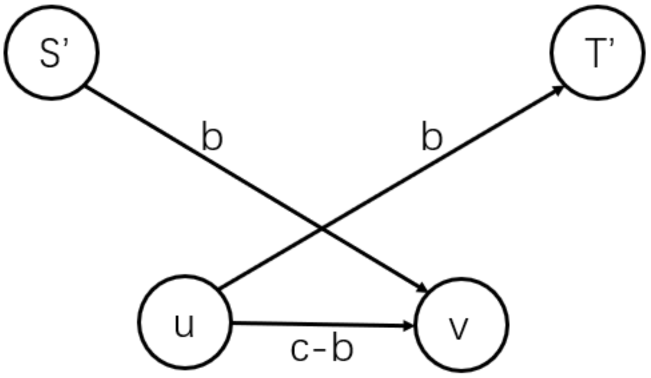
字典序最小的最小割

3.11.3 费用流

3.11.4 上下界网络流

有源汇上下界最大流

新建超级源汇S', T'，然后如图所示转化每一条边.



然后从S'到S, 从T到T'分别连容量为正无穷的边即可.

有源汇上下界最小流

按照上面的方法转换后先跑一遍最大流, 然后撤掉超级源汇, 反过来跑一次最大流退流, 最大流减去退掉的流量就是最小流.

无源汇上下界可行流

转化方法和上面的图是一样的, 只不过不需要考虑原有的源汇了.

在新图跑一遍最大流之后检查一遍辅助边, 如果有辅助边没满流则无解, 否则把每条边的流量加上b就是一组可行方案.

3.11.5 常见建图方法

3.11.6 例题

3.12 Prufer序列

对一棵有$n \geq 2$个结点的树, 它的Prufer编码是一个长为$n-2$, 且每个数都在$[1,n]$内的序列.

构造方法: 每次选取编号最小的叶子结点, 记录它的父亲, 然后把它删掉, 直到只剩两个点为止. (并且最后剩的两个点一定有一个是n号点.)

相应的, 由Prufer编码重构树的方法: 按顺序读入序列, 每次选取编号最小的且度数为1的结点, 把这个点和序列当前点连上, 然后两个点剩余度数同时-1.

Prufer编码的性质

- 每个至少2个结点的树都唯一对应一个Prufer编码. (当然也就可以做无根树哈希.)
- 每个点在Prufer序列中出现的次数恰好是度数-1. 所以如果给定某些点的度数然后求方案数, 就可以用简单的组合数解决.

最后, 构造和重构直接写都是$O(n \log n)$的, 想优化成线性需要一些技巧.

线性求Prufer序列代码:

```
// 0-based
vector<vector<int>> adj;
vector<int> parent;

void dfs(int v) {
    for (int u : adj[v]) {
        if (u != parent[v]) parent[u] = v, dfs(u);
    }
}

vector<int> pruefer_code() { // pruefer是德语
    int n = adj.size();
    parent.resize(n), parent[n - 1] = -1;
    dfs(n - 1);

    int ptr = -1;
    vector<int> degree(n);
    for (int i = 0; i < n; i++) {
        degree[i] = adj[i].size();
        if (degree[i] == 1 && ptr == -1) ptr = i;
    }

    vector<int> code(n - 2);
    int leaf = ptr;
    for (int i = 0; i < n - 2; i++) {
        int next = parent[leaf];
        code[i] = next;
        if (--degree[next] == 1 && next < ptr)
            leaf = next;
        else {
            ptr++;
            while (degree[ptr] != 1)
              ptr++;
            leaf = ptr;
        }
    }
    return code;
}
```

线性重构树代码:

```
// 0-based
vector<pair<int, int>> pruefer_decode(vector<int> const&
  ↪ code) {
    int n = code.size() + 2;
    vector<int> degree(n, 1);
    for (int i : code) degree[i]++;

    int ptr = 0;
    while (degree[ptr] != 1) ptr++;
    int leaf = ptr;

    vector<pair<int, int>> edges;
    for (int v : code) {
    edges.emplace_back(leaf, v);
    if (--degree[v] == 1 && v < ptr) {
        leaf = v;
    } else {
        ptr++;
        while (degree[ptr] != 1) ptr++;
        leaf = ptr;
    }
    }
    edges.emplace_back(leaf, n - 1);
    return edges;
}
```

3.13 弦图相关

Forked from templates of NEW CODE!!.

1. 团数 $\leq$ 色数，弦图团数 $=$ 色数
2. 设 $next(v)$ 表示 $N(v)$ 中最前的点．令 w* 表示所有满足 $A \in B$ 的 w 中最后的一个点，判断 $v \cup N(v)$ 是否为极大团，只需判断是否存在一个 w, 满足 $Next(w) = v$ 且 $|N(v)| + 1 \leq |N(w)|$ 即可．
3. 最小染色：完美消除序列从后往前依次给每个点染色，给每个点染上可以染的最小的颜色
4. 最大独立集：完美消除序列从前往后能选就选
5. 弦图最大独立集数 $=$ 最小团覆盖数，最小团覆盖：设最大独立集为 $\{p_1, p_2, \ldots, p_t\}$, 则 $\{p_1 \cup N(p_1), \ldots, p_t \cup N(p_t)\}$ 为最小团覆盖

4. 数据结构

4.1 线段树

4.1.1 非递归线段树

~~让fstqwq手撕~~

- 如果$M = 2^k$，则只能维护$[1, M - 2]$范围
- 找叶子：i对应的叶子就是$i + M$
- 单点修改：找到叶子然后向上跳
- 区间查询：左右区间各扩展一位，转换成开区间查询

```
int query(int l, int r) {
    l += M - 1;
    r += M + 1;

    int ans = 0;
    while (l ^ r != 1) {
        ans += sum[l ^ 1] + sum[r ^ 1];

        l >>= 1;
        r >>= 1;
    }

    return ans;
}
```

区间修改要标记永久化,并且求区间和和求最值的代码不太一样

区间加，区间求和

```
void update(int l, int r, int d) {
    int len = 1, cntl = 0, cntr = 0; // cntl, cntr是左右两边分别实际修改的区间长度
    for (l += n - 1, r += n + 1; l ^ r ^ 1; l >>= 1, r >>= 1, len <<= 1) {
        tree[l] += cntl * d, tree[r] += cntr * d;
        if (~l & 1) tree[l ^ 1] += d * len, mark[l ^ 1] += d, cntl += len;
        if (r & 1) tree[r ^ 1] += d * len, mark[r ^ 1] += d, cntr += len;
    }

    for (; l; l >>= 1, r >>= 1)
        tree[l] += cntl * d, tree[r] += cntr * d;
}

int query(int l, int r) {
    int ans = 0, len = 1, cntl = 0, cntr = 0;
    for (l += n - 1, r += n + 1; l ^ r ^ 1; l >>= 1, r >>= 1, len <<= 1) {
        ans += cntl * mark[l] + cntr * mark[r];
        if (~l & 1) ans += tree[l ^ 1], cntl += len;
        if (r & 1) ans += tree[r ^ 1], cntr += len;
    }

    for (; l; l >>= 1, r >>= 1)
        ans += cntl * mark[l] + cntr * mark[r];

    return ans;
}
```

区间加，区间求最大值

```
void update(int l, int r, int d) {
    for (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
        if (l < N) {
            tree[l] = max(tree[l << 1], tree[l << 1 | 1]) + mark[l];
            tree[r] = max(tree[r << 1], tree[r << 1 | 1]) + mark[r];
        }

        if (~l & 1) {
            tree[l ^ 1] += d;
            mark[l ^ 1] += d;
        }
        if (r & 1) {
            tree[r ^ 1] += d;
            mark[r ^ 1] += d;
        }
    }

    for (; l; l >>= 1, r >>= 1)
        if (l < N) tree[l] = max(tree[l << 1], tree[l << 1 | 1]) + mark[l],
                    tree[r] = max(tree[r << 1], tree[r << 1 | 1]) + mark[r];
}

void query(int l, int r) {
    int maxl = -INF, maxr = -INF;

    for (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
        maxl += mark[l];
        maxr += mark[r];

        if (~l & 1)
            maxl = max(maxl, tree[l ^ 1]);
        if (r & 1)
            maxr = max(maxr, tree[r ^ 1]);
    }

    while (l) {
        maxl += mark[l];
        maxr += mark[r];

        l >>= 1;
        r >>= 1;
    }

    return max(maxl, maxr);
}
```

4.1.2 线段树维护矩形并

为线段树的每个结点维护$cover_i$表示这个区间被完全覆盖的次数. 更新时分情况讨论，如果当前区间已被完全覆盖则长度就是区间长度，否则长度是左右儿子相加.

```
constexpr int maxn = 100005, maxm = maxn * 70;

int lc[maxm], rc[maxm], cover[maxm], sum[maxm], root, seg_cnt;
int s, t, d;

void refresh(int l, int r, int o) {
    if (cover[o])
        sum[o] = r - l + 1;
    else
        sum[o] = sum[lc[o]] + sum[rc[o]];
}

void modify(int l, int r, int &o) {
    if (!o)
        o = ++seg_cnt;

    if (s <= l && t >= r) {
        cover[o] += d;
        refresh(l, r, o);

        return;
    }

    int mid = (l + r) / 2;

    if (s <= mid)
        modify(l, mid, lc[o]);
    if (t > mid)
        modify(mid + 1, r, rc[o]);

    refresh(l, r, o);
}

struct modi {
    int x, l, r, d;

    bool operator < (const modi &o) {
        return x < o.x;
    }
} a[maxn * 2];

int main() {

    int n;
    scanf("%d", &n);

    for (int i = 1; i <= n; i++) {
        int lx, ly, rx, ry;
        scanf("%d%d%d%d", &lx, &ly, &rx, &ry);

        a[i * 2 - 1] = {lx, ly + 1, ry, 1};
        a[i * 2] = {rx, ly + 1, ry, -1};
    }

    sort(a + 1, a + n * 2 + 1);

    int last = -1;
    long long ans = 0;

    for (int i = 1; i <= n * 2; i++) {
        if (last != -1)
            ans += (long long)(a[i].x - last) * sum[1];
        last = a[i].x;

        s = a[i].l;
        t = a[i].r;
        d = a[i].d;

        modify(1, 1e9, root);
    }

    printf("%lld\n", ans);

    return 0;
}
```

4.1.3 主席树离线求区间mex(待完成)

4.2 陈丹琦分治

4.2.1 动态图连通性(分治并查集)

```
vector<pair<int, int> > seg[(1 << 22) + 5];

int s, t;
pair<int, int> d;

void add(int l, int r, int o) {
    if (s > t)
        return;

    if (s <= l && t >= r) {
        seg[o].push_back(d);
        return;
    }

    int mid = (l + r) / 2;

    if (s <= mid)
        add(l, mid, o * 2);
    if (t > mid)
        add(mid + 1, r, o * 2 + 1);
}

int ufs[maxn], sz[maxn], stk[maxn], top;

int findufs(int x) {
    while (ufs[x] != x)
        x = ufs[x];

    return ufs[x];
}

void link(int x, int y) {
    x = findufs(x);
    y = findufs(y);

    if (x == y)
        return;

    if (sz[x] < sz[y])
        swap(x, y);

    ufs[y] = x;
    sz[x] += sz[y];
    stk[++top] = y;
}

int ans[maxm];

void solve(int l, int r, int o) {
    int tmp = top;

    for (auto pi : seg[o])
        link(pi.first, pi.second);

    if (l == r)
        ans[l] = top;
    else {
        int mid = (l + r) / 2;

        solve(l, mid, o * 2);
        solve(mid + 1, r, o * 2 + 1);
    }

    for (int i = top; i > tmp; i--) {
        int x = stk[i];

        sz[ufs[x]] -= sz[x];
        ufs[x] = x;
    }

    top = tmp;
}
```

```cpp

map<pair<int, int>, int> mp;
```

4.2.2 四维偏序

```cpp
// 四维偏序

void CDQ1(int l, int r) {
    if (l >= r)
        return;

    int mid = (l + r) / 2;

    CDQ1(l, mid);
    CDQ1(mid + 1, r);

    int i = l, j = mid + 1, k = l;

    while (i <= mid && j <= r) {
        if (a[i].x < a[j].x) {
            a[i].ins = true;
            b[k++] = a[i++];
        }
        else {
            a[j].ins = false;
            b[k++] = a[j++];
        }
    }

    while (i <= mid) {
        a[i].ins = true;
        b[k++] = a[i++];
    }

    while (j <= r) {
        a[j].ins = false;
        b[k++] = a[j++];
    }

    copy(b + l, b + r + 1, a + l); // 后面的分治会破坏排序, 所以要复制一份

    CDQ2(l, r);
}

void CDQ2(int l, int r) {
    if (l >= r)
        return;

    int mid = (l + r) / 2;

    CDQ2(l, mid);
    CDQ2(mid + 1, r);

    int i = l, j = mid + 1, k = l;

    while (i <= mid && j <= r) {
        if (b[i].y < b[j].y) {
            if (b[i].ins)
                add(b[i].z, 1); // 树状数组

            t[k++] = b[i++];
        }
        else{
            if (!b[j].ins)
                ans += query(b[j].z - 1);

            t[k++] = b[j++];
        }
    }

    while (i <= mid) {
        if (b[i].ins)
            add(b[i].z, 1);

        t[k++] = b[i++];
    }

    while (j <= r) {
        if (!b[j].ins)
            ans += query(b[j].z - 1);

        t[k++] = b[j++];
    }

    for (i = l; i <= mid; i++)
        if (b[i].ins)
            add(b[i].z, -1);

    copy(t + l, t + r + 1, b + l);
}
```

4.3 整体二分

修改和询问都要划分, 备份一下, 递归之前copy回去.

如果是满足可减性的问题(例如查询区间k小数)可以直接在划分的时候把询问的k修改一下. 否则需要维护一个全局的数据结构, 一般来说可以先递归右边再递归左边, 具体维护方法视情况而定.

以下代码以ZJOI K大数查询为例(区间都添加一个数, 询问区间k大数).

```cpp
int op[maxn], ql[maxn], qr[maxn]; // 1: modify  2: query
long long qk[maxn]; // 修改和询问可以一起存

int ans[maxn];

void solve(int l, int r, vector<int> v) { // 如果想卡常可以用数组, 然后只需要传一个数组的l, r⍰ 递归的时候类似归并反过来, 开两个辅助数组, 处理完再复制回去即可
    if (v.empty())
        return;

    if (l == r) {
        for (int i : v)
            if (op[i] == 2)
                ans[i] = l;

        return;
    }

    int mid = (l + r) / 2;

    vector<int> vl, vr;

    for (int i : v) {
        if (op[i] == 1) {
            if (qk[i] <= mid)
                vl.push_back(i);
            else {
                update(ql[i], qr[i], 1); // update是区间加
                vr.push_back(i);
            }
        }
        else {
            long long tmp = query(ql[i], qr[i]);

            if (qk[i] <= tmp) // 因为是k大数查询
```

```
                vr.push_back(i);
            else {
                qk[i] -= tmp;
                vl.push_back(i);
            }
        }
    }

    for (int i : vr)
        if (op[i] == 1)
            update(ql[i], qr[i], -1);

    v.clear();

    solve(l, mid, vl);
    solve(mid + 1, r, vr);
}

int main() {
    int n, m;
    scanf("%d%d", &n, &m);

    M = 1;
    while (M < n + 2)
        M *= 2;

    for (int i = 1; i <= m; i++)
        scanf("%d%d%d%lld", &op[i], &ql[i], &qr[i],
          ↪ &qk[i]);

    vector<int> v;
    for (int i = 1; i <= m; i++)
        v.push_back(i);

    solve(1, 1e9, v);

    for (int i = 1; i <= m; i++)
        if (op[i] == 2)
            printf("%d\n", ans[i]);

    return 0;
}
```

4.4 平衡树

pb_ds平衡树在misc(倒数第二章)里.

4.4.1 Treap

```
// 注意: 相同键值可以共存

struct node { // 结点类定义
    int key, size, p; // 分别为键值, 子树大小, 优先度
    node *ch[2]; // 0表示左儿子, 1表示右儿子

    node(int key = 0) : key(key), size(1), p(rand()) {}

    void refresh() {
        size = ch[0] -> size + ch[1] -> size + 1;
    } // 更新子树大小(和附加信息, 如果有的话)
} null[maxn], *root = null, *ptr = null; // 数组名叫
  ↪ 做null是为了方便开哨兵节点
// 如果需要删除而空间不能直接开下所有结点, 则需要再写一个
  ↪ 垃圾回收
// 注意: 数组里的元素一定不能delete, 否则会导致RE

// 重要!在主函数最开始一定要加上以下预处理:
null -> ch[0] = null -> ch[1] = null;
null -> size = 0;

// 伪构造函数 O(1)
// 为了方便, 在结点类外面再定义一个伪构造函数
node *newnode(int x) { // 键值为x
    *++ptr = node(x);
    ptr -> ch[0] = ptr -> ch[1] = null;
    return ptr;
}

// 插入键值 期望O(\log n)
// 需要调用旋转
void insert(int x, node *&rt) { // rt为当前结点, 建议调用时
  ↪ 传入root, 下同
    if (rt == null) {
        rt = newnode(x);
        return;
    }

    int d = x > rt -> key;
    insert(x, rt -> ch[d]);
    rt -> refresh();

    if (rt -> ch[d] -> p < rt -> p)
        rot(rt, d ^ 1);
}

// 删除一个键值 期望O(\log n)
// 要求键值必须存在至少一个, 否则会导致RE
// 需要调用旋转
void erase(int x, node *&rt) {
    if (x == rt -> key) {
        if (rt -> ch[0] != null && rt -> ch[1] != null) {
            int d = rt -> ch[0] -> p < rt -> ch[1] -> p;
            rot(rt, d);
            erase(x, rt -> ch[d]);
        }
        else
            rt = rt -> ch[rt -> ch[0] == null];
    }
    else
        erase(x, rt -> ch[x > rt -> key]);

    if (rt != null)
        rt -> refresh();
}

// 求元素的排名(严格小于键值的个数 + 1) 期望O(\log n)
// 非递归
int rank(int x, node *rt) {
    int ans = 1, d;
    while (rt != null) {
        if ((d = x > rt -> key))
            ans += rt -> ch[0] -> size + 1;

        rt = rt -> ch[d];
    }

    return ans;
}

// 返回排名第k(从1开始)的键值对应的指针 期望O(\log n)
// 非递归
node *kth(int x, node *rt) {
    int d;
    while (rt != null) {
        if (x == rt -> ch[0] -> size + 1)
            return rt;

        if ((d = x > rt -> ch[0] -> size))
            x -= rt -> ch[0] -> size + 1;

        rt = rt -> ch[d];
    }

    return rt;
}

// 返回前驱(最大的比给定键值小的键值)对应的指针 期望O(\log n)
// 非递归
node *pred(int x, node *rt) {
    node *y = null;
    int d;

    while (rt != null) {
        if ((d = x > rt -> key))
            y = rt;

        rt = rt -> ch[d];
    }

    return y;
}

// 返回后继⍰最小的比给定键值大的键值⍰对应的指针 期望O(\log n)
// 非递归
node *succ(int x, node *rt) {
    node *y = null;
    int d;

    while (rt != null) {
        if ((d = x < rt -> key))
            y = rt;

        rt = rt -> ch[d ^ 1];
    }

    return y;
}

// 旋转(Treap版本) O(1)
// 平衡树基础操作
// 要求对应儿子必须存在，否则会导致后续各种莫名其妙的问题
void rot(node *&x, int d) { // x为被转下去的结点，会被修改以维护树结构
    node *y = x -> ch[d ^ 1];

    x -> ch[d ^ 1] = y -> ch[d];
    y -> ch[d] = x;

    x -> refresh();
    (x = y) -> refresh();
}
```

4.4.2 无旋Treap/可持久化Treap

```
struct node {
    int val, size;
    node *ch[2];

    node(int val) : val(val), size(1) {}

    inline void refresh() {
        size = ch[0] -> size + ch[1] -> size;
    }


} null[maxn];

node *copied(node *x) { // 如果不用可持久化的话，直接用就行了
    return new node(*x);
}

node *merge(node *x, node *y) {
    if (x == null)
        return y;
    if (y == null)
        return x;

    node *z;
    if (rand() % (x -> size + y -> size) < x -> size) {
        z = copied(y);
        z -> ch[0] = merge(x, y -> ch[0]);
    }
    else {
        z = copied(x);
        z -> ch[1] = merge(x -> ch[1], y);
    }

    z -> refresh(); // 因为每次只有一边会递归到儿子，所以z不可能取到null
    return z;
}

pair<node*, node*> split(node *x, int k) { // 左边大小为k
    if (x == null)
        return make_pair(null, null);

    pair<node*, node*> pi(null, null);

    if (k <= x -> ch[0] -> size) {
        pi = split(x -> ch[0], k);

        node *z = copied(x);
        z -> ch[0] = pi.second;
        z -> refresh();
        pi.second = z;
    }
    else {
        pi = split(x -> ch[1], k - x -> ch[0] -> size - 1);

        node *y = copied(x);
        y -> ch[1] = pi.first;
        y -> refresh();
        pi.first = y;
    }

    return pi;
}

// 记得初始化null
int main() {
    for (int i = 0; i <= n; i++)
        null[i].ch[0] = null[i].ch[1] = null;
    null -> size = 0;

    // do something

    return 0;
}
```

4.4.3 Splay

如果插入的话可以直接找到底然后splay一下，也可以直接splay前驱后继.

```cpp
#define dir(x) ((x) == (x) -> p -> ch[1])

struct node {
    int size;
    bool rev;
    node *ch[2],*p;

    node() : size(1), rev(false) {}

    void pushdown() {
        if(!rev)
            return;

        ch[0] -> rev ^= true;
        ch[1] -> rev ^= true;
        swap(ch[0], ch[1]);

        rev=false;
    }

    void refresh() {
        size = ch[0] -> size + ch[1] -> size + 1;
    }
} null[maxn], *root = null;

void rot(node *x, int d) {
    node *y = x -> ch[d ^ 1];

    if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
        y -> ch[d] -> p = x;
    ((y -> p = x -> p) != null ? x -> p -> ch[dir(x)] : root) = y;
    (y -> ch[d] = x) -> p = y;

    x -> refresh();
    y -> refresh();
}

void splay(node *x, node *t) {
    while (x -> p != t) {
        if (x -> p -> p == t) {
            rot(x -> p, dir(x) ^ 1);
            break;
        }

        if (dir(x) == dir(x -> p))
            rot(x -> p -> p, dir(x -> p) ^ 1);
        else
            rot(x -> p, dir(x) ^ 1);
        rot(x -> p, dir(x) ^ 1);
    }
}

node *kth(int k, node *o) {
    int d;
    k++; // 因为最左边有一个哨兵

    while (o != null) {
        o -> pushdown();

        if (k == o -> ch[0] -> size + 1)
            return o;

        if ((d = k > o -> ch[0] -> size))
            k -= o -> ch[0] -> size + 1;
        o = o -> ch[d];
    }

    return null;
}

void reverse(int l, int r) {
    splay(kth(l - 1));
    splay(kth(r + 1), root);

    root -> ch[1] -> ch[0] -> rev ^= true;
}

int n, m;

int main() {
    null -> size = 0;
    null -> ch[0] = null -> ch[1] = null -> p = null;

    scanf("%d%d", &n, &m);
    root = null + n + 1;
    root -> ch[0] = root -> ch[1] = root -> p = null;

    for (int i = 1; i <= n; i++) {
        null[i].ch[1] = null[i].p = null;
        null[i].ch[0] = root;
        root -> p = null + i;
        (root = null + i) -> refresh();
    }

    null[n + 2].ch[1] = null[n + 2].p = null;
    null[n + 2].ch[0] = root; // 这里直接建成一条链的，如果想减少常数也可以递归建一个平衡的树
    root -> p = null + n + 2; // 总之记得建两个哨兵，这样splay起来不需要特判
    (root = null + n + 2) -> refresh();

    // Do something

    return 0;
}
```

4.5 树链剖分

4.5.1 动态树形DP(最大权独立集)

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

using namespace std;

constexpr int maxn = 100005, maxm = 262155, inf = 0x3f3f3f3f;

struct binary_heap {
    priority_queue<int> q, t;

    binary_heap() {}

    void push(int x) {
        q.push(x);
    }

    void erase(int x) {
        t.push(x);
    }

    int top() {
        while (!t.empty() && q.top() == t.top()) {
            q.pop();
            t.pop();
        }

        return q.top();
    }
} heap;

int pool[maxm][2][2], (*pt)[2][2] = pool;

void merge(int a[2][2], int b[2][2]) {
    static int c[2][2];
    memset(c, 0, sizeof(c));

    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            for (int k = 0; k < 2; k++)
                if (!(j && k))
                    for (int t = 0; t < 2; t++)
                        c[i][t] = max(c[i][t], a[i][j] +
                        ↪ b[k][t]);

    memcpy(a, c, sizeof(c));
}

vector<pair<int, int> > tw;

struct seg_tree {
    int (*tr)[2][2], n;

    int s, d[2];

    seg_tree() {}

    void update(int o) {
        memcpy(tr[o], tr[o * 2], sizeof(int) * 4);
        merge(tr[o], tr[o * 2 + 1]);
    }

    void build(int l, int r, int o) {
        if (l == r) {
            tr[o][0][0] = tw[l - 1].first;
            tr[o][0][1] = tr[o][1][0] = -inf;
            tr[o][1][1] = tw[l - 1].second;

            return;
        }

        int mid = (l + r) / 2;

        build(l, mid, o * 2);
        build(mid + 1, r, o * 2 + 1);

        update(o);
    }

    void modify(int l, int r, int o) {
        if (l == r) {
            tr[o][0][0] = d[0];
            tr[o][0][1] = tr[o][1][0] = -inf;
            tr[o][1][1] = d[1];

            return;
        }

        int mid = (l + r) / 2;

        if (s <= mid)
            modify(l, mid, o * 2);
        else
            modify(mid + 1, r, o * 2 + 1);

        update(o);
    }

    int getval() {
        int ans = 0;
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 2; j++)
                ans = max(ans, tr[1][i][j]);

        return ans;
    }

    pair<int, int> getpair() {
        int ans[2] = {0};
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 2; j++)
                ans[i] = max(ans[i], tr[1][i][j]);

        return make_pair(ans[0], ans[1]);
    }

    void build(int len) {
        n = len;
        int N = 1;
        while (N < n * 2)
            N *= 2;

        tr = pt;
        pt += N;

        build(1, n, 1);
    }

    void modify(int x, int dat[2]) {
        s = x;
        for (int i = 0; i < 2; i++)
            d[i] = dat[i];
        modify(1, n, 1);
    }
} seg[maxn];

vector<int> G[maxn];

int p[maxn], d[maxn], sz[maxn], son[maxn], top[maxn];
int dp[maxn][2], dptr[maxn][2], w[maxn];

void dfs1(int x) {
    d[x] = d[p[x]] + 1;
    sz[x] = 1;

    for (int y : G[x])
        if (y != p[x]) {
            p[y] = x;
            dfs1(y);

            if (sz[y] > sz[son[x]])
                son[x] = y;

            sz[x] += sz[y];
        }
}

void dfs2(int x) {
    if (x == son[p[x]])
        top[x] = top[p[x]];
    else
        top[x] = x;

    for (int y : G[x])
        if (y != p[x])
            dfs2(y);

    dp[x][1] = w[x];
    for (int y : G[x])
        if (y != p[x] && y != son[x]) {
            dp[x][1] += dptr[y][0];
            dp[x][0] += max(dptr[y][0], dptr[y][1]);
        }

    if (top[x] == x) {
        tw.clear();

        for (int u = x; u; u = son[u])
            tw.push_back(make_pair(dp[u][0], dp[u][1]));

        seg[x].build((int)tw.size());

        tie(dptr[x][0], dptr[x][1]) = seg[x].getpair();

        heap.push(seg[x].getval());
    }
}

void modify(int x, int dat) {
    dp[x][1] -= w[x];
    dp[x][1] += (w[x] = dat);

    while (x) {
        if (p[top[x]]) {
            dp[p[top[x]]][0] -= max(dptr[top[x]][0], dptr[top[x]][1]);
            dp[p[top[x]]][1] -= dptr[top[x]][0];
        }

        heap.erase(seg[top[x]].getval());
        seg[top[x]].modify(d[x] - d[top[x]] + 1, dp[x]);
        heap.push(seg[top[x]].getval());

        tie(dptr[top[x]][0], dptr[top[x]][1]) = seg[top[x]].getpair();

        if (p[top[x]]) {
            dp[p[top[x]]][0] += max(dptr[top[x]][0], dptr[top[x]][1]);
            dp[p[top[x]]][1] += dptr[top[x]][0];
        }

        x = p[top[x]];
    }
}

int main() {

    int n, m;
    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++)
        scanf("%d", &w[i]);

    for (int i = 1; i < n; i++) {
        int x, y;
        scanf("%d%d", &x, &y);

        G[x].push_back(y);
        G[y].push_back(x);
    }

    dfs1(1);
    dfs2(1);

    while (m--) {
        int x, dat;
        scanf("%d%d", &x, &dat);

        modify(x, dat);

        printf("%d\n", heap.top());
    }

    return 0;
}
```

4.6 树分治

4.6.1 动态树分治

```
// 为了减小常数，这里采用bfs写法，实测预处理比dfs快将近一半
// 以下以维护一个点到每个黑点的距离之和为例

// 全局数组定义
vector<int> G[maxn], W[maxn];
int size[maxn], son[maxn], q[maxn];
int p[maxn], depth[maxn], id[maxn][20], d[maxn][20]; // id是对应层所在子树的根
int a[maxn], ca[maxn], b[maxn][20], cb[maxn][20]; // 维护距离和用的
bool vis[maxn], col[maxn];

// 建树 总计O(n\log n)
// 需要调用找重心和预处理距离，同时递归调用自身
void build(int x, int k, int s, int pr) { // 结点，深度，连通块大小，点分树上的父亲
    x = getcenter(x, s);
    vis[x] = true;
    depth[x] = k;
    p[x] = pr;

    for (int i = 0; i < (int)G[x].size(); i++)
        if (!vis[G[x][i]]) {
            d[G[x][i]][k] = W[x][i];
            p[G[x][i]] = x;

            getdis(G[x][i],k,G[x][i]); // bfs每个子树，预处理距离
        }

    for (int i = 0; i < (int)G[x].size(); i++)
        if (!vis[G[x][i]])
            build(G[x][i], k + 1, size[G[x][i]], x); // 递归建树
}

// 找重心 O(n)
int getcenter(int x, int s) {
    int head = 0, tail = 0;
    q[tail++] = x;

    while (head != tail) {
        x = q[head++];
        size[x] = 1; // 这里不需要清空，因为以后要用的话一定会重新赋值
        son[x] = 0;

        for (int i = 0; i < (int)G[x].size(); i++)
            if (!vis[G[x][i]] && G[x][i] != p[x]) {
                p[G[x][i]] = x;
                q[tail++] = G[x][i];
            }
    }

```

```cpp
    for (int i = tail - 1; i; i--) {
        x = q[i];
        size[p[x]] += size[x];

        if (size[x] > size[son[p[x]]])
            son[p[x]] = x;
    }

    x = q[0];
    while (son[x] && size[son[x]] * 2 >= s)
        x = son[x];

    return x;
}

// 预处理距离 O(n)
// 方便起见，这里直接用了笨一点的方法，O(n\log n)全存下来
void getdis(int x, int k, int rt) {
    int head = 0, tail = 0;
    q[tail++] = x;

    while (head != tail) {
        x = q[head++];
        size[x] = 1;
        id[x][k] = rt;

        for (int i = 0; i < (int)G[x].size(); i++)
            if (!vis[G[x][i]] && G[x][i] != p[x]) {
                p[G[x][i]] = x;
                d[G[x][i]][k] = d[x][k] + W[x][i];

                q[tail++] = G[x][i];
            }
    }

    for (int i = tail - 1; i; i--)
        size[p[q[i]]] += size[q[i]]; // 后面递归建树要用到子问题大小
}

// 修改 O(\log n)
void modify(int x) {
    if (col[x])
        ca[x]--;
    else
        ca[x]++; // 记得先特判自己作为重心的那层

    for (int u = p[x], k = depth[x] - 1; u; u = p[u], k--) {
        if (col[x]) {
            a[u] -= d[x][k];
            ca[u]--;

            b[id[x][k]][k] -= d[x][k];
            cb[id[x][k]][k]--;
        }
        else {
            a[u] += d[x][k];
            ca[u]++;

            b[id[x][k]][k] += d[x][k];
            cb[id[x][k]][k]++;
        }
    }

    col[x] ^= true;
}

// 询问 O(\log n)
int query(int x) {
    int ans = a[x]; // 特判自己是重心的那层

    for (int u = p[x], k = depth[x] - 1; u; u = p[u], k--)
        ans += a[u] - b[id[x][k]][k] + d[x][k] * (ca[u] - cb[id[x][k]][k]);

    return ans;
}
```

4.6.2 紫荆花之恋

```cpp
const int maxn = 100010;
const double alpha = 0.7;
struct node {
    static int randint() {
        static int a = 1213, b = 97818217, p = 998244353, x = 751815431;
        x = a * x + b;
        x %= p;
        return x < 0 ? (x += p) : x;
    }

    int data, size, p;
    node *ch[2];

    node(int d): data(d), size(1), p(randint()) {}

    inline void refresh() {
        size = ch[0]->size + ch[1]->size + 1;
    }
} *null = new node(0), *root[maxn], *root1[maxn][50];

void addnode(int, int);
void rebuild(int, int, int, int);
void dfs_getcenter(int, int, int &);
void dfs_getdis(int, int, int, int);
void dfs_destroy(int, int);
void insert(int, node *&);
int order(int, node *);
void destroy(node *&);
void rot(node *&, int);

vector<int>G[maxn], W[maxn];
int size[maxn] = {0}, siz[maxn][50] = {0}, son[maxn];
bool vis[maxn];
int depth[maxn], p[maxn], d[maxn][50], id[maxn][50];
int n, m, w[maxn], tmp;
long long ans = 0;

int main() {
    null->size = 0;
    null->ch[0] = null->ch[1] = null;

    scanf("%*d%d", &n);
    fill(vis, vis + n + 1, true);
    fill(root, root + n + 1, null);

    for (int i = 0; i <= n; i++)
        fill(root1[i], root1[i] + 50, null);

    scanf("%*d%*d%d", &w[1]);
    insert(-w[1], root[1]);
    size[1] = 1;
    printf("0\n");

    for (int i = 2; i <= n; i++) {
        scanf("%d%d%d", &p[i], &tmp, &w[i]);
        p[i] ^= (ans % (int)1e9);
        G[i].push_back(p[i]);
        W[i].push_back(tmp);
        G[p[i]].push_back(i);
        W[p[i]].push_back(tmp);
        addnode(i, tmp);
        printf("%lld\n", ans);
    }

    return 0;
}

void addnode(int x, int z) { //wj-dj>=di-wi
    depth[x] = depth[p[x]] + 1;
    size[x] = 1;
    insert(-w[x], root[x]);
    int rt = 0;

    for (int u = p[x], k = depth[p[x]]; u; u = p[u], k--) {
        if (u == p[x]) {
            id[x][k] = x;
            d[x][k] = z;
        }
        else {
            id[x][k] = id[p[x]][k];
            d[x][k] = d[p[x]][k] + z;
        }

        ans += order(w[x] - d[x][k], root[u]) - order(w[x] - d[x][k], root1[id[x][k]][k]);
        insert(d[x][k] - w[x], root[u]);
        insert(d[x][k] - w[x], root1[id[x][k]][k]);
        size[u]++;
        siz[id[x][k]][k]++;

        if (siz[id[x][k]][k] > size[u]*alpha + 5)
            rt = u;
    }

    id[x][depth[x]] = 0;
    d[x][depth[x]] = 0;

    if (rt) {
        dfs_destroy(rt, depth[rt]);
        rebuild(rt, depth[rt], size[rt], p[rt]);
    }
}

void rebuild(int x, int k, int s, int pr) {
    int u = 0;
    dfs_getcenter(x, s, u);
    vis[x = u] = true;
    p[x] = pr;
    depth[x] = k;
    size[x] = s;
    d[x][k] = id[x][k] = 0;
    destroy(root[x]);
    insert(-w[x], root[x]);

    if (s <= 1)
        return;

    for (int i = 0; i < (int)G[x].size(); i++)
        if (!vis[G[x][i]]) {
            p[G[x][i]] = 0;
            d[G[x][i]][k] = W[x][i];
            siz[G[x][i]][k] = p[G[x][i]] = 0;
            destroy(root1[G[x][i]][k]);
            dfs_getdis(G[x][i], x, G[x][i], k);
        }

    for (int i = 0; i < (int)G[x].size(); i++)
        if (!vis[G[x][i]])
            rebuild(G[x][i], k + 1, size[G[x][i]], x);
}

void dfs_getcenter(int x, int s, int &u) {
    size[x] = 1;
    son[x] = 0;

    for (int i = 0; i < (int)G[x].size(); i++)
        if (!vis[G[x][i]] && G[x][i] != p[x]) {
            p[G[x][i]] = x;
            dfs_getcenter(G[x][i], s, u);
            size[x] += size[G[x][i]];

            if (size[G[x][i]] > size[son[x]])
                son[x] = G[x][i];
        }

    if (!u || max(s - size[x], size[son[x]]) < max(s - size[u], size[son[u]]))
        u = x;
}

void dfs_getdis(int x, int u, int rt, int k) {
    insert(d[x][k] - w[x], root[u]);
    insert(d[x][k] - w[x], root1[rt][k]);
    id[x][k] = rt;
    siz[rt][k]++;
    size[x] = 1;

    for (int i = 0; i < (int)G[x].size(); i++)
        if (!vis[G[x][i]] && G[x][i] != p[x]) {
            p[G[x][i]] = x;
            d[G[x][i]][k] = d[x][k] + W[x][i];
            dfs_getdis(G[x][i], u, rt, k);
            size[x] += size[G[x][i]];
        }
}

void dfs_destroy(int x, int k) {
    vis[x] = false;

    for (int i = 0; i < (int)G[x].size(); i++)
        if (depth[G[x][i]] >= k && G[x][i] != p[x]) {
            p[G[x][i]] = x;
            dfs_destroy(G[x][i], k);
        }
}

void insert(int x, node *&rt) {
    if (rt == null) {
        rt = new node(x);
        rt->ch[0] = rt->ch[1] = null;
        return;
    }

    int d = x >= rt->data;
    insert(x, rt->ch[d]);
    rt->refresh();

    if (rt->ch[d]->p < rt->p)
        rot(rt, d ^ 1);
}

int order(int x, node *rt) {
    int ans = 0, d;
    x++;

    while (rt != null) {
        if ((d = x > rt->data))
            ans += rt->ch[0]->size + 1;

        rt = rt->ch[d];
    }

    return ans;
}

void destroy(node *&x) {
    if (x == null)
        return;

    destroy(x->ch[0]);
    destroy(x->ch[1]);
    delete x;
    x = null;
}

void rot(node *&x, int d) {
    node *y = x->ch[d ^ 1];
    x->ch[d ^ 1] = y->ch[d];
    y->ch[d] = x;
    x->refresh();
    (x = y)->refresh();
}
```

4.7 LCT动态树

4.7.1 不换根(弹飞绵羊)

```
#define isroot(x) ((x) != (x) -> p -> ch[0] && (x) != (x) -> p -> ch[1]) // 判断是不是Splay的根
#define dir(x) ((x) == (x) -> p -> ch[1]) // 判断它是它父亲的左 / 右儿子

struct node { // 结点类定义
    int size; // Splay的子树大小
    node *ch[2], *p;

    node() : size(1) {}
    void refresh() {
        size = ch[0] -> size + ch[1] -> size + 1;
    } // 附加信息维护
} null[maxn];

// 在主函数开头加上这句初始化
null -> size = 0;

// 初始化结点
void initalize(node *x) {
    x -> ch[0] = x -> ch[1] = x -> p = null;
}

// Access 均摊O(\log n)
// LCT核心操作，把结点到根的路径打通，顺便把与重儿子的连边变成轻边
// 需要调用splay
node *access(node *x) {
    node *y = null;

    while (x != null) {
        splay(x);

        x -> ch[1] = y;
        (y = x) -> refresh();

        x = x -> p;
    }

    return y;
}

// Link 均摊O(\log n)
// 把x的父亲设为y
// 要求x必须为所在树的根节点⍰否则会导致后续各种莫名其妙的问题
// 需要调用splay
void link(node *x, node *y) {
    splay(x);
    x -> p = y;
}

// Cut 均摊O(\log n)
// 把x与其父亲的连边断掉
// x可以是所在树的根节点，这时此操作没有任何实质效果
// 需要调用access和splay
void cut(node *x) {
    access(x);
    splay(x);

    x -> ch[0] -> p = null;
    x -> ch[0] = null;

    x -> refresh();
}

// Splay 均摊O(\log n)
// 需要调用旋转
void splay(node *x) {
    while (!isroot(x)) {
        if (isroot(x -> p)) {
            rot(x -> p, dir(x) ^ 1);
            break;
        }

        if (dir(x) == dir(x -> p))
            rot(x -> p -> p, dir(x -> p) ^ 1);
        else
            rot(x -> p, dir(x) ^ 1);
        rot(x -> p, dir(x) ^ 1);
    }
}

// 旋转(LCT版本) O(1)
// 平衡树基本操作
// 要求对应儿子必须存在，否则会导致后续各种莫名其妙的问题
void rot(node *x, int d) {
    node *y = x -> ch[d ^ 1];

    y -> p = x -> p;
    if (!isroot(x))
        x -> p -> ch[dir(x)] = y;

    if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
        y -> ch[d] -> p = x;
    (y -> ch[d] = x) -> p = y;

    x -> refresh();
    y -> refresh();
}
```

4.7.2 换根/维护生成树

```
#define isroot(x) ((x) -> p == null || ((x) -> p -> ch[0] != (x) && (x) -> p -> ch[1] != (x)))
#define dir(x) ((x) == (x) -> p -> ch[1])

using namespace std;

const int maxn = 200005;

struct node{
    int key, mx, pos;
    bool rev;
    node *ch[2], *p;
```

```cpp

    node(int key = 0): key(key), mx(key), pos(-1), rev(false) {}

    void pushdown() {
        if (!rev)
            return;

        ch[0] -> rev ^= true;
        ch[1] -> rev ^= true;
        swap(ch[0], ch[1]);

        if (pos != -1)
            pos ^= 1;

        rev = false;
    }

    void refresh() {
        mx = key;
        pos = -1;
        if (ch[0] -> mx > mx) {
            mx = ch[0] -> mx;
            pos = 0;
        }
        if (ch[1] -> mx > mx) {
            mx = ch[1] -> mx;
            pos = 1;
        }
    }
} null[maxn * 2];

void init(node *x, int k) {
    x -> ch[0] = x -> ch[1] = x -> p = null;
    x -> key = x -> mx = k;
}

void rot(node *x, int d) {
    node *y = x -> ch[d ^ 1];
    if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
        y -> ch[d] -> p = x;

    y -> p = x -> p;
    if (!isroot(x))
        x -> p -> ch[dir(x)] = y;

    (y -> ch[d] = x) -> p = y;

    x -> refresh();
    y -> refresh();
}

void splay(node *x) {
    x -> pushdown();

    while (!isroot(x)) {
        if (!isroot(x -> p))
            x -> p -> p -> pushdown();
        x -> p -> pushdown();
        x -> pushdown();

        if (isroot(x -> p)) {
            rot(x -> p, dir(x) ^ 1);
            break;
        }

        if (dir(x) == dir(x -> p))
            rot(x -> p -> p, dir(x -> p) ^ 1);
        else
            rot(x -> p, dir(x) ^ 1);

        rot(x -> p, dir(x) ^ 1);
    }
}

node *access(node *x) {
    node *y = null;

    while (x != null) {
        splay(x);

        x -> ch[1] = y;
        (y = x) -> refresh();

        x = x -> p;
    }

    return y;
}

void makeroot(node *x) {
    access(x);
    splay(x);
    x -> rev ^= true;
}

void link(node *x, node *y) {
    makeroot(x);
    x -> p = y;
}

void cut(node *x, node *y) {
    makeroot(x);
    access(y);
    splay(y);

    y -> ch[0] -> p = null;
    y -> ch[0] = null;
    y -> refresh();
}

node *getroot(node *x) {
    x = access(x);
    while (x -> pushdown(), x -> ch[0] != null)
        x = x -> ch[0];
    splay(x);
    return x;
}

node *getmax(node *x, node *y) {
    makeroot(x);
    x = access(y);

    while (x -> pushdown(), x -> pos != -1)
        x = x -> ch[x -> pos];
    splay(x);

    return x;
}

// 以下为主函数示例
for (int i = 1; i <= m; i++) {
    init(null + n + i, w[i]);
    if (getroot(null + u[i]) != getroot(null + v[i])) {
        ans[q + 1] -= k;
        ans[q + 1] += w[i];

        link(null + u[i], null + n + i);
        link(null + v[i], null + n + i);
        vis[i] = true;
    }
    else {
        int ii = getmax(null + u[i], null + v[i]) - null - n;
        if (w[i] >= w[ii])
            continue;

        cut(null + u[ii], null + n + ii);
        cut(null + v[ii], null + n + ii);

        link(null + u[i], null + n + i);
        link(null + v[i], null + n + i);

        ans[q + 1] -= w[ii];
        ans[q + 1] += w[i];
    }
}
```

4.7.3 维护子树信息

```
// 这个东西虽然只需要抄板子但还是极其难写，常数极其巨大，没必要的时候就不要用
// 如果维护子树最小值就需要套一个可删除的堆来维护，复杂度会变成O(n\log^2 n)
// 注意由于这道题与边权有关，需要边权拆点变点权

// 宏定义
#define isroot(x) ((x) -> p == null || ((x) != (x) -> p -> ch[0]&& (x) != (x) -> p -> ch[1]))
#define dir(x) ((x) == (x) -> p -> ch[1])

// 节点类定义
struct node { // 以维护子树中黑点到根距离和为例
    int w, chain_cnt, tree_cnt;
    long long sum, suml, sumr, tree_sum; // 由于换根需要子树反转，需要维护两个方向的信息
    bool rev, col;
    node *ch[2], *p;

    node() : w(0), chain_cnt(0), tree_cnt(0),sum(0),suml(0), sumr(0),
        tree_sum(0), rev(false), col(false) {}

    inline void pushdown() {
        if(!rev)
            return;

        ch[0]->rev ^= true;
        ch[1]->rev ^= true;
        swap(ch[0], ch[1]);
        swap(suml, sumr);

        rev = false;
    }

    inline void refresh() { // 如果不想这样特判就pushdown一下
        // pushdown();

        sum = ch[0] -> sum + ch[1] -> sum + w;
        suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] -> suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1] -> suml) + (tree_cnt + ch[1] -> chain_cnt) * (ch[0] -> sum + w) + tree_sum;
        sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] -> sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1] -> sumr) + (tree_cnt + ch[0] -> chain_cnt) * (ch[1] -> sum + w) + tree_sum;
        chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt + tree_cnt;
    }
} null[maxn * 2]; // 如果没有边权变点权就不用乘2了

// 封装构造函数
node *newnode(int w) {
    node *x = nodes.front(); // 因为有删边加边，可以用一个队列维护可用结点
    nodes.pop();
    initalize(x);
    x -> w = w;
    x -> refresh();
    return x;
}

// 封装初始化函数
// 记得在进行操作之前对所有结点调用一遍
inline void initalize(node *x) {
    *x = node();
    x -> ch[0] = x -> ch[1] = x -> p = null;
}

// 注意一下在Access的同时更新子树信息的方法
node *access(node *x) {
    node *y = null;

    while (x != null) {
        splay(x);

        x -> tree_cnt += x -> ch[1] -> chain_cnt - y -> chain_cnt;
        x->tree_sum += (x -> ch[1] -> rev ? x -> ch[1] -> sumr : x -> ch[1] -> suml) - y -> suml;
        x -> ch[1] = y;

        (y = x) -> refresh();
        x = x -> p;
    }

    return y;
}

// 找到一个点所在连通块的根
// 对比原版没有变化
node *getroot(node *x) {
    x = access(x);

    while (x -> pushdown(), x -> ch[0] != null)
        x = x -> ch[0];
    splay(x);

    return x;
}

// 换根，同样没有变化
void makeroot(node *x) {
    access(x);
    splay(x);
    x -> rev ^= true;
    x -> pushdown();
}

// 连接两个点
// !!! 注意这里必须把两者都变成根，因为只能修改根结点
void link(node *x, node *y) {
    makeroot(x);
    makeroot(y);

    x -> p = y;
    y -> tree_cnt += x -> chain_cnt;
    y -> tree_sum += x -> suml;
    y -> refresh();
}

// 删除一条边
// 对比原版没有变化
void cut(node *x, node *y) {
    makeroot(x);
    access(y);
    splay(y);

    y -> ch[0] -> p = null;
    y -> ch[0] = null;
    y -> refresh();
}

// 修改/询问一个点，这里以询问为例
// 如果是修改就在换根之后搞一些操作
long long query(node *x) {
    makeroot(x);
    return x -> suml;
}

// Splay函数
// 对比原版没有变化
void splay(node *x) {
    x -> pushdown();

    while (!isroot(x)) {
        if (!isroot(x -> p))
            x -> p -> p -> pushdown();
        x -> p -> pushdown();
        x -> pushdown();

        if (isroot(x -> p)) {
            rot(x -> p,dir(x) ^ 1);
            break;
        }

        if (dir(x) == dir(x -> p))
            rot(x -> p -> p, dir(x -> p) ^ 1);
        else
            rot(x -> p, dir(x) ^ 1);

        rot(x -> p, dir(x) ^ 1);
    }
}

// 旋转函数
// 对比原版没有变化
void rot(node *x, int d) {
    node *y = x -> ch[d ^ 1];

    if ((x -> ch[d^1] = y -> ch[d]) != null)
        y -> ch[d] -> p = x;

    y -> p = x -> p;
    if (!isroot(x))
        x -> p -> ch[dir(x)] = y;

    (y -> ch[d] = x) -> p = y;

    x -> refresh();
    y -> refresh();
}
```

4.7.4 模板题:动态QTREE4(询问树上相距最远点)

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>

#define isroot(x) ((x)->p==null||((x)!=(x)->p->ch[0]&&(x)!=(x)->p->ch[1]))
#define dir(x) ((x)==(x)->p->ch[1])

using namespace std;
using namespace __gnu_pbds;

const int maxn = 100010;
const long long INF = 1000000000000000000ll;

struct binary_heap {
    __gnu_pbds::priority_queue<long long, less<long long>, binary_heap_tag>q1, q2;
    binary_heap() {}

    void push(long long x) {
        if (x > (-INF) >> 2)
            q1.push(x);
    }

    void erase(long long x) {
        if (x > (-INF) >> 2)
            q2.push(x);
    }

    long long top() {
        if (empty())
            return -INF;

        while (!q2.empty() && q1.top() == q2.top()) {
            q1.pop();
            q2.pop();
        }

        return q1.top();
    }

    long long top2() {
        if (size() < 2)
            return -INF;

        long long a = top();
        erase(a);
        long long b = top();
        push(a);
        return a + b;
    }

    int size() {
        return q1.size() - q2.size();
    }

    bool empty() {
        return q1.size() == q2.size();
    }
} heap; // 全局堆维护每条链的最大子段和

struct node {
    long long sum, maxsum, prefix, suffix;
    int key;
    binary_heap heap; // 每个点的堆存的是它的子树中到它的最远距离，如果它是黑点的话还会包括自己
    node *ch[2], *p;
    bool rev;
    node(int k = 0): sum(k), maxsum(-INF), prefix(-INF),
        suffix(-INF), key(k), rev(false) {}
    inline void pushdown() {
        if (!rev)
            return;

        ch[0]->rev ^= true;
        ch[1]->rev ^= true;
        swap(ch[0], ch[1]);
        swap(prefix, suffix);
        rev = false;
    }
    inline void refresh() {
        pushdown();
        ch[0]->pushdown();
        ch[1]->pushdown();
        sum = ch[0]->sum + ch[1]->sum + key;
        prefix = max(ch[0]->prefix,
                     ch[0]->sum + key + ch[1]->prefix);
        suffix = max(ch[1]->suffix,
                     ch[1]->sum + key + ch[0]->suffix);
        maxsum = max(max(ch[0]->maxsum, ch[1]->maxsum),
                     ch[0]->suffix + key + ch[1]->prefix);

        if (!heap.empty()) {
            prefix = max(prefix,
                         ch[0]->sum + key + heap.top());
            suffix = max(suffix,
                         ch[1]->sum + key + heap.top());
            maxsum = max(maxsum, max(ch[0]->suffix,
                                     ch[1]->prefix) + key + heap.top());

            if (heap.size() > 1) {
                maxsum = max(maxsum, heap.top2() + key);
            }
        }
    }
} null[maxn << 1], *ptr = null;

void addedge(int, int, int);
void deledge(int, int);
void modify(int, int, int);
void modify_color(int);
node *newnode(int);
node *access(node *);
void makeroot(node *);
void link(node *, node *);
void cut(node *, node *);
void splay(node *);
void rot(node *, int);

queue<node *>freenodes;
tree<pair<int, int>, node *>mp;

bool col[maxn] = {false};
char c;
int n, m, k, x, y, z;

int main() {
    null->ch[0] = null->ch[1] = null->p = null;
    scanf("%d%d%d", &n, &m, &k);

    for (int i = 1; i <= n; i++)
        newnode(0);

    heap.push(0);

    while (k--) {
        scanf("%d", &x);

        col[x] = true;
        null[x].heap.push(0);
    }

    for (int i = 1; i < n; i++) {
        scanf("%d%d%d", &x, &y, &z);

        if (x > y)
            swap(x, y);
        addedge(x, y, z);
    }

    while (m--) {
        scanf(" %c%d", &c, &x);

        if (c == 'A') {
            scanf("%d", &y);

            if (x > y)
                swap(x, y);
            deledge(x, y);
        }
        else if (c == 'B') {
            scanf("%d%d", &y, &z);

            if (x > y)
                swap(x, y);
            addedge(x, y, z);
        }
        else if (c == 'C') {
            scanf("%d%d", &y, &z);

            if (x > y)
                swap(x, y);
            modify(x, y, z);
        }
        else
            modify_color(x);

        printf("%lld\n", (heap.top() > 0 ? heap.top() : -1));
    }

    return 0;
}
void addedge(int x, int y, int z) {
    node *tmp;
    if (freenodes.empty())
        tmp = newnode(z);
    else {
        tmp = freenodes.front();
        freenodes.pop();
        *tmp = node(z);
    }

    tmp->ch[0] = tmp->ch[1] = tmp->p = null;

    heap.push(tmp->maxsum);
    link(tmp, null + x);
    link(tmp, null + y);
    mp[make_pair(x, y)] = tmp;
}

void deledge(int x, int y) {
    node *tmp = mp[make_pair(x, y)];

    cut(tmp, null + x);
    cut(tmp, null + y);

    freenodes.push(tmp);
    heap.erase(tmp->maxsum);
    mp.erase(make_pair(x, y));
}
```

```
void modify(int x, int y, int z) {
    node *tmp = mp[make_pair(x, y)];
    makeroot(tmp);
    tmp->pushdown();

    heap.erase(tmp->maxsum);
    tmp->key = z;
    tmp->refresh();
    heap.push(tmp->maxsum);
}

void modify_color(int x) {
    makeroot(null + x);
    col[x] ^= true;

    if (col[x])
        null[x].heap.push(0);
    else
        null[x].heap.erase(0);

    heap.erase(null[x].maxsum);
    null[x].refresh();
    heap.push(null[x].maxsum);
}
node *newnode(int k) {
    *(++ptr) = node(k);
    ptr->ch[0] = ptr->ch[1] = ptr->p = null;
    return ptr;
}
node *access(node *x) {
    splay(x);
    heap.erase(x->maxsum);
    x->refresh();

    if (x->ch[1] != null) {
        x->ch[1]->pushdown();
        x->heap.push(x->ch[1]->prefix);
        x->refresh();
        heap.push(x->ch[1]->maxsum);
    }

    x->ch[1] = null;
    x->refresh();
    node *y = x;
    x = x->p;

    while (x != null) {
        splay(x);
        heap.erase(x->maxsum);

        if (x->ch[1] != null) {
            x->ch[1]->pushdown();
            x->heap.push(x->ch[1]->prefix);
            heap.push(x->ch[1]->maxsum);
        }

        x->heap.erase(y->prefix);
        x->ch[1] = y;
        (y = x)->refresh();
        x = x->p;
    }

    heap.push(y->maxsum);
    return y;
}
void makeroot(node *x) {
    access(x);
    splay(x);
    x->rev ^= true;
}
void link(node *x, node *y) { // 新添一条虚边，维护y对应的堆
    makeroot(x);
    makeroot(y);

    x->pushdown();
    x->p = y;
    heap.erase(y->maxsum);
    y->heap.push(x->prefix);
    y->refresh();
    heap.push(y->maxsum);
}
void cut(node *x, node *y) { // 断开一条实边，一条链变成两条链，需要维护全局堆
    makeroot(x);
    access(y);
    splay(y);

    heap.erase(y->maxsum);
    heap.push(y->ch[0]->maxsum);
    y->ch[0]->p = null;
    y->ch[0] = null;
    y->refresh();
    heap.push(y->maxsum);
}
void splay(node *x) {
    x->pushdown();

    while (!isroot(x)) {
        if (!isroot(x->p))
            x->p->p->pushdown();

        x->p->pushdown();
        x->pushdown();

        if (isroot(x->p)) {
            rot(x->p, dir(x) ^ 1);
            break;
        }

        if (dir(x) == dir(x->p))
            rot(x->p->p, dir(x->p) ^ 1);
        else
            rot(x->p, dir(x) ^ 1);

        rot(x->p, dir(x) ^ 1);
    }
}
void rot(node *x, int d) {
    node *y = x->ch[d ^ 1];

    if ((x->ch[d ^ 1] = y->ch[d]) != null)
        y->ch[d]->p = x;

    y->p = x->p;

    if (!isroot(x))
        x->p->ch[dir(x)] = y;

    (y->ch[d] = x)->p = y;

    x->refresh();
    y->refresh();
}
```

4.8 K-D树

4.8.1 动态K-D树(定期重构)

```
int l[2], r[2], x[B + 10][2], w[B + 10];
int n, op, ans = 0, cnt = 0, tmp = 0;
int d;

```

```
struct node {
    int x[2], l[2], r[2], w, sum;
    node *ch[2];

    bool operator < (const node &a) const {
        return x[d] < a.x[d];
    }

    void refresh() {
        sum = ch[0] -> sum + ch[1] -> sum + w;
        l[0] = min(x[0], min(ch[0] -> l[0], ch[1] -> l[0]));
        l[1] = min(x[1], min(ch[0] -> l[1], ch[1] -> l[1]));
        r[0] = max(x[0], max(ch[0] -> r[0], ch[1] -> r[0]));
        r[1] = max(x[1], max(ch[0] -> r[1], ch[1] -> r[1]));
    }
} null[maxn], *root = null;

void build(int l, int r, int k, node *&rt) {
    if (l > r) {
        rt = null;
        return;
    }

    int mid = (l + r) / 2;

    d = k;
    nth_element(null + l, null + mid, null + r + 1);

    rt = null + mid;
    build(l, mid - 1, k ^ 1, rt -> ch[0]);
    build(mid + 1, r, k ^ 1, rt -> ch[1]);

    rt -> refresh();
}

void query(node *rt) {
    if (l[0] <= rt -> l[0] && l[1] <= rt -> l[1] && rt -> r[0] <= r[0] && rt -> r[1] <= r[1]) {
        ans += rt -> sum;
        return;
    }
    else if (l[0] > rt -> r[0] || l[1] > rt -> r[1] || r[0] < rt -> l[0] || r[1] < rt -> l[1])
        return;

    if (l[0] <= rt -> x[0] && l[1] <= rt -> x[1] && rt -> x[0] <= r[0] && rt -> x[1] <= r[1])
        ans += rt -> w;

    query(rt -> ch[0]);
    query(rt -> ch[1]);
}

int main() {

    null -> l[0] = null -> l[1] = 10000000;
    null -> r[0] = null -> r[1] = -10000000;
    null -> sum = 0;
    null -> ch[0] = null -> ch[1] = null;
    scanf("%*d");

    while (scanf("%d", &op) == 1 && op != 3) {
        if (op == 1) {
            tmp++;
            scanf("%d%d%d", &x[tmp][0], &x[tmp][1], &w[tmp]);
            x[tmp][0] ^= ans;
            x[tmp][1] ^= ans;
            w[tmp] ^= ans;

            if (tmp == B) {
                for (int i = 1; i <= tmp; i++) {
                    null[cnt + i].x[0] = x[i][0];
                    null[cnt + i].x[1] = x[i][1];
                    null[cnt + i].w = w[i];
                }

                build(1, cnt += tmp, 0, root);
                tmp = 0;
            }
        }
        else {
            scanf("%d%d%d%d", &l[0], &l[1], &r[0], &r[1]);
            l[0] ^= ans;
            l[1] ^= ans;
            r[0] ^= ans;
            r[1] ^= ans;
            ans = 0;

            for (int i = 1; i <= tmp; i++)
                if (l[0] <= x[i][0] && l[1] <= x[i][1] && x[i][0] <= r[0] && x[i][1] <= r[1])
                    ans += w[i];

            query(root);
            printf("%d\n", ans);
        }
    }

    return 0;
}
```

4.9 LCA最近公共祖先

4.9.1 Tarjan LCA $O(n + m)$

```
vector<pair<int, int> > q[maxn];
int lca[maxn];

void dfs(int x) {
    dfn[x] = ++tim; // 其实求LCA是用不到DFS序的，但是一般其他步骤要用
    ufs[x] = x;

    for (auto pi : q[x]) {
        int y = pi.first, i = pi.second;
        if (dfn[y])
            lca[i] = findufs(y);
    }

    for (int y : G[x])
        if (y != p[x]) {
            p[y] = x;
            dfs(y);
        }

    ufs[x] = p[x];
}
```

4.10 虚树

```
struct Tree {
    vector<int>G[maxn], W[maxn];
```

```cpp
    int p[maxn], d[maxn], size[maxn], mn[maxn], mx[maxn];
    bool col[maxn];
    long long ans_sum;
    int ans_min, ans_max;

    void add(int x, int y, int z) {
        G[x].push_back(y);
        W[x].push_back(z);
    }

    void dfs(int x) {
        size[x] = col[x];
        mx[x] = (col[x] ? d[x] : -0x3f3f3f3f);
        mn[x] = (col[x] ? d[x] : 0x3f3f3f3f);

        for (int i = 0; i < (int)G[x].size(); i++) {
            d[G[x][i]] = d[x] + W[x][i];
            dfs(G[x][i]);
            ans_sum += (long long)size[x] * size[G[x][i]] * d[x];
            ans_max = max(ans_max, mx[x] + mx[G[x][i]] - (d[x] << 1));
            ans_min = min(ans_min, mn[x] + mn[G[x][i]] - (d[x] << 1));
            size[x] += size[G[x][i]];
            mx[x] = max(mx[x], mx[G[x][i]]);
            mn[x] = min(mn[x], mn[G[x][i]]);
        }
    }

    void clear(int x) {
        G[x].clear();
        W[x].clear();
        col[x] = false;
    }

    void solve(int rt) {
        ans_sum = 0;
        ans_max = 1 << 31;
        ans_min = (~0u) >> 1;
        dfs(rt);
        ans_sum <<= 1;
    }
} virtree;

void dfs(int);
int LCA(int, int);

vector<int>G[maxn];
int f[maxn][20], d[maxn], dfn[maxn], tim = 0;

bool cmp(int x, int y) {
    return dfn[x] < dfn[y];
}

int n, m, lgn = 0, a[maxn], s[maxn], v[maxn];

int main() {
    scanf("%d", &n);

    for (int i = 1, x, y; i < n; i++) {
        scanf("%d%d", &x, &y);
        G[x].push_back(y);
        G[y].push_back(x);
    }

    G[n + 1].push_back(1);
    dfs(n + 1);

    for (int i = 1; i <= n + 1; i++)
        G[i].clear();

    lgn--;

    for (int j = 1; j <= lgn; j++)
        for (int i = 1; i <= n; i++)
            f[i][j] = f[f[i][j - 1]][j - 1];

    scanf("%d", &m);

    while (m--) {
        int k;
        scanf("%d", &k);

        for (int i = 1; i <= k; i++)
            scanf("%d", &a[i]);

        sort(a + 1, a + k + 1, cmp);
        int top = 0, cnt = 0;
        s[++top] = v[++cnt] = n + 1;
        long long ans = 0;

        for (int i = 1; i <= k; i++) {
            virtree.col[a[i]] = true;
            ans += d[a[i]] - 1;
            int u = LCA(a[i], s[top]);

            if (s[top] != u) {
                while (top > 1 && d[s[top - 1]] >= d[u]) {
                    virtree.add(s[top - 1], s[top], d[s[top]] - d[s[top - 1]]);
                    top--;
                }

                if (s[top] != u) {
                    virtree.add(u, s[top], d[s[top]] - d[u]);
                    s[top] = v[++cnt] = u;
                }
            }

            s[++top] = a[i];
        }

        for (int i = top - 1; i; i--)
            virtree.add(s[i], s[i + 1], d[s[i + 1]] - d[s[i]]);

        virtree.solve(n + 1);
        ans *= k - 1;
        printf("%lld %d %d\n", ans - virtree.ans_sum, virtree.ans_min, virtree.ans_max);

        for (int i = 1; i <= k; i++)
            virtree.clear(a[i]);
        for (int i = 1; i <= cnt; i++)
            virtree.clear(v[i]);

    }

    return 0;
}

void dfs(int x) {
    dfn[x] = ++tim;
    d[x] = d[f[x][0]] + 1;

    while ((1 << lgn) < d[x])
        lgn++;

    for (int i = 0; i < (int)G[x].size(); i++)
        if (G[x][i] != f[x][0]) {
            f[G[x][i]][0] = x;
            dfs(G[x][i]);
        }
}

int LCA(int x, int y) {
    if (d[x] != d[y]) {
        if (d[x] < d[y])
            swap(x, y);

        for (int i = lgn; i >= 0; i--)
            if (((d[x] - d[y]) >> i) & 1)
                x = f[x][i];
    }

    if (x == y)
        return x;

    for (int i = lgn; i >= 0; i--)
        if (f[x][i] != f[y][i]) {
            x = f[x][i];
            y = f[y][i];
        }

    return f[x][0];
}
```

4.11 长链剖分

```
// 顾名思义，长链剖分是取最深的儿子作为重儿子

// O(n)维护以深度为下标的子树信息
vector<int> G[maxn], v[maxn];
int n, p[maxn], h[maxn], son[maxn], ans[maxn];

// 原题题意：求每个点的子树中与它距离是几的点最多，相同的取最大深度
// 由于vector只能在后面加入元素，为了写代码方便，这里反过来存
// 或者开一个结构体维护倒过来的vector
void dfs(int x) {
    h[x] = 1;

    for (int y : G[x])
        if (y != p[x]){
            p[y] = x;
            dfs(y);

            if (h[y] > h[son[x]])
                son[x] = y;
        }

    if (!son[x]) {
        v[x].push_back(1);
        ans[x] = 0;
        return;
    }

    h[x] = h[son[x]] + 1;
    swap(v[x],v[son[x]]);

    if (v[x][ans[son[x]]] == 1)
        ans[x] = h[x] - 1;
    else
        ans[x] = ans[son[x]];

    v[x].push_back(1);

    int mx = v[x][ans[x]];
    for (int y : G[x])
        if (y != p[x] && y != son[x]) {
            for (int j = 1; j <= h[y]; j++) {
                v[x][h[x] - j - 1] += v[y][h[y] - j];

                int t = v[x][h[x] - j - 1];
                if (t > mx || (t == mx && h[x] - j - 1 > ans[x])) {
                    mx = t;
                    ans[x] = h[x] - j - 1;
                }
            }

            v[y].clear();
        }
}
```

4.11.1 梯子剖分

```
// 在线求一个点的第k祖先 O(n\log n)-O(1)
// 理论基础：任意一个点x的k级祖先y所在长链长度一定>=k

// 全局数组定义
vector<int> G[maxn], v[maxn];
int d[maxn], mxd[maxn], son[maxn], top[maxn], len[maxn];
int f[19][maxn], log_tbl[maxn];

// 在主函数中两遍dfs之后加上如下预处理
log_tbl[0] = -1;
for (int i = 1; i <= n; i++)
    log_tbl[i] = log_tbl[i / 2] + 1;
for (int j = 1; (1 << j) < n; j++)
    for (int i = 1; i <= n; i++)
        f[j][i] = f[j - 1][f[j - 1][i]];

// 第一遍dfs，用于计算深度和找出重儿子
void dfs1(int x) {
    mxd[x] = d[x];

    for (int y : G[x])
        if (y != f[0][x]){
            f[0][y] = x;
            d[y] = d[x] + 1;

            dfs1(y);

            mxd[x] = max(mxd[x], mxd[y]);
            if (mxd[y] > mxd[son[x]])
                son[x] = y;
        }
}

// 第二遍dfs，用于进行剖分和预处理梯子剖分(每条链向上延伸一倍)数组
void dfs2(int x) {
    top[x] = (x == son[f[0][x]] ? top[f[0][x]] : x);

    for (int y : G[x])
        if (y != f[0][x])
            dfs2(y);

    if (top[x] == x) {
        int u = x;
        while (top[son[u]] == x)
            u = son[u];

        len[x] = d[u] - d[x];
        for (int i = 0; i < len[x]; i++, u = f[0][u])
            v[x].push_back(u);

        u = x;
        for (int i = 0; i < len[x] && u; i++, u = f[0][u])
```

```
            v[x].push_back(u);
    }
}

// 在线询问x的k级祖先 O(1)
// 不存在时返回0
int query(int x, int k) {
    if (!k)
        return x;
    if (k > d[x])
        return 0;

    x = f[log_tbl[k]][x];
    k ^= 1 << log_tbl[k];
    return v[top[x]][d[top[x]] + len[top[x]] - d[x] + k];
}
```

4.12 左偏树

(参见k短路板子.)

4.13 莫队

注意如果n和q不平衡, 块大小应该设为$\frac{n}{\sqrt{q}}$.
另外如果裸的莫队要卡常可以按块编号奇偶性分别对右端点正序或者倒序排序, 期望可以减少一半的移动次数.

4.13.1 回滚莫队(无删除莫队)(待完成)

4.13.2 莫队二次离线

适用范围: 询问的是点对相关(或者其它可以枚举每个点和区间算贡献)的信息, 并且可以离线; 更新时可以使用一些牺牲修改复杂度来改善询问复杂度的数据结构(如单点修改询问区间和).
先按照普通的莫队将区间排序. 考虑区间移动的情况, 以(l,r)向右移动右端点到(l,t)为例.
对于每个$i \in (r,t]$来说, 它都要对区间$[l,i)$算贡献. 可以拆成$[1,i)$和$[1,l)$两部分, 那么前一部分因为都是i和$[1,i)$做贡献的形式所以可以直接预处理.
考虑后一部分, i和$(1,l]$做贡献, 因为莫队的性质我们可以保证这样的询问次数不超过$O((n+m)\sqrt{n})$, 因此我们可以对每个l记录下来哪些i要和它询问. 并且每次移动时询问的i都是连续的, 所以对每个l开一个vector记录下对应的区间和编号就行了.
剩余的三种情况(右端点左移或者移动左端点)都是类似的, 具体可以看代码.
例: Yuno loves sqrt technology II (询问区间逆序对数)

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

using namespace std;

constexpr int maxn = 100005, B = 314;

struct Q {
    int l, r, d, id;

    Q() = default;

    Q(int l, int r, int d, int id) : l(l), r(r), d(d),
      ↪ id(id) {}

    friend bool operator < (const Q &a, const Q &b) {
        if (a.d != b.d)
            return a.d < b.d;

        return a.r < b.r;
    }
} q[maxn]; // 结构体可以复用, d既可以作为左端点块编号, 也
  ↪ 可以作为二次离线处理的倍数

int global_n, bid[maxn], L[maxn], R[maxn], cntb;

int sa[maxn], sb[maxn];

void addp(int x) { // sqrt(n)修改 O(1)查询
    for (int k = bid[x]; k <= cntb; k++)
        sb[k]++;

    for (int i = x; i <= R[bid[x]]; i++)
        sa[i]++;
}

int queryp(int x) {
    if (!x)
        return 0;

    return sa[x] + sb[bid[x] - 1];
}

void adds(int x) {
    for (int k = 1; k <= bid[x]; k++)
        sb[k]++;

    for (int i = L[bid[x]]; i <= x; i++)
        sa[i]++;
}

int querys(int x) {
    if (x > global_n)
        return 0; // 为了防止越界就判一下
    return sa[x] + sb[bid[x] + 1];
}

vector<Q> vp[maxn], vs[maxn]; // prefix, suffix

long long fp[maxn], fs[maxn]; // prefix, suffix

int a[maxn], b[maxn];

long long ta[maxn], ans[maxn];

int main() {

    int n, m;
    scanf("%d%d", &n, &m);

    global_n = n;

    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);

    memcpy(b, a, sizeof(int) * (n + 1));
    sort(b + 1, b + n + 1);

    for (int i = 1; i <= n; i++)
        a[i] = lower_bound(b + 1, b + n + 1, a[i]) - b;

    for (int i = 1; i <= n; i++) {
        bid[i] = (i - 1) / B + 1;

        if (!L[bid[i]])
            L[bid[i]] = i;

        R[bid[i]] = i;
        cntb = bid[i];
    }

    for (int i = 1; i <= m; i++) {
```

```cpp
        scanf("%d%d", &q[i].l, &q[i].r);

        q[i].d = bid[q[i].l];
        q[i].id = i;
    }

    sort(q + 1, q + m + 1);

    int l = 2, r = 1; // l, r是上一个询问的端点

    for (int i = 1; i <= m; i++) {
        int s = q[i].l, t = q[i].r; // s, t是当前要调整到的端点

        if (s < l)
            vs[r + 1].push_back(Q(s, l - 1, 1, i));
        else if (s > l)
            vs[r + 1].push_back(Q(l, s - 1, -1, i));

        l = s;

        if (t > r)
            vp[l - 1].push_back(Q(r + 1, t, 1, i));
        else if (t < r)
            vp[l - 1].push_back(Q(t + 1, r, -1, i));

        r = t;
    }

    for (int i = 1; i <= n; i++) { // 第一遍正着处理，解决关于前缀的询问
        fp[i] = fp[i - 1] + querys(a[i] + 1);

        adds(a[i]);

        for (auto q : vp[i]) {
            long long tmp = 0;
            for (int k = q.l; k <= q.r; k++)
                tmp += querys(a[k] + 1);

            ta[q.id] -= q.d * tmp;
        }
    }

    memset(sa, 0, sizeof(sa));
    memset(sb, 0, sizeof(sb));

    for (int i = n; i; i--) { // 第二遍倒着处理，解决关于后缀的询问
        fs[i] = fs[i + 1] + queryp(a[i] - 1);

        addp(a[i]);

        for (auto q : vs[i]) {
            long long tmp = 0;
            for (int k = q.l; k <= q.r; k++)
                tmp += queryp(a[k] - 1);

            ta[q.id] -= q.d * tmp;
        }
    }

    l = 2;
    r = 1;

    for (int i = 1; i <= m; i++) { // 求出fs和fp之后再加上这部分的贡献
        int s = q[i].l, t = q[i].r;

        ta[i] += fs[s] - fs[l];
        ta[i] += fp[t] - fp[r];

        l = s;
        r = t;

        ta[i] += ta[i - 1]; // 因为算出来的是相邻两个询问之间的贡献，所以要前缀和
        ans[q[i].id] = ta[i];
    }

    for (int i = 1; i <= m; i++)
        printf("%lld\n", ans[i]);

    return 0;
}
```

4.13.3 带修莫队在线化 $O(n^{\frac{5}{3}})$

最简单的带修莫队：块大小设成$n^{\frac{2}{3}}$，排序时第一关键字是左端点块编号，第二关键字是右端点块编号，第三关键字是时间. (记得把时间压缩成只有修改的时间.)

现在要求在线的同时支持修改，仍然以$B = n^{\frac{2}{3}}$分一块，预处理出两块之间的贡献，那么预处理复杂度就是$O(n^{\frac{5}{3}})$.

修改时最简单的方法是直接把$n^{\frac{2}{3}}$个区间全更新一遍. 嫌慢的话可以给每个区间打一个懒标记，询问的时候如果解了再更新区间的信息.

注意如果附加信息是可减的(比如每个数的出现次数)，那么就只需要存$O(n^{\frac{1}{3}})$个.

总复杂度仍然是$O(n^{\frac{5}{3}})$，如果打懒标记的话是跑不太满的. 如果附加信息可减则空间复杂度是$O(n^{\frac{4}{3}})$，否则和时间复杂度同阶.

4.13.4 莫队二次离线 在线化 $O((n + m)\sqrt{n})$

和之前的道理是一样的，i和$[1, i)$的贡献这部分仍然可以预处理掉，而前后缀对区间的贡献那部分只保存块端点处的信息.

按照莫队二次离线的转移方法操作之后发现只剩两边散块的贡献没有解决. 这时可以具体问题具体解决，例如求逆序对的话直接预处理出排序后的数组然后归并即可.

时空复杂度均为$O(n\sqrt{n})$.

以下代码以强制在线求区间逆序对为例(洛谷上被卡常了，正常情况下极限数据应该在1.5s内.)

```cpp
constexpr int maxn = 100005, B = 315, maxb = maxn / B + 5;

int n, bid[maxn], L[maxb], R[maxb], cntb;

struct DS { // O(sqrt(n))修改  O(1)查询
    int total;
    int sa[maxn], sb[maxb];

    void init(const DS &o) {
        total = o.total;
        memcpy(sa, o.sa, sizeof(int) * (n + 1));
        memcpy(sb, o.sb, sizeof(int) * (cntb + 1));
    }

    void add(int x) {
        total++;
        for (int k = 1; k <= bid[x]; k++)
            sb[k]++;
        for (int i = L[bid[x]]; i <= x; i++)
            sa[i]++;
    }

    int querys(int x) {
        if (x > n)
            return 0;

        return sb[bid[x] + 1] + sa[x];
    }

    int queryp(int x) {
        return total - querys(x + 1);
    }
} pr[maxb];

int c[maxn]; // 树状数组

void addc(int x, int d) {
    while (x) {
        c[x] += d;
        x -= x & -x;
    }
}

int queryc(int x) {
    int ans = 0;
    while (x <= n) {
        ans += c[x];
        x += x & -x;
    }
    return ans;
}

long long fp[maxn], fs[maxn];

int rnk[maxn], val[maxn][B + 5];

long long dat[maxb][maxb];

int a[maxn];

int main() {

    int m;
    cin >> n >> m;

    for (int i = 1; i <= n; i++) {
        cin >> a[i];

        bid[i] = (i - 1) / B + 1;
        if (!L[bid[i]])
            L[bid[i]] = i;
        R[bid[i]] = i;
        cntb = bid[i];

        rnk[i] = i;
    }

    for (int k = 1; k <= cntb; k++)
        sort(rnk + L[k], rnk + R[k] + 1, [](int x, int y){return a[x] < a[y];}); // 每个块排序

    for (int i = n; i; i--)
        for (int j = 2; i + j - 1 <= R[bid[i]]; j++) {
            val[i][j] = val[i + 1][j - 1] + val[i][j - 1] - val[i + 1][j - 2];
            if (a[i] > a[i + j - 1])
                val[i][j]++; // 块内用二维前缀和预处理
        }

    for (int k = 1; k <= cntb; k++) {
        for (int i = L[k]; i <= R[k]; i++) {
            dat[k][k] += queryc(a[i] + 1); // 单块内的逆序对直接用树状数组预处理
            addc(a[i], 1);
        }

        for (int i = L[k]; i <= R[k]; i++)
            addc(a[i], -1);
    }

    for (int i = 1; i <= n; i++) {
        if (i > 1 && i == L[bid[i]])
            pr[bid[i]].init(pr[bid[i] - 1]);

        fp[i] = fp[i - 1] + pr[bid[i]].querys(a[i] + 1);

        pr[bid[i]].add(a[i]);
    }

    for (int i = n; i; i--) {
        fs[i] = fs[i + 1] + (n - i - queryc(a[i] + 1));
        addc(a[i], 1);
    }

    for (int s = 1; s <= cntb; s++)
        for (int t = s + 1; t <= cntb; t++) {
            dat[s][t] = dat[s][t - 1] + dat[t][t];

            for (int i = L[t]; i <= R[t]; i++) // 块间的逆序对用刚才处理的分块求出
                dat[s][t] += pr[t - 1].querys(a[i] + 1) - pr[s - 1].querys(a[i] + 1);
        }

    long long ans = 0;

    while (m--) {
        long long s, t;
        cin >> s >> t;

        int l = s ^ ans, r = t ^ ans;

        if (bid[l] == bid[r])
            ans = val[l][r - l + 1];
        else {
            ans = dat[bid[l] + 1][bid[r] - 1];

            ans += fp[r] - fp[L[bid[r]] - 1];
            for (int i = L[bid[r]]; i <= r; i++)
                ans -= pr[bid[l]].querys(a[i] + 1);

            ans += fs[l] - fs[R[bid[l]] + 1];
            for (int i = l; i <= R[bid[l]]; i++)
                ans -= (a[i] - 1) - pr[bid[r] - 1].queryp(a[i] - 1);

            int i = L[bid[l]], j = L[bid[r]], w = 0; // 手写归并

            while (true) {
                while (i <= R[bid[l]] && rnk[i] < l)
                    i++;
                while (j <= R[bid[r]] && rnk[j] > r)
                    j++;

                if (i > R[bid[l]] && j > R[bid[r]])
                    break;

                int x = (i <= R[bid[l]] ? a[rnk[i]] : (int)1e9), y = (j <= R[bid[r]] ? a[rnk[j]] : (int)1e9);

                if (x < y) {
                    ans += w;
                    i++;
                }
                else {
                    j++;
                    w++;
                }
            }
        }

        cout << ans << '\n';
    }

    return 0;
}
```

4.14 常见根号思路

1. **通用**

- 出现次数大于$\sqrt{n}$的数不会超过$\sqrt{n}$个
- 对于带修改问题，如果不方便分治或者二进制分组，可以考虑对操作分块，每次查询时暴力最后的$\sqrt{n}$个修改并更正答案
- **根号分治**：如果分治时每个子问题需要$O(N)$(N是全局问题的大小)的时间，而规模较小的子问题可以$O(n^2)$解决，则可以使用根号分治
  - 规模大于$\sqrt{n}$的子问题用$O(N)$的方法解决，规模小于$\sqrt{n}$的子问题用$O(n^2)$暴力
  - 规模大于$\sqrt{n}$的子问题最多只有$\sqrt{n}$个
  - 规模不大于$\sqrt{n}$的子问题大小的平方和也必定不会超过$n\sqrt{n}$
- 如果输入规模之和不大于n(例如给定多个小字符串与大字符串进行询问)，那么规模超过$\sqrt{n}$的问题最多只有$\sqrt{n}$个

2. **序列**

- 某些维护序列的问题可以用分块/块状链表维护
- 对于静态区间询问问题，如果可以快速将左/右端点移动一位，可以考虑莫队
  - 如果强制在线可以分块预处理，但是一般空间需要$n\sqrt{n}$
    - 例题：询问区间中有几种数出现次数恰好为k，强制在线
  - 如果带修改可以试着想一想带修莫队，但是复杂度高达$n^{\frac{5}{3}}$
- 线段树可以解决的问题也可以用分块来做到$O(1)$询问或是$O(1)$修改，具体要看哪种操作更多

3. **树**

- 与序列类似，树上也有树分块和树上莫队
  - 树上带修莫队很麻烦，常数也大，最好不要先考虑
  - 树分块不要想当然
- 树分治也可以套根号分治，道理是一样的

4. **字符串**

- 循环节长度大于$\sqrt{n}$的子串最多只有$O(n)$个，如果是极长子串则只有$O(\sqrt{n})$个

5. 字符串

5.1 KMP

```cpp
char s[maxn], t[maxn];
int fail[maxn];
int n, m;

void init() { // 注意字符串是0-based，但是fail是1-based
    // memset(fail, 0, sizeof(fail));

    for (int i = 1; i < m; i++) {
        int j = fail[i];
        while (j && t[i] != t[j])
            j = fail[j];

        if (t[i] == t[j])
            fail[i + 1] = j + 1;
        else
            fail[i + 1] = 0;
    }
}

int KMP() {
    int cnt = 0, j = 0;

    for (int i = 0; i < n; i++) {
        while (j && s[i] != t[j])
            j = fail[j];

        if (s[i] == t[j])
            j++;
        if (j == m)
            cnt++;
    }

    return cnt;
}
```

5.1.1 ex-KMP

```cpp
//全局变量与数组定义
char s[maxn], t[maxn];
int n, m, a[maxn];

// 主过程 O(n + m)
// 把t的每个后缀与s的LCP输出到a中，s的后缀和自己的LCP存在nx中
// 0-based，s的长度是m，t的长度是n
void exKMP(const char *s, const char *t, int *a) {
    static int nx[maxn];

    memset(nx, 0, sizeof(nx));

    int j = 0;
    while (j + 1 < m && s[j] == s[j + 1])
        j++;
    nx[1] = j;

    for (int i = 2, k = 1;i < m; i++) {
        int pos = k + nx[k], len = nx[i - k];

        if (i + len < pos)
            nx[i] = len;
        else {
            j = max(pos - i, 0);
            while (i + j < m && s[j] == s[i + j])
                j++;

            nx[i] = j;
            k = i;
        }
    }

    j = 0;
    while (j < n && j < m && s[j] == t[j])
        j++;
    a[0] = j;

    for (int i = 1, k = 0; i < n; i++) {
        int pos = k + a[k], len = nx[i - k];
        if (i + len < pos)
            a[i] = len;
        else {
            j = max(pos - i, 0);
            while(j < m && i + j < n && s[j] == t[i + j])
                j++;

            a[i] = j;
            k = i;
        }
    }
}
```

5.2 AC自动机

```cpp
int ch[maxm][26], f[maxm][26], q[maxm], sum[maxm], cnt = 0;

// 在字典树中插入一个字符串 O(n)
int insert(const char *c) {
    int x = 0;
    while (*c) {
        if (!ch[x][*c - 'a'])
            ch[x][*c - 'a'] = ++cnt;
        x = ch[x][*c++ - 'a'];
    }
    return x;
}

// 建AC自动机 O(n * sigma)
void getfail() {
    int x, head = 0, tail = 0;

    for (int c = 0; c < 26; c++)
        if (ch[0][c])
            q[tail++] = ch[0][c]; // 把根节点的儿子加入队列

    while (head != tail) {
        x = q[head++];

        G[f[x][0]].push_back(x);
        fill(f[x] + 1, f[x] + 26, cnt + 1);

        for (int c = 0; c < 26; c++) {
            if (ch[x][c]) {
                int y = f[x][0];

                f[ch[x][c]][0] = ch[y][c];
                q[tail++] = ch[x][c];
            }
            else
                ch[x][c] = ch[f[x][0]][c];
        }
    }
    fill(f[0], f[0] + 26, cnt + 1);
}
```

5.3 后缀数组

5.3.1 倍增

```cpp
constexpr int maxn = 100005;

void get_sa(char *s, int n, int *sa, int *rnk, int *height) { // 1-base
    static int buc[maxn], id[maxn], p[maxn], t[maxn * 2];

    int m = 300;

    for (int i = 1; i <= n; i++)
        buc[rnk[i] = s[i]]++;
    for (int i = 1; i <= m; i++)
        buc[i] += buc[i - 1];
    for (int i = n; i; i--)
        sa[buc[rnk[i]]--] = i;

    memset(buc, 0, sizeof(int) * (m + 1));

    for (int k = 1, cnt = 0; cnt != n; k *= 2, m = cnt) {
        cnt = 0;
        for (int i = n; i > n - k; i--)
            id[++cnt] = i;

        for (int i = 1; i <= n; i++)
            if (sa[i] > k)
                id[++cnt] = sa[i] - k;

        for (int i = 1; i <= n; i++)
            buc[p[i] = rnk[id[i]]]++;
        for (int i = 1; i <= m; i++)
            buc[i] += buc[i - 1];
        for (int i = n; i; i--)
            sa[buc[p[i]]--] = id[i];

        memset(buc, 0, sizeof(int) * (m + 1));

        memcpy(t, rnk, sizeof(int) * (max(n, m) + 1));

        cnt = 0;
        for (int i = 1; i <= n; i++) {
            if (t[sa[i]] != t[sa[i - 1]] || t[sa[i] + k] != t[sa[i - 1] + k])
                cnt++;

            rnk[sa[i]] = cnt;
        }
    }

    for (int i = 1; i <= n; i++)
        sa[rnk[i]] = i;

    for (int i = 1, k = 0; i <= n; i++) { // 顺便求height
        if (k)
            k--;

        while (s[i + k] == s[sa[rnk[i] - 1] + k])
            k++;

        height[rnk[i]] = k; // height[i] = lcp(sa[i], sa[i - 1])
    }
}

char s[maxn];
int sa[maxn], rnk[maxn], height[maxn];

int main() {
    cin >> (s + 1);

    int n = strlen(s + 1);

    get_sa(s, n, sa, rnk, height);

    for (int i = 1; i <= n; i++)
        cout << sa[i] << (i < n ? ' ' : '\n');

    for (int i = 2; i <= n; i++)
        cout << height[i] << (i < n ? ' ' : '\n');

    return 0;
}
```

5.3.2 SA-IS

```cpp
// SA-IS求完的SA有效位只有1~n，但它是0-based，如果其他部分是1-based就抄一下封装

constexpr int maxn = 100005, l_type = 0, s_type = 1;

// 判断一个字符是否为LMS字符
bool is_lms(int *tp, int x) {
    return x > 0 && tp[x] == s_type && tp[x - 1] == l_type;
}

// 判断两个LMS子串是否相同
bool equal_substr(int *s, int x, int y, int *tp) {
    do {
        if (s[x] != s[y])
            return false;
        x++;
        y++;
    } while (!is_lms(tp, x) && !is_lms(tp, y));

    return s[x] == s[y];
}

// 诱导排序(从*型诱导到L型,从L型诱导到S型)
// 调用之前应将*型按要求放入SA中
void induced_sort(int *s, int *sa, int *tp, int *buc, int *lbuc, int *sbuc, int n, int m) {
    for (int i = 0; i <= n; i++)
        if (sa[i] > 0 && tp[sa[i] - 1] == l_type)
            sa[lbuc[s[sa[i] - 1]]++] = sa[i] - 1;

    for (int i = 1; i <= m; i++)
        sbuc[i] = buc[i] - 1;

    for (int i = n; ~i; i--)
        if (sa[i] > 0 && tp[sa[i] - 1] == s_type)
            sa[sbuc[s[sa[i] - 1]]--] = sa[i] - 1;
}

// s是输入字符串，n是字符串的长度，m是字符集的大小
int *sais(int *s, int len, int m) {
    int n = len - 1;

    int *tp = new int[n + 1];
    int *pos = new int[n + 1];
    int *name = new int[n + 1];
    int *sa = new int[n + 1];
    int *buc = new int[m + 1];
    int *lbuc = new int[m + 1];
    int *sbuc = new int[m + 1];

    memset(buc, 0, sizeof(int) * (m + 1));
    memset(lbuc, 0, sizeof(int) * (m + 1));
    memset(sbuc, 0, sizeof(int) * (m + 1));

    for (int i = 0; i <= n; i++)
        buc[s[i]]++;

    for (int i = 1; i <= m; i++) {
        buc[i] += buc[i - 1];

        lbuc[i] = buc[i - 1];
        sbuc[i] = buc[i] - 1;
    }

    tp[n] = s_type;
    for (int i = n - 1; ~i; i--) {
        if (s[i] < s[i + 1])
            tp[i] = s_type;
        else if (s[i] > s[i + 1])
            tp[i] = l_type;
        else
            tp[i] = tp[i + 1];
    }

    int cnt = 0;
    for (int i = 1; i <= n; i++)
        if (tp[i] == s_type && tp[i - 1] == l_type)
            pos[cnt++] = i;

    memset(sa, -1, sizeof(int) * (n + 1));
    for (int i = 0; i < cnt; i++)
        sa[sbuc[s[pos[i]]]--] = pos[i];
    induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);

    memset(name, -1, sizeof(int) * (n + 1));
    int lastx = -1, namecnt = 1;
    bool flag = false;

    for (int i = 1; i <= n; i++) {
        int x = sa[i];

        if (is_lms(tp, x)) {
            if (lastx >= 0 && !equal_substr(s, x, lastx, tp))
                namecnt++;

            if (lastx >= 0 && namecnt == name[lastx])
                flag = true;

            name[x] = namecnt;
            lastx = x;
        }
    }
    name[n] = 0;

    int *t = new int[cnt];
    int p = 0;
    for (int i = 0; i <= n; i++)
        if (name[i] >= 0)
            t[p++] = name[i];

    int *tsa;
    if (!flag) {
        tsa = new int[cnt];

        for (int i = 0; i < cnt; i++)
            tsa[t[i]] = i;
    }
    else
        tsa = sais(t, cnt, namecnt);

    lbuc[0] = sbuc[0] = 0;
    for (int i = 1; i <= m; i++) {
        lbuc[i] = buc[i - 1];
        sbuc[i] = buc[i] - 1;
    }

    memset(sa, -1, sizeof(int) * (n + 1));
    for (int i = cnt - 1; ~i; i--)
        sa[sbuc[s[pos[tsa[i]]]]--] = pos[tsa[i]];
    induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);

    // 多组数据的时候最好delete掉
    delete[] tp;
    delete[] pos;
    delete[] name;
    delete[] buc;
    delete[] lbuc;
    delete[] sbuc;
    delete[] t;
    delete[] tsa;

    return sa;
}

// 封装好的函数, 1-based
void get_sa(char *s, int n, int *sa, int *rnk, int *height) {
    static int a[maxn];

    for (int i = 1; i <= n; i++)
        a[i - 1] = s[i];

    a[n] = '$';

    int *t = sais(a, n + 1, 256);
    memcpy(sa, t, sizeof(int) * (n + 1));
    delete[] t;

    sa[0] = 0;
    for (int i = 1; i <= n; i++)
        rnk[++sa[i]] = i;

    for (int i = 1, k = 0; i <= n; i++) { // 求height
        if (k)
            k--;

        while (s[i + k] == s[sa[rnk[i] - 1] + k])
            k++;

        height[rnk[i]] = k; // height[i] = lcp(sa[i], sa[i - 1])
    }
}
```

5.3.3 SAMSA

```cpp
bool vis[maxn * 2];
char s[maxn];
int n, id[maxn * 2], ch[maxn * 2][26], height[maxn], tim = 0;

void dfs(int x) {
    if (id[x]) {
        height[tim++] = val[last];
        sa[tim] = id[x];

        last = x;
    }

    for (int c = 0; c < 26; c++)
        if (ch[x][c])
            dfs(ch[x][c]);

    last = par[x];
}

int main() {
    last = ++cnt;

    scanf("%s", s + 1);
    n = strlen(s + 1);

    for (int i = n; i; i--) {
        expand(s[i] - 'a');
        id[last] = i;
    }

    vis[1] = true;
    for (int i = 1; i <= cnt; i++)
        if (id[i])
            for (int x = i, pos = n; x && !vis[x]; x = par[x]) {
                vis[x] = true;
                pos -= val[x] - val[par[x]];
                ch[par[x]][s[pos + 1] - 'a'] = x;
            }

    dfs(1);

    for (int i = 1; i <= n; i++) {
        if (i > 1)
            printf(" ");
        printf("%d", sa[i]); // 1-based
    }
    printf("\n");

    for (int i = 1; i < n; i++) {
        if (i > 1)
            printf(" ");
        printf("%d", height[i]);
    }
    printf("\n");

    return 0;
}
```

5.4 后缀平衡树

如果不需要查询排名，只需要维护前驱后继关系的题目，可以直接用二分哈希+set去做.

一般的题目需要查询排名，这时候就需要写替罪羊树或者Treap维护tag. 插入后缀时如果首字母相同只需比较各自删除首字母后的tag大小即可.

(Treap也具有重量平衡树的性质，每次插入后影响到的子树大小期望是$O(\log n)$的，所以每次做完插入操作之后直接暴力重构子树内tag就行了.)

5.5 后缀自动机

```
// 在字符集比较小的时候可以直接开go数组，否则需要用map或者哈希表替换
// 注意!!!结点数要开成串长的两倍

// 全局变量与数组定义
int last, val[maxn], par[maxn], go[maxn][26], sam_cnt;
int c[maxn], q[maxn]; // 用来桶排序

// 在主函数开头加上这句初始化
last = sam_cnt = 1;

// 以下是按val进行桶排序的代码
for (int i = 1; i <= sam_cnt; i++)
    c[val[i] + 1]++;
for (int i = 1; i <= n; i++)
    c[i] += c[i - 1]; // 这里n是串长
for (int i = 1; i <= sam_cnt; i++)
    q[++c[val[i]]] = i;

//加入一个字符 均摊O(1)
void extend(int c) {
    int p = last, np = ++sam_cnt;
    val[np] = val[p] + 1;

    while (p && !go[p][c]) {
        go[p][c] = np;
        p = par[p];
    }

    if (!p)
        par[np] = 1;
    else {
        int q = go[p][c];

        if (val[q] == val[p] + 1)
            par[np] = q;
        else {
            int nq = ++sam_cnt;
            val[nq] = val[p] + 1;
            memcpy(go[nq], go[q], sizeof(go[q]));

            par[nq] = par[q];
            par[np] = par[q] = nq;

            while (p && go[p][c] == q){
                go[p][c] = nq;
                p = par[p];
            }
        }
    }

    last = np;
}
```

5.5.1 广义后缀自动机

下面的写法复杂度是Σ串长的，但是胜在简单. 如果建字典树然后BFS建自动机可以做到$O(n|\Sigma|)$(n是字典树结点数)，但是后者写起来比较麻烦.

```
int extend(int p, int c) {
    int np = 0;

    if (!go[p][c]) {
        np = ++sam_cnt;
        val[np] = val[p] + 1;
        while (p && !go[p][c]) {
            go[p][c] = np;
            p = par[p];
        }
    }

    if (!p)
        par[np] = 1;
    else {
        int q = go[p][c];

        if (val[q] == val[p] + 1) {
            if (np)
                par[np] = q;
            else
                return q;
        }
        else {
            int nq = ++sam_cnt;
            val[nq] = val[p] + 1;
            memcpy(go[nq], go[q], sizeof(go[q]));

            par[nq] = par[q];
            par[q] = nq;
            if (np)
                par[np] = nq;

            while (p && go[p][c] == q){
                go[p][c] = nq;
                p = par[p];
            }

            if (!np)
                return nq;
        }
    }

    return np;
}
// 调用的时候直接last = 1然后一路调用last = extend(last, c)就行了
```

5.5.2 区间本质不同子串计数(后缀自动机+LCT+线段树)

问题：给定一个字符串s，多次询问$[l,r]$区间的本质不同的子串个数，可能强制在线.

做法：考虑建出后缀自动机，然后枚举右端点，用线段树维护每个左端点的答案.

显然只有right集合在$[l,r]$中的串才有可能有贡献，所以我们可以只考虑每个串最大的right.

每次右端点+1时找到它对应的结点u，则u到根节点路径上的每个点，它的right集合都会被r更新.

对于某个特定的左端点l，我们需要保证本质不同的子串左端点不能越过它；因此对于一个结点p，我们知道它对应的子串长度$(val_{par_p}, val_p]$之后，在p的right集合最大值减去对应长度，这样对应的l内全部+1即可；这样询问时就只需要查询r对应的线段树中$[l,r]$的区间和.（当然旧的right对应的区间也要减掉）

实际上可以发现更新时都是把路径分成若干个整段更新right集合，因此可以用LCT维护这个过程.

时间复杂度$O(n\log^2 n)$，空间$O(n)$，当然如果强制在线的话，就把线段树改成主席树，空间复杂度就和时间复杂度同阶了.

```
int tim; // tim实际上就是当前的右端点

node *access(node *x) {
    node *y = null;

    while (x != null) {
        splay(x);

        x -> ch[1] = null;
        x -> refresh();

        if (x -> val) // val记录的是上次访问时间，也就是right集合最大值
            update(x -> val - val[x -> r] + 1, x -> val - val[par[x -> l]], -1);

        x -> val = tim;
        x -> lazy = true;

        update(x -> val - val[x -> r] + 1, x -> val - val[par[x -> l]], 1);

        x -> ch[1] = y;

        (y = x) -> refresh();

        x = x -> p;
    }

    return y;
}

// 以下是main函数中的用法
for (int i = 1; i <= n; i++) {
    tim++;
    access(null + id[i]);

    if (i >= m) // 例题询问长度是固定的，如果不固定的话就按照右端点离线即可
        ans[i - m + 1] = query(i - m + 1, i);
}
```

5.6 回文树

```
// 定理：一个字符串本质不同的回文子串个数是O(n)的
// 注意回文树只需要开一倍结点，另外结点编号也是一个可用的bfs序

// 全局数组定义
int val[maxn], par[maxn], go[maxn][26], last, cnt;
char s[maxn];

// 重要!在主函数最前面一定要加上以下初始化
par[0] = cnt = 1;
val[1] = -1;
// 这个初始化和广义回文树不一样，写普通题可以用，广义回文树就不要乱搞了

// extend函数 均摊O(1)
// 向后扩展一个字符
// 传入对应下标
void extend(int n) {
    int p = last, c = s[n] - 'a';
    while (s[n - val[p] - 1] != s[n])
        p = par[p];

    if (!go[p][c]) {
        int q = ++cnt, now = p;
        val[q] = val[p] + 2;

        do
            p=par[p];
        while (s[n - val[p] - 1] != s[n]);

        par[q] = go[p][c];
        last = go[now][c] = q;
    }
    else
        last = go[p][c];

    // a[last]++;
}
```

5.6.1 广义回文树

(代码是梯子剖分的版本，压力不大的题目换成直接倍增就好了，常数只差不到一倍)

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

using namespace std;

constexpr int maxn = 1000005, mod = 1000000007;

int val[maxn], par[maxn], go[maxn][26], fail[maxn][26], pam_last[maxn], pam_cnt;
int weight[maxn], pow_26[maxn];

int trie[maxn][26], trie_cnt, d[maxn], mxd[maxn], son[maxn], top[maxn], len[maxn], sum[maxn];
char chr[maxn];
int f[25][maxn], log_tbl[maxn];
vector<int> v[maxn];

vector<int> queries[maxn];

char str[maxn];
int n, m, ans[maxn];

int add(int x, int c) {
    if (!trie[x][c]) {
        trie[x][c] = ++trie_cnt;
        f[0][trie[x][c]] = x;
        chr[trie[x][c]] = c + 'a';
    }

    return trie[x][c];
}

int del(int x) {
    return f[0][x];
}

void dfs1(int x) {
    mxd[x] = d[x] = d[f[0][x]] + 1;

    for (int i = 0; i < 26; i++)
        if (trie[x][i]) {
            int y = trie[x][i];

            dfs1(y);

            mxd[x] = max(mxd[x], mxd[y]);
            if (mxd[y] > mxd[son[x]])
                son[x] = y;
        }
}

void dfs2(int x) {
    if (x == son[f[0][x]])
        top[x] = top[f[0][x]];
    else
        top[x] = x;

    for (int i = 0; i < 26; i++)
        if (trie[x][i]) {
            int y = trie[x][i];
            dfs2(y);
        }

    if (top[x] == x) {
        int u = x;
        while (top[son[u]] == x)
            u = son[u];

        len[x] = d[u] - d[x];
        for (int i = 0; i < len[x]; i++) {
            v[x].push_back(u);
            u = f[0][u];
        }

        u = x;
        for (int i = 0; i < len[x]; i++) { // 梯子剖分,要延长一倍
            v[x].push_back(u);
            u = f[0][u];
        }
    }
}

int get_anc(int x, int k) {
    if (!k)
        return x;
    if (k > d[x])
        return 0;

    x = f[log_tbl[k]][x];
    k ^= 1 << log_tbl[k];

    return v[top[x]][d[top[x]] + len[top[x]] - d[x] + k];
}

char get_char(int x, int k) { // 查询x前面k个的字符是哪个
    return chr[get_anc(x, k)];
}

int getfail(int x, int p) {
    if (get_char(x, val[p] + 1) == chr[x])
        return p;
    return fail[p][chr[x] - 'a'];
}

int extend(int x) {

    int p = pam_last[f[0][x]], c = chr[x] - 'a';

    p = getfail(x, p);

    int new_last;

    if (!go[p][c]) {
        int q = ++pam_cnt, now = p;
        val[q] = val[p] + 2;

        p = getfail(x, par[p]);

        par[q] = go[p][c];
        new_last = go[now][c] = q;

        for (int i = 0; i < 26; i++)
            fail[q][i] = fail[par[q]][i];

        if (get_char(x, val[par[q]]) >= 'a')
            fail[q][get_char(x, val[par[q]]) - 'a'] = par[q];

        if (val[q] <= n)
            weight[q] = (weight[par[q]] + (long long)(n - val[q] + 1) * pow_26[n - val[q]]) % mod;
        else
            weight[q] = weight[par[q]];
    }
    else
        new_last = go[p][c];

    pam_last[x] = new_last;

    return weight[pam_last[x]];
}

void bfs() {

    queue<int> q;

    q.push(1);

    while (!q.empty()) {
        int x = q.front();
        q.pop();

        sum[x] = sum[f[0][x]];
        if (x > 1)
            sum[x] = (sum[x] + extend(x)) % mod;

        for (int i : queries[x])
            ans[i] = sum[x];

        for (int i = 0; i < 26; i++)
            if (trie[x][i])
                q.push(trie[x][i]);
    }

}

int main() {

    pow_26[0] = 1;
    log_tbl[0] = -1;

    for (int i = 1; i <= 1000000; i++) {
        pow_26[i] = 26ll * pow_26[i - 1] % mod;
        log_tbl[i] = log_tbl[i / 2] + 1;
    }

    int T;
    scanf("%d", &T);

    while (T--) {
        scanf("%d%d%s", &n, &m, str);

        trie_cnt = 1;
        chr[1] = '#';

        int last = 1;
        for (char *c = str; *c; c++)
            last = add(last, *c - 'a');

        queries[last].push_back(0);

        for (int i = 1; i <= m; i++) {
            int op;
            scanf("%d", &op);

            if (op == 1) {
                char c;
                scanf(" %c", &c);

                last = add(last, c - 'a');
            }
            else
                last = del(last);

            queries[last].push_back(i);
        }

        dfs1(1);
        dfs2(1);

        for (int j = 1; j <= log_tbl[trie_cnt]; j++)
            for (int i = 1; i <= trie_cnt; i++)
                f[j][i] = f[j - 1][f[j - 1][i]];

        par[0] = pam_cnt = 1;


        for (int i = 0; i < 26; i++)
            fail[0][i] = fail[1][i] = 1;

        val[1] = -1;
        pam_last[1] = 1;

        bfs();

        for (int i = 0; i <= m; i++)
            printf("%d\n", ans[i]);

        for (int j = 0; j <= log_tbl[trie_cnt]; j++)
            memset(f[j], 0, sizeof(f[j]));

        for (int i = 1; i <= trie_cnt; i++) {
            chr[i] = 0;
            d[i] = mxd[i] = son[i] = top[i] = len[i] = pam_last[i] = sum[i] = 0;
            v[i].clear();
            queries[i].clear();

            memset(trie[i], 0, sizeof(trie[i]));
        }
        trie_cnt = 0;

        for (int i = 0; i <= pam_cnt; i++) {
            val[i] = par[i] = weight[i];

            memset(go[i], 0, sizeof(go[i]));
            memset(fail[i], 0, sizeof(fail[i]));
        }
        pam_cnt = 0;

    }

    return 0;
}
```

5.7 Manacher马拉车

```
//n为串长,回文半径输出到p数组中
//数组要开串长的两倍
void manacher(const char *t, int n) {
    static char s[maxn * 2];

    for (int i = n; i; i--)
        s[i * 2] = t[i];
    for (int i = 0; i <= n; i++)
        s[i * 2 + 1]='#';

    s[0] = '$';
    s[(n + 1) * 2] = '\0';
    n = n * 2 + 1;

    int mx = 0, j = 0;

    for (int i = 1; i <= n; i++) {
        p[i] = (mx > i ? min(p[j * 2 - i], mx - i) : 1);
        while (s[i - p[i]] == s[i + p[i]])
            p[i]++;

        if (i + p[i] > mx) {
            mx = i + p[i];
            j = i;
        }
    }
}
```

5.8 字符串原理

KMP和AC自动机的fail指针存储的都是它在串或者字典树上的最长后缀，因此要判断两个前缀是否互为后缀时可以直接用fail指针判断. 当然它不能做子串问题，也不能做最长公共后缀.
后缀数组利用的主要是LCP长度可以按照字典序做RMQ的性质，与某个串的LCP长度≥某个值的后缀形成一个区间. 另外一个比较好用的性质是本质不同的子串个数 = 所有子串数 - 字典序相邻的串的height.
后缀自动机实际上可以接受的是所有后缀，如果把中间状态也算上的话就是所有子串. 它的fail指针代表的也是当前串的后缀，不过注意每个状态可以代表很多状态，只要右端点在right集合中且长度处在$(val_{par_p}, val_p]$中的串都被它代表.
后缀自动机的fail树也就是**反串**的后缀树. 每个结点代表的串和后缀自动机同理，两个串的LCP长度也就是他们在后缀树上的LCA.

6. 动态规划

6.1 决策单调性$O(n \log n)$

```
int a[maxn], q[maxn], p[maxn], g[maxn]; // 存左端点,右端点就是下一个左端点 - 1

long long f[maxn], s[maxn];

int n, m;

long long calc(int l, int r) {
    if (r < l)
        return 0;

    int mid = (l + r) / 2;
    if ((r - l + 1) % 2 == 0)
        return (s[r] - s[mid]) - (s[mid] - s[l - 1]);
    else
        return (s[r] - s[mid]) - (s[mid - 1] - s[l - 1]);
}

int solve(long long tmp) {
    memset(f, 63, sizeof(f));
    f[0] = 0;

    int head = 1, tail = 0;

    for (int i = 1; i <= n; i++) {
        f[i] = calc(1, i);
        g[i] = 1;

        while (head < tail && p[head + 1] <= i)
            head++;
        if (head <= tail) {
            if (f[q[head]] + calc(q[head] + 1, i) < f[i]) {
                f[i] = f[q[head]] + calc(q[head] + 1, i);
                g[i] = g[q[head]] + 1;
            }
            while (head < tail && p[head + 1] <= i + 1)
                head++;
            if (head <= tail)
                p[head] = i + 1;
        }
        f[i] += tmp;

        int r = n;

        while(head <= tail) {
            if (f[q[tail]] + calc(q[tail] + 1, p[tail]) > f[i] + calc(i + 1, p[tail])) {
                r = p[tail] - 1;
                tail--;
            }
            else if (f[q[tail]] + calc(q[tail] + 1, r) <= f[i] + calc(i + 1, r)) {
                if (r < n) {
                    q[++tail] = i;
                    p[tail] = r + 1;
                }
                break;
            }
            else {
                int L = p[tail], R = r;
                while (L < R) {
                    int M = (L + R) / 2;

                    if (f[q[tail]] + calc(q[tail] + 1, M) <= f[i] + calc(i + 1, M))
                        L = M + 1;
                    else
                        R = M;
                }

                q[++tail] = i;
                p[tail] = L;

                break;
            }
        }
        if (head > tail) {
            q[++tail] = i;
            p[tail] = i + 1;
        }
    }

    return g[n];
}
```

6.2 例题(待完成)

7. 计算几何

7.1 Delaunay三角剖分 Voronoi图(待完成)

8. Miscellaneous

8.1 $O(1)$快速乘

如果对速度要求很高并且不能用指令集，可以去看fstqwq的模板.

```cpp
// long double 快速乘
// 在两数直接相乘会爆long long时才有必要使用
// 常数比直接long long乘法 + 取模大很多，非必要时不建议使用
long long mul(long long a, long long b, long long p) {
    a %= p;
    b %= p;
    return ((a * b - p * (long long)((long double)a / p * b + 0.5)) % p + p) % p;
}

// 指令集快速乘
// 试机记得测试能不能过编译
inline long long mul(const long long a, const long long b, const long long p) {
    long long ans;
    __asm__ __volatile__ ("\tmulq %%rbx\n\tdivq %%rcx\n" : "=d"(ans) : "a"(a), "b"(b), "c"(p));
    return ans;
}

// int乘法取模，大概比直接做快一倍
inline int mul_mod(int a, int b, int p) {
    int ans;
    __asm__ __volatile__ ("\tmull %%ebx\n\tdivl %%ecx\n" : "=d"(ans) : "a"(a), "b"(b), "c"(p));
    return ans;
}
```

8.2 Kahan求和算法(减少浮点数累加的误差)

当然一般来说是用不到的，累加被卡精度了才有必要考虑.

```cpp
double kahanSum(vector<double> vec) {
    double sum = 0, c = 0;
    for (auto x : vec) {
        double y = x - c;
        double t = sum + y;
        c = (t - sum) - y;
        sum = t;
    }
    return sum;
}
```

8.3 Python Decimal

```python
import decimal

decimal.getcontext().prec = 1234 # 有效数字位数

x = decimal.Decimal(2)
x = decimal.Decimal('50.5679') # 不要用float，因为float本身就不准确

x = decimal.Decimal('50.5679'). \
    quantize(decimal.Decimal('0.00')) # 保留两位小数，50.57
x = decimal.Decimal('50.5679'). \
    quantize(decimal.Decimal('0.00'), decimal.ROUND_HALF_UP) # 四舍五入
# 第二个参数可选如下:
# ROUND_HALF_UP 四舍五入
# ROUND_HALF_DOWN 五舍六入
# ROUND_HALF_EVEN 银行家舍入法，舍入到最近的偶数
# ROUND_UP 向绝对值大的取整
# ROUND_DOWN 向绝对值小的取整
# ROUND_CEILING 向正无穷取整
# ROUND_FLOOR 向负无穷取整
# ROUND_05UP (away from zero if last digit after rounding towards zero would have been 0 or 5; otherwise towards zero)

print('%f', x ) # 这样做只有float的精度
s = str(x)

decimal.is_finate(x) # x是否有穷(NaN也算)
decimal.is_infinate(x)
decimal.is_nan(x)
decimal.is_normal(x) # x是否正常
decimal.is_signed(x) # 是否为负数

decimal.fma(a, b, c) # a * b + c，精度更高

x.exp(), x.ln(), x.sqrt(), x.log10()

# 可以转复数，前提是要import complex
```

8.4 $O(n^2)$高精度

```cpp
// 注意如果只需要正数运算的话
// 可以只抄英文名的运算函数
// 按需自取
// 乘法O(n ^ 2)，除法O(10 * n ^ 2)

const int maxn = 1005;

struct big_decimal {
    int a[maxn];
    bool negative;

    big_decimal() {
        memset(a, 0, sizeof(a));
        negative = false;
    }

    big_decimal(long long x) {
        memset(a, 0, sizeof(a));
        negative = false;

        if (x < 0) {
            negative = true;
            x = -x;
        }

        while (x) {
            a[++a[0]] = x % 10;
            x /= 10;
        }
    }

    big_decimal(string s) {
        memset(a, 0, sizeof(a));
        negative = false;

        if (s == "")
            return;

        if (s[0] == '-') {
            negative = true;
            s = s.substr(1);
        }
        a[0] = s.size();
        for (int i = 1; i <= a[0]; i++)
```

```cpp
            a[i] = s[a[0] - i] - '0';

        while (a[0] && !a[a[0]])
            a[0]--;
    }

    void input() {
        string s;
        cin >> s;
        *this = s;
    }

    string str() const {
        if (!a[0])
            return "0";

        string s;
        if (negative)
            s = "-";

        for (int i = a[0]; i; i--)
            s.push_back('0' + a[i]);

        return s;
    }

    operator string () const {
        return str();
    }

    big_decimal operator - () const {
        big_decimal o = *this;
        if (a[0])
            o.negative ^= true;

        return o;
    }

    friend big_decimal abs(const big_decimal &u) {
        big_decimal o = u;
        o.negative = false;
        return o;
    }

    big_decimal &operator <<= (int k) {
        a[0] += k;

        for (int i = a[0]; i > k; i--)
            a[i] = a[i - k];

        for(int i = k; i; i--)
            a[i] = 0;

        return *this;
    }

    friend big_decimal operator << (const big_decimal &u, int k) {
        big_decimal o = u;
        return o <<= k;
    }

    big_decimal &operator >>= (int k) {
        if (a[0] < k)
            return *this = big_decimal(0);

        a[0] -= k;
        for (int i = 1; i <= a[0]; i++)
            a[i] = a[i + k];

        for (int i = a[0] + 1; i <= a[0] + k; i++)
            a[i] = 0;

        return *this;
    }

    friend big_decimal operator >> (const big_decimal &u, int k) {
        big_decimal o = u;
        return o >>= k;
    }

    friend int cmp(const big_decimal &u, const big_decimal &v) {
        if (u.negative || v.negative) {
            if (u.negative && v.negative)
                return -cmp(-u, -v);

            if (u.negative)
                return -1;

            if (v.negative)
                return 1;
        }

        if (u.a[0] != v.a[0])
            return u.a[0] < v.a[0] ? -1 : 1;

        for (int i = u.a[0]; i; i--)
            if (u.a[i] != v.a[i])
                return u.a[i] < v.a[i] ? -1 : 1;

        return 0;
    }

    friend bool operator < (const big_decimal &u, const big_decimal &v) {
        return cmp(u, v) == -1;
    }

    friend bool operator > (const big_decimal &u, const big_decimal &v) {
        return cmp(u, v) == 1;
    }

    friend bool operator == (const big_decimal &u, const big_decimal &v) {
        return cmp(u, v) == 0;
    }

    friend bool operator <= (const big_decimal &u, const big_decimal &v) {
        return cmp(u, v) <= 0;
    }

    friend bool operator >= (const big_decimal &u, const big_decimal &v) {
        return cmp(u, v) >= 0;
    }

    friend big_decimal decimal_plus(const big_decimal &u, const big_decimal &v) { // 保证u, v均为正数的话可以直接调用
        big_decimal o;

        o.a[0] = max(u.a[0], v.a[0]);

        for (int i = 1; i <= u.a[0] || i <= v.a[0]; i++) {
            o.a[i] += u.a[i] + v.a[i];

            if (o.a[i] >= 10) {
                o.a[i + 1]++;
                o.a[i] -= 10;
            }
        }

        if (o.a[o.a[0] + 1])
            o.a[0]++;

        return o;
    }

    friend big_decimal decimal_minus(const big_decimal &u, const big_decimal &v) { // 保证u, v均为正数的话可以直接调用
        int k = cmp(u, v);

        if (k == -1)
            return -decimal_minus(v, u);
        else if (k == 0)
            return big_decimal(0);

        big_decimal o;

        o.a[0] = u.a[0];

        for (int i = 1; i <= u.a[0]; i++) {
            o.a[i] += u.a[i] - v.a[i];

            if (o.a[i] < 0) {
                o.a[i] += 10;
                o.a[i + 1]--;
            }
        }

        while (o.a[0] && !o.a[o.a[0]])
            o.a[0]--;

        return o;
    }

    friend big_decimal decimal_multi(const big_decimal &u, const big_decimal &v) {
        big_decimal o;

        o.a[0] = u.a[0] + v.a[0] - 1;

        for (int i = 1; i <= u.a[0]; i++)
            for (int j = 1; j <= v.a[0]; j++)
                o.a[i + j - 1] += u.a[i] * v.a[j];

        for (int i = 1; i <= o.a[0]; i++)
            if (o.a[i] >= 10) {
                o.a[i + 1] += o.a[i] / 10;
                o.a[i] %= 10;
            }

        if (o.a[o.a[0] + 1])
            o.a[0]++;

        return o;
    }

    friend pair<big_decimal, big_decimal> decimal_divide(big_decimal u, big_decimal v) { // 整除
        if (v > u)
            return make_pair(big_decimal(0), u);

        big_decimal o;
        o.a[0] = u.a[0] - v.a[0] + 1;

        int m = v.a[0];
        v <<= u.a[0] - m;

        for (int i = u.a[0]; i >= m; i--) {
            while (u >= v) {
                u = u - v;
                o.a[i - m + 1]++;
            }

            v >>= 1;
        }

        while (o.a[0] && !o.a[o.a[0]])
            o.a[0]--;

        return make_pair(o, u);
    }

    friend big_decimal operator + (const big_decimal &u, const big_decimal &v) {
        if (u.negative || v.negative) {
            if (u.negative && v.negative)
                return -decimal_plus(-u, -v);

            if (u.negative)
                return v - (-u);

            if (v.negative)
                return u - (-v);
        }

        return decimal_plus(u, v);
    }

    friend big_decimal operator - (const big_decimal &u, const big_decimal &v) {
        if (u.negative || v.negative) {
            if (u.negative && v.negative)
                return -decimal_minus(-u, -v);

            if (u.negative)
                return -decimal_plus(-u, v);

            if (v.negative)
                return decimal_plus(u, -v);
        }

        return decimal_minus(u, v);
    }

    friend big_decimal operator * (const big_decimal &u, const big_decimal &v) {
        if (u.negative || v.negative) {
            big_decimal o = decimal_multi(abs(u), abs(v));

            if (u.negative ^ v.negative)
                return -o;
            return o;
        }

        return decimal_multi(u, v);
    }

    big_decimal operator * (long long x) const {
        if (x >= 10)
            return *this * big_decimal(x);

        if (negative)
            return -(*this * x);

        big_decimal o;

        o.a[0] = a[0];

        for (int i = 1; i <= a[0]; i++) {
            o.a[i] += a[i] * x;

            if (o.a[i] >= 10) {
                o.a[i + 1] += o.a[i] / 10;
                o.a[i] %= 10;
            }
        }

        if (o.a[a[0] + 1])
            o.a[0]++;

        return o;
    }

    friend pair<big_decimal, big_decimal> decimal_div(const big_decimal &u, const big_decimal &v) {
        if (u.negative || v.negative) {
            pair<big_decimal, big_decimal> o = decimal_div(abs(u), abs(v));

            if (u.negative ^ v.negative)
                return make_pair(-o.first, -o.second);
            return o;
        }

        return decimal_divide(u, v);
    }

    friend big_decimal operator / (const big_decimal &u, const big_decimal &v) { // v不能是0
        if (u.negative || v.negative) {
            big_decimal o = abs(u) / abs(v);

            if (u.negative ^ v.negative)
                return -o;
            return o;
        }

        return decimal_divide(u, v).first;
    }

    friend big_decimal operator % (const big_decimal &u, const big_decimal &v) {
        if (u.negative || v.negative) {
            big_decimal o = abs(u) % abs(v);

            if (u.negative ^ v.negative)
                return -o;
            return o;
        }

        return decimal_divide(u, v).second;
    }
};
```

8.5 笛卡尔树

```
int s[maxn], root, lc[maxn], rc[maxn];

int top = 0;
s[++top] = root = 1;
for (int i = 2; i <= n; i++) {
    s[top + 1] = 0;
    while (a[i] < a[s[top]]) // 小根笛卡尔树
        top--;

    if (top)
        rc[s[top]] = i;
    else
        root = i;

    lc[i] = s[top + 1];
    s[++top] = i;
}
```

8.6 GarsiaWachs算法($O(n\log n)$合并石子)

设序列是$\{a_i\}$，从左往右，找到一个最小的且满足$a_{k-1} \le a_{k+1}$的k，找到后合并a_k和a_{k-1}，再从当前位置开始向左找最大的j满足$a_j \ge a_k + a_{k-1}$(当然是指合并前的)，然后把$a_k + a_{k-1}$插到j的后面就行．一直重复，直到只剩下一堆石子就可以了．

另外在这个过程中，可以假设a_{-1}和a_n是正无穷的，可省略边界的判别．把a_0设为INF，a_{n+1}设为INF-1，可实现剩余一堆石子时自动结束．

8.7 常用NTT素数及原根

$p = r \times 2^k + 1$	r	k	最小原根
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
985661441	235	22	3
998244353	119	23	3
1004535809	479	21	3
1005060097*	1917	19	**5**
2013265921	15	27	**31**
2281701377	17	27	3
31525197391593473	7	52	3
180143985094819841	5	55	**6**
1945555039024054273	27	56	**5**
4179340454199820289	29	57	3

*注：1005060097有点危险，在变化长度大于$524288 = 2^{19}$时不可用．

8.8 xorshift

```
ull k1, k2;
const int mod = 10000000;
ull xorShift128Plus() {
    ull k3 = k1, k4 = k2;
    k1 = k4;
    k3 ^= (k3 << 23);
    k2 = k3 ^ k4 ^ (k3 >> 17) ^ (k4 >> 26);
    return k2 + k4;
}
void gen(ull _k1, ull _k2) {
    k1 = _k1, k2 = _k2;
    int x = xorShift128Plus() % threshold + 1;
    // do sth
}


```

```
uint32_t xor128(void) {
    static uint32_t x = 123456789;
    static uint32_t y = 362436069;
    static uint32_t z = 521288629;
    static uint32_t w = 88675123;
    uint32_t t;

    t = x ^ (x << 11);
    x = y; y = z; z = w;
    return w = w ^ (w >> 19) ^ (t ^ (t >> 8));
}
```

8.9 枚举子集

(注意这是$t \neq 0$的写法，如果可以等于0需要在循环里手动`break`)

```
for (int t = s; t; (--t) &= s) {
    // do something
}
```

8.10 STL

8.10.1 vector

- `vector(int nSize)`: 创建一个vector，元素个数为nSize
- `vector(int nSize, const T &value)`: 创建一个vector，元素个数为nSize，且值均为value
- `vector(begin, end)`: 复制[begin, end)区间内另一个数组的元素到vector中
- `void assign(int n, const T &x)`: 设置向量中前n个元素的值为x
- `void assign(const_iterator first, const_iterator last)`: 向量中[first, last)中元素设置成当前向量元素

8.10.2 list

- `assign()` 给list赋值
- `back()` 返回最后一个元素
- `begin()` 返回指向第一个元素的迭代器
- `clear()` 删除所有元素
- `empty()` 如果list是空的则返回true
- `end()` 返回末尾的迭代器
- `erase()` 删除一个元素
- `front()` 返回第一个元素
- `insert()` 插入一个元素到list中
- `max_size()` 返回list能容纳的最大元素数量
- `merge()` 合并两个list
- `pop_back()` 删除最后一个元素
- `pop_front()` 删除第一个元素
- `push_back()` 在list的末尾添加一个元素
- `push_front()` 在list的头部添加一个元素
- `rbegin()` 返回指向第一个元素的逆向迭代器
- `remove()` 从list删除元素
- `remove_if()` 按指定条件删除元素
- `rend()` 指向list末尾的逆向迭代器
- `resize()` 改变list的大小
- `reverse()` 把list的元素倒转
- `size()` 返回list中的元素个数
- `sort()` 给list排序
- `splice()` 合并两个list
- `swap()` 交换两个list
- `unique()` 删除list中重复的元

8.10.3 unordered_set / unordered_map

- `unordered_map<int, int, hash>`: 自定义哈希函数，其中hash是一个带重载括号的类.

8.11 Public Based DataStructure(PB_DS)

8.11.1 哈希表

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace __gnu_pbds;

cc_hash_table<string, int> mp1; // 拉链法
gp_hash_table<string, int> mp2; // 查探法(快一些)
```

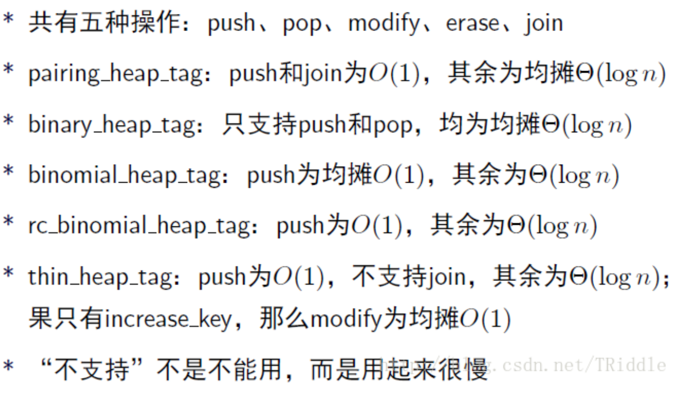
8.11.2 堆

默认也是大根堆，和`std::priority_queue`保持一致.

```
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;

__gnu_pbds::priority_queue<int> q;
__gnu_pbds::priority_queue<int, greater<int>, pairing_heap_tag> pq;
```

效率参考:



* 共有五种操作：push、pop、modify、erase、join

* pairing_heap_tag：push和join为$O(1)$，其余为均摊$\Theta(\log n)$

* binary_heap_tag：只支持push和pop，均为均摊$\Theta(\log n)$

* binomial_heap_tag：push为均摊$O(1)$，其余为$\Theta(\log n)$

* rc_binomial_heap_tag：push为$O(1)$，其余为$\Theta(\log n)$

* thin_heap_tag：push为$O(1)$，不支持join，其余为$\Theta(\log n)$；果只有increase_key，那么modify为均摊$O(1)$

* “不支持”不是不能用，而是用起来很慢

csdn.net/TRiddle

常用操作:

- `push()`: 向堆中压入一个元素，返回迭代器
- `pop()`: 将堆顶元素弹出
- `top()`: 返回堆顶元素
- `size()`: 返回元素个数
- `empty()`: 返回是否非空
- `modify(point_iterator, const key)`: 把迭代器位置的 key 修改为传入的 key
- `erase(point_iterator)`: 把迭代器位置的键值从堆中删除
- `join(__gnu_pbds::priority_queue &other)`: 把 other 合并到 `*this`，并把 other 清空

8.11.3 平衡树

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> t;

// rb_tree_tag 红黑树(还有splay_tree_tag和ov_tree_tag, 后者不知道是什么)
```

注意第五个参数要填`tree_order_statistics_node_update`才能使用排名操作.

- `insert(x)`: 向树中插入一个元素x，返回`pair<point_iterator, bool>`

- `erase(x)`: 从树中删除一个元素/迭代器x, 返回一个 `bool` 表明是否删除成功
- `order_of_key(x)`: 返回x的排名, 0-based
- `find_by_order(x)`: 返回排名(0-based)所对应元素的迭代器
- `lower_bound(x) / upper_bound(x)`: 返回第一个≥或者>x的元素的迭代器
- `join(x)`: 将x树并入当前树, 前提是两棵树的类型一样, 并且二者值域不能重叠, x树会被删除
- `split(x,b)`: 分裂成两部分, 小于等于x的属于当前树, 其余的属于b树
- `empty()`: 返回是否为空
- `size()`: 返回大小

(注意平衡树不支持多重值, 如果需要多重值, 可以再开一个`unordered_map`来记录值出现的次数, 将`x<<32`后加上出现的次数后插入. 注意此时应该为long long类型.)

8.12 rope

```
#include <ext/rope>
using namespace __gnu_cxx;

push_back(x); // 在末尾添加x
insert(pos, x); // 在pos插入x, 自然支持整个char数组的一次插入
erase(pos, x); // 从pos开始删除x个, 不要只传一个参数, 有毒
copy(pos, len, x); // 从pos开始到pos + len为止的部分, 赋值给x
replace(pos, x); // 从pos开始换成x
substr(pos, x); // 提取pos开始x个
at(x) / [x]; // 访问第x个元素
```

8.13 其他C++相关

8.13.1 <cmath>

- `std::log1p(x)`: (注意是数字1)返回$\ln(1+x)$的值, x非常接近0时比直接exp精确得多.
- `std::hypot(x, y[, z])`: 返回平方和的平方根, 或者说到原点的欧几里德距离.

8.13.2 <algorithm>

- `std::all_of(begin, end, f)`: 检查范围内元素调用函数f后是否全返回真. 类似地还有`std::any_of`和`std::none_of`.
- `std::for_each(begin, end, f)`: 对范围内所有元素调用一次f. 如果传入的是引用, 也可以用f修改. (例如`for_each(a, a + n, [](int &x){cout << ++x << "\n";})`)
- `std::for_each_n(begin, n, f)`: 同上, 只不过范围改成了从begin开始的n个元素.
- `std::copy(), std::copy_n()`: 用法谁都会, 但标准里说如果元素是可平凡复制的(比如int), 那么它会避免批量赋值, 并且调用`std::memmove()`之类的快速复制函数. (一句话总结: 它跑得快)
- `std::rotate(begin, mid, end)`: 循环移动, 移动后mid位置的元素会跑到first位置. C++11起会返回begin位置的元素移动后的位置.
- `std::unique(begin, end)`: 去重, 返回去重后的end.
- `std::partition(begin, end, f)`: 把f为true的放在前面, false的放在后面, 返回值是第二部分的开头, **不保持相对顺序**. 如果要保留相对顺序可以用`std::stable_partition()`, 比如写整体二分.
- `std::partition_copy(begin, end, begin_t, begin_f, f)`: 不修改原数组, 把true的扔到begin_t, false的扔到begin_f. 返回值是两部分结尾的迭代器的pair.
- `std::equal_range(begin, end, x)`: 在已经排好序的数组里找到等于x的范围.
- `std::minmax(a, b)`: 返回`pair(min(a, b), max(a, b))`. 比如`tie(l, r) = minmax(l, r)`.

8.13.3 std::tuple

- `std::make_tuple(...)`: 返回构造好的tuple
- `std::get<i>(tup)`: 返回tuple的第i项
- `std::tuple_cat(...)`: 传入几个tuple, 返回按顺序连起来的tuple
- `std::tie(x, y, z, ...)`: 把传入的变量的左值引用绑起来作为tuple返回, 例如可以`std::tie(x, y, z) = std::make_tuple(a, b, c)`.

8.13.4 <complex>

- `complex<double> imaginary = 1i, x = 2 + 3i`: 可以这样直接构造复数.
- `real/imag(x)`: 返回实部/虚部.
- `conj(x)`: 返回共轭复数.
- `arg(x)`: 返回辐角.
- `norm(x)`: 返回模的平方. (直接求模用`abs(x)`.)
- `polar(len, theta)`: 用绝对值和辐角构造复数.

8.14 一些游戏

8.14.1 炉石传说

两个随从(a_i, h_i)和(a_j, h_j)皇城PK, 最后只有$a_i \times h_i$较大的一方才有可能活下来, 当然也有可能一起死.

8.15 OEIS

如果没有特殊说明, 那么以下数列都从第0项开始, 除非没有定义也没有好的办法解释第0项的意义.

8.15.1 计数相关

1. **卡特兰数(A000108)**
   1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, …
   性质见“数学”部分.
2. **(大)施罗德数(A006318)**
   1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, 5293446, 27297738, 142078746, 745387038, … (0-based)
   性质同样见“数学”部分, 和卡特兰数放在一起.
3. **小施罗德数(A001003)**
   1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859, 2646723, 13648869, 71039373, 372693519, … (0-based)
   性质位置同上.
   小施罗德数除了第0项以外都是施罗德数的一半.
4. **默慈金数(Motzkin numbers, A001006)**
   1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634, 310572, 853467, 2356779, … (0-based)
   性质位置同上.
5. **将点按顺序排成一圈后不自交的树的个数(A001764)**
   1, 1, 3, 12, 55, 273, 1428, 7752, 43263, 246675, 1430715, 8414640, 50067108, 300830572, 1822766520, … (0-based)
   $a_n = \frac{\binom{3n}{n}}{2n+1}$
   也就是说, 在圆上按顺序排列的n个点之间连$n-1$条不相交(除端点外)的弦, 组成一棵树的方案数.
   也等于每次只能向右或向上, 并且不能高于$y = 2x$这条直线, 从$(0,0)$走到$(n, 2n)$的方案数.
   扩展: 如果改成不能高于$y = kx$这条直线, 走到(n, kn)的方案数, 那么答案就是$\frac{\binom{(k+1)n}{n}}{kn+1}$.

6. **n个点的圆上画不相交的弦的方案数(A054726)**

   1, 1, 2, 8, 48, 352, 2880, 25216, 231168, 2190848, 21292032, 211044352, 2125246464, 21681954816, … (0-based)

   $a_n = 2^n s_{n-2}$ $(n > 2)$, s_n是上面的小施罗德数.

   和上面的区别在于, 这里可以不连满$n-1$条边. 另外默慈金数画的弦不能共享端点, 但是这里可以.

7. **Wedderburn-Etherington numbers(A001190)**

   0, 1, 1, 1, 2, 3, 6, 11, 23, 46, 98, 207, 451, 983, 2179, 4850, 10905, 24631, 56011, 127912, 293547, … (0-based)

   每个结点都有0或者2个儿子, 且总共有n个叶子结点的二叉树方案数. (**无标号**)

   同时也是$n-1$个结点的**无标号**二叉树个数.

   $A(x) = x + \frac{A(x)^2 + A(x^2)}{2} = 1 - \sqrt{1 - 2x - A(x^2)}$

8. **划分数(A000041)**

   1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, 56, 77, 101, 135, 176, 231, 297, 385, 490, 627, 792, 1002, … (0-based)

9. **贝尔数(A000110)**

   1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, 1382958545, … (0-based)

10. **错位排列数(A0000166)**

    1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841, 2290792932, 32071101049, … (0-based)

11. **交替阶乘(A005165)**

    0, 1, 1, 5, 19, 101, 619, 4421, 35899, 326981, 3301819, 36614981, 442386619, 5784634181, 81393657019, …

    $$n! - (n-1)! + (n-2)! - \ldots 1! = \sum_{i=0}^{n-1} (-1)^i (n-i)!.$$

    $a_0 = 0,\ a_n = n! - a_{n-1}.$

8.15.2 线性递推数列

1. **Lucas数(A000032)**

   2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5778, 9349, 15127, …

2. **斐波那契数(A000045)**

   0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, …

3. **泰波那契数(Tribonacci, A000071)**

   0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, 927, 1705, 3136, 5768, 10609, 19513, 35890, …

   $a_0 = a_1 = 0,\ a_2 = 1,\ a_n = a_{n-1} + a_{n-2} + a_{n-3}.$

4. **Pell数(A0000129)**

   0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, 33461, 80782, 195025, 470832, 1136689, …

   $a_0 = 0,\ a_1 = 1,\ a_n = 2a_{n-1} + a_{n-2}.$

5. **帕多万(Padovan)数(A0000931)**

   1, 0, 0, 1, 0, 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37, 49, 65, 86, 114, 151, 200, 265, 351, 465, 616, 816, 1081, 1432, 1897, 2513, 3329, 4410, 5842, 7739, 10252, 13581, 17991, 23833, 31572, …

   $a_0 = 1,\ a_1 = a_2 = 0,\ a_n = a_{n-2} + a_{n-3}.$

6. **Jacobsthal numbers(A001045)**

   0, 1, 1, 3, 5, 11, 21, 43, 85, 171, 341, 683, 1365, 2731, 5461, 10923, 21845, 43691, 87381, 174763, …

   $a_0 = 0,\ a_1 = 1.\ a_n = a_{n-1} + 2a_{n-2}$

   同时也是最接近$\frac{2^n}{3}$的整数.

7. **佩林数(A001608)**

   3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17, 22, 29, 39, 51, 68, 90, 119, 158, 209, 277, 367, 486, 644, 853, …

   $a_0 = 3,\ a_1 = 0,\ a_2 = 2,\ a_n = a_{n-2} + a_{n-3}$

8.15.3 数论相关

1. **Carmichael数, 伪质数(A002997)**

   561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633, 62745, 63973, 75361, 101101, 115921, 126217, 162401, 172081, 188461, 252601, 278545, 294409, 314821, 334153, 340561, 399001, 410041, 449065, 488881, 512461, …

   满足$\forall$与n互质的a, 都有$a^(n-1) = 1 \pmod n$的所有**合数**n被称为Carmichael数.

   Carmichael数在10^8以内只有255个.

2. **反质数(A002182)**

   1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840, 1260, 1680, 2520, 5040, 7560, 10080, 15120, 20160, 25200, 27720, 45360, 50400, 55440, 83160, 110880, 166320, 221760, 277200, 332640, 498960, 554400, 665280, 720720, 1081080, 1441440, 2162160, …

   比所有更小的数的约数数量都更多的数.

3. **前n个质数的乘积(A002110)**

   1, 2, 6, 30, 210, 2310, 30030, 510510, 9699690, 223092870, 6469693230, 200560490130, 7420738134810, …

4. **梅森质数(A000668)**

   3, 7, 31, 127, 8191, 131071, 524287, 2147483647, 2305843009213693951, 618970019642690137449562111, 162259276829213363391578010288127, 170141183460469231731687303715884105727

   p是质数, 同时$2^p - 1$也是质数.

8.15.4 其他

1. **伯努利数(A027641)**

   见"数学/常见数列"部分.

2. **四个柱子的汉诺塔(A007664)**

   0, 1, 3, 5, 9, 13, 17, 25, 33, 41, 49, 65, 81, 97, 113, 129, 161, 193, 225, 257, 289, 321, 385, 449, …

   差分之后可以发现其实就是1次+1, 2次+2, 3次+4, 4次+8…的规律.

3. **乌拉姆数(Ulam numbers, A002858)**

   1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, 57, 62, 69, 72, 77, 82, 87, 97, 99, 102, 106, 114, 126, 131, 138, 145, 148, 155, 175, 177, 180, 182, 189, 197, 206, 209, 219, 221, 236, 238, 241, 243, 253, 258, 260, 273, 282, 309, 316, 319, 324, 339 …

   $a_1 = 1,\ a_2 = 2,\ a_n$表示在所有$> a_{n-1}$的数中, 最小的, 能被表示成(前面的两个不同的元素的和)的数.

8.16 编译选项

- `-O2 -g -std=c++14`: 狗都知道
- `-Wall -Wextra -Wshadow -Wconversion`: 更多警告
  - `-Werror`: 强制将所有Warning变成Error
- `-fsanitize=(address/undefined)`: 检查有符号整数溢出(算ub)/数组越界
  - 注意无符号类型溢出不算ub.
- `-fno-ms-extensions`: 关闭一些和msvc保持一致的特性, 例如, 不标返回值类型的函数会报CE而不是默认为`int`.
  - 但是不写return的话它还是管不了.

8.17 注意事项

8.17.1 常见下毒手法

- 0/1base是不是搞混了
- 高精度高低位搞反了吗
- 线性筛抄对了吗
- 快速乘抄对了吗
- `i <= n, j <= m`
- sort比较函数是不是比了个寂寞
- 该取模的地方都取模了吗
- 边界情况(+1-1之类的)有没有想清楚
- **特判是否有必要，确定写对了吗**

8.17.2 场外相关

- 安顿好之后查一下附近的咖啡店,打印店,便利店之类的位置,以备不时之需
- 热身赛记得检查一下编译注意事项中的代码能否过编译,还有熟悉比赛场地,清楚洗手间在哪儿,测试打印机(如果可以)
- 比赛前至少要翻一遍板子,尤其要看原理与例题
- 比赛前一两天不要摸鱼,要早睡,有条件最好洗个澡;比赛当天不要起太晚,维持好的状态
- 赛前记得买咖啡,最好直接安排三人份,记得要咖啡因比较足的;如果主办方允许,就带些巧克力之类的高热量零食
- 入场之后记得检查机器,尤其要逐个检查键盘按键有没有坏的;如果可以的话,调一下gedit设置
- 开赛之前调整好心态,比赛而已,不必心急.

8.17.3 做题策略与心态调节

- 拿到题后立刻按照商量好的顺序读题，前半小时最好跳过题意太复杂的题(除非被过穿了)
- 签到题写完不要激动，稍微检查一下最可能的下毒点再交，避免无谓的罚时

  一两行的那种傻逼题就算了
- 读完题及时输出题意，一方面避免重复读题，一方面也可以让队友有一个初步印象，方便之后决定开题顺序
- 如果不能确定题意就不要贸然输出甚至上机，尤其是签到题，因为样例一般都很弱
- 一个题如果卡了很久又有其他题可以写，那不妨先放掉写更容易的题，不要在一棵树上吊死

  不要被一两道题搞得心态爆炸，一方面急也没有意义，一方面你很可能真的离AC就差一步
- 榜是不会骗人的，一个题如果被不少人过了就说明这个题很可能并没有那么难;如果不是有十足的把握就不要轻易开没什么人交的题;另外不要忘记最后一小时会封榜
- 想不出题/找不出毒自然容易犯困，一定不要放任自己昏昏欲睡，最好去洗手间冷静一下，没有条件就站起来踱步
- 思考的时候不要挂机，一定要在草稿纸上画一画，最好说出声来最不容易断掉思路
- 出完算法一定要check一下样例和一些trivial的情况，不然容易写了半天发现写了个假算法
- 上机前有时间就提前给需要思考怎么写的地方打草稿，不要浪费机时
- 查毒时如果最难的地方反复check也没有问题，就从头到脚仔仔细细查一遍，不要放过任何细节，即使是并查集和sort这种东西也不能想当然
- 后半场如果时间不充裕就不要冒险开难题，除非真的无事可做

  如果是没写过的东西也不要轻举妄动，在有其他好写的题的时候就等一会再说
- 大多数时候都要听队长安排，虽然不一定最正确但可以保持组织性
- 任何时候都不要着急，着急不能解决问题，不要当喆国王
- 输了游戏，还有人生;赢了游戏，还有人生.

8.18 附录：vscode相关

8.18.1 插件

- Chinese (Simplified) (简体中文语言包)
- C/C++
- C++ Intellisense (前提是让用)
- Better C++ Syntax
- Python
- Pylance (前提是让用)
- Rainbow Brackets (前提是让用)

8.18.2 设置选项

- Editor: Insert Spaces (取消勾选，改为tab缩进)
- Editor: Line Warp (开启折行)
- 改配色，“深色+：默认深色”
- 自动保存(F1 → “auto”)

8.18.3 快捷键

- F1 / Ctrl+Shift+P: 万能键，打开命令面板
- F8: 下一个Error Shift+F8: 上一个Error
- Ctrl+\: 水平分栏，最多3栏
- Ctrl+1/2/3: 切到对应栏
- Ctrl+[/]: 当前行向左/右缩进
- Alt+F12: 查看定义的缩略图(显示小窗，不跳过去)
- Ctrl+H: 查找替换
- Ctrl+D: 下一个匹配的也被选中(用于配合Ctrl+F)
- Ctrl+U: 回退上一个光标操作(防止光标飞了找不回去)
- Ctrl+/: 切换行注释
- Ctrl+‘(键盘左上角的倒引号): 显示终端

8.19 附录：骂人的艺术 —梁实秋

古今中外没有一个不骂人的人．骂人就是有道德观念的意思，因为在骂人的时候，至少在骂人者自己总觉得那人有该骂的地方．何者该骂，何者不该骂，这个抉择的标准，是极道德的．所以根本不骂人，大可不必．骂人是一种发泄感情的方法，尤其是那一种怨怒的感情．想骂人的时候而不骂，时常在身体上弄出毛病，所以想骂人时，骂骂何妨？

但是，骂人是一种高深的学问，不是人人都可以随便试的．有因为骂人挨嘴巴的，有因为骂人吃官司的，有因为骂人反被人骂的，这都是不会骂人的原故．今以研究所得，公诸同好，或可为骂人时之一助乎？

1. 知己知彼

   骂人是和动手打架一样的，你如其敢打人一拳，你先要自己忖度下，你吃得起别人的一拳否．这叫做知己知彼．骂人也是一样．譬如你骂他是“屈死”，你先要反省，自己和“屈死”有无分别．你骂别人荒唐，你自己想想曾否吃喝嫖赌．否则别人回敬你一二句，你就受不了．所以别人有着某种短处，而足下也正有同病，那么你在骂他的时候只得割爱．

2. 无骂不如己者

   要骂人须要挑比你大一点的人物，比你漂亮一点的或者比你坏得万倍而比你得势的人物，总之，你要骂人，那人无论在好的一方面或坏的一方面都要能胜过你，你才不吃亏．你骂大人物，就怕他不理你，他一回骂，你就算骂着了．因为身份相同的人才肯对骂．在坏的一方面胜过你的，你骂他就如教训一般，他既便回骂，一般人仍不会理会他的．假如你骂一个无关痛痒的人，你越骂他他越得意，时常可以把一个无名小卒骂出名了，你看冤与不冤？

3. 适可而止

   骂大人物骂到他回骂的时候，便不可再骂 再骂则一般人对你必无同情，以为你是无理取闹．骂小人物骂到他不能回骂的时候，便不可再骂 再骂下去则一般人对你也必无同情，以为你是欺负弱者．

4. 旁敲侧击
   他偷东西, 你骂他是贼 他抢东西, 你骂他是盗, 这是笨伯. 骂人必须先明虚实掩映之法, 须要烘托旁衬, 旁敲侧击, 于要紧处只一语便得, 所谓杀人于咽喉处着刀. 越要骂他你越要原谅他, 即便说些恭维话亦不为过, 这样的骂法才能显得你所骂的句句是真实确凿, 让旁人看起来也可见得你的度量.

5. 态度镇定
   骂人最忌浮躁. 一语不合, 面红筋跳, 暴躁如雷, 此灌夫骂座, 泼妇骂街之术, 不足以言骂人. 善骂者必须态度镇静, 行若无事. 普通一般骂人, 谁的声音高便算谁占理, 谁的来势猛便算谁骂赢, 惟真善骂人者, 乃能避其锋而击其懈. 你等他骂得疲倦的时候, 你只消轻轻的回敬他一句, 让他再狂吼一阵. 在他暴躁不堪的时候, 你不妨对他冷笑几声, 包管你不费力气, 把他气得死去活来, 骂得他针针见血.

6. 出言典雅
   骂人要骂得微妙含蓄, 你骂他一句要使他不甚觉得是骂, 等到想过一遍才慢慢觉悟这句话不是好话, 让他笑着的面孔由白而红, 由红而紫, 由紫而灰, 这才是骂人的上乘. 欲达到此种目的, 深刻之用意固不可少, 而典雅之言词则尤为重要. 言词典雅可使听者不致刺耳. 如要骂人骂得典雅, 则首先要在骂时万万别提起女人身上的某一部分, 万万不要涉及生理学范围. 骂人一骂到生理学范围以内, 底下再有什么话都不好说了. 譬如你骂某甲, 千万别提起他的令堂令妹. 因为那样一来, 便无是非可言, 并且你自己也不免有令堂令妹, 他若回敬起来, 岂非势均力敌, 半斤八两? 再者骂人的时候, 最好不要加人以种种难堪的名词, 称呼起来总要客气, 即使他是极卑鄙的小人, 你也不妨称他先生, 越客气, 越骂得有力量. 骂得时节最好引用他自己的词句, 这不但可以使他难堪, 还可以减轻他对你骂的力量. 俗话少用, 因为俗话一览无遗, 不若典雅古文曲折含蓄.

7. 以退为进
   两人对骂, 而自己亦有理屈之处, 则处于开骂伊始, 特宜注意, 最好是毅然将自己理屈之处完全承认下来, 即使道歉认错均不妨事. 先把自己理屈之处轻轻遮掩过去, 然后你再重整旗鼓, 着着逼人, 方可无后顾之忧. 即使自己没有理屈的地方, 也绝不可自行夸张, 务必要谦逊不遑, 把自己的位置降到一个不可再降的位置, 然后骂起人来, 自有一种公正光明的态度. 否则你骂他一两句, 他便以你个人的事反唇相讥, 一场对骂, 会变成两人私下口角, 是非曲直, 无从判断. 所以骂人者自己要低声下气, 此所谓以退为进.

8. 预设埋伏
   你把这句话骂过去, 你便要想想看, 他将用什么话骂回来. 有眼光的骂人者, 便处处留神, 或是先将他要骂你的话替他说出来, 或是预先安设埋伏, 令他骂回来的话失去效力. 他骂你的话, 你替他说出来, 这便等于缴了他的械一般. 预设埋伏, 便是在要攻击你的地方, 你先轻轻的安下话根, 然后他骂过来就等于枪弹打在沙包上, 不能中伤.

9. 小题大做
   如对方有该骂之处, 而题目身小, 不值一骂, 或你所知不多, 不足一骂, 那时节你便可用小题大做的方法, 来扩大题目. 先用诚恳而怀疑的态度引申对方的意思, 由不紧要之点引到大题目上去, 处处用严谨的逻辑逼他说出不逻辑的话来, 或是逼他说出合于逻辑但不合乎理的话来, 然后你再大举骂他, 骂到体无完肤为止, 而原来惹动你的小题目, 轻轻一提便了.

10. 远交近攻
    一个时候, 只能骂一个人, 或一种人, 或一派人. 决不宜多树敌. 所以骂人的时候, 万勿连累旁人, 即使必须牵涉多人, 你也要表示好意, 否则回骂之声纷至沓来, 使你无从应付.

骂人的艺术, 一时所能想起来的有上面十条, 信手拈来, 并无条理. 我做此文的用意, 是助人骂人. 同时也是想把骂人的技术揭破一点, 供爱骂人者参考. 挨骂的人看看, 骂人的心理原来是这样的, 也算是揭破一张黑幕给你瞧瞧!

8.20 附录: Cheat Sheet

见最后几页.

Theoretical Computer Science Cheat Sheet

Definitions	
$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim\limits_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\left[{n \atop k}\right]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\left\{{n \atop k}\right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle{n \atop k}\right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with k ascents.
$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\Big((i+1)^{m+1} - i^{m+1} - (m+1)i^m\Big)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n} \frac{1}{i}, \quad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \quad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\binom{n}{k} = \binom{n}{n-k}$,

4. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

12. $\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

14. $\left[{n \atop 1}\right] = (n-1)!$, **15.** $\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\left[{n \atop n}\right] = 1$, **17.** $\left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

18. $\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

25. $\left\langle{0 \atop k}\right\rangle = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

31. $\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\left\langle\!\left\langle{n \atop 0}\right\rangle\!\right\rangle = 1$, **33.** $\left\langle\!\left\langle{n \atop n}\right\rangle\!\right\rangle = 0 \quad \text{for } n \ne 0$,

34. $\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle + (2n-1-k)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle$, **35.** $\sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. $\left[\begin{matrix}n+1\\m+1\end{matrix}\right] = \sum_k \left[\begin{matrix}n\\k\end{matrix}\right]\binom{k}{m} = \sum_{k=0}^{n} \left[\begin{matrix}k\\m\end{matrix}\right] n^{\underline{n-k}} = n!\sum_{k=0}^{n}\frac{1}{k!}\left[\begin{matrix}k\\m\end{matrix}\right],$

39. $\left[\begin{matrix}x\\x-n\end{matrix}\right] = \sum_{k=0}^{n}\left\langle\!\!\left\langle\begin{matrix}n\\k\end{matrix}\right\rangle\!\!\right\rangle\binom{x+k}{2n},$

40. $\left\{\begin{matrix}n\\m\end{matrix}\right\} = \sum_k \binom{n}{k}\left\{\begin{matrix}k+1\\m+1\end{matrix}\right\}(-1)^{n-k},$

41. $\left[\begin{matrix}n\\m\end{matrix}\right] = \sum_k \left[\begin{matrix}n+1\\k+1\end{matrix}\right]\binom{k}{m}(-1)^{m-k},$

42. $\left\{\begin{matrix}m+n+1\\m\end{matrix}\right\} = \sum_{k=0}^{m} k\left\{\begin{matrix}n+k\\k\end{matrix}\right\},$

43. $\left[\begin{matrix}m+n+1\\m\end{matrix}\right] = \sum_{k=0}^{m} k(n+k)\left[\begin{matrix}n+k\\k\end{matrix}\right],$

44. $\binom{n}{m} = \sum_k \left\{\begin{matrix}n+1\\k+1\end{matrix}\right\}\left[\begin{matrix}k\\m\end{matrix}\right](-1)^{m-k},$

45. $(n-m)!\binom{n}{m} = \sum_k \left[\begin{matrix}n+1\\k+1\end{matrix}\right]\left\{\begin{matrix}k\\m\end{matrix}\right\}(-1)^{m-k}, \quad \text{for } n \ge m,$

46. $\left\{\begin{matrix}n\\n-m\end{matrix}\right\} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[\begin{matrix}m+k\\k\end{matrix}\right],$

47. $\left[\begin{matrix}n\\n-m\end{matrix}\right] = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{\begin{matrix}m+k\\k\end{matrix}\right\},$

48. $\left\{\begin{matrix}n\\\ell+m\end{matrix}\right\}\binom{\ell+m}{\ell} = \sum_k \left\{\begin{matrix}k\\\ell\end{matrix}\right\}\left\{\begin{matrix}n-k\\m\end{matrix}\right\}\binom{n}{k},$

49. $\left[\begin{matrix}n\\\ell+m\end{matrix}\right]\binom{\ell+m}{\ell} = \sum_k \left[\begin{matrix}k\\\ell\end{matrix}\right]\left[\begin{matrix}n-k\\m\end{matrix}\right]\binom{n}{k}.$

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:

$$\sum_{i=1}^{n} 2^{-d_i} \le 1,$$

and equality holds only if every internal node has 2 sons.

Recurrences

Master method:

$$T(n) = aT(n/b) + f(n), \quad a \ge 1, b > 1$$

If $\exists\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then

$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then

$$T(n) = \Theta(n^{\log_b a}\log_2 n).$$

If $\exists\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \le cf(n)$ for large n, then

$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence

$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have

$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get

$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find

$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$. Summing factors (example): Consider the following recurrence

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side

$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$\begin{gathered}1\big(T(n) - 3T(n/2) = n\big)\\ 3\big(T(n/2) - 3T(n/4) = n/2\big)\\ \vdots \quad \vdots \quad \vdots\\ 3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big)\end{gathered}$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get

$$\sum_{i=0}^{m-1}\frac{n}{2^i}3^i = n\sum_{i=0}^{m-1}\left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have

$$\begin{aligned} n\sum_{i=0}^{m-1} c^i &= n\left(\frac{c^m - 1}{c - 1}\right)\\ &= 2n(c^{\log_2 n} - 1)\\ &= 2n(c^{(k-1)\log_c n} - 1)\\ &= 2n^k - 2n,\end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider

$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that

$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find

$$\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j\\ &= T_i.\end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:

$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:

$$\sum_{i\ge0} g_{i+1}x^i = \sum_{i\ge0} 2g_i x^i + \sum_{i\ge0} x^i.$$

We choose $G(x) = \sum_{i\ge0} x^i g_i$. Rewrite in terms of $G(x)$:

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i\ge0} x^i.$$

Simplify:

$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:

$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:

$$\begin{aligned} G(x) &= x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right)\\ &= x\left(2\sum_{i\ge0} 2^i x^i - \sum_{i\ge0} x^i\right)\\ &= \sum_{i\ge0}(2^{i+1} - 1)x^{i+1}.\end{aligned}$$

So $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet

$\pi \approx 3.14159$, $e \approx 2.71828$, $\gamma \approx 0.57721$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$, $\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

i	2^i	p_i
1	2	2
2	4	3
3	8	5
4	16	7
5	32	11
6	64	13
7	128	17
8	256	19
9	512	23
10	1,024	29
11	2,048	31
12	4,096	37
13	8,192	41
14	16,384	43
15	32,768	47
16	65,536	53
17	131,072	59
18	262,144	61
19	524,288	67
20	1,048,576	71
21	2,097,152	73
22	4,194,304	79
23	8,388,608	83
24	16,777,216	89
25	33,554,432	97
26	67,108,864	101
27	134,217,728	103
28	268,435,456	107
29	536,870,912	109
30	1,073,741,824	113
31	2,147,483,648	127
32	4,294,967,296	131

Pascal's Triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):

$$B_0 = 1,\ B_1 = -\tfrac{1}{2},\ B_2 = \tfrac{1}{6},\ B_4 = -\tfrac{1}{30},\ B_6 = \tfrac{1}{42},\ B_8 = -\tfrac{1}{30},\ B_{10} = \tfrac{5}{66}.$$

Change of base, quadratic formula:

$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number e:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$

$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

$$1,\ \tfrac{3}{2},\ \tfrac{11}{6},\ \tfrac{25}{12},\ \tfrac{137}{60},\ \tfrac{49}{20},\ \tfrac{363}{140},\ \tfrac{761}{280},\ \tfrac{7129}{2520},\ \ldots$$

$$\ln n < H_n < \ln n + 1,$$

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:

1, 2, 6, 24, 120, 720, 5040, 40320, 362880, …

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:

$$a(i,j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$$

$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are n different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all n types is

$$nH_n.$$

Probability

Continuous distributions: If

$$\Pr[a < X < b] = \int_a^b p(x)\,dx,$$

then p is the probability density function of X. If

$$\Pr[X < a] = P(a),$$

then P is the distribution function of X. If P and p both exist then

$$P(a) = \int_{-\infty}^{a} p(x)\,dx.$$

Expectation: If X is discrete

$$\mathrm{E}[g(X)] = \sum_x g(x) \Pr[X = x].$$

If X continuous then

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx = \int_{-\infty}^{\infty} g(x)\,dP(x).$$

Variance, standard deviation:

$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$

$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events A and B:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$

$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff A and B are independent.

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables X and Y:

$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X] \cdot \mathrm{E}[Y],$$

if X and Y are independent.

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$

$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:

$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n} \Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:

$$\Pr\Big[\bigvee_{i=1}^{n} X_i\Big] = \sum_{i=1}^{n} \Pr[X_i] + \sum_{k=2}^{n} (-1)^{k+1} \sum_{i_i < \cdots < i_k} \Pr\Big[\bigwedge_{j=1}^{k} X_{i_j}\Big].$$

Moment inequalities:

$$\Pr\big[|X| \geq \lambda\,\mathrm{E}[X]\big] \leq \frac{1}{\lambda},$$

$$\Pr\Big[\big|X - \mathrm{E}[X]\big| \geq \lambda \cdot \sigma\Big] \leq \frac{1}{\lambda^2}.$$

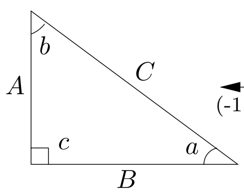
Geometric distribution:

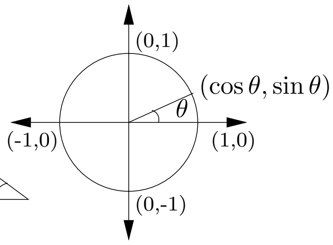
$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$

Theoretical Computer Science Cheat Sheet

Trigonometry





Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \qquad \cos a = B/C,$$

$$\csc a = C/A, \qquad \sec a = C/B,$$

$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \qquad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\tfrac{1}{2}AB, \qquad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$

$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$

$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$

$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$

$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$

$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$

$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$

$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$

$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$

$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$

$$\sin 2x = 2 \sin x \cos x, \qquad \sin 2x = \frac{2 \tan x}{1 + \tan^2 x},$$

$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$

$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$

$$\tan 2x = \frac{2 \tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2 \cot x},$$

$$\sin(x + y)\sin(x - y) = \sin^2 x - \sin^2 y,$$

$$\cos(x + y)\cos(x - y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i \sin x, \qquad e^{i\pi} = -1.$$

v2.02 ©1994 by Steve Seiden
`sseiden@acm.org`
`http://www.csc.lsu.edu/~seiden`

Matrices

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$

$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$

$$= \begin{array}{c} aei + bfg + cdh \\ -\, ceg - fha - ibd. \end{array}$$

Permanents:

$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$

$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

Identities:

$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$

$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$

$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$

$$\sinh(x + y) = \sinh x \cosh y + \cosh x \sinh y,$$

$$\cosh(x + y) = \cosh x \cosh y + \sinh x \sinh y,$$

$$\sinh 2x = 2 \sinh x \cosh x,$$

$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$

$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$

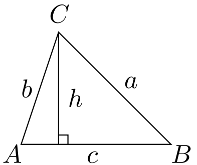
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$

$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

… in mathematics you don't understand things, you just get used to them.
– J. von Neumann

More Trig.



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab \cos C.$$

Area:

$$A = \tfrac{1}{2}hc,$$

$$= \tfrac{1}{2}ab \sin C,$$

$$= \frac{c^2 \sin A \sin B}{2 \sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$

$$s = \tfrac{1}{2}(a + b + c),$$

$$s_a = s - a,$$

$$s_b = s - b,$$

$$s_c = s - c.$$

More identities:

$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$

$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$

$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$

$$= \frac{1 - \cos x}{\sin x},$$

$$= \frac{\sin x}{1 + \cos x},$$

$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$

$$= \frac{1 + \cos x}{\sin x},$$

$$= \frac{\sin x}{1 - \cos x},$$

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$

$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$

$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$

$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$

$$\sin x = \frac{\sinh ix}{i},$$

$$\cos x = \cosh ix,$$

$$\tan x = \frac{\tanh ix}{i}.$$

Theoretical Computer Science Cheat Sheet

Number Theory

The Chinese remainder theorem: There exists a number C such that:

$$\begin{aligned} C &\equiv r_1 \bmod m_1 \\ \vdots\ &\ \vdots\ \ \vdots \\ C &\equiv r_n \bmod m_n \end{aligned}$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^{n} p_i^{e_i-1}(p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^{n} \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of} \\ & r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\Big(\frac{a}{d}\Big).$$

Prime numbers:

$$p_n = n \ln n + n \ln\ln n - n + n\frac{\ln\ln n}{\ln n} + O\Big(\frac{n}{\ln n}\Big),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\Big(\frac{n}{(\ln n)^4}\Big).$$

Graph Theory

Definitions:

Loop	An edge connecting a vertex to itself.
Directed	Each edge has a direction.
Simple	Graph with no loops or multi-edges.
Walk	A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$.
Trail	A walk with distinct edges.
Path	A trail with distinct vertices.
Connected	A graph where there exists a path between any two vertices.
Component	A maximal connected subgraph.
Tree	A connected acyclic graph.
Free tree	A tree with no root.
DAG	Directed acyclic graph.
Eulerian	Graph with a trail visiting each edge exactly once.
Hamiltonian	Graph with a cycle visiting each vertex exactly once.
Cut	A set of edges whose removal increases the number of components.
Cut-set	A minimal cut.
Cut edge	A size 1 cut.
k-Connected	A graph connected with the removal of any $k-1$ vertices.
k-Tough	$\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \|S\|$.
k-Regular	A graph where all vertices have degree k.
k-Factor	A k-regular spanning subgraph.
Matching	A set of edges, no two of which are adjacent.
Clique	A set of vertices, all of which are adjacent.
Ind. set	A set of vertices, none of which are adjacent.
Vertex cover	A set of vertices which cover all edges.
Planar graph	A graph which can be embeded in the plane.
Plane graph	An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5.

Notation:

$E(G)$	Edge set
$V(G)$	Vertex set
$c(G)$	Number of components
$G[S]$	Induced subgraph
$\deg(v)$	Degree of v
$\Delta(G)$	Maximum degree
$\delta(G)$	Minimum degree
$\chi(G)$	Chromatic number
$\chi_E(G)$	Edge chromatic number
G^c	Complement graph
K_n	Complete graph
K_{n_1,n_2}	Complete bipartite graph
$\mathrm{r}(k, \ell)$	Ramsey number

Geometry

Projective coordinates: triples (x, y, z), not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian	Projective
(x, y)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

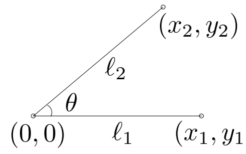
$$\big[|x_1 - x_0|^p + |y_1 - y_0|^p\big]^{1/p},$$

$$\lim_{p \to \infty} \big[|x_1 - x_0|^p + |y_1 - y_0|^p\big]^{1/p}.$$

Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\tfrac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1):

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

Theoretical Computer Science Cheat Sheet

π

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\tfrac{\pi}{4} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cdots}}}}$$

Gregrory's series:

$$\tfrac{\pi}{4} = 1 - \tfrac{1}{3} + \tfrac{1}{5} - \tfrac{1}{7} + \tfrac{1}{9} - \cdots$$

Newton's series:

$$\tfrac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\tfrac{\pi}{6} = \frac{1}{\sqrt{3}} \left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots \right)$$

Euler's series:

$$\tfrac{\pi^2}{6} = \tfrac{1}{1^2} + \tfrac{1}{2^2} + \tfrac{1}{3^2} + \tfrac{1}{4^2} + \tfrac{1}{5^2} + \cdots$$

$$\tfrac{\pi^2}{8} = \tfrac{1}{1^2} + \tfrac{1}{3^2} + \tfrac{1}{5^2} + \tfrac{1}{7^2} + \tfrac{1}{9^2} + \cdots$$

$$\tfrac{\pi^2}{12} = \tfrac{1}{1^2} - \tfrac{1}{2^2} + \tfrac{1}{3^2} - \tfrac{1}{4^2} + \tfrac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x. We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D, divide N by D, obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of N' is less than that of D. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[\frac{N(x)}{D(x)} \right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!} \left[\frac{d^k}{dx^k} \left(\frac{N(x)}{D(x)} \right) \right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
– George Bernard Shaw

Calculus

Derivatives:

1. $\dfrac{d(cu)}{dx} = c\dfrac{du}{dx}$, **2.** $\dfrac{d(u+v)}{dx} = \dfrac{du}{dx} + \dfrac{dv}{dx}$, **3.** $\dfrac{d(uv)}{dx} = u\dfrac{dv}{dx} + v\dfrac{du}{dx}$,

4. $\dfrac{d(u^n)}{dx} = nu^{n-1}\dfrac{du}{dx}$, **5.** $\dfrac{d(u/v)}{dx} = \dfrac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$, **6.** $\dfrac{d(e^{cu})}{dx} = ce^{cu}\dfrac{du}{dx}$,

7. $\dfrac{d(c^u)}{dx} = (\ln c)c^u\dfrac{du}{dx}$, **8.** $\dfrac{d(\ln u)}{dx} = \dfrac{1}{u}\dfrac{du}{dx}$,

9. $\dfrac{d(\sin u)}{dx} = \cos u\dfrac{du}{dx}$, **10.** $\dfrac{d(\cos u)}{dx} = -\sin u\dfrac{du}{dx}$,

11. $\dfrac{d(\tan u)}{dx} = \sec^2 u\dfrac{du}{dx}$, **12.** $\dfrac{d(\cot u)}{dx} = \csc^2 u\dfrac{du}{dx}$,

13. $\dfrac{d(\sec u)}{dx} = \tan u \, \sec u\dfrac{du}{dx}$, **14.** $\dfrac{d(\csc u)}{dx} = -\cot u \, \csc u\dfrac{du}{dx}$,

15. $\dfrac{d(\arcsin u)}{dx} = \dfrac{1}{\sqrt{1-u^2}}\dfrac{du}{dx}$, **16.** $\dfrac{d(\arccos u)}{dx} = \dfrac{-1}{\sqrt{1-u^2}}\dfrac{du}{dx}$,

17. $\dfrac{d(\arctan u)}{dx} = \dfrac{1}{1+u^2}\dfrac{du}{dx}$, **18.** $\dfrac{d(\operatorname{arccot} u)}{dx} = \dfrac{-1}{1+u^2}\dfrac{du}{dx}$,

19. $\dfrac{d(\operatorname{arcsec} u)}{dx} = \dfrac{1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **20.** $\dfrac{d(\operatorname{arccsc} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$,

21. $\dfrac{d(\sinh u)}{dx} = \cosh u\dfrac{du}{dx}$, **22.** $\dfrac{d(\cosh u)}{dx} = \sinh u\dfrac{du}{dx}$,

23. $\dfrac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\dfrac{du}{dx}$, **24.** $\dfrac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\dfrac{du}{dx}$,

25. $\dfrac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \, \tanh u\dfrac{du}{dx}$, **26.** $\dfrac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \, \coth u\dfrac{du}{dx}$,

27. $\dfrac{d(\operatorname{arcsinh} u)}{dx} = \dfrac{1}{\sqrt{1+u^2}}\dfrac{du}{dx}$, **28.** $\dfrac{d(\operatorname{arccosh} u)}{dx} = \dfrac{1}{\sqrt{u^2-1}}\dfrac{du}{dx}$,

29. $\dfrac{d(\operatorname{arctanh} u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx}$, **30.** $\dfrac{d(\operatorname{arccoth} u)}{dx} = \dfrac{1}{u^2-1}\dfrac{du}{dx}$,

31. $\dfrac{d(\operatorname{arcsech} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **32.** $\dfrac{d(\operatorname{arccsch} u)}{dx} = \dfrac{-1}{|u|\sqrt{1+u^2}}\dfrac{du}{dx}$.

Integrals:

1. $\displaystyle\int cu \, dx = c\int u \, dx$, **2.** $\displaystyle\int (u+v) \, dx = \int u \, dx + \int v \, dx$,

3. $\displaystyle\int x^n \, dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1$, **4.** $\displaystyle\int \frac{1}{x} dx = \ln x$, **5.** $\displaystyle\int e^x \, dx = e^x$,

6. $\displaystyle\int \frac{dx}{1+x^2} = \arctan x$, **7.** $\displaystyle\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

8. $\displaystyle\int \sin x \, dx = -\cos x$, **9.** $\displaystyle\int \cos x \, dx = \sin x$,

10. $\displaystyle\int \tan x \, dx = -\ln|\cos x|$, **11.** $\displaystyle\int \cot x \, dx = \ln|\cos x|$,

12. $\displaystyle\int \sec x \, dx = \ln|\sec x + \tan x|$, **13.** $\displaystyle\int \csc x \, dx = \ln|\csc x + \cot x|$,

14. $\displaystyle\int \arcsin\tfrac{x}{a} dx = \arcsin\tfrac{x}{a} + \sqrt{a^2 - x^2}, \quad a > 0$,

Calculus Cont.

15. $\int \arccos\frac{x}{a}dx = \arccos\frac{x}{a} - \sqrt{a^2-x^2}, \quad a>0,$

16. $\int \arctan\frac{x}{a}dx = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2), \quad a>0,$

17. $\int \sin^2(ax)dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

18. $\int \cos^2(ax)dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

19. $\int \sec^2 x\,dx = \tan x,$

20. $\int \csc^2 x\,dx = -\cot x,$

21. $\displaystyle\int \sin^n x\,dx = -\frac{\sin^{n-1}x\cos x}{n} + \frac{n-1}{n}\int \sin^{n-2}x\,dx,$

22. $\displaystyle\int \cos^n x\,dx = \frac{\cos^{n-1}x\sin x}{n} + \frac{n-1}{n}\int \cos^{n-2}x\,dx,$

23. $\displaystyle\int \tan^n x\,dx = \frac{\tan^{n-1}x}{n-1} - \int \tan^{n-2}x\,dx, \quad n\neq 1,$

24. $\displaystyle\int \cot^n x\,dx = -\frac{\cot^{n-1}x}{n-1} - \int \cot^{n-2}x\,dx, \quad n\neq 1,$

25. $\displaystyle\int \sec^n x\,dx = \frac{\tan x\sec^{n-1}x}{n-1} + \frac{n-2}{n-1}\int \sec^{n-2}x\,dx, \quad n\neq 1,$

26. $\displaystyle\int \csc^n x\,dx = -\frac{\cot x\csc^{n-1}x}{n-1} + \frac{n-2}{n-1}\int \csc^{n-2}x\,dx, \quad n\neq 1,$

27. $\int \sinh x\,dx = \cosh x,$

28. $\int \cosh x\,dx = \sinh x,$

29. $\int \tanh x\,dx = \ln|\cosh x|,$

30. $\int \coth x\,dx = \ln|\sinh x|,$

31. $\int \operatorname{sech} x\,dx = \arctan\sinh x,$

32. $\int \operatorname{csch} x\,dx = \ln\left|\tanh\frac{x}{2}\right|,$

33. $\int \sinh^2 x\,dx = \frac{1}{4}\sinh(2x) - \frac{1}{2}x,$

34. $\int \cosh^2 x\,dx = \frac{1}{4}\sinh(2x) + \frac{1}{2}x,$

35. $\int \operatorname{sech}^2 x\,dx = \tanh x,$

36. $\int \operatorname{arcsinh}\frac{x}{a}dx = x\operatorname{arcsinh}\frac{x}{a} - \sqrt{x^2+a^2}, \quad a>0,$

37. $\int \operatorname{arctanh}\frac{x}{a}dx = x\operatorname{arctanh}\frac{x}{a} + \frac{a}{2}\ln|a^2-x^2|,$

38. $\displaystyle\int \operatorname{arccosh}\frac{x}{a}dx = \begin{cases} x\operatorname{arccosh}\dfrac{x}{a} - \sqrt{x^2+a^2}, & \text{if } \operatorname{arccosh}\frac{x}{a} > 0 \text{ and } a>0, \\ x\operatorname{arccosh}\dfrac{x}{a} + \sqrt{x^2+a^2}, & \text{if } \operatorname{arccosh}\frac{x}{a} < 0 \text{ and } a>0, \end{cases}$

39. $\displaystyle\int \frac{dx}{\sqrt{a^2+x^2}} = \ln\left(x+\sqrt{a^2+x^2}\right), \quad a>0,$

40. $\displaystyle\int \frac{dx}{a^2+x^2} = \frac{1}{a}\arctan\frac{x}{a}, \quad a>0,$

41. $\int \sqrt{a^2-x^2}\,dx = \frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a}, \quad a>0,$

42. $\int (a^2-x^2)^{3/2}dx = \frac{x}{8}(5a^2-2x^2)\sqrt{a^2-x^2} + \frac{3a^4}{8}\arcsin\frac{x}{a}, \quad a>0,$

43. $\displaystyle\int \frac{dx}{\sqrt{a^2-x^2}} = \arcsin\frac{x}{a}, \quad a>0,$

44. $\displaystyle\int \frac{dx}{a^2-x^2} = \frac{1}{2a}\ln\left|\frac{a+x}{a-x}\right|,$

45. $\displaystyle\int \frac{dx}{(a^2-x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2-x^2}},$

46. $\int \sqrt{a^2\pm x^2}\,dx = \frac{x}{2}\sqrt{a^2\pm x^2} \pm \frac{a^2}{2}\ln\left|x+\sqrt{a^2\pm x^2}\right|,$

47. $\displaystyle\int \frac{dx}{\sqrt{x^2-a^2}} = \ln\left|x+\sqrt{x^2-a^2}\right|, \quad a>0,$

48. $\displaystyle\int \frac{dx}{ax^2+bx} = \frac{1}{a}\ln\left|\frac{x}{a+bx}\right|,$

49. $\displaystyle\int x\sqrt{a+bx}\,dx = \frac{2(3bx-2a)(a+bx)^{3/2}}{15b^2},$

50. $\displaystyle\int \frac{\sqrt{a+bx}}{x}dx = 2\sqrt{a+bx} + a\int \frac{1}{x\sqrt{a+bx}}dx,$

51. $\displaystyle\int \frac{x}{\sqrt{a+bx}}dx = \frac{1}{\sqrt{2}}\ln\left|\frac{\sqrt{a+bx}-\sqrt{a}}{\sqrt{a+bx}+\sqrt{a}}\right|, \quad a>0,$

52. $\displaystyle\int \frac{\sqrt{a^2-x^2}}{x}dx = \sqrt{a^2-x^2} - a\ln\left|\frac{a+\sqrt{a^2-x^2}}{x}\right|,$

53. $\int x\sqrt{a^2-x^2}\,dx = -\frac{1}{3}(a^2-x^2)^{3/2},$

54. $\int x^2\sqrt{a^2-x^2}\,dx = \frac{x}{8}(2x^2-a^2)\sqrt{a^2-x^2} + \frac{a^4}{8}\arcsin\frac{x}{a}, \quad a>0,$

55. $\displaystyle\int \frac{dx}{\sqrt{a^2-x^2}} = -\frac{1}{a}\ln\left|\frac{a+\sqrt{a^2-x^2}}{x}\right|,$

56. $\displaystyle\int \frac{x\,dx}{\sqrt{a^2-x^2}} = -\sqrt{a^2-x^2},$

57. $\displaystyle\int \frac{x^2\,dx}{\sqrt{a^2-x^2}} = -\frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a,} \quad a>0,$

58. $\displaystyle\int \frac{\sqrt{a^2+x^2}}{x}dx = \sqrt{a^2+x^2} - a\ln\left|\frac{a+\sqrt{a^2+x^2}}{x}\right|,$

59. $\displaystyle\int \frac{\sqrt{x^2-a^2}}{x}dx = \sqrt{x^2-a^2} - a\arccos\frac{a}{|x|}, \quad a>0,$

60. $\int x\sqrt{x^2\pm a^2}\,dx = \frac{1}{3}(x^2\pm a^2)^{3/2},$

61. $\displaystyle\int \frac{dx}{x\sqrt{x^2+a^2}} = \frac{1}{a}\ln\left|\frac{x}{a+\sqrt{a^2+x^2}}\right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

62. $\displaystyle\int \frac{dx}{x\sqrt{x^2-a^2}} = \tfrac{1}{a}\arccos\tfrac{a}{|x|}, \quad a>0,$

63. $\displaystyle\int \frac{dx}{x^2\sqrt{x^2\pm a^2}} = \mp\frac{\sqrt{x^2\pm a^2}}{a^2x},$

64. $\displaystyle\int \frac{x\,dx}{\sqrt{x^2\pm a^2}} = \sqrt{x^2\pm a^2},$

65. $\displaystyle\int \frac{\sqrt{x^2\pm a^2}}{x^4}\,dx = \mp\frac{(x^2+a^2)^{3/2}}{3a^2x^3},$

66. $\displaystyle\int \frac{dx}{ax^2+bx+c} = \begin{cases} \dfrac{1}{\sqrt{b^2-4ac}}\ln\left|\dfrac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|, & \text{if } b^2>4ac,\\[2ex] \dfrac{2}{\sqrt{4ac-b^2}}\arctan\dfrac{2ax+b}{\sqrt{4ac-b^2}}, & \text{if } b^2<4ac,\end{cases}$

67. $\displaystyle\int \frac{dx}{\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|, & \text{if } a>0,\\[2ex] \dfrac{1}{\sqrt{-a}}\arcsin\dfrac{-2ax-b}{\sqrt{b^2-4ac}}, & \text{if } a<0,\end{cases}$

68. $\displaystyle\int \sqrt{ax^2+bx+c}\,dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ax-b^2}{8a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$

69. $\displaystyle\int \frac{x\,dx}{\sqrt{ax^2+bx+c}} = \frac{\sqrt{ax^2+bx+c}}{a} - \frac{b}{2a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$

70. $\displaystyle\int \frac{dx}{x\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{-1}{\sqrt{c}}\ln\left|\dfrac{2\sqrt{c}\sqrt{ax^2+bx+c}+bx+2c}{x}\right|, & \text{if } c>0,\\[2ex] \dfrac{1}{\sqrt{-c}}\arcsin\dfrac{bx+2c}{|x|\sqrt{b^2-4ac}}, & \text{if } c<0,\end{cases}$

71. $\displaystyle\int x^3\sqrt{x^2+a^2}\,dx = (\tfrac{1}{3}x^2-\tfrac{2}{15}a^2)(x^2+a^2)^{3/2},$

72. $\displaystyle\int x^n\sin(ax)\,dx = -\tfrac{1}{a}x^n\cos(ax)+\tfrac{n}{a}\int x^{n-1}\cos(ax)\,dx,$

73. $\displaystyle\int x^n\cos(ax)\,dx = \tfrac{1}{a}x^n\sin(ax)-\tfrac{n}{a}\int x^{n-1}\sin(ax)\,dx,$

74. $\displaystyle\int x^ne^{ax}\,dx = \frac{x^ne^{ax}}{a}-\tfrac{n}{a}\int x^{n-1}e^{ax}\,dx,$

75. $\displaystyle\int x^n\ln(ax)\,dx = x^{n+1}\left(\frac{\ln(ax)}{n+1}-\frac{1}{(n+1)^2}\right),$

76. $\displaystyle\int x^n(\ln ax)^m\,dx = \frac{x^{n+1}}{n+1}(\ln ax)^m - \frac{m}{n+1}\int x^n(\ln ax)^{m-1}\,dx.$

$$\begin{array}{lclcl}
x^1 = & x^{\underline{1}} & = & x^{\overline{1}} \\
x^2 = & x^{\underline{2}}+x^{\underline{1}} & = & x^{\overline{2}}-x^{\overline{1}} \\
x^3 = & x^{\underline{3}}+3x^{\underline{2}}+x^{\underline{1}} & = & x^{\overline{3}}-3x^{\overline{2}}+x^{\overline{1}} \\
x^4 = & x^{\underline{4}}+6x^{\underline{3}}+7x^{\underline{2}}+x^{\underline{1}} & = & x^{\overline{4}}-6x^{\overline{3}}+7x^{\overline{2}}-x^{\overline{1}} \\
x^5 = & x^{\underline{5}}+15x^{\underline{4}}+25x^{\underline{3}}+10x^{\underline{2}}+x^{\underline{1}} & = & x^{\underline{5}}-15x^{\overline{4}}+25x^{\overline{3}}-10x^{\overline{2}}+x^{\overline{1}}
\end{array}$$

$$\begin{array}{lclc}
x^{\overline{1}} = & x^1 & x^{\underline{1}} = & x^1 \\
x^{\overline{2}} = & x^2+x^1 & x^{\underline{2}} = & x^2-x^1 \\
x^{\overline{3}} = & x^3+3x^2+2x^1 & x^{\underline{3}} = & x^3-3x^2+2x^1 \\
x^{\overline{4}} = & x^4+6x^3+11x^2+6x^1 & x^{\underline{4}} = & x^4-6x^3+11x^2-6x^1 \\
x^{\overline{5}} = & x^5+10x^4+35x^3+50x^2+24x^1 & x^{\underline{5}} = & x^5-10x^4+35x^3-50x^2+24x^1
\end{array}$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1)-f(x),$$
$$\mathrm{E}\,f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x)+C.$$
$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$\Delta(cu) = c\Delta u, \qquad \Delta(u+v) = \Delta u+\Delta v,$

$\Delta(uv) = u\Delta v+\mathrm{E}\,v\Delta u,$

$\Delta(x^{\underline{n}}) = nx^{\underline{n-1}},$

$\Delta(H_x) = x^{\underline{-1}}, \qquad \Delta(2^x) = 2^x,$

$\Delta(c^x) = (c-1)c^x, \qquad \Delta\binom{x}{m} = \binom{x}{m-1}.$

Sums:

$\sum cu\,\delta x = c\sum u\,\delta x,$

$\sum(u+v)\,\delta x = \sum u\,\delta x+\sum v\,\delta x,$

$\sum u\Delta v\,\delta x = uv-\sum \mathrm{E}\,v\Delta u\,\delta x,$

$\sum x^{\underline{n}}\,\delta x = \frac{x^{\underline{n+1}}}{m+1}, \qquad \sum x^{\underline{-1}}\,\delta x = H_x,$

$\sum c^x\,\delta x = \frac{c^x}{c-1}, \qquad \sum\binom{x}{m}\,\delta x = \binom{x}{m+1}.$

Falling Factorial Powers:

$x^{\underline{n}} = x(x-1)\cdots(x-n+1), \quad n>0,$

$x^{\underline{0}} = 1,$

$x^{\underline{n}} = \dfrac{1}{(x+1)\cdots(x+|n|)}, \quad n<0,$

$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$

Rising Factorial Powers:

$x^{\overline{n}} = x(x+1)\cdots(x+n-1), \quad n>0,$

$x^{\overline{0}} = 1,$

$x^{\overline{n}} = \dfrac{1}{(x-1)\cdots(x-|n|)}, \quad n<0,$

$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$

Conversion:

$$\begin{aligned} x^{\underline{n}} &= (-1)^n(-x)^{\overline{n}} = (x-n+1)^{\overline{n}} \\ &= 1/(x+1)^{\overline{-n}}, \\ x^{\overline{n}} &= (-1)^n(-x)^{\underline{n}} = (x+n-1)^{\underline{n}} \\ &= 1/(x-1)^{\underline{-n}}, \end{aligned}$$

$$x^n = \sum_{k=1}^n \left\{ {n \atop k} \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ {n \atop k} \right\} (-1)^{n-k}x^{\overline{k}},$$

$$x^{\underline{n}} = \sum_{k=1}^n \left[{n \atop k} \right] (-1)^{n-k}x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \left[{n \atop k} \right] x^k.$$

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{i=0}^{\infty} x^i,$$

$$\frac{1}{1-cx} = 1 + cx + c^2x^2 + c^3x^3 + \cdots = \sum_{i=0}^{\infty} c^i x^i,$$

$$\frac{1}{1-x^n} = 1 + x^n + x^{2n} + x^{3n} + \cdots = \sum_{i=0}^{\infty} x^{ni},$$

$$\frac{x}{(1-x)^2} = x + 2x^2 + 3x^3 + 4x^4 + \cdots = \sum_{i=0}^{\infty} i x^i,$$

$$x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) = x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots = \sum_{i=0}^{\infty} i^n x^i,$$

$$e^x = 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

$$\ln(1+x) = x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i},$$

$$\ln \frac{1}{1-x} = x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots = \sum_{i=1}^{\infty} \frac{x^i}{i},$$

$$\sin x = x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$$

$$\cos x = 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},$$

$$\tan^{-1} x = x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},$$

$$(1+x)^n = 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{n}{i} x^i,$$

$$\frac{1}{(1-x)^{n+1}} = 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,$$

$$\frac{x}{e^x - 1} = 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},$$

$$\frac{1}{2x}(1 - \sqrt{1-4x}) = 1 + x + 2x^2 + 5x^3 + \cdots = \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}} = 1 + x + 2x^2 + 6x^3 + \cdots = \sum_{i=0}^{\infty} \binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,$$

$$\frac{1}{1-x} \ln \frac{1}{1-x} = x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots = \sum_{i=1}^{\infty} H_i x^i,$$

$$\frac{1}{2}\left(\ln \frac{1}{1-x}\right)^2 = \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots = \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i},$$

$$\frac{x}{1-x-x^2} = x + x^2 + 2x^3 + 3x^4 + \cdots = \sum_{i=0}^{\infty} F_i x^i,$$

$$\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} = F_n x + F_{2n} x^2 + F_{3n} x^3 + \cdots = \sum_{i=0}^{\infty} F_{ni} x^i.$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y) \sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$xA'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x)\,dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker

Theoretical Computer Science Cheat Sheet

Series

Expansions:

$$\frac{1}{(1-x)^{n+1}} \ln \frac{1}{1-x} = \sum_{i=0}^{\infty} (H_{n+i} - H_n) \binom{n+i}{i} x^i, \qquad \left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty} \left\{ {i \atop n} \right\} x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty} \left[{n \atop i} \right] x^i, \qquad (e^x - 1)^n = \sum_{i=0}^{\infty} \left\{ {i \atop n} \right\} \frac{n! x^i}{i!},$$

$$\left(\ln \frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty} \left[{i \atop n} \right] \frac{n! x^i}{i!}, \qquad x \cot x = \sum_{i=0}^{\infty} \frac{(-4)^i B_{2i} x^{2i}}{(2i)!},$$

$$\tan x = \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i}(2^{2i}-1) B_{2i} x^{2i-1}}{(2i)!}, \qquad \zeta(x) = \sum_{i=1}^{\infty} \frac{1}{i^x},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty} \frac{\mu(i)}{i^x}, \qquad \frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty} \frac{\phi(i)}{i^x},$$

$$\zeta(x) = \prod_p \frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty} \frac{d(i)}{x^i} \quad \text{where } d(n) = \textstyle\sum_{d|n} 1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty} \frac{S(i)}{x^i} \quad \text{where } S(n) = \textstyle\sum_{d|n} d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!} \pi^{2n}, \quad n \in \mathbb{N},$$

$$\frac{x}{\sin x} = \sum_{i=0}^{\infty} (-1)^{i-1} \frac{(4^i - 2) B_{2i} x^{2i}}{(2i)!},$$

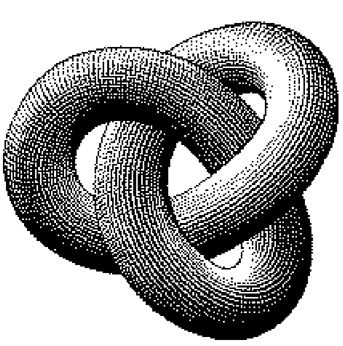
$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty} \frac{n(2i+n-1)!}{i!(n+i)!} x^i,$$

$$e^x \sin x = \sum_{i=1}^{\infty} \frac{2^{i/2} \sin \frac{i\pi}{4}}{i!} x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty} \frac{(4i)!}{16^i \sqrt{2}(2i)!(2i+1)!} x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i+1)(2i+1)!} x^{2i}.$$

Escher's Knot



Stieltjes Integration

If G is continuous in the interval $[a,b]$ and F is nondecreasing then

$$\int_a^b G(x)\,dF(x)$$

exists. If $a \le b \le c$ then

$$\int_a^c G(x)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_b^c G(x)\,dF(x).$$

If the integrals involved exist

$$\int_a^b \big(G(x) + H(x)\big)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_a^b H(x)\,dF(x),$$

$$\int_a^b G(x)\,d\big(F(x) + H(x)\big) = \int_a^b G(x)\,dF(x) + \int_a^b G(x)\,dH(x),$$

$$\int_a^b c \cdot G(x)\,dF(x) = \int_a^b G(x)\,d\big(c \cdot F(x)\big) = c \int_a^b G(x)\,dF(x),$$

$$\int_a^b G(x)\,dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x)\,dG(x).$$

If the integrals involved exist, and F possesses a derivative F' at every point in $[a,b]$ then

$$\int_a^b G(x)\,dF(x) = \int_a^b G(x)F'(x)\,dx.$$

Cramer's Rule

If we have equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ \vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \quad& \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

Let $A = (a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A \ne 0$. Let A_i be A with column i replaced by B. Then

$$x_i = \frac{\det A_i}{\det A}.$$

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.
– William Blake (The Marriage of Heaven and Hell)

00	47	18	76	29	93	85	34	61	52
86	11	57	28	70	39	94	45	02	63
95	80	22	67	38	71	49	56	13	04
59	96	81	33	07	48	72	60	24	15
73	69	90	82	44	17	58	01	35	26
68	74	09	91	83	55	27	12	46	30
37	08	75	19	92	84	66	23	50	41
14	25	36	40	51	62	03	77	88	99
21	32	43	54	65	06	10	89	97	78
42	53	64	05	16	20	31	98	79	87

The Fibonacci number system: Every integer n has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where $k_i \ge k_{i+1} + 2$ for all i, $1 \le i < m$ and $k_m \ge 2$.

Fibonacci Numbers

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$

Definitions:

$$F_i = F_{i-1} + F_{i-2}, \quad F_0 = F_1 = 1,$$

$$F_{-i} = (-1)^{i-1} F_i,$$

$$F_i = \tfrac{1}{\sqrt{5}} \left(\phi^i - \hat{\phi}^i\right),$$

Cassini's identity: for $i > 0$:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n,$$

$$F_{2n} = F_n F_{n+1} + F_{n-1} F_n.$$

Calculation by matrices:

$$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$