



All-in at the River

Standard Code Library

Shanghai Jiao Tong University

Desprado2 fstqwq AntiLeaf

Last Compiled: 21:55 Jun 11, 2024

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)	1	
1.1.4 多项式操作	2	
1.1.5 多点求值应用: $O(\sqrt{n} \log^2 n)$ 快速求阶乘	4	
1.1.6 多项式快速插值	4	
1.1.7 Bostan-Mori (求多项式分式第 n 项)	5	
1.1.8 快速线性递推 $O(k \log k \log n)$	6	
1.1.9 多项式复合逆	6	
1.1.10 多项式复合	8	
1.1.11 分治 FFT	9	
1.1.12 半在线卷积	9	
1.2 插值	10	
1.2.1 牛顿插值	10	
1.2.2 拉格朗日 (Lagrange) 插值	10	
1.3 FWT 快速沃尔什变换	10	
1.3.1 三行 FWT	10	
1.4 线性代数	11	
1.4.1 矩阵乘法	11	
1.4.2 高斯消元	11	
1.4.3 行列式取模	11	
1.4.4 线性基 (消成对角)	11	
1.4.5 线性代数知识	11	
1.4.6 矩阵树定理, BEST 定理	12	
1.5 $O(k^2 \log n)$ 齐次线性递推	12	
1.6 Berlekamp-Massey 最小递推式	12	
1.6.1 优化矩阵快速幂 DP	13	
1.6.2 求矩阵最小多项式	13	
1.6.3 求稀疏矩阵的行列式	13	
1.6.4 求稀疏矩阵的秩	13	
1.6.5 解稀疏方程组	13	
1.7 单纯形	13	
1.7.1 线性规划对偶原理	14	
1.8 博弈论	14	
1.8.1 SG 定理	14	
1.8.2 纳什均衡	14	
1.8.3 经典博弈	15	
1.8.4 例题	15	
1.9 自适应 Simpson 积分	15	
1.10 常见数列	15	
1.10.1 斐波那契数卢卡斯数	15	
1.10.2 伯努利数, 自然数幂次和	16	
1.10.3 分拆数	16	
1.10.4 斯特林数	16	
1.10.5 贝尔数	17	
1.10.6 欧拉数 (Eulerian Number)	17	
1.10.7 卡特兰数, 施罗德数, 默慈金数	17	
1.11 常用公式及结论	18	
1.11.1 方差	18	
1.11.2 min-max 反演	18	
1.11.3 单位根反演 (展开整除条件 $[n k]$)	18	
1.11.4 康托展开 (排列的排名)	18	
1.11.5 连通图计数	18	
1.11.6 常系数齐次线性递推求通项	18	
1.12 常用生成函数变换	18	
2 数论	19	
2.1 $O(n)$ 预处理逆元	19	
2.2 线性筛	19	
2.3 杜教筛	19	
2.4 Powerful Number 筛	19	
2.5 洲阁筛	20	
2.6 min25 筛	21	
2.7 Miller-Rabin	22	
2.8 Pollard's Rho	23	
2.9 快速阶乘算法	23	
2.10 扩展欧几里德 exgcd	23	
2.10.1 求通解的方法	24	
2.10.2 类欧几里德算法 (直线下整点个数)	24	
2.11 中国剩余定理	24	
2.11.1 ex-CRT	24	
2.12 二次剩余	24	
2.13 原根阶	24	
2.14 常用数论公式	25	
2.14.1 莫比乌斯反演	25	
2.14.2 降幂公式	25	
2.14.3 其他常用公式	25	
3 图论	26	
3.1 最小生成树	26	
3.1.1 Boruvka 算法	26	
3.1.2 动态最小生成树	26	
3.1.3 最小树形图	27	
3.1.4 Steiner Tree 斯坦纳树	28	
3.1.5 最小直径生成树	29	
3.2 最短路	30	
3.2.1 Dijkstra	30	
3.2.2 Johnson 算法 (负权图多源最短路)	30	
3.2.3 k 短路	30	
3.3 Tarjan 算法	31	
3.3.1 强连通分量	31	
3.3.2 割点点双	31	
3.3.3 桥边双	32	
3.4 欧拉回路	32	
3.5 仙人掌	32	
3.5.1 仙人掌 DP	32	
3.6 二分图	33	
3.6.1 匈牙利	33	
3.6.2 Hopcroft-Karp 二分图匹配	33	
3.6.3 KM 二分图最大权匹配	34	
3.6.4 二分图原理	35	
3.7 一般图匹配	35	
3.7.1 高斯消元	35	
3.7.2 带花树	36	
3.7.3 带权带花树	37	
3.7.4 原理	39	
3.8 支配树	39	
3.9 2-SAT	40	
3.10 最大流	40	
3.10.1 Dinic	40	
3.10.2 ISAP	41	
3.10.3 HLPP 最高标号预流推进	42	
3.11 费用流	43	
3.11.1 SPFA 费用流	43	
3.11.2 Dijkstra 费用流	43	
3.11.3 预流推进费用流 (可处理负环) $O(nm \log C)$	44	
3.12 网络流原理	46	
3.12.1 最大流	46	
3.12.2 最小割	46	
3.12.3 上下界网络流	46	
3.12.4 常见建图方法	47	
3.12.5 例题	47	
3.13 Prufer 序列	47	
3.14 弦图相关	47	
3.15 其他	48	
3.15.1 Stoer-Wagner 全局最小割	48	

4 数据结构	49	5.6.1 广义回文树	81
4.1 线段树	49	5.7 Manacher 马拉车	82
4.1.1 非递归线段树	49	5.8 字符串原理	83
4.1.2 线段树维护矩形并	49		
4.1.3 历史和	50		
4.2 陈丹琦分治	51		
4.2.1 动态图连通性 (分治并查集)	51		
4.2.2 四维偏序	52		
4.3 整体二分	52		
4.4 平衡树	53		
4.4.1 Treap	53		
4.4.2 无旋 Treap / 可持久化 Treap	54		
4.4.3 Splay	55		
4.5 树链剖分	55		
4.5.1 动态树形 DP (最大权独立集)	55		
4.6 树分治	57		
4.6.1 动态树分治	57		
4.6.2 紫荆花之恋	58		
4.7 LCT动态树	61		
4.7.1 不换根 (弹飞绵羊)	61		
4.7.2 换根/维护生成树	61		
4.7.3 维护子树信息	63		
4.7.4 模板题: 动态QTREE4	64		
4.8 K-D树	66		
4.8.1 动态 K-D 树 (定期重构)	66		
4.9 LCA 最近公共祖先	67		
4.9.1 Tarjan LCA $O(n + m)$	67		
4.10 虚树	67		
4.11 长链剖分	69		
4.11.1 梯子剖分	69		
4.12 堆	70		
4.12.1 左偏树	70		
4.12.2 二叉堆	70		
4.13 莫队	70		
4.13.1 莫队二次离线	70		
4.13.2 带修莫队在线化 $O(n^{\frac{5}{3}})$	72		
4.13.3 莫队二次离线在线化 $O((n + m)\sqrt{n})$	72		
4.14 常见根号思路	73		
5 字符串	74		
5.1 KMP	74		
5.1.1 ex-KMP	74		
5.2 AC 自动机	74		
5.3 后缀数组	74		
5.3.1 倍增	74		
5.3.2 SA-IS	75		
5.3.3 SAMSA	76		
5.4 后缀平衡树	77		
5.5 后缀自动机	77		
5.5.1 广义后缀自动机	77		
5.5.2 区间本质不同子串计数	77		
5.6 回文树	80		
		5.6.1 广义回文树	81
		5.7 Manacher 马拉车	82
		5.8 字符串原理	83
6 动态规划	84		
6.1 决策单调性 $O(n \log n)$	84		
6.2 例题	84		
6.2.1 103388A Assigning Prizes 容斥	84		
7 计算几何	86		
7.1 Delaunay 三角剖分	86		
7.2 最近点对	88		
8 杂项	89		
8.1 $O(1)$ 快速乘	89		
8.2 Kahan求和算法 (减少浮点数累加的误差)	89		
8.3 Python Decimal	89		
8.4 $O(n^2)$ 高精度	89		
8.5 笛卡尔树	92		
8.6 GarsiaWachs 算法 ($O(n \log n)$ 合并石子)	92		
8.7 常用 NTT 素数及原根	92		
8.8 xorshift	92		
8.9 枚举子集	93		
8.10 STL	93		
8.10.1 vector	93		
8.10.2 list	93		
8.10.3 unordered_set/map	93		
8.10.4 自定义 Hash	93		
8.11 Public Based DataStructure (PB_DS)	93		
8.11.1 哈希表	93		
8.11.2 堆	93		
8.11.3 平衡树	94		
8.12 rope	94		
8.13 其他 C++ 相关	94		
8.13.1 <cmath>	94		
8.13.2 <algorithm>	94		
8.13.3 std::tuple	94		
8.13.4 <complex>	94		
8.14 一些游戏	94		
8.14.1 德州扑克	94		
8.14.2 炉石传说	97		
8.15 OEIS	97		
8.15.1 计数相关	97		
8.15.2 线性递推数列	98		
8.15.3 数论相关	98		
8.15.4 其他	98		
8.16 编译选项	98		
8.17 附录: VScode 相关	99		
8.17.1 插件	99		
8.17.2 设置选项	99		
8.17.3 快捷键	99		
8.18 附录: 骂人的艺术—梁实秋	100		
8.19 附录: Cheat Sheet	100		

1 数学

1.1 多项式

1.1.1 FFT

```

1 using cp = complex<double>;
2 const double PI = acos(-1.0);
3
4 vector<cp> omega[25];
5
6 void fft_init(int n) {
7     for (int k = 2, d = 0; k <= n; k *= 2, d++) {
8         omega[d].resize(k + 1);
9         for (int i = 0; i <= k; i++)
10            omega[d][i] = polar(1.0, 2 * PI * i / k);
11    }
12}
13
14 void fft(cp* a, int n, int t) {
15     for (int i = 1, j = 0; i < n - 1; i++) {
16         int k = n;
17         do
18             j ^= (k >>= 1);
19         while (j < k);
20
21         if (i < j)
22             swap(a[i], a[j]);
23    }
24
25     for (int k = 1, d = 0; k < n; k *= 2, d++) {
26         for (int i = 0; i < n; i += k * 2)
27             for (int j = 0; j < k; j++) {
28                 cp w = omega[d][t > 0 ? j : k * 2 - j];
29                 cp u = a[i + j], v = w * a[i + j + k];
30                 a[i + j] = u + v;
31                 a[i + j + k] = u - v;
32            }
33
34     if (t < 0)
35         for (int i = 0; i < n; i++)
36             a[i] /= n;
37}

```

```

22     for (int i = 2; i < n; i++)
23         inv[i] = (ll)(p - p / i) * inv[p % i] % p;
24    }
25
26 // 传入的必须是 [0, p) 范围内, 不能有负的, 不然会溢出
27 // 不能保证就把 d == 16 改成 d % 8 == 0 之类
28 void ntt(int *c, int n, int t) {
29     static ull a[MAXN];
30     for (int i = 0; i < n; i++) a[i] = c[i];
31
32     for (int i = 1, j = 0; i < n - 1; i++) {
33         int k = n;
34         do
35             j ^= (k >>= 1);
36         while (j < k);
37
38         if (i < j)
39             swap(a[i], a[j]);
40    }
41
42     for (int k = 1, d = 0; k < n; k *= 2, d++) {
43         if (d == 16)
44             for (int i = 0; i < n; i++)
45                 a[i] %= p;
46
47         for (int i = 0; i < n; i += k * 2)
48             for (int j = 0; j < k; j++) {
49                 int w = omega[d][t > 0 ? j : k * 2 - j];
50                 ull u = a[i + j], v = w * a[i + j + k];
51                 a[i + j] = u + v;
52                 a[i + j + k] = u - v + p;
53            }
54
55         if (t > 0) {
56             for (int i = 0; i < n; i++)
57                 c[i] = a[i] % p;
58        }
59
60     else {
61         int inv = qpow(n, p - 2);
62         for (int i = 0; i < n; i++)
63             c[i] = a[i] * inv % p;
64    }
65}

```

1.1.2 NTT

```

1 using ll = long long;
2 using ull = unsigned long long;
3
4 int inv[MAXN]; // 逆元, 如果需要积分就顺便预处理出来
5 poly omega[25]; // 单位根
6
7 // n 是 DFT 的最大长度
8 // 例如如果最多有两个长为 k 的多项式相乘, 或者求逆的长度为
9 // → k, 那么 n 需要 >= 2k
10
11 void ntt_init(int n) {
12     for (int k = 2, d = 0; k <= n; k *= 2, d++) {
13         omega[d].resize(k + 1);
14
15         int wn = qpow(3, (p - 1) / k), tmp = 1;
16         for (int i = 0; i <= k; i++) {
17             omega[d][i] = tmp;
18             tmp = (long long)tmp * wn % p;
19        }
20
21     inv[1] = 1;
22}

```

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.

```

1 void dft(cp* a, cp* b, int n) {
2     static cp c[MAXN];
3     for (int i = 0; i < n; i++)
4         c[i] = cp(a[i].real(), b[i].real());
5
6     fft(c, n, 1);
7     for (int i = 0; i < n; i++) {
8         int j = (n - i) & (n - 1);
9         a[i] = (c[i] + conj(c[j])) * 0.5;
10        b[i] = (c[i] - conj(c[j])) * -0.5i;
11    }
12 }
13
14 void idft(cp* a, cp* b, int n) {
15     static cp c[MAXN];
16     for (int i = 0; i < n; i++)
17         c[i] = a[i] + 1i * b[i];
18
19     fft(c, n, -1);
20     for (int i = 0; i < n; i++) {
21         a[i] = c[i].real();
22         b[i] = c[i].imag();
23     }
24 }
25
26 vector<int> multiply(const vector<int>& u, const
27     → vector<int>& v, int mod) {
28     static cp a[2][MAXN], b[2][MAXN], c[3][MAXN];
29
30     int base = ceil(sqrt(mod));
31     int n = (int)u.size(), m = (int)v.size();
32
33     int fft_n = 1;
34     while (fft_n < n + m - 1)
35         fft_n *= 2;
36
37     for (int i = 0; i < 2; i++) {
38         fill(a[i], a[i] + fft_n, 0);
39         fill(b[i], b[i] + fft_n, 0);
40     }
41     for (int i = 0; i < 3; i++)
42         fill(c[i], c[i] + fft_n, 0);
43
44     for (int i = 0; i < n; i++) {
45         a[0][i] = (u[i] % mod) % base;
46         a[1][i] = (u[i] % mod) / base;
47     }
48
49     for (int i = 0; i < m; i++) {
50         b[0][i] = (v[i] % mod) % base;
51         b[1][i] = (v[i] % mod) / base;
52     }
53
54     dft(a[0], a[1], fft_n);
55     dft(b[0], b[1], fft_n);
56
57     for (int i = 0; i < fft_n; i++) {
58         c[0][i] = a[0][i] * b[0][i];
59         c[1][i] = a[0][i] * b[1][i] + a[1][i] * b[0]
60             → [i];
61         c[2][i] = a[1][i] * b[1][i];
62     }
63
64     fft(c[1], fft_n, -1);
65     idft(c[0], c[2], fft_n);
66
67     int base2 = base * base % mod;
68     vector<int> ans(n + m - 1);
69
70     for (int i = 0; i < n + m - 1; i++)
71         ans[i] = ((ll)(c[0][i].real() + 0.5) +
72             (ll)(c[1][i].real() + 0.5) % mod * base +
73             (ll)(c[2][i].real() + 0.5) % mod * base2) %
74                 → mod;
75
76     return ans;
77 }
```

1.1.4 多项式操作

```

1 int get_ntt_n(int n) { // not inclusive
2     int ntt_n = 1;
3     while (ntt_n < n)
4         ntt_n *= 2;
5     return ntt_n;
6 }
7
8 using poly = vector<int>;
9
10 // u, v 长度要相同, 返回长度是两倍
11 poly poly_calc(const poly& u, const poly& v,
12     → function<int(int, int)> op) {
13     static int a[MAXN], b[MAXN], c[MAXN];
14
15     int n = (int)u.size();
16
17     memcpy(a, u.data(), sizeof(int) * n);
18     fill(a + n, a + n * 2, 0);
19     memcpy(b, v.data(), sizeof(int) * n);
20     fill(b + n, b + n * 2, 0);
21
22     ntt(a, n * 2, 1);
23     ntt(b, n * 2, 1);
24
25     for (int i = 0; i < n * 2; i++)
26         c[i] = op(a[i], b[i]);
27
28     ntt(c, n * 2, -1);
29
30     return poly(c, c + n * 2);
31 }
32
33 // 乘法, 返回长度是两倍
34 poly poly_mul(const poly& u, const poly& v) {
35     return poly_calc(u, v, [](int a, int b) { return
36         → (ll)a * b % p; });
37 }
38
39 // 求逆, 返回长度不变
40 poly poly_inv(const poly& a) {
41     poly c{qpow(a[0], p - 2)}; // 常数项一般都是 1
42
43     for (int k = 2; k <= (int)a.size(); k *= 2) {
44         c.resize(k);
45
46         poly b(a.begin(), a.begin() + k);
47         c = poly_calc(b, c, [](int bi, int ci) {
48             → return ((2 - (ll)bi * ci) % p + p) * ci %
49             → p;
50         });
51         memset(c.data() + k, 0, sizeof(int) * k);
52
53         c.resize(a.size());
54     }
55 }
```

```

55 // 开根, 返回长度不变
56 poly poly_sqrt(const poly& a) {
57     poly c{1}; // 常数项不是 1 的话要写二次剩余
58
59     for (int k = 2; k <= (int)a.size(); k *= 2) {
60         c.resize(k);
61
62         poly b(a.begin(), a.begin() + k);
63         b = poly_mul(b, poly_inv(c));
64
65         for (int i = 0; i < k; i++)
66             c[i] = (ll)(c[i] + b[i]) * inv_2 % p; // ← inv_2 是 2 的逆元
67     }
68
69     c.resize(a.size());
70     return c;
71 }
72
73 // 求导
74 poly poly_derivative(const poly& a) {
75     poly c(a.size());
76     for (int i = 1; i < (int)a.size(); i++)
77         c[i - 1] = (ll)a[i] * i % p;
78     return c;
79 }
80
81 // 不定积分, 最好预处理逆元
82 poly poly_integrate(const poly& a) {
83     poly c(a.size());
84     for (int i = 1; i < (int)a.size(); i++)
85         c[i] = (ll)a[i - 1] * inv[i] % p;
86     return c;
87 }
88
89 // ln, 常数项不能为 0, 返回长度不变
90 poly poly_ln(const poly& a) {
91     poly c = poly_mul(poly_derivative(a), poly_inv(a));
92     c.resize(a.size());
93     return poly_integrate(c);
94 }
95
96 // exp, 常数项必须是 0, 返回长度不变
97 // 常数很大并且总代码很长, 一般可以改用分治 FFT
98 // 依据: 设  $G(x) = \exp F(x)$ , 则  $g_i = \frac{1}{i} \sum_{k=1}^{i-1} g_{i-k} k f_k$ 
99 poly poly_exp(const poly& a) {
100    poly c{1};
101
102    for (int k = 2; k <= (int)a.size(); k *= 2) {
103        c.resize(k);
104
105        poly b = poly_ln(c);
106        for (int i = 0; i < k; i++) {
107            b[i] = a[i] - b[i];
108            if (b[i] < 0)
109                b[i] += p;
110        }
111        (++b[0]) %= p;
112
113        c = poly_mul(b, c);
114        memset(c.data() + k, 0, sizeof(int) * k);
115    }
116
117    c.resize(a.size());
118    return c;
119 }
120
121 // k 次幂, 返回长度不变
122 // 注意常数项必须是 0, 一次项必须是 1, 否则需要在调用前处理一下
123 poly poly_pow(const poly& a, int k) {
124     poly c = poly_ln(a);
125     for (int i = 0; i < (int)c.size(); i++)
126         c[i] = (ll)c[i] * k % p;
127     return poly_exp(c);
128 }
129
130 // 自动判断长度的乘法
131 poly poly_auto_mul(poly a, poly b) {
132     int res_len = (int)a.size() + (int)b.size() - 1;
133     int ntt_n = get_ntt_n(res_len);
134
135     a.resize(ntt_n);
136     b.resize(ntt_n);
137
138     ntt(a.data(), ntt_n, 1);
139     ntt(b.data(), ntt_n, 1);
140
141     for (int i = 0; i < ntt_n; i++)
142         a[i] = (ll)a[i] * b[i] % p;
143
144     ntt(a.data(), ntt_n, -1);
145     a.resize(res_len);
146     return a;
147 }
148
149 // 多项式除法, a 和 b 长度可以任意
150 // 商的长度是  $n - m + 1$ , 余数的长度是  $m - 1$ 
151 poly poly_div(const poly& a, const poly& b) {
152     int n = (int)a.size(), m = (int)b.size();
153     if (n < m)
154         return {};
155
156     int ntt_n = get_ntt_n(n - m + 1);
157
158     poly f(ntt_n), g(ntt_n);
159     for (int i = 0; i < n - m + 1; i++)
160         f[i] = a[n - i - 1];
161     for (int i = 0; i < m && i < n - m + 1; i++)
162         g[i] = b[m - i - 1];
163
164     poly g_inv = poly_inv(g);
165     fill(g_inv.begin() + n - m + 1, g_inv.end(), 0);
166     poly c = poly_mul(f, g_inv);
167     c.resize(n - m + 1);
168     reverse(c.begin(), c.end());
169     return c;
170 }
171
172 // 多项式取模, a 和 b 长度可以任意, 返回 (余数, 商)
173 pair<poly, poly> poly_mod(const poly& a, const poly& b)
174     → {
175     int n = (int)a.size(), m = (int)b.size();
176     if (n < m)
177         return {a, {}};
178
179     poly d = poly_div(a, b);
180     poly c = poly_auto_mul(b, d);
181
182     poly r(m - 1);
183     for (int i = 0; i < m - 1; i++)
184         r[i] = (a[i] - c[i] + p) % p;
185     return {r, d};
186
187 // 多项式多点求值, f 是多项式, x 是询问
188 struct poly_eval {
189     poly f;
190     vector<int> x;

```

```

191     vector<poly> gs;
192     vector<int> ans;
193
194     poly_eval(const poly& f, const vector<int>& x) :
195         → f(f), x(x) {}
196
197     void pretreat(int l, int r, int o) {
198         poly& g = gs[o];
199
200         if (l == r) {
201             g = poly{p - x[l], 1};
202             return;
203         }
204
205         int mid = (l + r) / 2;
206         pretreat(l, mid, o * 2);
207         pretreat(mid + 1, r, o * 2 + 1);
208
209         if (o > 1)
210             g = poly_auto_mul(gs[o * 2], gs[o * 2 +
211             → 1]);
212
213     void solve(int l, int r, int o, const poly& f) {
214         if (l == r) {
215             ans[l] = f[0];
216             return;
217         }
218
219         int mid = (l + r) / 2;
220         solve(l, mid, o * 2, poly_mod(f, gs[o *
221             → 2]).first);
222         solve(mid + 1, r, o * 2 + 1, poly_mod(f, gs[o *
223             → 2 + 1]).first);
224
225     vector<int> operator() () {
226         int n = (int)f.size(), m = (int)x.size();
227         if (m <= n)
228             x.resize(m = n + 1);
229         else if (n < m - 1)
230             f.resize(n = m - 1);
231
232         int bit_ceil = get_ntt_n(m);
233         ntt_init(bit_ceil * 2); // 注意这里初始化了
234
235         gs.resize(2 * bit_ceil + 1);
236         pretreat(0, m - 1, 1);
237
238         ans.resize(m);
239         solve(0, m - 1, 1, f);
240         return ans;
241     }
242 };

```

1.1.5 多点求值应用: $O(\sqrt{n} \log^2 n)$ 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 $O(\sqrt{p})$.

1.1.6 多项式快速插值

问题 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.

```

1 int qx[maxn], qy[maxn];
2 int th[25][maxn * 2], ansf[maxn]; // th存的是各阶段
3                                     → 的M(x)
4
5 void pretreat2(int l, int r, int k) { // 预处理
6     static int A[maxn], B[maxn];
7     int *h = th[k] + l * 2;
8
9     if (l == r) {
10        h[0] = p - qx[l];
11        h[1] = 1;
12        return;
13    }
14
15    int mid = (l + r) / 2;
16    pretreat2(l, mid, k * 2);
17    pretreat2(mid + 1, r, k * 2 + 1);
18
19    for (int i = l; i <= r; i++) {
20        h[i] = (A[i] * h[i * 2] + B[i]) % p;
21    }
22
23    for (int i = l; i <= r; i++) {
24        A[i] = h[i];
25        B[i] = h[i * 2];
26    }
27
28    for (int i = l; i <= r; i++) {
29        h[i] = (A[i] * h[i * 2] + B[i]) % p;
30    }
31
32    for (int i = l; i <= r; i++) {
33        A[i] = h[i];
34        B[i] = h[i * 2];
35    }
36
37    for (int i = l; i <= r; i++) {
38        h[i] = (A[i] * h[i * 2] + B[i]) % p;
39    }
40
41    for (int i = l; i <= r; i++) {
42        A[i] = h[i];
43        B[i] = h[i * 2];
44    }
45
46    for (int i = l; i <= r; i++) {
47        h[i] = (A[i] * h[i * 2] + B[i]) % p;
48    }
49
50    for (int i = l; i <= r; i++) {
51        A[i] = h[i];
52        B[i] = h[i * 2];
53    }
54
55    for (int i = l; i <= r; i++) {
56        h[i] = (A[i] * h[i * 2] + B[i]) % p;
57    }
58
59    for (int i = l; i <= r; i++) {
60        A[i] = h[i];
61        B[i] = h[i * 2];
62    }
63
64    for (int i = l; i <= r; i++) {
65        h[i] = (A[i] * h[i * 2] + B[i]) % p;
66    }
67
68    for (int i = l; i <= r; i++) {
69        A[i] = h[i];
70        B[i] = h[i * 2];
71    }
72
73    for (int i = l; i <= r; i++) {
74        h[i] = (A[i] * h[i * 2] + B[i]) % p;
75    }
76
77    for (int i = l; i <= r; i++) {
78        A[i] = h[i];
79        B[i] = h[i * 2];
80    }
81
82    for (int i = l; i <= r; i++) {
83        h[i] = (A[i] * h[i * 2] + B[i]) % p;
84    }
85
86    for (int i = l; i <= r; i++) {
87        A[i] = h[i];
88        B[i] = h[i * 2];
89    }
90
91    for (int i = l; i <= r; i++) {
92        h[i] = (A[i] * h[i * 2] + B[i]) % p;
93    }
94
95    for (int i = l; i <= r; i++) {
96        A[i] = h[i];
97        B[i] = h[i * 2];
98    }
99
100   for (int i = l; i <= r; i++) {
101      h[i] = (A[i] * h[i * 2] + B[i]) % p;
102  }
103
104  for (int i = l; i <= r; i++) {
105      A[i] = h[i];
106      B[i] = h[i * 2];
107  }
108
109  for (int i = l; i <= r; i++) {
110      h[i] = (A[i] * h[i * 2] + B[i]) % p;
111  }
112
113  for (int i = l; i <= r; i++) {
114      A[i] = h[i];
115      B[i] = h[i * 2];
116  }
117
118  for (int i = l; i <= r; i++) {
119      h[i] = (A[i] * h[i * 2] + B[i]) % p;
120  }
121
122  for (int i = l; i <= r; i++) {
123      A[i] = h[i];
124      B[i] = h[i * 2];
125  }
126
127  for (int i = l; i <= r; i++) {
128      h[i] = (A[i] * h[i * 2] + B[i]) % p;
129  }
130
131  for (int i = l; i <= r; i++) {
132      A[i] = h[i];
133      B[i] = h[i * 2];
134  }
135
136  for (int i = l; i <= r; i++) {
137      h[i] = (A[i] * h[i * 2] + B[i]) % p;
138  }
139
140  for (int i = l; i <= r; i++) {
141      A[i] = h[i];
142      B[i] = h[i * 2];
143  }
144
145  for (int i = l; i <= r; i++) {
146      h[i] = (A[i] * h[i * 2] + B[i]) % p;
147  }
148
149  for (int i = l; i <= r; i++) {
150      A[i] = h[i];
151      B[i] = h[i * 2];
152  }
153
154  for (int i = l; i <= r; i++) {
155      h[i] = (A[i] * h[i * 2] + B[i]) % p;
156  }
157
158  for (int i = l; i <= r; i++) {
159      A[i] = h[i];
160      B[i] = h[i * 2];
161  }
162
163  for (int i = l; i <= r; i++) {
164      h[i] = (A[i] * h[i * 2] + B[i]) % p;
165  }
166
167  for (int i = l; i <= r; i++) {
168      A[i] = h[i];
169      B[i] = h[i * 2];
170  }
171
172  for (int i = l; i <= r; i++) {
173      h[i] = (A[i] * h[i * 2] + B[i]) % p;
174  }
175
176  for (int i = l; i <= r; i++) {
177      A[i] = h[i];
178      B[i] = h[i * 2];
179  }
180
181  for (int i = l; i <= r; i++) {
182      h[i] = (A[i] * h[i * 2] + B[i]) % p;
183  }
184
185  for (int i = l; i <= r; i++) {
186      A[i] = h[i];
187      B[i] = h[i * 2];
188  }
189
190  for (int i = l; i <= r; i++) {
191      h[i] = (A[i] * h[i * 2] + B[i]) % p;
192  }
193
194  for (int i = l; i <= r; i++) {
195      A[i] = h[i];
196      B[i] = h[i * 2];
197  }
198
199  for (int i = l; i <= r; i++) {
200      h[i] = (A[i] * h[i * 2] + B[i]) % p;
201  }
202
203  for (int i = l; i <= r; i++) {
204      A[i] = h[i];
205      B[i] = h[i * 2];
206  }
207
208  for (int i = l; i <= r; i++) {
209      h[i] = (A[i] * h[i * 2] + B[i]) % p;
210  }
211
212  for (int i = l; i <= r; i++) {
213      A[i] = h[i];
214      B[i] = h[i * 2];
215  }
216
217  for (int i = l; i <= r; i++) {
218      h[i] = (A[i] * h[i * 2] + B[i]) % p;
219  }
220
221  for (int i = l; i <= r; i++) {
222      A[i] = h[i];
223      B[i] = h[i * 2];
224  }
225
226  for (int i = l; i <= r; i++) {
227      h[i] = (A[i] * h[i * 2] + B[i]) % p;
228  }
229
230  for (int i = l; i <= r; i++) {
231      A[i] = h[i];
232      B[i] = h[i * 2];
233  }
234
235  for (int i = l; i <= r; i++) {
236      h[i] = (A[i] * h[i * 2] + B[i]) % p;
237  }
238
239  for (int i = l; i <= r; i++) {
240      A[i] = h[i];
241      B[i] = h[i * 2];
242  }
243
244  for (int i = l; i <= r; i++) {
245      h[i] = (A[i] * h[i * 2] + B[i]) % p;
246  }
247
248  for (int i = l; i <= r; i++) {
249      A[i] = h[i];
250      B[i] = h[i * 2];
251  }
252
253  for (int i = l; i <= r; i++) {
254      h[i] = (A[i] * h[i * 2] + B[i]) % p;
255  }
256
257  for (int i = l; i <= r; i++) {
258      A[i] = h[i];
259      B[i] = h[i * 2];
260  }
261
262  for (int i = l; i <= r; i++) {
263      h[i] = (A[i] * h[i * 2] + B[i]) % p;
264  }
265
266  for (int i = l; i <= r; i++) {
267      A[i] = h[i];
268      B[i] = h[i * 2];
269  }
270
271  for (int i = l; i <= r; i++) {
272      h[i] = (A[i] * h[i * 2] + B[i]) % p;
273  }
274
275  for (int i = l; i <= r; i++) {
276      A[i] = h[i];
277      B[i] = h[i * 2];
278  }
279
280  for (int i = l; i <= r; i++) {
281      h[i] = (A[i] * h[i * 2] + B[i]) % p;
282  }
283
284  for (int i = l; i <= r; i++) {
285      A[i] = h[i];
286      B[i] = h[i * 2];
287  }
288
289  for (int i = l; i <= r; i++) {
290      h[i] = (A[i] * h[i * 2] + B[i]) % p;
291  }
292
293  for (int i = l; i <= r; i++) {
294      A[i] = h[i];
295      B[i] = h[i * 2];
296  }
297
298  for (int i = l; i <= r; i++) {
299      h[i] = (A[i] * h[i * 2] + B[i]) % p;
300  }
301
302  for (int i = l; i <= r; i++) {
303      A[i] = h[i];
304      B[i] = h[i * 2];
305  }
306
307  for (int i = l; i <= r; i++) {
308      h[i] = (A[i] * h[i * 2] + B[i]) % p;
309  }
310
311  for (int i = l; i <= r; i++) {
312      A[i] = h[i];
313      B[i] = h[i * 2];
314  }
315
316  for (int i = l; i <= r; i++) {
317      h[i] = (A[i] * h[i * 2] + B[i]) % p;
318  }
319
320  for (int i = l; i <= r; i++) {
321      A[i] = h[i];
322      B[i] = h[i * 2];
323  }
324
325  for (int i = l; i <= r; i++) {
326      h[i] = (A[i] * h[i * 2] + B[i]) % p;
327  }
328
329  for (int i = l; i <= r; i++) {
330      A[i] = h[i];
331      B[i] = h[i * 2];
332  }
333
334  for (int i = l; i <= r; i++) {
335      h[i] = (A[i] * h[i * 2] + B[i]) % p;
336  }
337
338  for (int i = l; i <= r; i++) {
339      A[i] = h[i];
340      B[i] = h[i * 2];
341  }
342
343  for (int i = l; i <= r; i++) {
344      h[i] = (A[i] * h[i * 2] + B[i]) % p;
345  }
346
347  for (int i = l; i <= r; i++) {
348      A[i] = h[i];
349      B[i] = h[i * 2];
350  }
351
352  for (int i = l; i <= r; i++) {
353      h[i] = (A[i] * h[i * 2] + B[i]) % p;
354  }
355
356  for (int i = l; i <= r; i++) {
357      A[i] = h[i];
358      B[i] = h[i * 2];
359  }
360
361  for (int i = l; i <= r; i++) {
362      h[i] = (A[i] * h[i * 2] + B[i]) % p;
363  }
364
365  for (int i = l; i <= r; i++) {
366      A[i] = h[i];
367      B[i] = h[i * 2];
368  }
369
370  for (int i = l; i <= r; i++) {
371      h[i] = (A[i] * h[i * 2] + B[i]) % p;
372  }
373
374  for (int i = l; i <= r; i++) {
375      A[i] = h[i];
376      B[i] = h[i * 2];
377  }
378
379  for (int i = l; i <= r; i++) {
380      h[i] = (A[i] * h[i * 2] + B[i]) % p;
381  }
382
383  for (int i = l; i <= r; i++) {
384      A[i] = h[i];
385      B[i] = h[i * 2];
386  }
387
388  for (int i = l; i <= r; i++) {
389      h[i] = (A[i] * h[i * 2] + B[i]) % p;
390  }
391
392  for (int i = l; i <= r; i++) {
393      A[i] = h[i];
394      B[i] = h[i * 2];
395  }
396
397  for (int i = l; i <= r; i++) {
398      h[i] = (A[i] * h[i * 2] + B[i]) % p;
399  }
400
401  for (int i = l; i <= r; i++) {
402      A[i] = h[i];
403      B[i] = h[i * 2];
404  }
405
406  for (int i = l; i <= r; i++) {
407      h[i] = (A[i] * h[i * 2] + B[i]) % p;
408  }
409
410  for (int i = l; i <= r; i++) {
411      A[i] = h[i];
412      B[i] = h[i * 2];
413  }
414
415  for (int i = l; i <= r; i++) {
416      h[i] = (A[i] * h[i * 2] + B[i]) % p;
417  }
418
419  for (int i = l; i <= r; i++) {
420      A[i] = h[i];
421      B[i] = h[i * 2];
422  }
423
424  for (int i = l; i <= r; i++) {
425      h[i] = (A[i] * h[i * 2] + B[i]) % p;
426  }
427
428  for (int i = l; i <= r; i++) {
429      A[i] = h[i];
430      B[i] = h[i * 2];
431  }
432
433  for (int i = l; i <= r; i++) {
434      h[i] = (A[i] * h[i * 2] + B[i]) % p;
435  }
436
437  for (int i = l; i <= r; i++) {
438      A[i] = h[i];
439      B[i] = h[i * 2];
440  }
441
442  for (int i = l; i <= r; i++) {
443      h[i] = (A[i] * h[i * 2] + B[i]) % p;
444  }
445
446  for (int i = l; i <= r; i++) {
447      A[i] = h[i];
448      B[i] = h[i * 2];
449  }
450
451  for (int i = l; i <= r; i++) {
452      h[i] = (A[i] * h[i * 2] + B[i]) % p;
453  }
454
455  for (int i = l; i <= r; i++) {
456      A[i] = h[i];
457      B[i] = h[i * 2];
458  }
459
460  for (int i = l; i <= r; i++) {
461      h[i] = (A[i] * h[i * 2] + B[i]) % p;
462  }
463
464  for (int i = l; i <= r; i++) {
465      A[i] = h[i];
466      B[i] = h[i * 2];
467  }
468
469  for (int i = l; i <= r; i++) {
470      h[i] = (A[i] * h[i * 2] + B[i]) % p;
471  }
472
473  for (int i = l; i <= r; i++) {
474      A[i] = h[i];
475      B[i] = h[i * 2];
476  }
477
478  for (int i = l; i <= r; i++) {
479      h[i] = (A[i] * h[i * 2] + B[i]) % p;
480  }
481
482  for (int i = l; i <= r; i++) {
483      A[i] = h[i];
484      B[i] = h[i * 2];
485  }
486
487  for (int i = l; i <= r; i++) {
488      h[i] = (A[i] * h[i * 2] + B[i]) % p;
489  }
490
491  for (int i = l; i <= r; i++) {
492      A[i] = h[i];
493      B[i] = h[i * 2];
494  }
495
496  for (int i = l; i <= r; i++) {
497      h[i] = (A[i] * h[i * 2] + B[i]) % p;
498  }
499
500  for (int i = l; i <= r; i++) {
501      A[i] = h[i];
502      B[i] = h[i * 2];
503  }
504
505  for (int i = l; i <= r; i++) {
506      h[i] = (A[i] * h[i * 2] + B[i]) % p;
507  }
508
509  for (int i = l; i <= r; i++) {
510      A[i] = h[i];
511      B[i] = h[i * 2];
512  }
513
514  for (int i = l; i <= r; i++) {
515      h[i] = (A[i] * h[i * 2] + B[i]) % p;
516  }
517
518  for (int i = l; i <= r; i++) {
519      A[i] = h[i];
520      B[i] = h[i * 2];
521  }
522
523  for (int i = l; i <= r; i++) {
524      h[i] = (A[i] * h[i * 2] + B[i]) % p;
525  }
526
527  for (int i = l; i <= r; i++) {
528      A[i] = h[i];
529      B[i] = h[i * 2];
530  }
531
532  for (int i = l; i <= r; i++) {
533      h[i] = (A[i] * h[i * 2] + B[i]) % p;
534  }
535
536  for (int i = l; i <= r; i++) {
537      A[i] = h[i];
538      B[i] = h[i * 2];
539  }
540
541  for (int i = l; i <= r; i++) {
542      h[i] = (A[i] * h[i * 2] + B[i]) % p;
543  }
544
545  for (int i = l; i <= r; i++) {
546      A[i] = h[i];
547      B[i] = h[i * 2];
548  }
549
550  for (int i = l; i <= r; i++) {
551      h[i] = (A[i] * h[i * 2] + B[i]) % p;
552  }
553
554  for (int i = l; i <= r; i++) {
555      A[i] = h[i];
556      B[i] = h[i * 2];
557  }
558
559  for (int i = l; i <= r; i++) {
560      h[i] = (A[i] * h[i * 2] + B[i]) % p;
561  }
562
563  for (int i = l; i <= r; i++) {
564      A[i] = h[i];
565      B[i] = h[i * 2];
566  }
567
568  for (int i = l; i <= r; i++) {
569      h[i] = (A[i] * h[i * 2] + B[i]) % p;
570  }
571
572  for (int i = l; i <= r; i++) {
573      A[i] = h[i];
574      B[i] = h[i * 2];
575  }
576
577  for (int i = l; i <= r; i++) {
578      h[i] = (A[i] * h[i * 2] + B[i]) % p;
579  }
580
581  for (int i = l; i <= r; i++) {
582      A[i] = h[i];
583      B[i] = h[i * 2];
584  }
585
586  for (int i = l; i <= r; i++) {
587      h[i] = (A[i] * h[i * 2] + B[i]) % p;
588  }
589
590  for (int i = l; i <= r; i++) {
591      A[i] = h[i];
592      B[i] = h[i * 2];
593  }
594
595  for (int i = l; i <= r; i++) {
596      h[i] = (A[i] * h[i * 2] + B[i]) % p;
597  }
598
599  for (int i = l; i <= r; i++) {
600      A[i] = h[i];
601      B[i] = h[i * 2];
602  }
603
604  for (int i = l; i <= r; i++) {
605      h[i] = (A[i] * h[i * 2] + B[i]) % p;
606  }
607
608  for (int i = l; i <= r; i++) {
609      A[i] = h[i];
610      B[i] = h[i * 2];
611  }
612
613  for (int i = l; i <= r; i++) {
614      h[i] = (A[i] * h[i * 2] + B[i]) % p;
615  }
616
617  for (int i = l; i <= r; i++) {
618      A[i] = h[i];
619      B[i] = h[i * 2];
620  }
621
622  for (int i = l; i <= r; i++) {
623      h[i] = (A[i] * h[i * 2] + B[i]) % p;
624  }
625
626  for (int i = l; i <= r; i++) {
627      A[i] = h[i];
628      B[i] = h[i * 2];
629  }
630
631  for (int i = l; i <= r; i++) {
632      h[i] = (A[i] * h[i * 2] + B[i]) % p;
633  }
634
635  for (int i = l; i <= r; i++) {
636      A[i] = h[i];
637      B[i] = h[i * 2];
638  }
639
640  for (int i = l; i <= r; i++) {
641      h[i] = (A[i] * h[i * 2] + B[i]) % p;
642  }
643
644  for (int i = l; i <= r; i++) {
645      A[i] = h[i];
646      B[i] = h[i * 2];
647  }
648
649  for (int i = l; i <= r; i++) {
650      h[i] = (A[i] * h[i * 2] + B[i]) % p;
651  }
652
653  for (int i = l; i <= r; i++) {
654      A[i] = h[i];
655      B[i] = h[i * 2];
656  }
657
658  for (int i = l; i <= r; i++) {
659      h[i] = (A[i] * h[i * 2] + B[i]) % p;
660  }
661
662  for (int i = l; i <= r; i++) {
663      A[i] = h[i];
664      B[i] = h[i * 2];
665  }
666
667  for (int i = l; i <= r; i++) {
668      h[i] = (A[i] * h[i * 2] + B[i]) % p;
669  }
670
671  for (int i = l; i <= r; i++) {
672      A[i] = h[i];
673      B[i] = h[i * 2];
674  }
675
676  for (int i = l; i <= r; i++) {
677      h[i] = (A[i] * h[i * 2] + B[i]) % p;
678  }
679
680  for (int i = l; i <= r; i++) {
681      A[i] = h[i];
682      B[i] = h[i * 2];
683  }
684
685  for (int i = l; i <= r; i++) {
686      h[i] = (A[i] * h[i * 2] + B[i]) % p;
687  }
688
689  for (int i = l; i <= r; i++) {
690      A[i] = h[i];
691      B[i] = h[i * 2];
692  }
693
694  for (int i = l; i <= r; i++) {
695      h[i] = (A[i] * h[i * 2] + B[i]) % p;
696  }
697
698  for (int i = l; i <= r; i++) {
699      A[i] = h[i];
700      B[i] = h[i * 2];
701  }
702
7
```

```

12 }
13
14     int mid = (l + r) / 2;
15
16     pretreat2(l, mid, k + 1);
17     pretreat2(mid + 1, r, k + 1);
18
19     int N = 1;
20     while (N <= r - l + 1)
21         N *= 2;
22
23     int *hl = th[k + 1] + l * 2, *hr = th[k + 1] + (mid
24         -> + 1) * 2;
25
26     memset(A, 0, sizeof(int) * N);
27     memset(B, 0, sizeof(int) * N);
28
29     memcpy(A, hl, sizeof(int) * (mid - l + 2));
30     memcpy(B, hr, sizeof(int) * (r - mid + 1));
31
32     NTT(A, N, 1);
33     NTT(B, N, 1);
34
35     for (int i = 0; i < N; i++)
36         A[i] = (long long)A[i] * B[i] % p;
37
38     NTT(A, N, -1);
39
40     for (int i = 0; i <= r - l + 1; i++)
41         h[i] = A[i];
42 }
43 void solve2(int l, int r, int k) { // 分治
44     static int A[maxn], B[maxn], t[maxn];
45
46     if (l == r)
47         return;
48
49     int mid = (l + r) / 2;
50
51     solve2(l, mid, k + 1);
52     solve2(mid + 1, r, k + 1);
53
54     int *hl = th[k + 1] + l * 2, *hr = th[k + 1] + (mid
55         -> + 1) * 2;
56
57     int N = 1;
58
59     while (N < r - l + 1)
60         N *= 2;
61
62     memset(A, 0, sizeof(int) * N);
63     memset(B, 0, sizeof(int) * N);
64
65     memcpy(A, ansf + l, sizeof(int) * (mid - l + 1));
66     memcpy(B, hr, sizeof(int) * (r - mid + 1));
67
68     NTT(A, N, 1);
69     NTT(B, N, 1);
70
71     for (int i = 0; i < N; i++)
72         t[i] = (long long)A[i] * B[i] % p;
73
74     memset(A, 0, sizeof(int) * N);
75     memset(B, 0, sizeof(int) * N);
76
77     memcpy(A, ansf + mid + 1, sizeof(int) * (r - mid));
78     memcpy(B, hl, sizeof(int) * (mid - l + 2));
79
80     NTT(A, N, 1);
81
82     for (int i = 0; i < N; i++)
83         t[i] = (t[i] + (long long)A[i] * B[i]) % p;
84
85     NTT(t, N, -1);
86
87     memcpy(ansf + l, t, sizeof(int) * (r - l + 1));
88 }
89
90 // 主过程
91 // 如果x, y传nullptr表示询问已经存在了qx, qy里
92 void interpolation(int *x, int *y, int n, int *f =
93     nullptr) {
94     static int d[maxn];
95
96     if (x)
97         memcpy(qx, x, sizeof(int) * n);
98     if (y)
99         memcpy(qy, y, sizeof(int) * n);
100
101    pretreat2(0, n - 1, 0);
102
103    get_derivative(th[0], d, n + 1);
104
105    multipoint_eval(d, qx, nullptr, n, n);
106
107    for (int i = 0; i < n; i++)
108        ansf[i] = (long long)qy[i] * qpow(ans[i], p -
109            2) % p;
110
111    solve2(0, n - 1, 0);
112
113    if (f)
114        memcpy(f, ansf, sizeof(int) * n);
115 }
```

```

80     NTT(B, N, 1);
81
82     for (int i = 0; i < N; i++)
83         t[i] = (t[i] + (long long)A[i] * B[i]) % p;
84
85     NTT(t, N, -1);
86
87     memcpy(ansf + l, t, sizeof(int) * (r - l + 1));
88 }
89
90 // 主过程
91 // 如果x, y传nullptr表示询问已经存在了qx, qy里
92 void interpolation(int *x, int *y, int n, int *f =
93     nullptr) {
94     static int d[maxn];
95
96     if (x)
97         memcpy(qx, x, sizeof(int) * n);
98     if (y)
99         memcpy(qy, y, sizeof(int) * n);
100
101    pretreat2(0, n - 1, 0);
102
103    get_derivative(th[0], d, n + 1);
104
105    multipoint_eval(d, qx, nullptr, n, n);
106
107    for (int i = 0; i < n; i++)
108        ansf[i] = (long long)qy[i] * qpow(ans[i], p -
109            2) % p;
110
111    solve2(0, n - 1, 0);
112
113    if (f)
114        memcpy(f, ansf, sizeof(int) * n);
115 }
```

1.1.7 Bostan-Mori (求多项式分式第 n 项)

问题 已知多项式 $f(x)$ 与 $g(x)$, 求 $\frac{f(x)}{g(x)}$ 的第 n 项系数。
上下同乘 $g(-x)$, 则底下的 $g(x)g(-x)$ 只有偶数项, 所以上面的奇/偶数项乘完之后奇偶性是不变的。然后就可以直接按照 n 的奇偶性分情况只取出奇数项或者偶数项, 这样就在 n 不变的情况下把 k 折半了, 一直做到 $k = 0$ 然后输出常数项即可。

$$\begin{aligned}
 [x^k] \frac{f(x)}{g(x)} &= [x^k] \frac{f(x)g(-x)}{g(x)g(-x)} = [x^k] \frac{F(x^2) + xG(x^2)}{H(x^2)} \\
 &= \begin{cases} [x^{\lfloor k/2 \rfloor}] \frac{F(x)}{H(x)} & (k \text{ is even}) \\ [x^{\lfloor k/2 \rfloor}] \frac{G(x)}{H(x)} & (k \text{ is odd}) \end{cases}
 \end{aligned}$$

复杂度 $O(k \log k \log n)$ 。

```

1 int bostan_mori(int k, poly a, poly b) {
2     int n = (int)a.size();
3
4     while (k) {
5         poly c = b;
6         for (int i = 1; i < n; i += 2)
7             c[i] = (p - c[i]) % p;
8
9         a = poly_mul(a, c);
10        b = poly_mul(b, c);
11
12        for (int i = 0; i < n; i++) {
13            a[i] = a[i] * 2 + k % 2;
14            b[i] = b[i] * 2;
15        }
16    }
17
18    return a[0];
19 }
```

```

15     }
16
17     a.resize(n);
18     b.resize(n);
19     k /= 2;
20 }
21
22 return (ll)a[0] * qpow(b[0], p - 2) % p;
23 }
```

1.1.8 快速线性递推 $O(k \log k \log n)$

多项式除法 需要的代码参见 1.1.4.多项式操作 (第 2 页)。

```

1 poly poly_power_mod(ll k, const poly& m) { //  $x^k \bmod m$ 
2     poly ans{1}, a{0, 1};
3
4     while (k) {
5         if (k & 1)
6             ans = poly_mod(poly_auto_mul(ans, a),
7                             m).first;
7         a = poly_mod(poly_auto_mul(a, a), m).first;
8         k /= 2;
9     }
10
11    return ans;
12 }
13
14 //  $a_n = \sum_{i=1}^m c_i a_{n-i}$  ( $c_0 = 0$ )
15 struct linear_recurrence {
16     poly f; // f是预处理结果
17
18     linear_recurrence(const poly& c, ll n) {
19         assert(c[0] == 0); // c[0] 是没有用的
20         int m = (int)c.size() - 1;
21
22         int ntt_n = 1;
23         while (ntt_n < m * 2)
24             ntt_n *= 2;
25         ntt_init(ntt_n); // 图省事就直接 ntt_init(1 <
26                                     // 18)
26
27         poly t(m + 1);
28         t[m] = 1;
29
30         for (int i = 0; i < m; i++)
31             t[i] = (p - c[m - i]) % p;
32         f = poly_power_mod(n, t);
33     }
34
35     int operator()(const vector<int>& a) { // 0~m-1项初
36         始值
37         assert(a.size() == f.size()); int ans = 0;
38         for (int i = 0; i < (int)a.size(); i++)
39             ans = (ans + (ll)f[i] * a[i]) % p;
40         return ans;
41     }
41 };
```

Bostan-Mori 具体参见 1.1.7.Bostan-Mori (第 5 页)。

这个做法只需要抄多项式乘法，并且常数更小。

```

1 //  $a_n = \sum_{i=1}^m f_i a_{n-i}$  ( $f_0 = 0$ )
2 // f.size() = a.size() + 1
3 int linear_recurrence(int n, poly f, poly a) {
4     int m = (int)a.size();
5
6     int ntt_n = 1;
```

```

7     while (ntt_n <= m)
8         ntt_n *= 2;
9
10    ntt_init(ntt_n * 2);
11
12    f.resize(ntt_n);
13    a.resize(ntt_n);
14
15    for (int i = 1; i <= m; i++)
16        f[i] = (p - f[i]) % p;
17    f[0] = 1;
18
19    a = poly_mul(a, f);
20    a.resize(ntt_n);
21    fill(a.data() + m, a.data() + ntt_n, 0);
22
23    return bostan_mori(n, a, f);
24 }
```

1.1.9 多项式复合逆

拉格朗日反演 如果 $f(x)$ 与 $g(x)$ 互为复合逆，则有

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

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

这样可以得到复合逆的一项。如果需要 $0 \dots n$ 项的所有系数，就麻烦一些。

推导过程省略，直接上结论：

$$f(x) \equiv x \left(\sum_{k=1}^n x^{n-k} \frac{n}{k} [x^n] g^k(x) \right)^{-1/n} \pmod{x^{n+1}}$$

$$\equiv \frac{x}{g_1} \left(\sum_{k=1}^n \frac{n}{k} \left(\frac{x}{g_1} \right)^{n-k} [z^n] \left(\frac{g(z)}{g_1} \right)^k \right)^{-1/n}$$

g_1 是 $g(x)$ 的一次项。把它提出来是为了把要开根的式子常数项化成 1，避免考虑 $n \bmod \varphi(p)$ 的逆元。

现在唯一的难点就在于求 $[x^n] \left(\frac{g(x)}{g_1} \right)^k$ 。

考虑更一般的问题，即 对所有 $k \in [1, n]$ ，如何分别求出 $[x^n] A^k(x)$ 。引入另一个自变量 y ，代表 $A(x)$ 的次数这一维，得到一个二元生成函数：

$$\sum_i x^i \sum_j y^j [x^i] A^j(x) = \sum_j y^j A^j(x) = \frac{1}{1 - yA(x)}$$

x^n 项的系数即为所求。

针对这个子问题，再考虑更一般的问题，即求 $[x^n] \frac{P(x,y)}{Q(x,y)}$ 。

按照 Bostan-Mori (1.1.7, 第 5 页) 一样的思路，上下同乘 $Q(-x, y)$ ，又会发现分母 $Q(x, y)Q(-x, y)$ 里 x 只有偶数项，只取出奇数项或者偶数项就可以把 n 折半，然后递归下去即可。边界是 $\frac{P(0,y)}{Q(0,y)}$ 。

在这里初始时 x 最高项是 n 次，而 y 只有一次。每步 x 的最大次数都会减半，而 y 的最大次数会翻倍，所以总的项数一直是 $O(n)$ 的。总的复杂度是 $O(n \log^2 n)$ 。

这里有一个优化：做完 k 次迭代之后 y 的最高次数是 2^k ，也就是说有 $2^k + 1$ 项，在做卷积的时候就会白白多做一倍的长度。实际上可以注意到， $P(x, 0)$ 和 $Q(x, 0)$ 的 y^0 项只能是 0 或者 1，不会包含 x 。所以可以把它提出来，这样 y 这一维的长度就刚好是 2^k 了。

$Q(x, 0)$ 因为每次都保留偶数项，所以常数项一直都是 1。 $P(x, 0)$ 初始常数项也是 1，但在保留一次奇数项之后就一直是 0 了。

设 $P(x, 0)$ 的常数项是 k ，按照 $(Py + k)(Qy + 1) = (PQy + P + kQ)y + k$ 计算即可。

解决上面的所有子问题之后别忘了还要有一个多项式开 n 次根，总代码量还是很大的。

```

1 // 对二元多项式做 DFT
2 void dft_2d(vector<poly>& a, int ntt_n, int ntt_m) {
3     int n = (int)a.size();
4
5     a.resize(ntt_n);
6     for (int i = 0; i < ntt_n; i++) {
7         a[i].resize(ntt_m);
8
9         if (i < n)
10            ntt(a[i].data(), ntt_m, 1);
11    }
12
13    for (int j = 0; j < ntt_m; j++) {
14        poly t(ntt_n);
15
16        for (int i = 0; i < n; i++)
17            t[i] = a[i][j];
18
19        ntt(t.data(), ntt_n, 1);
20
21        for (int i = 0; i < ntt_n; i++)
22            a[i][j] = t[i];
23    }
24}
25
26 // 对二元多项式做 DFT, 只保留前 n 行中 filter = true 的行
27 void idft_2d(vector<poly>& a, int n,
28   → function<bool(int)> filter) {
29     int ntt_n = (int)a.size(), m = (int)a[0].size();
30
31     for (int j = 0; j < m; j++) {
32         poly t(ntt_n);
33         for (int i = 0; i < ntt_n; i++)
34             t[i] = a[i][j];
35
36         ntt(t.data(), ntt_n, -1);
37
38         for (int i = 0; i < n; i++)
39             a[i][j] = t[i];
40    }
41
42    a.resize(n);
43    for (int i = 0; i < n; i++)
44        if (filter(i))
45            ntt(a[i].data(), m, -1);
46}
47
48 // 对所有 k ∈ [1, n], 求 [x^n]f^k(x)
49 // 注意这里 n 是最高次数, 即 f 的长度为 n+1
50 poly bostan_mori(const poly& f) {
51     int n = (int)f.size() - 1;
52     vector<poly> a(n + 1), b(n + 1);
53
54     for (int i = 0; i <= n; i++) {
55         a[i].push_back(0);
56         b[i].push_back((p - f[i]) % p);
57    }
58
59    bool a_constant = true;
60    int m = 1;
61    while (n) {
62        vector<poly> ac(a), bc(b);
63        for (int i = 0; i <= n; i++) {
64            for (int j = 0; j < m; j++) {
65                if (a_constant) {
66                    int d = i % 2 ? p - b[i][j] : b[i]
67                    → [j];
68                    (a[i][j] += d) %= p;
69                } else {
70                    if (i % 2 == 0)
71                        b[i][j] = b[i][j] * 2 % p;
72                    else
73                        b[i][j] = 0;
74                }
75            }
76            int ntt_n = get_ntt_n(n * 2 + 1);
77            dft_2d(ac, ntt_n, m * 2);
78            dft_2d(bc, ntt_n, m * 2);
79
80            for (int i = 0; i < ntt_n; i++) {
81                for (int j = 0; j < m * 2; j++) // Q(-x, y)
82                    → 的 DFT 直接从 Q(x, y) 的 DFT 转化过来就
83                    → 行了
84                    ac[i][j] = (ll)ac[i][j] * bc[(i + ntt_n
85                    → / 2) % ntt_n][j] % p;
86
87            for (int i = 0; i < ntt_n / 2; i++) // 因为 Q(x,
88            → y) Q(-x, y) 只有 x^2k 项, 所以只需做一半
89            for (int j = 0; j < m * 2; j++) // DFT 的后
90            → 一半一定和前一半一样
91            bc[i][j] = (ll)bc[i][j] * bc[i + ntt_n
92            → / 2][j] % p;
93
94            bc.resize(ntt_n / 2);
95
96            idft_2d(ac, n + 1, [n] (int i) { return i % 2
97            → == n % 2; });
98            idft_2d(bc, n / 2 + 1, [] (int i) { return
99            → true; });
100
101           for (int i = 0; i <= n; i++) {
102               if (i % 2 == n % 2) {
103                   a[i].resize(m * 2);
104
105                   for (int j = 1; j < m * 2; j++)
106                       (a[i][j] += ac[i][j - 1]) %= p;
107
108                   a[i / 2].swap(a[i]);
109               }
110
111           for (int i = 0; i <= n / 2; i++) {
112               b[i].swap(b[i * 2]);
113               b[i].resize(m * 2);
114
115               for (int j = 1; j < m * 2; j++)
116                   (b[i][j] += bc[i][j - 1]) %= p;
117
118           a_constant &= !(n % 2);
119           n /= 2;
120           m *= 2;
121
122           a.resize(n + 1);
123           b.resize(n + 1);
124
125           for (int i = (int)f.size() - 1; i; i--) {
126               a[0][i] = a[0][i - 1];
127               b[0][i] = b[0][i - 1];
128
129           a[0][0] = b[0][0] = 1;
130
131           poly res = poly_mul(a[0], poly_inv(b[0])); // 因为 m
132           → 每次都翻倍, 所以此时 FFT 的长度一定就是 m
133           res.resize(f.size() + 1);
134
135           return res;
136       }
137   }
138 }
```

```

128 }
129
130 // 返回长度和 g 相同
131 // 因为做 DFT 的时候没有二维转一维，所以还是只需要初始化
132 // → get_ntt_n(n + 1) * 2 就行了
133 poly poly_composite_inversion(const poly& g) {
134     assert(!g[0] && g[1]);
135
136     int n = (int)g.size() - 1;
137
138     int g1_inv = qpow(g[1], p - 2);
139     poly t(n + 1);
140     for (int i = 1; i <= n; i++)
141         t[i] = (ll)g[i] * g1_inv % p;
142
143     poly res = bostan_mori(t);
144
145     for (int i = 0, pw = 1; i < n; i++) {
146         t[i] = (ll)n * inv[n - i] % p * res[n - i] % p
147             → * pw % p;
148         pw = (ll)pw * g1_inv % p;
149     }
150
151     int ntt_n = 1;
152     while (ntt_n <= n)
153         ntt_n *= 2;
154     t.resize(ntt_n);
155
156     t = poly_pow(t, p - inv[n]);
157
158     poly f(n + 1);
159     for (int i = 1; i <= n; i++)
160         f[i] = (ll)t[i - 1] * g1_inv % p;
161     return f;
162 }
```

1.1.10 多项式复合

问题 给出两个最高次数均为 x^n 的多项式 $F(x)$ 和 $G(x)$, 求 $F(G(x))$ 的 $0 \dots n$ 项系数。

首先要注意形式幂级数的复合只有在 $G(x)$ 没有常数项时才是良定义的, 不然 $F(G(0))$ 不一定收敛。不过如果只是多项式复合就无所谓了。

做法 设 $H(x) = F(G(x))$, 也就是

$$H(x) = \sum_{i=0}^n f_i G(x)^i$$

点积是不好做的, 不过可以把 $F(x)$ 的系数反过来变成卷积。设 $r_i = f_{n-i}$, 那么

$$\begin{aligned} H(x) &\equiv \sum_{i=0}^n r_{n-i} G(x)^i \\ &\equiv \sum_{i=0}^n r_{n-i} [y^i] \frac{1}{1 - yG(x)} \\ &\equiv [y^n] \frac{R(y)}{1 - yG(x)} \pmod{x^{n+1}} \end{aligned}$$

发现这个形式和多项式复合逆的子问题很像, 不过这里要求的是 y^n 项的系数。仍然按照 Bostan-Mori 算法的思路, 上下同乘 $Q(-x, y)$:

$$\begin{aligned} [y^n] \frac{R(y)}{Q(x, y)} &\equiv [y^n] \frac{R(y)}{Q(x, y)Q(-x, y)} Q(-x, y) \\ &\equiv [y^n] \frac{R(y)}{V(x^2, y)} Q(-x, y) \pmod{x^{n+1}} \end{aligned}$$

那么 $\frac{R(y)}{V(x^2, y)} \pmod{x^{n+1}}$ 就变成了一个 x 的最高次数减半的子问题, 这时后面的 $\pmod{x^{n+1}}$ 就用得上了。因为乘了一次所以 y 的最高次数会翻倍, 一直递归到底才会达到 y^n 。注意递归的时候 $R(y)$ 是完全不变的, 因此为了保证复杂度需要先递归地做下去, 返回的时候再乘 $Q(-x, y)$ 。边界是 $Q(x, y) \equiv R(y)Q(0, y)^{-1} \pmod{x^1}$ 。

返回的时候 x 的最高次数要翻一倍, 不需要处理, 最多把超过当前层 n 的部分截掉就行了。而因为我们最终只需要 y^n 项, 假设当前 $Q(-x, y)$ 的 y 次数是 2^k , 那么当前层返回的结果的 y^i 项最多会影响到最终的第 $i + 1 + 2 + 4 + \dots + 2^{k-1} = i + 2^k - 1$ 项, 因此每层只需要返回最高的 2^k 项。每层的总项数都是 $O(n)$ 的, 总复杂度是 $O(n \log^2 n)$ 。

类似多项式复合逆, 这里也有一个优化: 在递归的时候 y 的最高次数是 2^k , 项数就是 $2^k + 1$, 做卷积的时候会多做一倍。但可以注意到 $Q(x, y)$ 的 y_0 项始终是 1, 所以可以把它提出来, 这样项数就刚好是 2^k 了, 每步按照 $(Ay + 1)(By + 1) = (ABy + A + B)y + 1$ 计算即可。

这样优化的话, 返回时会变成取一个 2^{k+1} 项多项式与一个 2^k 项多项式的乘积的较高 2^k 项。实际上按照循环卷积的原理, 直接做长为 2^{k+1} 的 FFT 就行了, 这时较高的 2^k 项在循环卷积下是不受影响的。

另外因为返回时还涉及到 $\frac{R(y)}{V(x^2, y)}$ 的 FFT, 因为 FFT 本质就是单位根处的点值, 所以可以发现 x 这一维的 DFT 实际上就是 $\frac{R(y)}{V(x, y)}$ 的 DFT 再重复一次。这样返回时可以少做一半 FFT, 减少一些常数。

代码里的 `dft_2d` 和 `idft_2d` 和多项式复合逆里的是一样的, 去抄一下就行了。

```

1 // b.size() = n + 1
2 vector<poly> bostan_mori_comp(int n, const poly& a,
3     → vector<poly> b) {
4     if (!n) { // a 的长度一定是最初的 n + 1
5         int n0 = (int)a.size(), m = (int)b[0].size();
6
7         b[0].insert(b[0].begin(), 1);
8         b[0].resize(get_ntt_n(n0));
9         poly b_inv = poly_inv(b[0]);
10        b_inv.resize(a.size());
11
12        poly t = poly_auto_mul(a, b_inv);
13        poly res = poly(m);
14        for (int i = 0; i < n0; i++)
15            res[i + m - n0] = t[i];
16
17        return {res};
18    }
19
20    int ntt_n = get_ntt_n(n * 2 + 1);
21    int m = (int)b[0].size();
22
23    vector<poly> q = b;
24    dft_2d(b, ntt_n, m * 2);
25
26    vector<poly> c(ntt_n);
27    for (int i = 0; i < ntt_n; i++) // Q(-x, y) 的 DFT
28        → 直接从 Q(x, y) 的 DFT 转化过来就行了
29        c[i] = b[(i + ntt_n / 2) % ntt_n];
30
31    b.resize(ntt_n / 2);
32
33    for (int i = 0; i < ntt_n / 2; i++)
34        for (int j = 0; j < m * 2; j++)
35            b[i][j] = (ll)b[i][j] * c[i][j] % p;
36
37    idft_2d(b, n / 2 + 1, [] (int i) { return true; });
38
39    for (int i = 0; i <= n / 2; i++) {
```

```

38     for (int j = m * 2; j; j--)
39         b[i][j] = b[i][j - 1];
40     b[i][0] = 0;
41
42     for (int j = 0; j < m; j++)
43         b[i][j] = (b[i][j] + q[i * 2][j] * 2 % p) %
44             → p;
45 }
46
47 b.resize(n / 2 + 1);
48
49 vector<poly> res = bostan_mori_comp(n / 2, a,
50     → move(b));
51
52
53 for (int i = 0; i < ntt_n; i++) // 因为 V(x^2, y) 只
54     → 有 x^2k 项, 所以只需做一半
55     for (int j = 0; j < m * 2; j++) // DFT 的后一半
56         → 一定和前一半一样
57         c[i][j] = (ll)t[i % (ntt_n / 2)][j] * c[i]
58             → [j] % p;
59
60 idft_2d(c, n + 1, [] (int i) { return true; });
61
62 t = vector<poly>(n + 1);
63 for (int i = 0; i <= n; i++) {
64     t[i].resize(m);
65
66     for (int j = 0; j < m; j++)
67         t[i][j] = c[i][j + m - 1];
68
69     if (i % 2 == 0)
70         for (int j = 0; j < m; j++)
71             t[i][j] = (t[i][j] + res[i / 2][j + m])
72                 → % p;
73 }
74
75 return t;
76
77 // 求 F(G(x)), 长度要相同
78 poly poly_composition(const poly& f, const poly& g) {
79     int n = (int)f.size() - 1;
80
81     poly a(n + 1);
82     for (int i = 0; i <= n; i++)
83         a[i] = f[n - i];
84
85     vector<poly> b(n + 1);
86     for (int i = 0; i <= n; i++)
87         b[i] = {(p - g[i]) % p};
88
89     vector<poly> t = bostan_mori_comp(n, a, b);
90
91     poly res(n + 1);
92     for (int i = 0; i <= n; i++)
93         res[i] = t[i][0];
94     return res;
95 }
```

1.1.11 分治 FFT

```

1 void solve(int l, int r) {
2     if (l == r)
3         return;
4
5     int mid = (l + r) / 2;
```

```

6         solve(l, mid);
7
8         int N = 1;
9         while (N <= r - l + 1)
10             N *= 2;
11
12         for (int i = l; i <= mid; i++)
13             B[i - l] = (long long)A[i] * fac_inv[i] % p;
14         fill(B + mid - l + 1, B + N, 0);
15         for (int i = 0; i < N; i++)
16             C[i] = fac_inv[i];
17
18         NTT(B, N, 1);
19         NTT(C, N, 1);
20
21         for (int i = 0; i < N; i++)
22             B[i] = (long long)B[i] * C[i] % p;
23
24         NTT(B, N, -1);
25
26         for (int i = mid + 1; i <= r; i++)
27             A[i] = (A[i] + B[i - l] * 2 % p * (long
28                 → long)fac[i] % p) % p;
29
30         solve(mid + 1, r);
31 }
```

1.1.12 半在线卷积

```

1 void solve(int l, int r) {
2     if (r <= m)
3         return;
4
5     if (r - l == 1) {
6         if (l == m)
7             f[l] = a[m];
8         else
9             f[l] = (long long)f[l] * inv[l - m] % p;
10
11     for (int i = l, t = (long long)l * f[l] % p; i
12         → <= n; i += l)
13         g[i] = (g[i] + t) % p;
14
15     return;
16 }
17
18 int mid = (l + r) / 2;
19
20 solve(l, mid);
21
22 if (l == 0) {
23     for (int i = 1; i < mid; i++) {
24         A[i] = f[i];
25         B[i] = (c[i] + g[i]) % p;
26     }
27     NTT(A, r, 1);
28     NTT(B, r, 1);
29     for (int i = 0; i < r; i++)
30         A[i] = (long long)A[i] * B[i] % p;
31     NTT(A, r, -1);
32
33     for (int i = mid; i < r; i++)
34         f[i] = (f[i] + A[i]) % p;
35 }
36 else {
37     for (int i = 0; i < r - l; i++)
38         A[i] = f[i];
```

```

38     for (int i = l; i < mid; i++)
39         B[i - l] = (c[i] + g[i]) % p;
40     NTT(A, r - l, 1);
41     NTT(B, r - l, 1);
42     for (int i = 0; i < r - l; i++)
43         A[i] = (long long)A[i] * B[i] % p;
44     NTT(A, r - l, -1);

45
46     for (int i = mid; i < r; i++)
47         f[i] = (f[i] + A[i - l]) % p;

48
49     memset(A, 0, sizeof(int) * (r - l));
50     memset(B, 0, sizeof(int) * (r - l));

51
52     for (int i = l; i < mid; i++)
53         A[i - l] = f[i];
54     for (int i = 0; i < r - l; i++)
55         B[i] = (c[i] + g[i]) % p;
56     NTT(A, r - l, 1);
57     NTT(B, r - l, 1);
58     for (int i = 0; i < r - l; i++)
59         A[i] = (long long)A[i] * B[i] % p;
60     NTT(A, r - l, -1);

61
62     for (int i = mid; i < r; i++)
63         f[i] = (f[i] + A[i - l]) % p;
64 }

65
66     memset(A, 0, sizeof(int) * (r - l));
67     memset(B, 0, sizeof(int) * (r - l));

68
69     solve(mid, r);
70 }
```

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:

```

1 for (int i = 0; i <= k; i++)
2     r[i] = f(i);
3 for (int j = 0; j < k; j++)
4     for (int i = k; i > j; i--)
5         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

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

1.3 FWT 快速沃尔什变换

```

1 // 注意 FWT 常数比较小, 这点与 FFT/NTT 不同
2 // 以下代码均以模质数情况为例, 其中 n 为变换长度, t 表示
3 // → 正/逆变换

4 // 按位或版本
5 void FWT_or(int *A, int n, int t) {
6     for (int k = 2; k <= n; k *= 2)
7         for (int i = 0; i < n; i += k)
8             for (int j = 0; j < k / 2; j++) {
9                 if (t > 0)
10                     A[i + j + k / 2] = (A[i + j + k / 2] +
11                                         A[i + j]) % p;
12                 else
13                     A[i + j + k / 2] = (A[i + j + k / 2] -
14                                         A[i + j] + p) % p;
15             }
16 }

17 // 按位与版本
18 void FWT_and(int *A, int n, int t) {
19     for (int k = 2; k <= n; k *= 2)
20         for (int i = 0; i < n; i += k)
21             for (int j = 0; j < k / 2; j++) {
22                 if (t > 0)
23                     A[i + j] = (A[i + j] + A[i + j + k / 2]) % p;
24                 else
25                     A[i + j] = (A[i + j] - A[i + j + k / 2] + p) % p;
26             }
27 }

28 // 按位异或版本
29 void FWT_xor(int *A, int n, int t) {
30     for (int k = 2; k <= n; k *= 2)
31         for (int i = 0; i < n; i += k)
32             for (int j = 0; j < k / 2; j++) {
33                 int a = A[i + j], b = A[i + j + k / 2];
34                 A[i + j] = (a + b) % p;
35                 A[i + j + k / 2] = (a - b + p) % p;
36             }
37

38     if (t < 0) {
39         int inv = qpow(n % p, p - 2); // n 的逆元, 在不
40         // 取模时需要用每层除以 2 代替
41         for (int i = 0; i < n; i++)
42             A[i] = A[i] * inv % p;
43 }
```

1.3.1 三行 FWT

```

1 void fwt_or(int *a, int n, int tp) {
2     for (int j = 0; (1 << j) < n; j++)
3         for (int i = 0; i < n; i++)
4             if (i >> j & 1) {
5                 if (tp > 0)
6                     a[i] += a[i ^ (1 << j)];
7                 else
8                     a[i] -= a[i ^ (1 << j)];
9             }
10 }
```

```

11 // and自然就是or反过来
12 void fwt_and(int *a, int n, int tp) {
13     for (int j = 0; (1 << j) < n; j++) {
14         for (int i = 0; i < n; i++) {
15             if (!(i > j & 1)) {
16                 if (tp > 0)
17                     a[i] += a[i | (1 << j)];
18                 else
19                     a[i] -= a[i | (1 << j)];
20             }
21         }
22     }
23 // xor同理

```

1.4 线性代数

稀疏矩阵操作参见1.6.Berlekamp-Massey 最小递推式 (第 13页).

1.4.1 矩阵乘法

```

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

```

1.4.2 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.

```

1 void Gauss_Jordan(int A[][maxn], int n) {
2     for (int i = 1; i <= n; i++) {
3         int ii = i;
4         for (int j = i + 1; j <= n; j++)
5             if (fabs(A[j][i]) > fabs(A[ii][i]))
6                 ii = j;
7
8         if (ii != i) // 这里没有判是否无解, 如果有可能无
9             →解的话要判一下
10        for (int j = i; j <= n + 1; j++)
11            swap(A[i][j], A[ii][j]);
12
13        for (int j = 1; j <= n; j++)
14            if (j != i) // 消成对角
15                for (int k = n + 1; k >= i; k--)
16                    A[j][k] -= A[j][i] / A[i][i] * A[i]
17                    →[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.

1.4.3 行列式取模

```

1 int p;
2
3 int Gauss(int A[maxn][maxn], int n) {
4     int det = 1;

```

```

5
6     for (int i = 1; i <= n; i++) {
7         for (int j = i + 1; j <= n; j++) {
8             while (A[j][i]) {
9                 int t = (p - A[i][i] / A[j][i]) % p;
10                for (int k = i; k <= n; k++)
11                    A[i][k] = (A[i][k] + (long
12                      →long)A[j][k] * t) % p;
13
14                swap(A[i], A[j]);
15                det = (p - det) % p; // 交换一次之后行列
16                →式取负
17            }
18
19            if (!A[i][i])
20                return 0;
21
22        }
23    }
24

```

1.4.4 线性基 (消成对角)

```

1 void add(unsigned long long x) {
2     for (int i = 63; i >= 0; i--) {
3         if (x >> i & 1) {
4             if (b[i])
5                 x ^= b[i];
6             else {
7                 b[i] = x;
8
9                 for (int j = i - 1; j >= 0; j--) {
10                     if (b[j] && (b[i] >> j & 1))
11                         b[i] ^= b[j];
12
13                     for (int j = i + 1; j < 64; j++) {
14                         if (b[j] >> i & 1)
15                             b[j] ^= b[i];
16
17                     break;
18                 }
19             }
20         }
}

```

1.4.5 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:

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

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).

1.4.6 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.

1.5 $O(k^2 \log n)$ 齐次线性递推

first里面是第 1 到 k 项, trans从低到高分别是 a_{n-1} 到 a_{n-k} 的系数.

```

1 struct LinearRecurrence {
2     vector<int> first, trans;
3     vector<vector<int>> bin;
4
5     vector<int> multi(const vector<int> &a, const
6         → vector<int> &b) {
7         int n = (int)a.size() - 1;
8
9         vector<int> c(n * 2 + 1);
10
11        for (int i = 0; i <= n; i++) {
12            for (int j = 0; j <= n; j++)
13                c[i + j] = (c[i + j] + (long long)a[i]
14                    → * b[j]) % p;
15
16            for (int i = n * 2; i > n; i--) {
17                for (int j = 0; j < n; j++)
18                    c[i - 1 - j] = (c[i - 1 - j] + (long
19                        → long)c[i] * trans[j]) % p;
20
21                c[i] = 0;
22            }
23
24            c.resize(n + 1);
25            return c;
26        }
27
28        LinearRecurrence(vector<int> &first, vector<int>
29            → &trans) : first(first), trans(trans) {
30            int n = (int)first.size();
31
32            vector<int> a(n + 1);
33            a[1] = 1;
34            bin.push_back(a);
35
36            for (int i = 1; i < 64; i++)
37                bin.push_back(multi(bin[i - 1], bin[i -
38                    → 1]));
39
40            int calc(long long k) {
41                int n = (int)first.size();
42
43                vector<int> a(n + 1);
44                a[0] = 1;
45
46                for (int i = 0; i < n; i++)
47                    a[i] = 0;
48
49                for (int i = 0; i < k; i++)
50                    a[i] = 1;
51
52            }
53
54        }
55    }
56
57    vector<int> calc() {
58        vector<int> a(first);
59
60        for (int i = 0; i < trans.size(); i++)
61            a = multi(a, trans[i]);
62
63        return a;
64    }
65
66    void print() {
67        cout << "First: ";
68        for (int i = 0; i < first.size(); i++)
69            cout << first[i] << " ";
70
71        cout << endl;
72
73        cout << "Trans: ";
74        for (int i = 0; i < trans.size(); i++)
75            cout << trans[i] << " ";
76
77        cout << endl;
78
79        cout << "Bin: ";
80        for (int i = 0; i < bin.size(); i++)
81            cout << bin[i] << " ";
82
83        cout << endl;
84
85        cout << "Calc: ";
86        cout << calc() << endl;
87
88    }
89
90    void print(int i) {
91        cout << "First[" << i << "]: ";
92        cout << first[i] << endl;
93
94        cout << "Trans[" << i << "]: ";
95        cout << trans[i] << endl;
96
97        cout << "Bin[" << i << "]: ";
98        cout << bin[i] << endl;
99
100    }
101
102    void print(int i, int j) {
103        cout << "Bin[" << i << "]["
104            << j << "]: ";
105        cout << bin[i][j] << endl;
106
107    }
108
109    void print(int i, int j, int k) {
110        cout << "Bin[" << i << "]["
111            << j << "]["
112            << k << "]: ";
113        cout << bin[i][j][k] << endl;
114
115    }
116
117    void print(int i, int j, int k, int l) {
118        cout << "Bin[" << i << "]["
119            << j << "]["
120            << k << "]["
121            << l << "]: ";
122        cout << bin[i][j][k][l] << endl;
123
124    }
125
126    void print(int i, int j, int k, int l, int m) {
127        cout << "Bin[" << i << "]["
128            << j << "]["
129            << k << "]["
130            << l << "]["
131            << m << "]: ";
132        cout << bin[i][j][k][l][m] << endl;
133
134    }
135
136    void print(int i, int j, int k, int l, int m, int n) {
137        cout << "Bin[" << i << "]["
138            << j << "]["
139            << k << "]["
140            << l << "]["
141            << m << "]["
142            << n << "]: ";
143        cout << bin[i][j][k][l][m][n] << endl;
144
145    }
146
147    void print(int i, int j, int k, int l, int m, int n, int o) {
148        cout << "Bin[" << i << "]["
149            << j << "]["
150            << k << "]["
151            << l << "]["
152            << m << "]["
153            << n << "]["
154            << o << "]: ";
155        cout << bin[i][j][k][l][m][n][o] << endl;
156
157    }
158
159    void print(int i, int j, int k, int l, int m, int n, int o, int p) {
160        cout << "Bin[" << i << "]["
161            << j << "]["
162            << k << "]["
163            << l << "]["
164            << m << "]["
165            << n << "]["
166            << o << "]["
167            << p << "]: ";
168        cout << bin[i][j][k][l][m][n][o][p] << endl;
169
170    }
171
172    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q) {
173        cout << "Bin[" << i << "]["
174            << j << "]["
175            << k << "]["
176            << l << "]["
177            << m << "]["
178            << n << "]["
179            << o << "]["
180            << p << "]["
181            << q << "]: ";
182        cout << bin[i][j][k][l][m][n][o][p][q] << endl;
183
184    }
185
186    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r) {
187        cout << "Bin[" << i << "]["
188            << j << "]["
189            << k << "]["
190            << l << "]["
191            << m << "]["
192            << n << "]["
193            << o << "]["
194            << p << "]["
195            << q << "]["
196            << r << "]: ";
197        cout << bin[i][j][k][l][m][n][o][p][q][r] << endl;
198
199    }
200
201    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s) {
202        cout << "Bin[" << i << "]["
203            << j << "]["
204            << k << "]["
205            << l << "]["
206            << m << "]["
207            << n << "]["
208            << o << "]["
209            << p << "]["
210            << q << "]["
211            << r << "]["
212            << s << "]: ";
213        cout << bin[i][j][k][l][m][n][o][p][q][r][s] << endl;
214
215    }
216
217    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t) {
218        cout << "Bin[" << i << "]["
219            << j << "]["
220            << k << "]["
221            << l << "]["
222            << m << "]["
223            << n << "]["
224            << o << "]["
225            << p << "]["
226            << q << "]["
227            << r << "]["
228            << s << "]["
229            << t << "]: ";
230        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t] << endl;
231
232    }
233
234    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u) {
235        cout << "Bin[" << i << "]["
236            << j << "]["
237            << k << "]["
238            << l << "]["
239            << m << "]["
240            << n << "]["
241            << o << "]["
242            << p << "]["
243            << q << "]["
244            << r << "]["
245            << s << "]["
246            << t << "]["
247            << u << "]: ";
248        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u] << endl;
249
250    }
251
252    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v) {
253        cout << "Bin[" << i << "]["
254            << j << "]["
255            << k << "]["
256            << l << "]["
257            << m << "]["
258            << n << "]["
259            << o << "]["
260            << p << "]["
261            << q << "]["
262            << r << "]["
263            << s << "]["
264            << t << "]["
265            << u << "]["
266            << v << "]: ";
267        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v] << endl;
268
269    }
270
271    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w) {
272        cout << "Bin[" << i << "]["
273            << j << "]["
274            << k << "]["
275            << l << "]["
276            << m << "]["
277            << n << "]["
278            << o << "]["
279            << p << "]["
280            << q << "]["
281            << r << "]["
282            << s << "]["
283            << t << "]["
284            << u << "]["
285            << v << "]["
286            << w << "]: ";
287        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w] << endl;
288
289    }
290
291    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x) {
292        cout << "Bin[" << i << "]["
293            << j << "]["
294            << k << "]["
295            << l << "]["
296            << m << "]["
297            << n << "]["
298            << o << "]["
299            << p << "]["
300            << q << "]["
301            << r << "]["
302            << s << "]["
303            << t << "]["
304            << u << "]["
305            << v << "]["
306            << w << "]["
307            << x << "]: ";
308        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x] << endl;
309
310    }
311
312    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y) {
313        cout << "Bin[" << i << "]["
314            << j << "]["
315            << k << "]["
316            << l << "]["
317            << m << "]["
318            << n << "]["
319            << o << "]["
320            << p << "]["
321            << q << "]["
322            << r << "]["
323            << s << "]["
324            << t << "]["
325            << u << "]["
326            << v << "]["
327            << w << "]["
328            << x << "]["
329            << y << "]: ";
330        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y] << endl;
331
332    }
333
334    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z) {
335        cout << "Bin[" << i << "]["
336            << j << "]["
337            << k << "]["
338            << l << "]["
339            << m << "]["
340            << n << "]["
341            << o << "]["
342            << p << "]["
343            << q << "]["
344            << r << "]["
345            << s << "]["
346            << t << "]["
347            << u << "]["
348            << v << "]["
349            << w << "]["
350            << x << "]["
351            << y << "]["
352            << z << "]: ";
353        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z] << endl;
354
355    }
356
357    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z, int a) {
358        cout << "Bin[" << i << "]["
359            << j << "]["
360            << k << "]["
361            << l << "]["
362            << m << "]["
363            << n << "]["
364            << o << "]["
365            << p << "]["
366            << q << "]["
367            << r << "]["
368            << s << "]["
369            << t << "]["
370            << u << "]["
371            << v << "]["
372            << w << "]["
373            << x << "]["
374            << y << "]["
375            << z << "]["
376            << a << "]: ";
377        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z][a] << endl;
378
379    }
380
381    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z, int a, int b) {
382        cout << "Bin[" << i << "]["
383            << j << "]["
384            << k << "]["
385            << l << "]["
386            << m << "]["
387            << n << "]["
388            << o << "]["
389            << p << "]["
390            << q << "]["
391            << r << "]["
392            << s << "]["
393            << t << "]["
394            << u << "]["
395            << v << "]["
396            << w << "]["
397            << x << "]["
398            << y << "]["
399            << z << "]["
400            << a << "]["
401            << b << "]: ";
402        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z][a][b] << endl;
403
404    }
405
406    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z, int a, int b, int c) {
407        cout << "Bin[" << i << "]["
408            << j << "]["
409            << k << "]["
410            << l << "]["
411            << m << "]["
412            << n << "]["
413            << o << "]["
414            << p << "]["
415            << q << "]["
416            << r << "]["
417            << s << "]["
418            << t << "]["
419            << u << "]["
420            << v << "]["
421            << w << "]["
422            << x << "]["
423            << y << "]["
424            << z << "]["
425            << a << "]["
426            << b << "]["
427            << c << "]: ";
428        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z][a][b][c] << endl;
429
430    }
431
432    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z, int a, int b, int c, int d) {
433        cout << "Bin[" << i << "]["
434            << j << "]["
435            << k << "]["
436            << l << "]["
437            << m << "]["
438            << n << "]["
439            << o << "]["
440            << p << "]["
441            << q << "]["
442            << r << "]["
443            << s << "]["
444            << t << "]["
445            << u << "]["
446            << v << "]["
447            << w << "]["
448            << x << "]["
449            << y << "]["
450            << z << "]["
451            << a << "]["
452            << b << "]["
453            << c << "]["
454            << d << "]: ";
455        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z][a][b][c][d] << endl;
456
457    }
458
459    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z, int a, int b, int c, int d, int e) {
460        cout << "Bin[" << i << "]["
461            << j << "]["
462            << k << "]["
463            << l << "]["
464            << m << "]["
465            << n << "]["
466            << o << "]["
467            << p << "]["
468            << q << "]["
469            << r << "]["
470            << s << "]["
471            << t << "]["
472            << u << "]["
473            << v << "]["
474            << w << "]["
475            << x << "]["
476            << y << "]["
477            << z << "]["
478            << a << "]["
479            << b << "]["
480            << c << "]["
481            << d << "]["
482            << e << "]: ";
483        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z][a][b][c][d][e] << endl;
484
485    }
486
487    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z, int a, int b, int c, int d, int e, int f) {
488        cout << "Bin[" << i << "]["
489            << j << "]["
490            << k << "]["
491            << l << "]["
492            << m << "]["
493            << n << "]["
494            << o << "]["
495            << p << "]["
496            << q << "]["
497            << r << "]["
498            << s << "]["
499            << t << "]["
500            << u << "]["
501            << v << "]["
502            << w << "]["
503            << x << "]["
504            << y << "]["
505            << z << "]["
506            << a << "]["
507            << b << "]["
508            << c << "]["
509            << d << "]["
510            << e << "]["
511            << f << "]: ";
512        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z][a][b][c][d][e][f] << endl;
513
514    }
515
516    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z, int a, int b, int c, int d, int e, int f, int g) {
517        cout << "Bin[" << i << "]["
518            << j << "]["
519            << k << "]["
520            << l << "]["
521            << m << "]["
522            << n << "]["
523            << o << "]["
524            << p << "]["
525            << q << "]["
526            << r << "]["
527            << s << "]["
528            << t << "]["
529            << u << "]["
530            << v << "]["
531            << w << "]["
532            << x << "]["
533            << y << "]["
534            << z << "]["
535            << a << "]["
536            << b << "]["
537            << c << "]["
538            << d << "]["
539            << e << "]["
540            << f << "]["
541            << g << "]: ";
542        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z][a][b][c][d][e][f][g] << endl;
543
544    }
545
546    void print(int i, int j, int k, int l, int m, int n, int o, int p, int q, int r, int s, int t, int u, int v, int w, int x, int y, int z, int a, int b, int c, int d, int e, int f, int g, int h) {
547        cout << "Bin[" << i << "]["
548            << j << "]["
549            << k << "]["
550            << l << "]["
551            << m << "]["
552            << n << "]["
553            << o << "]["
554            << p << "]["
555            << q << "]["
556            << r << "]["
557            << s << "]["
558            << t << "]["
559            << u << "]["
560            << v << "]["
561            << w << "]["
562            << x << "]["
563            << y << "]["
564            << z << "]["
565            << a << "]["
566            << b << "]["
567            << c << "]["
568            << d << "]["
569            << e << "]["
570            << f << "]["
571            << g << "]["
572            << h << "]: ";
573        cout << bin[i][j][k][l][m][n][o][p][q][r][s][t][u][v][w][x][y][z][a][b][c][
```

```

42     }
43 }
44
45 for (auto &x : v) // 一般是需要最小递推式的, 所以处理
46     ←一下
47     x = (p - x) % p;
48     v.insert(v.begin(), 1);
49
50 }  


```

如果要求向量序列的递推式, 就把每位乘一个随机权值 (或者说是乘一个随机行向量 v^T) 变成求数列递推式即可.

如果是矩阵序列的话就随机一个行向量 u^T 和列向量 v , 然后把矩阵变成 $u^T A v$ 的数列就行了.

1.6.1 优化矩阵快速幂DP

如果 f_i 是一个向量, 并且转移是一个矩阵, 那显然 $\{f_i\}$ 是一个线性递推序列.

假设 f_i 有 n 维, 先暴力求出 $f_{0 \sim 2n-1}$, 然后跑Berlekamp-Massey, 最后调用前面的快速齐次线性递推 (6页) 即可. (快速齐次线性递推的结果是一个序列, 某个给定初值的结果就是点乘, 所以只需要跑一次.)

如果要求 f_m , 并且矩阵有 k 个非零项的话, 复杂度就是 $O(nk + n \log m \log n)$. (因为暴力求前 $2n-1$ 个 f_i 需要 $O(nk)$ 时间.)

1.6.2 求矩阵最小多项式

矩阵 A 的最小多项式是次数最小的并且 $f(A) = 0$ 的多项式 f .

实际上最小多项式就是 $\{A^i\}$ 的最小递推式, 所以直接调用Berlekamp-Massey就好了, 并且显然它的次数不超过 n .

瓶颈在于求出 A^i , 实际上我们只要处理 $A^i v$ 就行了, 每次对向量做递推.

假设 A 有 k 个非零项, 则复杂度为 $O(kn + n^2)$.

1.6.3 求稀疏矩阵的行列式

如果能求出特征多项式, 则常数项乘上 $(-1)^n$ 就是行列式, 但是最小多项式不一定就是特征多项式.

把 A 乘上一个随机对角阵 B (实际上就是每行分别乘一个随机数), 则 AB 的最小多项式有很大概率就是特征多项式, 最后再除掉 $\det B$ 就行了.

设 A 有 k 个非零项, 则复杂度为 $O(kn + n^2)$.

1.6.4 求稀疏矩阵的秩

设 A 是一个 $n \times m$ 的矩阵, 首先随机一个 $n \times n$ 的对角阵 P 和一个 $m \times m$ 的对角阵 Q , 然后计算 $QAP^{-1}Q^TQ$ 的最小多项式即可.

实际上不用计算这个矩阵, 因为求最小多项式时要用它乘一个向量, 我们依次把这几个矩阵乘到向量里就行了. 答案就是最小多项式除掉所有 x 因子后剩下的次数.

设 A 有 k 个非零项, 复杂度为 $O(kn + n^2)$.

1.6.5 解稀疏方程组

问题 $Ax = b$, 其中 A 是一个 $n \times n$ 的满秩稀疏矩阵, b 和 x 是 $1 \times n$ 的列向量, A, b 已知, 需要解出 x .

做法 显然 $x = A^{-1}b$. 如果我们能求出 $\{A^i b\} (i \geq 0)$ 的最小递推式 $\{r_{0 \sim m-1}\} (m \leq n)$, 那么就有结论

$$A^{-1}b = -\frac{1}{r_{m-1}} \sum_{i=0}^{m-2} A^i b r_{m-2-i}$$

因为 A 是稀疏矩阵, 直接按定义递推出 $b \sim A^{2n-1}b$ 即可. 设 A 中有 k 个非零项, 则复杂度为 $O(kn + n^2)$.

```

1 vector<int> solve_sparse_equations(const
2     → vector<tuple<int, int, int>> &A, const vector<int>
3     → &b) {
4         int n = (int)b.size(); // 0-based
5
6         vector<vector<int>> f({b});
7
8         for (int i = 1; i < 2 * n; i++) {
9             vector<int> v(n);
10            auto &u = f.back();
11
12            for (auto [x, y, z] : A) // [x, y, value]
13                v[x] = (v[x] + (long long)u[y] * z) % p;
14
15            f.push_back(v);
16        }
17
18        vector<int> w(n);
19        mt19937 gen;
20        for (auto &x : w)
21            x = uniform_int_distribution<int>(1, p - 1)
22                → (gen);
23
24        vector<int> a(2 * n);
25        for (int i = 0; i < 2 * n; i++)
26            for (int j = 0; j < n; j++)
27                a[i] = (a[i] + (long long)f[i][j] * w[j]) %
28                    p;
29
30        auto c = berlekamp_massey(a);
31        int m = (int)c.size();
32
33        vector<int> ans(n);
34
35        for (int i = 0; i < m - 1; i++)
36            for (int j = 0; j < n; j++)
37                ans[j] = (ans[j] + (long long)c[m - 2 - i]
38                    → * f[i][j]) % p;
39
40        int inv = qpow(p - c[m - 1], p - 2);
41
42        for (int i = 0; i < n; i++)
43            ans[i] = (long long)ans[i] * inv % p;
44
45        return ans;
46    }

```

1.7 单纯形

```

1 const double eps = 1e-10;
2
3 double A[maxn][maxn], x[maxn];
4 int n, m, t, id[maxn * 2];
5
6 // 方便起见, 这里附上主函数
7 int main() {
8     scanf("%d%d%d", &n, &m, &t);
9
10    for (int i = 1; i <= n; i++) {
11        scanf("%lf", &A[0][i]);
12        id[i] = i;
13    }
14
15    for (int i = 1; i <= m; i++) {
16        for (int j = 1; j <= n; j++)
17            scanf("%lf", &A[i][j]);
18

```

```

19     scanf("%lf", &A[i][0]);
20 }
21
22 if (!initialize())
23     printf("Infeasible"); // 无解
24 else if (!simplex())
25     printf("Unbounded"); // 最优解无限大
26
27 else {
28     printf("%.15lf\n", -A[0][0]);
29     if (t) {
30         for (int i = 1; i <= m; i++)
31             x[id[i + n]] = A[i][0];
32         for (int i = 1; i <= n; i++)
33             printf("%.15lf ", x[i]);
34     }
35 }
36 return 0;
37 }

//初始化
//对于初始解可行的问题，可以把初始化省略掉
41 bool initialize() {
42     while (true) {
43         double t = 0.0;
44         int l = 0, e = 0;
45
46         for (int i = 1; i <= m; i++) {
47             if (A[i][0] + eps < t) {
48                 t = A[i][0];
49                 l = i;
50             }
51
52             if (!l)
53                 return true;
54
55             for (int i = 1; i <= n; i++) {
56                 if (A[l][i] < -eps && (!e || id[i] <
57                     → id[e]))
58                     e = i;
59
60             if (!e)
61                 return false;
62
63             pivot(l, e);
64     }
65
66 //求解
67 bool simplex() {
68     while (true) {
69         int l = 0, e = 0;
70         for (int i = 1; i <= n; i++)
71             if (A[0][i] > eps && (!e || id[i] < id[e]))
72                 e = i;
73
74             if (!e)
75                 return true;
76
77             double t = 1e50;
78             for (int i = 1; i <= m; i++) {
79                 if (A[i][e] > eps && A[i][0] / A[i][e] < t)
80                     → {
81                         l = i;
82                         t = A[i][0] / A[i][e];
83                     }
84
85             if (!l)
86                 return false;
87     }
88 }

```

```

86         pivot(l, e);
87     }
88 }
89
90 //转轴操作，本质是在凸包上沿着一条棱移动
91 void pivot(int l, int e) {
92     swap(id[e], id[n + l]);
93     double t = A[l][e];
94     A[l][e] = 1.0;
95
96     for (int i = 0; i <= n; i++)
97         A[l][i] /= t;
98
99     for (int i = 0; i <= m; i++)
100        if (i != l) {
101            t = A[i][e];
102            A[i][e] = 0.0;
103            for (int j = 0; j <= n; j++)
104                A[i][j] -= t * A[l][j];
105        }
106    }
107 }

```

1.7.1 线性规划对偶原理

给定一个原始线性规划:

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^n c_j x_j \\ \text{Subject to} \quad & \sum_{j=1}^n a_{ij} x_j \geq b_i, \\ & x_j \geq 0 \end{aligned}$$

定义它的对偶线性规划为:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^m b_i y_i \\ \text{Subject to} \quad & \sum_{i=1}^m a_{ij} y_i \leq c_j, \\ & y_i \geq 0 \end{aligned}$$

用矩阵可以更形象地表示为:

$$\begin{array}{ll} \text{Minimize} & \mathbf{c}^T \mathbf{x} \\ \text{Subject to} & \mathbf{Ax} \geq \mathbf{b}, \\ & \mathbf{x} \geq 0 \end{array} \iff \begin{array}{ll} \text{Maximize} & \mathbf{b}^T \mathbf{y} \\ \text{Subject to} & \mathbf{A}^T \mathbf{y} \leq \mathbf{c}, \\ & \mathbf{y} \geq 0 \end{array}$$

1.8 博弈论

1.8.1 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1.8.3.经典博弈(15页)。

1.8.2 纳什均衡

纯策略，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。

1.8.3 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。

实际上只要路径长度唯一就可以，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 $\text{mod}(k+1)$ ，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。也等价于两个人移动一个只能向左上方走的皇后，不能动的输。

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

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

7. 无向图游戏

在一个无向图上的某个点上摆一个棋子，两个人轮流把棋子移动到相邻的点，并且每个点只能走一次，不能操作的输。

结论 如果某个点一定在最大匹配中，则先手必胜，否则先手必败。

1.8.4 例题

1. 黑白棋游戏

一些棋子排成一列，棋子两面分别是黑色和白色。两个人轮流行动，每次可以选择一个白色朝上的棋子，把它和它左边的所有棋子都翻转，不能行动的输。

结论 只需要看最左边的棋子即可，因为每次操作最左边的棋子都一定会被翻转。

二维情况同理，如果每次是把左上角的棋子全部翻转，那么就只需要看左上角的那个棋子。

1.9 自适应 Simpson 积分

Forked from fstqwq's template.

```

1 // Adaptive Simpson's method : double simpson::solve
2   ↪ (double (*f) (double), double l, double r, double
3     ↪ eps) : integrates f over (l, r) with error eps.
4
5 double area (double (*f) (double), double l, double r)
6   ↪ {
7     double m = l + (r - l) / 2;
8     return (f(l) + 4 * f(m) + f(r)) * (r - l) / 6;
9   }
10
11 double solve (double (*f) (double), double l, double r,
12   ↪ double eps, double a) {
13   double m = l + (r - l) / 2;
14   double left = area(f, l, m), right = area(f, m, r);
15   if (fabs(left + right - a) <= 15 * eps)
16     return left + right + (left + right - a) /
17       ↪ 15.0;
18   return solve(f, l, m, eps / 2, left) + solve(f, m,
19     ↪ r, eps / 2, right);
20 }
21
22 double solve (double (*f) (double), double l, double r,
23   ↪ double eps) {
24   return solve(f, l, r, eps, area (f, l, r));
25 }
```

1.10 常见数列

查表参见8.15.OEIS（第97页）。

1.10.1 斐波那契数卢卡斯数

斐波那契数 $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$
 $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \dots$

卢卡斯数 $L_0 = 2, L_1 = 1$

$2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, \dots$

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

```

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

```

7   if (n & 1)
8     return {d, c + d};
9   else
10    return {c, d};
11 }

```

1.10.2 伯努利数, 自然数幂次和

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

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

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

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

$B_0 = 1, B_1 = -\frac{1}{2}, B_4 = -\frac{1}{30}, B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, \dots$
(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:

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

1.10.3 分拆数

```

1 int b = sqrt(n);
2 ans[0] = tmp[0] = 1;
3
4 for (int i = 1; i <= b; ++i) {
5   for (int rep = 0; rep < 2; ++rep)
6     for (int j = i; j <= n - i * i; ++j)
7       add(tmp[j], tmp[j - i]);
8
9   for (int j = i * i; j <= n; ++j)
10    add(ans[j], tmp[j - i * i]);
11 }
12
13 // -----
14
15 long long a[100010];
16 long long p[50005]; // 欧拉五边形数定理
17
18 int main() {
19   p[0] = 1;
20   p[1] = 1;
21   p[2] = 2;
22   int i;
23   for (i = 1; i < 50005; i++) { // 递推式系
24     a[2 * i] = i * (i * 3 - 1) / 2; // 五边形数
25     a[2 * i + 1] = i * (i * 3 + 1) / 2;
26   }
27   for (i = 3; i < 50005; i++) { //
28     p[n] = p[n-1] + p[n-2] - p[n-5] - p[n-7] + p[12] + p[15] - ...
29     p[n-i*(3i-1)/2] + p[n-i*(3i+1)/2]
      p[i] = 0;
      int j;
}

```

```

30   for (j = 2; a[j] <= i; j++) { // 可能为负数, 式
31     ↪ 中加1000007
32     if (j & 2)
33       p[i] = (p[i] + p[i - a[j]] + 1000007) %
34       ↪ 1000007;
35     else
36       p[i] = (p[i] - p[i - a[j]] + 1000007) %
37       ↪ 1000007;
38   }
39   int n;
40   while (~scanf("%d", &n))
41     printf("%lld\n", p[n]);
}

```

1.10.4 斯特林数

1. 第一类斯特林数

$\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$ 表示 n 个元素划分成 k 个轮换的方案数.

递推式: $\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \left\{ \begin{matrix} n-1 \\ k-1 \end{matrix} \right\} + (n-1) \left\{ \begin{matrix} n-1 \\ k \end{matrix} \right\}$.

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).

$$\sum_{k=0}^n \left[\begin{matrix} n \\ k \end{matrix} \right] x^k = \prod_{i=0}^{n-1} (x+i)$$

求同一列: 用一个轮换的指数生成函数做 k 次幂

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

2. 第二类斯特林数

$\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$ 表示 n 个元素划分成 k 个子集的方案数.

递推式: $\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \left\{ \begin{matrix} n-1 \\ k-1 \end{matrix} \right\} + k \left\{ \begin{matrix} n-1 \\ k \end{matrix} \right\}$.

求一个: 容斥, 狗都会做

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

求同一行: FFT, 狗都会做

求同一列: 指数生成函数

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

普通生成函数

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

3. 斯特林反演

$$f(n) = \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} g(k) \iff g(n) = \sum_{k=0}^n (-1)^{n-k} \left[\begin{matrix} n \\ k \end{matrix} \right] f(k)$$

4. 幂的转换

上升幂与普通幂的转换

$$x^{\bar{n}} = \sum_k \left[\begin{matrix} n \\ k \end{matrix} \right] x^k$$

$$x^n = \sum_k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} (-1)^{n-k} x^{\bar{k}}$$

下降幂与普通幂的转换

$$x^n = \sum_k \binom{n}{k} x^k = \sum_k \binom{x}{k} \binom{n}{k} k!$$

$$x^n = \sum_k \binom{n}{k} (-1)^{n-k} x^k$$

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.

5. 斯特林多项式 (斯特林数关于斜线的性质)

定义:

$$\sigma_n(x) = \frac{\begin{bmatrix} x \\ n \end{bmatrix}}{x(x-1)\dots(x-n)}$$

$\sigma_n(x)$ 的最高次数是 x^{n-1} . (所以作为唯一的特例, $\sigma_0(x) = \frac{1}{x}$ 不是多项式.)

斯特林多项式实际上非常神奇, 它与两类斯特林数都有关系.

$$\begin{bmatrix} n \\ n-k \end{bmatrix} = n^{k+1} \sigma_k(n)$$

$$\left\{ \begin{array}{c} n \\ n-k \end{array} \right\} = (-1)^{k+1} n^{k+1} \sigma_k(-(n-k))$$

不过它并不好求. 可以 $O(k^2)$ 直接计算前几个点值然后插值, 或者如果要推式子的话可以用后面提到的二阶欧拉数.

1.10.5 贝尔数

$$\begin{aligned} B_0 &= 1, B_1 = 1, B_2 = 2, B_3 = 5, \\ B_4 &= 15, B_5 = 52, B_6 = 203, \dots \end{aligned}$$

$$B_n = \sum_{k=0}^n \left\{ \begin{array}{c} n \\ k \end{array} \right\}$$

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:

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

1.10.6 欧拉数 (Eulerian Number)

1. 欧拉数

$\langle \rangle$: n 个数的排列, 有 k 个上升的方案数.

$$\left\langle \begin{array}{c} n \\ k \end{array} \right\rangle = (n-k) \left\langle \begin{array}{c} n-1 \\ k-1 \end{array} \right\rangle + (k+1) \left\langle \begin{array}{c} n-1 \\ k \end{array} \right\rangle$$

$$\left\langle \begin{array}{c} n \\ k \end{array} \right\rangle = \sum_{i=0}^{k+1} (-1)^i \binom{n+1}{i} (k+1-i)^n$$

$$\sum_{k=0}^{n-1} \left\langle \begin{array}{c} n \\ k \end{array} \right\rangle = n!$$

$$x^n = \sum_{k=0}^{n-1} \left\langle \begin{array}{c} n \\ k \end{array} \right\rangle \binom{x+k}{n}$$

$$k! \left\{ \begin{array}{c} n \\ k \end{array} \right\} = \sum_{i=0}^{n-1} \left\langle \begin{array}{c} n \\ i \end{array} \right\rangle \binom{i}{n-k}$$

2. 二阶欧拉数

$\langle\langle \rangle\rangle$: 每个数都出现两次的多重排列, 并且每个数两次出现之间的数都比它要大. 在此前提下有 k 个上升的方案数.

$$\langle\langle \begin{array}{c} n \\ k \end{array} \rangle\rangle = (2n-k-1) \langle\langle \begin{array}{c} n-1 \\ k-1 \end{array} \rangle\rangle + (k+1) \langle\langle \begin{array}{c} n-1 \\ k \end{array} \rangle\rangle$$

$$\sum_{k=0}^{n-1} \langle\langle \begin{array}{c} n \\ k \end{array} \rangle\rangle = (2n-1)!! = \frac{(2n)!}{2^n}$$

3. 二阶欧拉数与斯特林数的关系

$$\left\{ \begin{array}{c} x \\ x-n \end{array} \right\} = \sum_{k=0}^{n-1} \langle\langle \begin{array}{c} n \\ k \end{array} \rangle\rangle \binom{x+n-k-1}{2n}$$

$$\left[\begin{array}{c} x \\ x-n \end{array} \right] = \sum_{k=0}^{n-1} \langle\langle \begin{array}{c} n \\ k \end{array} \rangle\rangle \binom{x+k}{2n}$$

1.10.7 卡特兰数, 施罗德数, 默慈金数

1. 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:

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

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

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

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

2. 施罗德数

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

$$(n+1)S_n = (6n-3)S_{n-1} - (n-2)S_{n-2}$$

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.)

3. 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。

1.11 常用公式及结论

1.11.1 方差

m 个数的方差：

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

随机变量的方差： $D^2(x) = E(x^2) - E^2(x)$

1.11.2 min-max 反演

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

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

推广：求第 k 大

$$k\text{-}\max(S) = \sum_{T \subset S} (-1)^{|T|-k} \binom{|T|-1}{k-1} \min(T)$$

显然只有大小至少为 k 的子集才是有用的。

1.11.3 单位根反演（展开整除条件 $[n|k]$ ）

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

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

1.11.4 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。

1.11.5 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 。

1.11.6 常系数齐次线性递推求通项

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

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

- (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 $\alpha_1, \dots, \alpha_s$ 互异， $e_1 + \dots + e_s = r$ 则 u_n 有表达式

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

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



1.12 常用生成函数变换

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

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

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

（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 $\binom{n}{k} k!$ 之后其余过程完全相同。

2 数论

2.1 $O(n)$ 预处理逆元

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

2.2 线性筛

```

// 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;
        }
    }
}

```

2.3 杜教筛

$$S_\varphi(n) = \frac{n(n+1)}{2} - \sum_{d=2}^n S_\varphi\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$$

$$S_\mu(n) = 1 - \sum_{d=2}^n S_\mu\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$$

```

1 // 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
2 // → 和 (有点绕)
3 // 有些题只要求  $n \leq 10^9$ , 这时就没必要开 long long 了, 但
4 // → 记得乘法时强转
5 // 常量/全局变量/数组定义
6 const int maxn = 5000005, table_size = 5000000, p =
7     → 1000000007, inv_2 = (p + 1) / 2;
8 bool notp[maxn];
9 int prime[maxn / 20], phi[maxn], tbl[100005];
10 // tbl用来顶替哈希表, 其实开到  $n^{\{1 / 3\}}$  就够了, 不过保
11 // → 风险起见开成  $\sqrt{n}$  比较好
12 long long N;
13
14 // 主函数前面加上这么一句
15 memset(tbl, -1, sizeof(tbl));
16
17 // 线性筛预处理部分略去
18
19 // 杜教筛主过程 总计  $O(n^{\{2 / 3\}})$ 
20 // 递归调用自身
21 // 递推式还需具体情况具体分析, 这里以求欧拉函数前缀和
22 // → ( $\bmod 10^9 + 7$ ) 为例
23 int S(long long n) {
24     if (n <= table_size)
25         return phi[n];
26     else if (~tbl[N / n])
27         return tbl[N / n];
28     // 原理: n除以所有可能的数的结果一定互不相同
29
30     int ans = 0;
31     for (long long i = 2, last; i <= n; i = last + 1) {
32         last = n / (n / i);
33         ans = (ans + (last - i + 1) % p * S(n / i)) %
34             → p; // 如果n是int范围的话记得强转
35     }
36
37     ans = (n % p * ((n + 1) % p) % p * inv_2 - ans + p)
38         → % p; // 同上
39     return tbl[N / n] = ans;
40 }

```

2.4 Powerful Number 篇

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** $a(n)$, 使得 $a(p) = f(p)$.

那么就存在一个积性函数

那么就存在一个特征函数 $h = j * g^{-1}$ ，也就是 $j = g^{-1} * h$ 。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

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

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 $O(\sqrt{n})$.

例题:

- $f(p^k) = p^k(p^k - 1) : g(n) = \text{id}(n)\varphi(n).$

- $f(p^k) = p \text{ xor } k : n$ 为偶数时 $g(n) = 3\varphi(n)$, 否则 $g(n) = \varphi(n).$

2.5 洲阁筛

(比较难写, 不建议用. 最好用后面的 min25 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

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

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

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

边界显然是自然数幂次和, 转移是

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

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 $p_i \leq j < p_i^2$, 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 $p_i^2 \leq j$, 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 $\leq \sqrt{n}$

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

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

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.

```

1 constexpr int maxn = 200005, p = 1000000007;
2
3 long long N, val[maxn]; // 询问的n和存储所有整除结果的表
4 int sqrt;
5
6 inline int getid(long long x) {
7     if (x <= sqrt)
8         return x;
9
10    return val[0] - N / x + 1;
11}
12
13 bool notp[maxn];
14 int prime[maxn], prime_cnt, rem[maxn]; // 线性筛用数组
15
16 int f[maxn], pr[maxn], g[2][maxn], dp[maxn];
17 int l[maxn], r[maxn];
18
19 // 线性筛省略
20
21 inline int get_sum(long long n, int k) {
22     n %= p;
23
24     if (k == 1)
25         return n * (n + 1) % p * ((p + 1) / 2) % p;
26
27     else
28         return n * (n + 1) % p * (2 * n + 1) % p * ((p
29             + 1) / 6) % p;
30 }
31
32 void get_dp(long long n, int k, int *dp) {
33     for (int j = 1; j <= val[0]; j++)
34         dp[j] = get_sum(val[j], k);
35
36     for (int i = 1; i <= prime_cnt; i++) {
37         long long lb = (long long)prime[i] * prime[i];
38         int pw = (k == 1 ? prime[i] : (int)(lb % p));
39
40         pr[i] = (pr[i - 1] + pw) % p;
41
42         for (int j = val[0]; j && val[j] >= lb; j--) {
43             int t = getid(val[j] / prime[i]);
44
45             int tmp = dp[t];
46             if (l[t] < i)
47                 tmp = (tmp - pr[min(i - 1, r[t])] +
48                     pr[l[t]]) % p;
49
50             dp[j] = (dp[j] - (long long)pw * tmp) % p;
51             if (dp[j] < 0)
52                 dp[j] += p;
53         }
54
55         for (int j = 1; j <= val[0]; j++) {
56             dp[j] = (dp[j] - pr[r[j]] + pr[l[j]]) % p;
57
58             dp[j] = (dp[j] + p - 1) % p; // 因为DP数组是
59             // 有1的, 但后面计算不应该有1
60         }
61
62         int calc1(long long n) {
63             get_dp(n, 1, g[0]);
64             get_dp(n, 2, g[1]);
65
66             int ans = 0;
67
68             for (int j = 1; j <= val[0]; j++)
69                 ans += g[0][j] * g[1][j];
70
71             return ans;
72         }
73
74         cout << calc1(N) << endl;
75     }
76 }
77
78 int main() {
79     cin << N;
80
81     for (int i = 2; i <= maxn; i++) {
82         if (!notp[i])
83             prime[prime_cnt++] = i;
84
85         for (int j = i * i; j <= maxn; j += i)
86             notp[j] = true;
87     }
88
89     cout << get_dp(N, 1, g[0]) + get_dp(N, 2, g[1]) << endl;
90 }
```

```

67  for (int i = 1; i <= sqrtN; i++)
68    ans = (ans + (long long)f[i] * (g[1][getid(N /
69      - i)] - g[0][getid(N / i)])) % p;
70
71  if (ans < 0)
72    ans += p;
73
74  return ans;
75
76 int calc2(long long n) {
77  for (int j = 1; j <= val[0]; j++)
78    dp[j] = 1;
79
80  for (int i = 1; i <= prime_cnt; i++)
81    pr[i] = (pr[i - 1] + f[prime[i]]) % p;
82
83  for (int i = prime_cnt; i; i--) {
84    long long lb = (long long)prime[i] * prime[i];
85
86    for (int j = val[0]; j && val[j] >= lb; j--)
87      for (long long pc = prime[i]; pc <= val[j];
88            -> pc *= prime[i]) {
89        int t = getid(val[j] / pc);
90
91        int tmp = dp[t];
92        if (r[t] > i)
93          tmp = (tmp + pr[r[t]] - pr[max(i,
94            -> l[t])]) % p;
95
96        dp[j] = (dp[j] + pc % p * ((pc - 1) %
97          -> p) % p * tmp) % p;
98      }
99
100 return (long long)(dp[val[0]] + pr[r[val[0]]] -
101   -> pr[l[val[0]]] + p) % p;
102
103 int main() {
104
105  // ios::sync_with_stdio(false);
106
107  cin >> N;
108
109  sqrtN = (int)sqrt(N);
110
111  get_table(sqrtN);
112
113  for (int i = 1; i <= sqrtN; i++)
114    val[+val[0]] = i;
115
116  for (int i = 1; i <= sqrtN; i++)
117    val[+val[0]] = N / i;
118
119  sort(val + 1, val + val[0] + 1);
120
121  val[0] = unique(val + 1, val + val[0] + 1) - val -
122    -> 1;
123
124  int li = 0, ri = 0;
125  for (int j = 1; j <= val[0]; j++) {
126    while (ri < prime_cnt && prime[ri + 1] <
127      -> val[j])
128      ri++;
129
130    while (li <= prime_cnt && (long long)prime[li]
131      -> * prime[li] <= val[j])
132      li++;
133
134    l[j] = li - 1;
135    r[j] = ri;
136  }
137
138  cout << (calc1(N) + calc2(N)) % p << endl;
139
140  return 0;
141}

```

2.6 min25 篩

假设要求的是

$$\sum_{i=1}^n f(i)$$

设 \sqrt{n} 以内的质数为 $p_1 \dots p_m$, 记

$$g(n) = \sum_{i=1}^n [i \in \mathbb{P}] f(i)$$

也就是只考虑质数项的和.

为了方便求出 g , 构造一个多项式 $F(x) = \sum a_i x^i$, 满足 $F(p) = f(p)$, 这样每个次幂就可以分开算贡献. ($f(p^c)$ 的形式无所谓.)

再令

$$h_k(i, n) = \sum_{x=2}^n [x \in \mathbb{P} \text{ 或 } x \text{ 与前 } i \text{ 个质数互质}] x^k$$

显然 $g(n) = \sum_k a_k h_k(\pi(\sqrt{n}), n)$, 递推求出所有 h_k 即可得到 g . 考虑 h 的转移, 当 $p_i > \sqrt{n}$ 时显然有 $h_k(i, n) = h_k(i - 1, n)$, 否则有

$$h_k(i, n) = h_k(i - 1, n) - p_i^k h_k\left(i - 1, \left\lfloor \frac{n}{p_i} \right\rfloor\right) + p_i^k \sum_{j=1}^{i-1} p_j^k$$

边界为 $h_k(0, n) = \sum_{i=2}^n i^k$.

求出 g 之后, 为了得到所有 $f(i)$ 之和还需要一次递推. 设

$$S(i, n) = \sum_{k=2}^n [k \text{ 与前 } (i - 1) \text{ 个质数互质}] f(k)$$

则

$$S(i, n) = g(n) - \sum_{k=1}^{i-1} f(p_k) + \sum_{k=i}^{p_k \leq \sqrt{n}} \sum_{c=1}^{p_k^{c+1} \leq n} S\left(k + 1, \left\lfloor \frac{n}{p_k^c} \right\rfloor\right) f(p_k^c) + f(p_k^{c+1})$$

这里直接递归即可, 注意边界应设为 $p_i > n$ 或 $n < 1$ 时 $S(i, n) = 0$. 最后的答案即为 $ans = S(1, n) + f(1)$.

也可以从大到小枚举 i 递推, 考虑到只有 $p_i \leq \sqrt{n}$ 时才有递推, 可以后缀和优化. 不优化的复杂度是 $O(n^{1-\epsilon})$, 优化之后是 $O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$, 不过一般是不优化更快.

```

1 constexpr int maxn = 200005, p = (int)1e9 + 7;
2
3 bool notp[maxn];
4 int prime[maxn / 5], prime_cnt;
5
6 // 线性筛省略
7
8 long long val[maxn]; // n 的所有整除结果
9 int sqrtN;

```

```

10 void get_val(long long n) {
11     for (int i = 1; i <= sqrtn; i++)
12         val[+val[0]] = i;
13
14     for (int i = sqrtn; i; i--)
15         val[+val[0]] = n / i;
16
17     assert(is_sorted(val + 1, val + val[0] + 1));
18     val[0] = unique(val + 1, val + val[0] + 1) - val -
19         → 1;
20 }
21
22 int getid(long long x) { // val[val[0]] 就是 n
23     return x <= sqrtn ? x : (val[0] - val[val[0]]) / x +
24         → 1;
25 }
26
27 int f(long long n) {
28     n %= p;
29     return n * (n - 1) % p;
30 }
31
32 int g[maxn];
33 int dp[maxn], prime_sum[maxn];
34
35 void get_dp(function<int(long long)> pw,
36             → function<int(long long)> sum) {
37     memset(dp, 0, sizeof(dp));
38     memset(prime_sum, 0, sizeof(prime_sum));
39
40     for (int i = 1; i <= prime_cnt; i++)
41         prime_sum[i] = (prime_sum[i - 1] +
42             → pw(prime[i])) % p;
43
44     for (int i = 1; i <= val[0]; i++)
45         dp[i] = (sum(val[i]) + p - 1) % p;
46
47     for (int i = 1; i <= prime_cnt; i++) {
48         int pi = prime[i];
49
50         for (int j = val[0]; (long long)pi * pi <=
51             → val[j]; j--) {
52             int k = getid(val[j] / pi);
53
54             dp[j] = (dp[j] + (long long)pw(pi) *
55                 → (prime_sum[i - 1] - dp[k])) % p;
56             if (dp[j] < 0)
57                 dp[j] += p;
58         }
59     }
60
61 void calc(long long n) {
62     get_val(n);
63
64     get_dp([] (long long x) {
65         x %= p;
66         return x * x % p;
67     }, [] (long long n) {
68         n %= p;
69         return n * (n + 1) % p * (2 * n + 1) % p * ((p
70             → + 1) / 6) % p;
71     }); // x ^ 2
72
73     for (int i = 1; i <= val[0]; i++)
74         g[i] = dp[i];
75
76     get_dp([] (long long x) {
77         return x % p;
78     });
79
80     memset(prime_sum, 0, sizeof(prime_sum));
81     for (int i = 1; i <= prime_cnt; i++)
82         prime_sum[i] = (prime_sum[i - 1] + f(prime[i]))
83             → % p;
84 }
85
86 int S(int i, long long n) {
87     if (prime[i] > n || n < 1)
88         return 0;
89
90     int sq = sqrt(n + 0.5);
91     int tmp = (g[getid(n)] - prime_sum[i - 1] + p) % p;
92
93     for (int k = i; k <= prime_cnt && prime[k] <= sq;
94         → k++) {
95         int pk = prime[k];
96         long long pw = pk;
97
98         for (int c = 1; pw * pk <= n; c++, pw *= pk)
99             tmp = (tmp + (long long)S(k + 1, n / pw) *
100                 → f(pw) + f(pw * pk)) % p;
101    }
102 }
103
104 int main() {
105     long long n;
106     cin >> n;
107
108     sqrtn = 1; // sqrtn 是全局的
109     while ((long long)(sqrtn + 1) * (sqrtn + 1) <= n)
110         sqrtn++;
111
112     get_table(sqrtn);
113
114     calc(n);
115
116     int ans = (S(1, n) + 1) % p;
117
118     cout << ans << endl;
119
120     return 0;
121 }
122 }
```

```

73     }, [] (long long n) {
74         n %= p;
75         return n * (n + 1) / 2 % p;
76    }); // x
77
78     for (int i = 1; i <= val[0]; i++)
79         g[i] = (g[i] - dp[i] + p) % p;
80
81     memset(prime_sum, 0, sizeof(prime_sum));
82     for (int i = 1; i <= prime_cnt; i++)
83         prime_sum[i] = (prime_sum[i - 1] + f(prime[i]))
84             → % p;
85
86     int S(int i, long long n) {
87         if (prime[i] > n || n < 1)
88             return 0;
89
90         int sq = sqrt(n + 0.5);
91         int tmp = (g[getid(n)] - prime_sum[i - 1] + p) % p;
92
93         for (int k = i; k <= prime_cnt && prime[k] <= sq;
94             → k++) {
95             int pk = prime[k];
96             long long pw = pk;
97
98             for (int c = 1; pw * pk <= n; c++, pw *= pk)
99                 tmp = (tmp + (long long)S(k + 1, n / pw) *
100                     → f(pw) + f(pw * pk)) % p;
101        }
102    }
103
104 int main() {
105     long long n;
106     cin >> n;
107
108     sqrtn = 1; // sqrtn 是全局的
109     while ((long long)(sqrtn + 1) * (sqrtn + 1) <= n)
110         sqrtn++;
111
112     get_table(sqrtn);
113
114     calc(n);
115
116     int ans = (S(1, n) + 1) % p;
117
118     cout << ans << endl;
119
120     return 0;
121 }
122 }
```

2.7 Miller-Rabin

```

1 // 复杂度可以认为是常数
2
3 // 用一个数检测
4 // 需要调用long long快速幂和O(1) 快速乘
5 bool check(long long n, long long b) { // b: base
6     long long a = n - 1;
7     int k = 0;
8
9     while (a % 2 == 0) {
10         a /= 2;
11         k++;
12     }
13 }
```

```

14     long long t = qpow(b, a, n); // 这里的快速幂函数需要
15         → 写 O(1) 快速乘
16     if (t == 1 || t == n - 1)
17         return true;
18
19     while (k--) {
20         t = mul(t, t, n); // mul是O(1) 快速乘函数
21         if (t == n - 1)
22             return true;
23     }
24
25     return false;
26 }
27
28 // 封装好的函数体
29 // 需要调用check
30 bool Miller_Rabin(long long n) {
31     if (n == 1)
32         return false;
33     if (n == 2)
34         return true;
35     if (n % 2 == 0)
36         return false;
37
38     // int范围内只需要检查 {2, 7, 61}
39     // long long范围内只需要检查 {2, 325, 9375, 28178,
40     // → 450775, 9780504, 1795265022}
41
42     for (int i : {2, 325, 9375, 28178, 450775, 9780504,
43     // → 1795265022}) {
44         if (i >= n)
45             break;
46         if (!check(n, i))
47             return false;
48     }
49
50     return true;
51 }
```

```

25     do
26         p = Pollards_Rho(n);
27     while (!p); // p是任意一个非平凡因子
28
29     if (p == n) {
30         v.push_back(p); // 说明n本身就是质数
31         return;
32     }
33
34     solve(p, v); // 递归分解两半
35     solve(n / p, v);
36 }
37
38 // Pollard's Rho主过程
39 // 需要使用Miller-Rabin作为子算法
40 // 同时需要调用O(1) 快速乘和gcd函数
41 long long Pollards_Rho(long long n) {
42     // assert(n > 1);
43
44     if (Miller_Rabin(n))
45         return n;
46
47     long long c = rand() % (n - 2) + 1, i = 1, k = 2, x
48         → = rand() % (n - 3) + 2, u = 2; // 注意这里rand函
49         → 数需要重定义一下
50     while (true) {
51         i++;
52         x = (mul(x, x, n) + c) % n; // mul是O(1) 快速乘
53         → 函数
54
55         long long g = gcd((u - x + n) % n, n);
56         if (g > 1 && g < n)
57             return g;
58
59         if (u == x)
60             return 0; // 失败, 需要重新调用
61
62         if (i == k) {
63             u = x;
64             k *= 2;
65         }
66     }
67 }
```

2.8 Pollard's Rho

```

1 // 注意, 虽然Pollard's Rho的理论复杂度是O(n ^ {1 / 4})
2     → 的,
3 // 但实际跑起来比较慢, 一般用于做long long范围内的质因数
4     → 分解
5
6 // 封装好的函数体
7 // 需要调用solve
8 void factorize(long long n, vector<long long> &v) { //
9     → v用于存分解出来的质因子, 重复的会放多个
10    for (int i : {2, 3, 5, 7, 11, 13, 17, 19})
11        while (n % i == 0) {
12            v.push_back(i);
13            n /= i;
14        }
15
16        solve(n, v);
17        sort(v.begin(), v.end()); // 从小到大排序后返回
18    }
19
20 // 递归过程
21 // 需要调用Pollard's Rho主过程, 同时递归调用自身
22 void solve(long long n, vector<long long> &v) {
23     if (n == 1)
24         return;
25
26     long long p;
```

2.9 快速阶乘算法

参见1.1.5.多点求值应用: $O(\sqrt{n} \log^2 n)$ 快速求阶乘 (第4页).

2.10 扩展欧几里德 exgcd

```

1 void exgcd(LL a, LL b, LL &c, LL &x, LL &y) {
2     if (b == 0) {
3         c = a;
4         x = 1;
5         y = 0;
6         return;
7     }
8
9     exgcd(b, a % b, c, x, y);
10
11     LL tmp = x;
12     x = y;
13     y = tmp - (a / b) * y;
14 }
```

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

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

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)

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

```

1 int solve(int n, int a, int b, int m) {
2     if (!b)
3         return n * (a / m);
4     if (a >= m)
5         return n * (a / m) + solve(n, a % m, b, m);
6     if (b >= m)
7         return (n - 1) * n / 2 * (b / m) + solve(n, a,
8             → b % m, m);
9
10    return solve((a + b * n) / m, (a + b * n) % m, m,
11        → b);
12 }
```

2.11 中国剩余定理

$$x \equiv a_i \pmod{m_i}$$

$$M = \prod_i m_i, M_i = \frac{M}{m_i}$$

$$M'_i \equiv M_i^{-1} \pmod{m_i}$$

$$x \equiv \sum_i a_i M_i M'_i \pmod{M}$$

2.11.1 ex-CRT

设两个方程分别是 $x \equiv a_1 \pmod{m_1}$ 和 $x \equiv a_2 \pmod{m_2}$.

将它们转化为不定方程 $x = m_1 p + a_1 = m_2 q + a_2$, 其中 p, q 是整数, 则有 $m_1 p - m_2 q = a_2 - a_1$.

当 $a_2 - a_1$ 不能被 $\gcd(m_1, m_2)$ 整除时无解, 否则可以通过扩展欧几里德解出来一组可行解 (p, q) .

则原来的两方程组成的模方程组的解为 $x \equiv b \pmod{M}$, 其中 $b = m_1 p + a_1$, $M = \text{lcm}(m_1, m_2)$.

2.12 二次剩余

```

1 int p, w;
2
3 struct pi {
4     int a, b; // a + b * sqrt(w)
5
6     pi(int a = 0, int b = 0) : a(a), b(b) {}
7
8     friend pi operator * (const pi &u, const pi &v) {
9         return pi(((long long)u.a * v.a + (long
10            → long)u.b * v.b % p * w) % p,
11            ((long long)u.a * v.b + (long long)u.b *
12            → v.a) % p);
13 }
14 pi qpow(pi a, int b) {
15     pi ans(1, 0);
16 }
```

```

16
17     while (b) {
18         if (b & 1)
19             ans = ans * a;
20
21         b >>= 1;
22         a = a * a;
23     }
24
25     return ans;
26 }
27
28 int qpow(int a, int b) {
29     int ans = 1;
30
31     while (b) {
32         if (b & 1)
33             ans = (long long)ans * a % p;
34
35         b >>= 1;
36         a = (long long)a * a % p;
37     }
38
39     return ans;
40 }
41
42 int my_legendre(int a) { // std有同名函数, 最好换个名字,
43     ↪ 不然传了两个数都查不出来
44     return qpow(a, (p - 1) / 2);
45 }
46
47 int quadratic_residual(int b, int mod) {
48     p = mod;
49
50     if (p == 2)
51         return 1;
52
53     if (my_legendre(b) == p - 1)
54         return -1; // 无解
55
56     int a;
57     do {
58         a = rand() % p;
59         w = ((long long)a * a - b) % p;
60         if (w < 0)
61             w += p;
62     } while (my_legendre(w) != p - 1);
63
64     return qpow(pi(a, 1), (p + 1) / 2).a;
65 }
```

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 在阶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 $\gcd(\delta_p(a), \delta_p(b)) = 1$.

另外, 如果 a 与 p 互质, 则有 $\delta_p(a^k) = \frac{\delta_p(a)}{\gcd(\delta_p(a), k)}$. (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:

```

1 def split(n): # 分解质因数
2     i = 2
3     a = []
4     while i * i <= n:
5         if n % i == 0:
6             a.append(i)
7
8             while n % i == 0:
9                 n /= i
10
11            i += 1
12
13        if n > 1:
14            a.append(n)
15
16    return a
17
18 def getg(p): # 找原根
19     def judge(g):
20         for i in d:
21             if pow(g, (p - 1) / i, p) == 1:
22                 return False
23         return True
24
25     d = split(p - 1)
26     g = 2
27
28     while not judge(g):
29         g += 1
30
31     return g
32
33 print(getg(int(input())))

```

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

2.14 常用数论公式

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 降幂公式

$$a^k \equiv a^{k \bmod \varphi(p)+\varphi(p)}, k \geq \varphi(p)$$

2.14.3 其他常用公式

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

$$\varphi * I = id$$

$$\mu * id = \varphi$$

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

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

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

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

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

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 $+\infty$, 跑MST后把非树边删掉, 减少无用边
- Contraction: 待修改边标为 $-\infty$, 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 $(n, m) \rightarrow (n, n + k - 1) \rightarrow (k + 1, 2k)$.

```

1 // 全局结构体与数组定义
2 struct edge { // 边的定义
3     int u, v, w, id; // id表示边在原图中的编号
4     bool vis; // 在Kruskal时用, 记录这条边是否是树边
5     bool operator < (const edge &e) const { return w <
6         → e.w; }
7 } e[20][maxn], t[maxn]; // 为了便于回滚, 在每层分治存一
8 → 个副本
9
10 // 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
11 struct A {
12     int id, u, v;
13 } a[maxn];
14
15 int id[20][maxn]; // 每条边在当前图中的编号
16 int p[maxn], size[maxn], stk[maxn], top; // p和size是并
17 → 查集数组, stk是用来撤销的栈
18 int n, m, q; // 点数, 边数, 修改数
19
20 // 方便起见, 附上可能需要用到的预处理代码
21 int main() {
22     for (int i = 1; i <= n; i++) { // 并查集初始化
23         p[i] = i;
24         size[i] = 1;
25     }
26
27     for (int i = 1; i <= m; i++) { // 读入与预标号
28         scanf("%d%d%d", &e[0][i].u, &e[0][i].v, &e[0]
29             → [i].w);
30         e[0][i].id = i;
31         id[0][i] = i;
32     }
33
34     for (int i = 1; i <= q; i++) { // 预处理出调用数组
            scanf("%d%d", &a[i].id, &a[i].v);
        }
```

```

35     a[i].u = e[0][a[i].id].w;
36     e[0][a[i].id].w = a[i].v;
37 }
38
39 for(int i = q; i; i--)
40     e[0][a[i].id].w = a[i].u;
41
42 CDQ(1, q, 0, m, 0); // 这是调用方法
43 }

44
45 // 分治主过程 O(n log^2 n)
46 // 需要调用Reduction和Contraction
47 void CDQ(int l, int r, int d, int m, long long ans) {
48     ← // CDQ分治
49     if (l == r) { // 区间长度已减小到1, 输出答案, 退出
50         e[d][id[d][a[l].id]].w = a[l].v;
51         printf("%lld\n", ans + Kruskal(m, e[d]));
52         e[d][id[d][a[l].id]].w = a[l].u;
53         return;
54     }
55
56     int tmp = top;
57
58     Reduction(l, r, d, m);
59     ans += Contraction(l, r, d, m); // R-C
60
61     int mid = (l + r) / 2;
62
63     copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
64     for (int i = 1; i <= m; i++)
65         id[d + 1][e[d][i].id] = i; // 准备好下一层要用的
66 → 数组
67
68     CDQ(l, mid, d + 1, m, ans);
69
70     for (int i = l; i <= mid; i++)
71         e[d][id[d][a[i].id]].w = a[i].v; // 进行左边的修
72 → 改
73
74     copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
75     for (int i = 1; i <= m; i++)
76         id[d + 1][e[d][i].id] = i; // 重新准备下一层要用
77 → 的数组
78
79     CDQ(mid + 1, r, d + 1, m, ans);
80
81     for (int i = top; i > tmp; i--)
82         cut(stk[i]); // 撤销所有操作
83     top = tmp;
84
85 // Reduction(减少无用边): 待修改边标为+INF, 跑MST后把非树
86 → 边删掉, 减少无用边
87 // 需要调用Kruskal
88 void Reduction(int l, int r, int d, int &m) {
89     for (int i = l; i <= r; i++)
90         e[d][id[d][a[i].id]].w = INF; // 待修改的边标为INF
91
92     Kruskal(m, e[d]);
93
94     copy(e[d] + 1, e[d] + m + 1, t + 1);
95
96     int cnt = 0;
97     for (int i = 1; i <= m; i++)
98         if (t[i].w == INF || t[i].vis){ // 非树边扔掉
99             id[d][t[i].id] = ++cnt; // 给边重新编号
100            e[d][cnt] = t[i];
101        }
102 }
```

```

100
101    for (int i = r; i >= l; i--)
102        e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边
103        ↪ 改回去
104
105    m = cnt;
106
107
108 // Contraction(缩必须边): 待修改边标为- $\text{INF}$ , 跑MST后缩除待
109 // → 修改边之外的所有树边
110 // 返回缩掉的边的总权值
111 // 需要调用Kruskal
112 long long Contraction(int l, int r, int d, int &m) {
113     long long ans = 0;
114
115     for (int i = l; i <= r; i++)
116         e[d][id[d][a[i].id]].w = - $\text{INF}$ ; // 待修改边标
117         ↪ 为- $\text{INF}$ 
118
119     Kruskal(m, e[d]);
120     copy(e[d] + 1, e[d] + m + 1, t + 1);
121
122     int cnt = 0;
123     for (int i = 1; i <= m; i++) {
124
125         if (t[i].w != - $\text{INF}$  && t[i].vis) { // 必须边
126             ans += t[i].w;
127             mergeset(t[i].u, t[i].v);
128         }
129         else { // 不确定边
130             id[d][t[i].id] = ++cnt;
131             e[d][cnt] = t[i];
132         }
133
134     for (int i = r; i >= l; i--) {
135         e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边
136         ↪ 改回去
137         e[d][id[d][a[i].id]].vis = false;
138
139     m = cnt;
140
141     return ans;
142
143
144 // Kruskal算法  $O(m \log n)$ 
145 // 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
146 // → 撤销即可
147 long long Kruskal(int m, edge *e) {
148     int tmp = top;
149     long long ans = 0;
150
151     sort(e + 1, e + m + 1); // 比较函数在结构体中定义过
152     ↪ 了
153
154     for (int i = 1; i <= m; i++) {
155         if (findroot(e[i].u) != findroot(e[i].v)) {
156             e[i].vis = true;
157             ans += e[i].w;
158             mergeset(e[i].u, e[i].v);
159         }
160         else
161             e[i].vis = false;
162
163     for (int i = top; i > tmp; i--)
164         cut(stk[i]); // 撤销所有操作
165
166     top = tmp;
167
168
169 // 以下是并查集相关函数
170 int findroot(int x) { // 因为需要撤销, 不写路径压缩
171     while (p[x] != x)
172         x = p[x];
173
174     return x;
175
176 }
177
178 void mergeset(int x, int y) { // 按size合并, 如果想跑得
179 // → 更快就写一个按秩合并
180     x = findroot(x); // 但是按秩合并要再开一个栈记录合并
181     ↪ 之前的秩
182     y = findroot(y);
183
184     if (x == y)
185         return;
186
187     if (size[x] > size[y])
188         swap(x, y);
189
190     p[x] = y;
191     size[y] += size[x];
192     stk[++top] = x;
193
194
195 void cut(int x) { // 并查集撤销
196     int y = x;
197
198     do
199         size[y = p[y]] -= size[x];
200     while (p[y] != y);
201
202     p[x] = x;
203 }

```

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

```

1 constexpr int maxn = 105, maxe = 10005, inf =
2     ↪ 0x3f3f3f3f;
3
4 struct edge {
5     int u, v, w;
6 } e[maxe];
7
8 int mn[maxn], pr[maxn], ufs[maxn], vis[maxn];
9 bool alive[maxn];
10
11 int edmonds(int n, int m, int rt) {
12     for (int i = 1; i <= n; i++)
13         alive[i] = true;
14
15     int ans = 0;
16
17     while (true) {
18         memset(mn, 63, sizeof(int) * (n + 1));
19         memset(pr, 0, sizeof(int) * (n + 1));
20         memset(ufs, 0, sizeof(int) * (n + 1));
21
22         for (int i = 1; i <= m; i++)
23             if (mn[e[i].u] > e[i].w)
24                 mn[e[i].v] = e[i].w;
25
26         for (int i = 1; i <= n; i++)
27             if (mn[i] == 63)
28                 break;
29
30         if (i == n)
31             return ans;
32
33         for (int i = 1; i <= m; i++)
34             if (mn[e[i].u] > e[i].w)
35                 mn[e[i].v] = e[i].w;
36
37         for (int i = 1; i <= n; i++)
38             if (mn[i] == 63)
39                 break;
40
41         if (i == n)
42             return ans;
43
44         for (int i = 1; i <= m; i++)
45             if (mn[e[i].u] > e[i].w)
46                 mn[e[i].v] = e[i].w;
47
48         for (int i = 1; i <= n; i++)
49             if (mn[i] == 63)
50                 break;
51
52         if (i == n)
53             return ans;
54
55         for (int i = 1; i <= m; i++)
56             if (mn[e[i].u] > e[i].w)
57                 mn[e[i].v] = e[i].w;
58
59         for (int i = 1; i <= n; i++)
60             if (mn[i] == 63)
61                 break;
62
63         if (i == n)
64             return ans;
65
66         for (int i = 1; i <= m; i++)
67             if (mn[e[i].u] > e[i].w)
68                 mn[e[i].v] = e[i].w;
69
70         for (int i = 1; i <= n; i++)
71             if (mn[i] == 63)
72                 break;
73
74         if (i == n)
75             return ans;
76
77         for (int i = 1; i <= m; i++)
78             if (mn[e[i].u] > e[i].w)
79                 mn[e[i].v] = e[i].w;
80
81         for (int i = 1; i <= n; i++)
82             if (mn[i] == 63)
83                 break;
84
85         if (i == n)
86             return ans;
87
88         for (int i = 1; i <= m; i++)
89             if (mn[e[i].u] > e[i].w)
90                 mn[e[i].v] = e[i].w;
91
92         for (int i = 1; i <= n; i++)
93             if (mn[i] == 63)
94                 break;
95
96         if (i == n)
97             return ans;
98
99         for (int i = 1; i <= m; i++)
100            if (mn[e[i].u] > e[i].w)
101                mn[e[i].v] = e[i].w;
102
103         for (int i = 1; i <= n; i++)
104             if (mn[i] == 63)
105                 break;
106
107         if (i == n)
108             return ans;
109
110         for (int i = 1; i <= m; i++)
111             if (mn[e[i].u] > e[i].w)
112                 mn[e[i].v] = e[i].w;
113
114         for (int i = 1; i <= n; i++)
115             if (mn[i] == 63)
116                 break;
117
118         if (i == n)
119             return ans;
120
121         for (int i = 1; i <= m; i++)
122             if (mn[e[i].u] > e[i].w)
123                 mn[e[i].v] = e[i].w;
124
125         for (int i = 1; i <= n; i++)
126             if (mn[i] == 63)
127                 break;
128
129         if (i == n)
130             return ans;
131
132         for (int i = 1; i <= m; i++)
133             if (mn[e[i].u] > e[i].w)
134                 mn[e[i].v] = e[i].w;
135
136         for (int i = 1; i <= n; i++)
137             if (mn[i] == 63)
138                 break;
139
140         if (i == n)
141             return ans;
142
143
144 // Kruskal算法  $O(m \log n)$ 
145 // 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
146 // → 撤销即可
147 long long Kruskal(int m, edge *e) {
148     int tmp = top;
149     long long ans = 0;
150
151     sort(e + 1, e + m + 1); // 比较函数在结构体中定义过
152     ↪ 了
153
154     for (int i = 1; i <= m; i++) {
155         if (findroot(e[i].u) != findroot(e[i].v)) {
156             e[i].vis = true;
157             ans += e[i].w;
158             mergeset(e[i].u, e[i].v);
159         }
160         else
161             e[i].vis = false;
162
163     for (int i = top; i > tmp; i--)
164         cut(stk[i]); // 撤销所有操作
165
166     top = tmp;
167
168
169 // 以下是并查集相关函数
170 int findroot(int x) { // 因为需要撤销, 不写路径压缩
171     while (p[x] != x)
172         x = p[x];
173
174     return x;
175
176 }
177
178 void mergeset(int x, int y) { // 按size合并, 如果想跑得
179 // → 更快就写一个按秩合并
180     x = findroot(x); // 但是按秩合并要再开一个栈记录合并
181     ↪ 之前的秩
182     y = findroot(y);
183
184     if (x == y)
185         return;
186
187     if (size[x] > size[y])
188         swap(x, y);
189
190     p[x] = y;
191     size[y] += size[x];
192     stk[++top] = x;
193
194
195 void cut(int x) { // 并查集撤销
196     int y = x;
197
198     do
199         size[y = p[y]] -= size[x];
200     while (p[y] != y);
201
202     p[x] = x;
203 }

```

```

20     memset(vis, 0, sizeof(int) * (n + 1));
21
22     mn[rt] = 0;
23
24     for (int i = 1; i <= m; i++) {
25         if (e[i].u != e[i].v && e[i].w <
26             → mn[e[i].v]) {
27             mn[e[i].v] = e[i].w;
28             pr[e[i].v] = e[i].u;
29         }
30
31         for (int i = 1; i <= n; i++) {
32             if (alive[i]) {
33                 if (mn[i] >= inf)
34                     return -1; // 不存在最小树形图
35
36             ans += mn[i];
37         }
38
39         bool flag = false;
40
41         for (int i = 1; i <= n; i++) {
42             if (!alive[i])
43                 continue;
44
45             int x = i;
46             while (x && !vis[x]) {
47                 vis[x] = i;
48                 x = pr[x];
49             }
50
51             if (x && vis[x] == i) {
52                 flag = true;
53                 for (int u = x; !ufs[u]; u = pr[u])
54                     ufs[u] = x;
55             }
56         }
57
58         for (int i = 1; i <= m; i++) {
59             e[i].w -= mn[e[i].v];
60
61             if (ufs[e[i].u])
62                 e[i].u = ufs[e[i].u];
63             if (ufs[e[i].v])
64                 e[i].v = ufs[e[i].v];
65
66             if (!flag)
67                 return ans;
68
69             for (int i = 1; i <= n; i++)
70                 if (ufs[i] && i != ufs[i])
71                     alive[i] = false;
72     }
73 }
```

$O(m \log n)$ 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:

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

2. 新加一条边:

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

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 $m \log n$.

```

1 constexpr int maxn = 105, inf = 0x3f3f3f3f;
2
3 int dp[maxn][(1 << 10) + 1];
4 int g[maxn][maxn], a[15];
5 bool inq[maxn];
6
7 int main() {
8
9     int n, m, k;
10    scanf("%d%d%d", &n, &m, &k);
11
12    memset(g, 63, sizeof(g));
13
14    while (m--) {
15        int u, v, c;
16        scanf("%d%d%d", &u, &v, &c);
17
18        g[u][v] = g[v][u] = min(g[u][v], c); // 不要忘了
19        → 是双向边
20    }
21
22    memset(dp, 63, sizeof(dp));
23
24    for (int i = 0; i < k; i++) {
25        scanf("%d", &a[i]);
26
27        dp[a[i]][1 << i] = 0;
28    }
29
30    for (int s = 1; s < (1 << k); s++) {
31        for (int i = 1; i <= n; i++) {
32            for (int t = (s - 1) & s; t; (--t) &= s)
33                dp[i][s] = min(dp[i][s], dp[i][t] +
34                    → dp[i][s ^ t]);
35
36            // SPFA
37            queue<int> q;
38            for (int i = 1; i <= n; i++) {
39                if (dp[i][s] < inf) {
40                    q.push(i);
41                    inq[i] = true;
42                }
43
44            while (!q.empty()) {
45                int i = q.front();
46                q.pop();
47                inq[i] = false; // 最终结束时inq一定全0, 所
48                → 以不用清空
49
50                for (int j = 1; j <= n; j++) {
51                    if (dp[i][s] + g[i][j] < dp[j][s]) {
52                        dp[j][s] = dp[i][s] + g[i][j];
53                        if (!inq[j]) {
54                            q.push(j);
55                            inq[j] = true;
56                        }
57                    }
58                }
59            }
60        }
61    }
62 }
```

```

57
58     int ans = inf;
59     for (int i = 1; i <= n; i++)
60         ans = min(ans, dp[i][(1 << k) - 1]);
61
62     printf("%d\n", ans);
63
64     return 0;
65 }
```

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.

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 505;
6 constexpr long long inf = 0x3f3f3f3f3f3f3f3fll;
7
8 int g[maxn][maxn], id[maxn][maxn], pr[maxn]; // g是邻接
9 → 矩阵
10 long long f[maxn][maxn], d[maxn];
11 bool vis[maxn];
12
13 vector<pair<int, int>>
14 → minimum_diameter_spanning_tree(int n) { // 1-based
15     for (int i = 1; i <= n; i++)
16         for (int j = 1; j <= n; j++)
17             g[i][j] *= 2; // 输入的边权都要乘2
18
19     memset(f, 63, sizeof(f));
20
21     for (int i = 1; i <= n; i++)
22         f[i][i] = 0;
23
24     for (int i = 1; i <= n; i++)
25         for (int j = 1; j <= n; j++)
26             if (g[i][j])
27                 f[i][j] = g[i][j];
28
29     for (int k = 1; k <= n; k++)
30         for (int i = 1; i <= n; i++)
31             for (int j = 1; j <= n; j++)
32                 f[i][j] = min(f[i][j], f[i][k] + f[k]
33 → [j]);
34
35     for (int i = 1; i <= n; i++) {
36         for (int j = 1; j <= n; j++)
37             id[i][j] = j; // 距离i第j近的点
38
39         sort(id[i] + 1, id[i] + n + 1, [&i] (int x, int
40 → y) {
41             return f[i][x] < f[i][y];
42         });
43     }
44
45     int o = 0;
46     long long ansv = inf; // vertex
47
48     for (int i = 1; i <= n; i++)
49         if (f[i][id[i][n]] * 2 < ansv) {
50             ansv = f[i][id[i][n]] * 2;
51             o = i;
52         }
53
54     int u = 0, v = 0;
55     long long disu = -inf, disv = -inf, anse = inf;
```

```

52
53     for (int x = 1; x <= n; x++)
54         for (int y = 1; y <= n; y++)
55             if (g[x][y]) { // 如果 g[x][y] = 0 说明没有边
56                 int w = g[x][y];
57
58                 for (int i = n - 1, j = n; i; i--)
59                     if (f[y][id[x][i]] > f[y][id[x]
60 → [j]]) {
61                         long long tmp = f[x][id[x][i]]
62 → + f[y][id[x][j]] + w;
63                         if (tmp < anse) {
64                             anse = tmp;
65                             u = x;
66                             v = y;
67
68                             disu = tmp / 2 - f[x][id[x]
69 → [i]];
69                             disv = w - disu;
70                         }
71                     }
72
73             printf("%lld\n", min(ansv, anse) / 2); // 直径
74
75             memset(d, 63, sizeof(d));
76
77             if (ansv <= anse)
78                 d[o] = 0;
79             else {
80                 d[u] = disu;
81                 d[v] = disv;
82             }
83
84             for (int k = 1; k <= n; k++) { // Dijkstra
85                 int x = 0;
86                 for (int i = 1; i <= n; i++)
87                     if (!vis[i] && d[i] < d[x])
88                         x = i;
89
90                 vis[x] = true;
91                 for (int y = 1; y <= n; y++)
92                     if (g[x][y] && !vis[y]) {
93                         if (d[y] > d[x] + g[x][y]) {
94                             d[y] = d[x] + g[x][y];
95                             pr[y] = x;
96                         }
97                     else if (d[y] == d[x] + g[x][y] &&
98 → d[pr[y]] < d[x])
99                         pr[y] = x;
100
101             vector<pair<int, int>> vec;
102             for (int i = 1; i <= n; i++)
103                 if (pr[i])
104                     vec.emplace_back(i, pr[i]);
105
106             if (ansv > anse)
107                 vec.emplace_back(u, v);
108
109             return vec;
110         }
111     }
112
113     int main() {
114
115         int n, m;
```

```

116     scanf("%d%d", &n, &m);
117
118     while (m--) {
119         int x, y, z;
120         scanf("%d%d%d", &x, &y, &z);
121
122         g[x][y] = g[y][x] = z; // 无向图
123     }
124
125     auto vec = minimum_diameter_spanning_tree(n);
126     for (auto [x, y] : vec)
127         printf("%d %d\n", x, y);
128
129     return 0;
130 }
```

3.2 最短路

3.2.1 Dijkstra

参见3.2.3.k 短路(第 30 页), 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 , 跑一遍SPFA, 计算每个点的势 $h_u = d_{s,u}$, 然后将每条边 $u \rightarrow v$ 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

```

1 // 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
2 // →最好还是用A*
3 // DAG和有环的情况都可以, 有重边或自环也无所谓, 但不能有
4 // →零环
5 // 以下代码以Dijkstra + 可持久化左偏树为例
6
7 constexpr int maxn = 1005, maxe = 10005, maxm = maxe *
8 // → 30; // 点数, 边数, 左偏树结点数
9
10 struct A { // 用来求最短路
11     int x, d;
12
13     A(int x, int d) : x(x), d(d) {}
14
15     bool operator < (const A &a) const {
16         return d > a.d;
17     }
18
19 struct node { // 左偏树结点
20     int w, i, d; // i: 最后一条边的编号 d: 左偏树附加信息
21     node *lc, *rc;
22
23     node() {}
24
25     node(int w, int i) : w(w), i(i), d(0) {}
26
27     void refresh(){
28         d = rc -> d + 1;
29     }
30     null[maxm], *ptr = null, *root[maxn];
31
32 struct B { // 维护答案用
33     int x, w; // x是结点编号, w表示之前已经产生的权值
```

```

33     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
34
35     B(int x, node *rt, int w) : x(x), w(w), rt(rt) {}
36
37     bool operator < (const B &a) const {
38         return w + rt -> w > a.w + a.rt -> w;
39     }
40 }
41
42 // 全局变量和数组定义
43 vector<int> G[maxn], W[maxn], id[maxn]; // 最开始要存反
44 // →向图, 然后把G清空作为儿子列表
45 bool vis[maxn], used[maxe]; // used表示边是否在最短路树
46 // →上
47 int u[maxe], v[maxe], w[maxe]; // 存下每条边, 注意是有向
48 // →边
49 int d[maxn], p[maxn]; // p表示最短路树上每个点的父边
50 int n, m, k, s, t; // s, t分别表示起点和终点
51
52 // 以下是主函数中较关键的部分
53 for (int i = 0 ; i <= n; i++)
54     root[i] = null; // 一定要加上!!!
55
56 // (读入& 建反向图)
57 Dijkstra();
58
59 // (清空G, W, id)
60 for (int i = 1; i <= n; i++)
61     if (p[i]) {
62         used[p[i]] = true; // 在最短路树上
63         G[v[p[i]]].push_back(i);
64     }
65
66 for (int i = 1; i <= m; i++) {
67     w[i] -= d[u[i]] - d[v[i]]; // 现在的w[i] 表示这条边
68 // →能使路径长度增加多少
69     if (!used[i])
70         root[u[i]] = merge(root[u[i]], newnode(w[i],
71 // →i));
72
73 dfs(t);
74 priority_queue<B> heap;
75 heap.push(B(s, root[s], 0)); // 初始状态是找贡献最小的边
76 // →加进去
77 printf("%d\n", d[s]); // 第1短路需要特判
78 while (--k) { // 其余k - 1短路径用二叉堆维护
79     if (heap.empty())
80         printf("-1\n");
81     else {
82         int x = heap.top().x, w = heap.top().w;
83         node *rt = heap.top().rt;
84         heap.pop();
85
86         printf("%d\n", d[s] + w + rt -> w);
87
88         if (rt -> lc != null || rt -> rc != null)
89             heap.push(B(x, merge(rt -> lc, rt -> rc,
90 // →w))); // pop掉当前边, 换成另一条贡献大
91 // →点的边
92         if (root[v[rt -> i]] != null)
93             heap.push(B(v[rt -> i], root[v[rt -> i]], w
94 // →+ rt -> w)); // 保留当前边, 往后面再接上
95 // →另一条边
96     }
97 }
```

```

93 }
94 // 主函数到此结束
95
96
97 // Dijkstra预处理最短路 O(m\log n)
98 void Dijkstra() {
99     memset(d, 63, sizeof(d));
100    d[t] = 0;
101    priority_queue<A> heap;
102    heap.push(A(t, 0));
103
104    while (!heap.empty()) {
105        int x = heap.top().x;
106        heap.pop();
107
108        if(vis[x])
109            continue;
110
111        vis[x] = true;
112        for (int i = 0; i < (int)G[x].size(); i++)
113            if (!vis[G[x][i]] && d[G[x][i]] > d[x] +
114                → W[x][i]) {
115                d[G[x][i]] = d[x] + W[x][i];
116                p[G[x][i]] = id[x][i];
117
118                heap.push(A(G[x][i], d[G[x][i]]));
119            }
120    }
121
122 // dfs求出每个点的堆 总计O(m\log n)
123 // 需要调用merge, 同时递归调用自身
124 void dfs(int x) {
125     root[x] = merge(root[x], root[v[p[x]]]);
126
127     for (int i = 0; i < (int)G[x].size(); i++)
128         dfs(G[x][i]);
129 }
130
131 // 包装过的new node() O(1)
132 node *newnode(int w, int i) {
133     *ptr = node(w, i);
134     ptr -> lc = ptr -> rc = null;
135     return ptr;
136 }
137
138 // 带可持久化的左偏树合并 总计O(\log n)
139 // 递归调用自身
140 node *merge(node *x, node *y) {
141     if (x == null)
142         return y;
143     if (y == null)
144         return x;
145
146     if (x -> w > y -> w)
147         swap(x, y);
148
149     node *z = newnode(x -> w, x -> i);
150     z -> lc = x -> lc;
151     z -> rc = merge(x -> rc, y);
152
153     if (z -> lc -> d < z -> rc -> d)
154         swap(z -> lc, z -> rc);
155     z -> refresh();
156
157     return z;
158 }

```

3.3 Tarjan 算法

3.3.1 强连通分量

```

1 int dfn[maxn], low[maxn], tim = 0;
2 vector<int> G[maxn], scc[maxn];
3 int sccid[maxn], scc_cnt = 0, stk[maxn];
4 bool instk[maxn];
5
6 void dfs(int x) {
7     dfn[x] = low[x] = ++tim;
8
9     stk[++stk[0]] = x;
10    instk[x] = true;
11
12    for (int y : G[x]) {
13        if (!dfn[y]) {
14            dfs(y);
15            low[x] = min(low[x], low[y]);
16        }
17        else if (instk[y])
18            low[x] = min(low[x], dfn[y]);
19    }
20
21    if (dfn[x] == low[x]) {
22        scc_cnt++;
23
24        int u;
25        do {
26            u = stk[stk[0]--];
27            instk[u] = false;
28            sccid[u] = scc_cnt;
29            scc[scc_cnt].push_back(u);
30        } while (u != x);
31    }
32 }
33
34 void tarjan(int n) {
35     for (int i = 1; i <= n; i++)
36         if (!dfn[i])
37             dfs(i);
38 }

```

3.3.2 割点点双

```

1 vector<int> G[maxn], bcc[maxn];
2 int dfn[maxn], low[maxn], tim = 0, bccid[maxn], bcc_cnt
3     ←= 0;
4 bool iscut[maxn];
5
6 pair<int, int> stk[maxn];
7 int stk_cnt = 0;
8
9 void dfs(int x, int pr) {
10    int child = 0;
11    dfn[x] = low[x] = ++tim;
12
13    for (int y : G[x]) {
14        if (!dfn[y]) {
15            stk[++stk_cnt] = make_pair(x, y);
16            child++;
17            dfs(y, x);
18            low[x] = min(low[x], low[y]);
19
20            if (low[y] >= dfn[x]) {
21                iscut[x] = true;
22                bcc_cnt++;
23            }
24        }
25    }
26
27    if (child == 1 && iscut[x])
28        bcc[bccid[x]].push_back(x);
29
30    while (true) {
31        if (stk[stk_cnt] == make_pair(x, y))
32            break;
33        stk[stk_cnt] = make_pair(y, x);
34        child--;
35    }
36
37    if (child == 0)
38        bcc[bccid[x]].push_back(x);
39
40    if (child == 1)
41        bcc[bccid[x]].push_back(y);
42
43    if (child == 0)
44        bcc[bccid[x]].push_back(y);
45
46    if (child == 1)
47        bcc[bccid[x]].push_back(x);
48
49    if (child == 0)
50        bcc[bccid[x]].push_back(x);
51
52    if (child == 1)
53        bcc[bccid[x]].push_back(y);
54
55    if (child == 0)
56        bcc[bccid[x]].push_back(y);
57
58    if (child == 1)
59        bcc[bccid[x]].push_back(x);
60
61    if (child == 0)
62        bcc[bccid[x]].push_back(x);
63
64    if (child == 1)
65        bcc[bccid[x]].push_back(y);
66
67    if (child == 0)
68        bcc[bccid[x]].push_back(y);
69
70    if (child == 1)
71        bcc[bccid[x]].push_back(x);
72
73    if (child == 0)
74        bcc[bccid[x]].push_back(x);
75
76    if (child == 1)
77        bcc[bccid[x]].push_back(y);
78
79    if (child == 0)
80        bcc[bccid[x]].push_back(y);
81
82    if (child == 1)
83        bcc[bccid[x]].push_back(x);
84
85    if (child == 0)
86        bcc[bccid[x]].push_back(x);
87
88    if (child == 1)
89        bcc[bccid[x]].push_back(y);
90
91    if (child == 0)
92        bcc[bccid[x]].push_back(y);
93
94    if (child == 1)
95        bcc[bccid[x]].push_back(x);
96
97    if (child == 0)
98        bcc[bccid[x]].push_back(x);
99
100   if (child == 1)
101      bcc[bccid[x]].push_back(y);
102
103   if (child == 0)
104      bcc[bccid[x]].push_back(y);
105
106   if (child == 1)
107      bcc[bccid[x]].push_back(x);
108
109   if (child == 0)
110      bcc[bccid[x]].push_back(x);
111
112   if (child == 1)
113      bcc[bccid[x]].push_back(y);
114
115   if (child == 0)
116      bcc[bccid[x]].push_back(y);
117
118   if (child == 1)
119      bcc[bccid[x]].push_back(x);
120
121   if (child == 0)
122      bcc[bccid[x]].push_back(x);
123
124   if (child == 1)
125      bcc[bccid[x]].push_back(y);
126
127   if (child == 0)
128      bcc[bccid[x]].push_back(y);
129
130   if (child == 1)
131      bcc[bccid[x]].push_back(x);
132
133   if (child == 0)
134      bcc[bccid[x]].push_back(x);
135
136   if (child == 1)
137      bcc[bccid[x]].push_back(y);
138
139   if (child == 0)
140      bcc[bccid[x]].push_back(y);
141
142   if (child == 1)
143      bcc[bccid[x]].push_back(x);
144
145   if (child == 0)
146      bcc[bccid[x]].push_back(x);
147
148   if (child == 1)
149      bcc[bccid[x]].push_back(y);
150
151   if (child == 0)
152      bcc[bccid[x]].push_back(y);
153
154   if (child == 1)
155      bcc[bccid[x]].push_back(x);
156
157   if (child == 0)
158      bcc[bccid[x]].push_back(x);
159
160   if (child == 1)
161      bcc[bccid[x]].push_back(y);
162
163   if (child == 0)
164      bcc[bccid[x]].push_back(y);
165
166   if (child == 1)
167      bcc[bccid[x]].push_back(x);
168
169   if (child == 0)
170      bcc[bccid[x]].push_back(x);
171
172   if (child == 1)
173      bcc[bccid[x]].push_back(y);
174
175   if (child == 0)
176      bcc[bccid[x]].push_back(y);
177
178   if (child == 1)
179      bcc[bccid[x]].push_back(x);
180
181   if (child == 0)
182      bcc[bccid[x]].push_back(x);
183
184   if (child == 1)
185      bcc[bccid[x]].push_back(y);
186
187   if (child == 0)
188      bcc[bccid[x]].push_back(y);
189
190   if (child == 1)
191      bcc[bccid[x]].push_back(x);
192
193   if (child == 0)
194      bcc[bccid[x]].push_back(x);
195
196   if (child == 1)
197      bcc[bccid[x]].push_back(y);
198
199   if (child == 0)
200      bcc[bccid[x]].push_back(y);
201
202   if (child == 1)
203      bcc[bccid[x]].push_back(x);
204
205   if (child == 0)
206      bcc[bccid[x]].push_back(x);
207
208   if (child == 1)
209      bcc[bccid[x]].push_back(y);
210
211   if (child == 0)
212      bcc[bccid[x]].push_back(y);
213
214   if (child == 1)
215      bcc[bccid[x]].push_back(x);
216
217   if (child == 0)
218      bcc[bccid[x]].push_back(x);
219
220   if (child == 1)
221      bcc[bccid[x]].push_back(y);
222
223   if (child == 0)
224      bcc[bccid[x]].push_back(y);
225
226   if (child == 1)
227      bcc[bccid[x]].push_back(x);
228
229   if (child == 0)
230      bcc[bccid[x]].push_back(x);
231
232   if (child == 1)
233      bcc[bccid[x]].push_back(y);
234
235   if (child == 0)
236      bcc[bccid[x]].push_back(y);
237
238   if (child == 1)
239      bcc[bccid[x]].push_back(x);
240
241   if (child == 0)
242      bcc[bccid[x]].push_back(x);
243
244   if (child == 1)
245      bcc[bccid[x]].push_back(y);
246
247   if (child == 0)
248      bcc[bccid[x]].push_back(y);
249
250   if (child == 1)
251      bcc[bccid[x]].push_back(x);
252
253   if (child == 0)
254      bcc[bccid[x]].push_back(x);
255
256   if (child == 1)
257      bcc[bccid[x]].push_back(y);
258
259   if (child == 0)
260      bcc[bccid[x]].push_back(y);
261
262   if (child == 1)
263      bcc[bccid[x]].push_back(x);
264
265   if (child == 0)
266      bcc[bccid[x]].push_back(x);
267
268   if (child == 1)
269      bcc[bccid[x]].push_back(y);
270
271   if (child == 0)
272      bcc[bccid[x]].push_back(y);
273
274   if (child == 1)
275      bcc[bccid[x]].push_back(x);
276
277   if (child == 0)
278      bcc[bccid[x]].push_back(x);
279
280   if (child == 1)
281      bcc[bccid[x]].push_back(y);
282
283   if (child == 0)
284      bcc[bccid[x]].push_back(y);
285
286   if (child == 1)
287      bcc[bccid[x]].push_back(x);
288
289   if (child == 0)
290      bcc[bccid[x]].push_back(x);
291
292   if (child == 1)
293      bcc[bccid[x]].push_back(y);
294
295   if (child == 0)
296      bcc[bccid[x]].push_back(y);
297
298   if (child == 1)
299      bcc[bccid[x]].push_back(x);
300
301   if (child == 0)
302      bcc[bccid[x]].push_back(x);
303
304   if (child == 1)
305      bcc[bccid[x]].push_back(y);
306
307   if (child == 0)
308      bcc[bccid[x]].push_back(y);
309
310   if (child == 1)
311      bcc[bccid[x]].push_back(x);
312
313   if (child == 0)
314      bcc[bccid[x]].push_back(x);
315
316   if (child == 1)
317      bcc[bccid[x]].push_back(y);
318
319   if (child == 0)
320      bcc[bccid[x]].push_back(y);
321
322   if (child == 1)
323      bcc[bccid[x]].push_back(x);
324
325   if (child == 0)
326      bcc[bccid[x]].push_back(x);
327
328   if (child == 1)
329      bcc[bccid[x]].push_back(y);
330
331   if (child == 0)
332      bcc[bccid[x]].push_back(y);
333
334   if (child == 1)
335      bcc[bccid[x]].push_back(x);
336
337   if (child == 0)
338      bcc[bccid[x]].push_back(x);
339
340   if (child == 1)
341      bcc[bccid[x]].push_back(y);
342
343   if (child == 0)
344      bcc[bccid[x]].push_back(y);
345
346   if (child == 1)
347      bcc[bccid[x]].push_back(x);
348
349   if (child == 0)
350      bcc[bccid[x]].push_back(x);
351
352   if (child == 1)
353      bcc[bccid[x]].push_back(y);
354
355   if (child == 0)
356      bcc[bccid[x]].push_back(y);
357
358   if (child == 1)
359      bcc[bccid[x]].push_back(x);
360
361   if (child == 0)
362      bcc[bccid[x]].push_back(x);
363
364   if (child == 1)
365      bcc[bccid[x]].push_back(y);
366
367   if (child == 0)
368      bcc[bccid[x]].push_back(y);
369
370   if (child == 1)
371      bcc[bccid[x]].push_back(x);
372
373   if (child == 0)
374      bcc[bccid[x]].push_back(x);
375
376   if (child == 1)
377      bcc[bccid[x]].push_back(y);
378
379   if (child == 0)
380      bcc[bccid[x]].push_back(y);
381
382   if (child == 1)
383      bcc[bccid[x]].push_back(x);
384
385   if (child == 0)
386      bcc[bccid[x]].push_back(x);
387
388   if (child == 1)
389      bcc[bccid[x]].push_back(y);
390
391   if (child == 0)
392      bcc[bccid[x]].push_back(y);
393
394   if (child == 1)
395      bcc[bccid[x]].push_back(x);
396
397   if (child == 0)
398      bcc[bccid[x]].push_back(x);
399
400   if (child == 1)
401      bcc[bccid[x]].push_back(y);
402
403   if (child == 0)
404      bcc[bccid[x]].push_back(y);
405
406   if (child == 1)
407      bcc[bccid[x]].push_back(x);
408
409   if (child == 0)
410      bcc[bccid[x]].push_back(x);
411
412   if (child == 1)
413      bcc[bccid[x]].push_back(y);
414
415   if (child == 0)
416      bcc[bccid[x]].push_back(y);
417
418   if (child == 1)
419      bcc[bccid[x]].push_back(x);
420
421   if (child == 0)
422      bcc[bccid[x]].push_back(x);
423
424   if (child == 1)
425      bcc[bccid[x]].push_back(y);
426
427   if (child == 0)
428      bcc[bccid[x]].push_back(y);
429
430   if (child == 1)
431      bcc[bccid[x]].push_back(x);
432
433   if (child == 0)
434      bcc[bccid[x]].push_back(x);
435
436   if (child == 1)
437      bcc[bccid[x]].push_back(y);
438
439   if (child == 0)
440      bcc[bccid[x]].push_back(y);
441
442   if (child == 1)
443      bcc[bccid[x]].push_back(x);
444
445   if (child == 0)
446      bcc[bccid[x]].push_back(x);
447
448   if (child == 1)
449      bcc[bccid[x]].push_back(y);
450
451   if (child == 0)
452      bcc[bccid[x]].push_back(y);
453
454   if (child == 1)
455      bcc[bccid[x]].push_back(x);
456
457   if (child == 0)
458      bcc[bccid[x]].push_back(x);
459
460   if (child == 1)
461      bcc[bccid[x]].push_back(y);
462
463   if (child == 0)
464      bcc[bccid[x]].push_back(y);
465
466   if (child == 1)
467      bcc[bccid[x]].push_back(x);
468
469   if (child == 0)
470      bcc[bccid[x]].push_back(x);
471
472   if (child == 1)
473      bcc[bccid[x]].push_back(y);
474
475   if (child == 0)
476      bcc[bccid[x]].push_back(y);
477
478   if (child == 1)
479      bcc[bccid[x]].push_back(x);
480
481   if (child == 0)
482      bcc[bccid[x]].push_back(x);
483
484   if (child == 1)
485      bcc[bccid[x]].push_back(y);
486
487   if (child == 0)
488      bcc[bccid[x]].push_back(y);
489
490   if (child == 1)
491      bcc[bccid[x]].push_back(x);
492
493   if (child == 0)
494      bcc[bccid[x]].push_back(x);
495
496   if (child == 1)
497      bcc[bccid[x]].push_back(y);
498
499   if (child == 0)
500      bcc[bccid[x]].push_back(y);
501
502   if (child == 1)
503      bcc[bccid[x]].push_back(x);
504
505   if (child == 0)
506      bcc[bccid[x]].push_back(x);
507
508   if (child == 1)
509      bcc[bccid[x]].push_back(y);
510
511   if (child == 0)
512      bcc[bccid[x]].push_back(y);
513
514   if (child == 1)
515      bcc[bccid[x]].push_back(x);
516
517   if (child == 0)
518      bcc[bccid[x]].push_back(x);
519
520   if (child == 1)
521      bcc[bccid[x]].push_back(y);
522
523   if (child == 0)
524      bcc[bccid[x]].push_back(y);
525
526   if (child == 1)
527      bcc[bccid[x]].push_back(x);
528
529   if (child == 0)
530      bcc[bccid[x]].push_back(x);
531
532   if (child == 1)
533      bcc[bccid[x]].push_back(y);
534
535   if (child == 0)
536      bcc[bccid[x]].push_back(y);
537
538   if (child == 1)
539      bcc[bccid[x]].push_back(x);
540
541   if (child == 0)
542      bcc[bccid[x]].push_back(x);
543
544   if (child == 1)
545      bcc[bccid[x]].push_back(y);
546
547   if (child == 0)
548      bcc[bccid[x]].push_back(y);
549
550   if (child == 1)
551      bcc[bccid[x]].push_back(x);
552
553   if (child == 0)
554      bcc[bccid[x]].push_back(x);
555
556   if (child == 1)
557      bcc[bccid[x]].push_back(y);
558
559   if (child == 0)
560      bcc[bccid[x]].push_back(y);
561
562   if (child == 1)
563      bcc[bccid[x]].push_back(x);
564
565   if (child == 0)
566      bcc[bccid[x]].push_back(x);
567
568   if (child == 1)
569      bcc[bccid[x]].push_back(y);
570
571   if (child == 0)
572      bcc[bccid[x]].push_back(y);
573
574   if (child == 1)
575      bcc[bccid[x]].push_back(x);
576
577   if (child == 0)
578      bcc[bccid[x]].push_back(x);
579
580   if (child == 1)
581      bcc[bccid[x]].push_back(y);
582
583   if (child == 0)
584      bcc[bccid[x]].push_back(y);
585
586   if (child == 1)
587      bcc[bccid[x]].push_back(x);
588
589   if (child == 0)
590      bcc[bccid[x]].push_back(x);
591
592   if (child == 1)
593      bcc[bccid[x]].push_back(y);
594
595   if (child == 0)
596      bcc[bccid[x]].push_back(y);
597
598   if (child == 1)
599      bcc[bccid[x]].push_back(x);
600
601   if (child == 0)
602      bcc[bccid[x]].push_back(x);
603
604   if (child == 1)
605      bcc[bccid[x]].push_back(y);
606
607   if (child == 0)
608      bcc[bccid[x]].push_back(y);
609
610   if (child == 1)
611      bcc[bccid[x]].push_back(x);
612
613   if (child == 0)
614      bcc[bccid[x]].push_back(x);
615
616   if (child == 1)
617      bcc[bccid[x]].push_back(y);
618
619   if (child == 0)
620      bcc[bccid[x]].push_back(y);
621
622   if (child == 1)
623      bcc[bccid[x]].push_back(x);
624
625   if (child == 0)
626      bcc[bccid[x]].push_back(x);
627
628   if (child == 1)
629      bcc[bccid[x]].push_back(y);
630
631   if (child == 0)
632      bcc[bccid[x]].push_back(y);
633
634   if (child == 1)
635      bcc[bccid[x]].push_back(x);
636
637   if (child == 0)
638      bcc[bccid[x]].push_back(x);
639
640   if (child == 1)
641      bcc[bccid[x]].push_back(y);
642
643   if (child == 0)
644      bcc[bccid[x]].push_back(y);
645
646   if (child == 1)
647      bcc[bccid[x]].push_back(x);
648
649   if (child == 0)
650      bcc[bccid[x]].push_back(x);
651
652   if (child == 1)
653      bcc[bccid[x]].push_back(y);
654
655   if (child == 0)
656      bcc[bccid[x]].push_back(y);
657
658   if (child == 1)
659      bcc[bccid[x]].push_back(x);
660
661   if (child == 0)
662      bcc[bccid[x]].push_back(x);
663
664   if (child == 1)
665      bcc[bccid[x]].push_back(y);
666
667   if (child == 0)
668      bcc[bccid[x]].push_back(y);
669
670   if (child == 1)
671      bcc[bccid[x]].push_back(x);
672
673   if (child == 0)
674      bcc[bccid[x]].push_back(x);
675
676   if (child == 1)
677      bcc[bccid[x]].push_back(y);
678
679   if (child == 0)
680      bcc[bccid[x]].push_back(y);
681
682   if (child == 1)
683      bcc[bccid[x]].push_back(x);
684
685   if (child == 0)
686      bcc[bccid[x]].push_back(x);
687
688   if (child == 1)
689      bcc[bccid[x]].push_back(y);
690
691   if (child == 0)
692      bcc[bccid[x]].push_back(y);
693
694   if (child == 1)
695      bcc[bccid[x]].push_back(x);
696
697   if (child == 0)
698      bcc[bccid[x]].push_back(x);
699
700   if (child == 1)
701      bcc[bccid[x]].push_back(y);
702
703   if (child == 0)
704      bcc[bccid[x]].push_back(y);
705
706   if (child == 1)
707      bcc[bccid[x]].push_back(x);
708
709   if (child == 0)
710      bcc[bccid[x]].push_back(x);
711
712   if (child == 1)
713      bcc[bccid[x]].push_back(y);
714
715   if (child == 0)
716      bcc[bccid[x]].push_back(y);
717
718   if (child == 1)
719      bcc[bccid[x]].push_back(x);
720
721   if (child == 0)
722      bcc[bccid[x]].push_back(x);
723
724   if (child == 1)
725      bcc[bccid[x]].push_back(y);
726
727   if (child == 0)
728      bcc[bccid[x]].push_back(y);
729
730   if (child == 1)
731      bcc[bccid[x]].push_back(x);
732
733   if (child == 0)
734      bcc[bccid[x]].push_back(x);
735
736   if (child == 1)
737      bcc[bccid[x]].push_back(y);
738
739   if (child == 0)
740      bcc[bccid[x]].push_back(y);
741
742   if (child == 1)
743      bcc[bccid[x]].push_back(x);
744
745   if (child == 0)
746      bcc[bccid[x]].push_back(x);
747
748   if (child == 1)
749      bcc[bccid[x]].push_back(y);
750
751   if (child == 0)
752      bcc[bccid[x]].push_back(y);
753
754   if (child == 1)
755      bcc[bccid[x]].push_back(x);
756
757   if (child == 0)
758      bcc[bccid[x]].push_back(x);
759
760   if (child == 1)
761      bcc[bccid[x]].push_back(y);
762
763   if (child == 0)
764      bcc[bccid[x]].push_back(y);
765
766   if (child == 1)
767      bcc[bccid[x]].push_back(x);
768
769   if (child == 0)
770      bcc[bccid[x]].push_back(x);
771
772   if (child == 1)
773      bcc[bccid[x]].push_back(y);
774
775   if (child == 0)
776      bcc[bccid[x]].push_back(y);
777
778   if (child == 1)
779      bcc[bccid[x]].push_back(x);
780
781   if (child == 0)
782      bcc[bccid[x]].push_back(x);
783
784   if (child == 1)
785      bcc[bccid[x]].push_back(y);
786
787   if (child == 0)
788      bcc[bccid[x]].push_back(y);
789
790   if (child == 1)
791      bcc[bccid[x]].push_back(x);
792
793   if (child == 0)
794      bcc[bccid[x]].push_back(x);
795
796   if (child == 1)
797      bcc[bccid[x]].push_back(y);
798
799   if (child == 0)
800      bcc[bccid[x]].push_back(y);
801
802   if (child == 1)
803      bcc[bccid[x]].push_back(x);
804
805   if (child == 0)
806      bcc[bccid[x]].push_back(x);
807
808   if (child == 1)
809      bcc[bccid[x]].push_back(y);
810
811   if (child == 0)
812      bcc[bccid[x]].push_back(y);
813
814   if (child == 1)
815      bcc[bccid[x]].push_back(x);
816
817   if (child == 0)
818      bcc[bccid[x]].push_back(x);
819
820   if (child == 1)
821      bcc[bccid[x]].push_back(y);
822
823   if (child == 0)
824      bcc[bccid[x]].push_back(y);
825
826   if (child == 1)
827      bcc[bccid[x]].push_back(x);
828
829   if (child == 0)
830      bcc[bccid[x]].push_back(x);
831
832   if (child == 1)
833      bcc[bccid[x]].push_back(y);
834
835   if (child == 0)
836      bcc[bccid[x]].push_back(y);
837
838   if (child == 1)
839      bcc[bccid[x]].push_back(x);
840
841   if (child == 0)
842      bcc[bccid[x]].push_back(x);
843
844   if (child == 1)
845      bcc[bccid[x]].push_back(y);
846
847   if (child == 0)
848      bcc[bccid[x]].push_back(y);
849
850   if (child == 1)
851      bcc[bccid[x]].push_back(x);
852
853   if (child == 0)
854      bcc[bccid[x]].push_back(x);
855
856   if (child == 1)
857      bcc[bccid[x]].push_back(y);
858
859   if (child == 0)
860      bcc[bccid[x]].push_back(y);
861
862   if (child == 1)
863      bcc[bccid[x]].push_back(x);
864
865   if (child == 0)
866      bcc[bccid[x]].push_back(x);
867
868   if (child == 1)
869      bcc[bccid[x]].push_back(y);
870
871   if (child == 0)
872      bcc[bccid[x]].push_back(y);
873
874   if (child == 1)
875      bcc[bccid[x]].push_back(x);
876
877   if (child == 0)
878      bcc[bccid[x]].push_back(x);
879
880   if (child == 1)
881      bcc[bccid[x]].push_back(y);
882
883   if (child == 0)
884      bcc[bccid[x]].push_back(y);
885
886   if (child == 1)
887      bcc[bccid[x]].push_back(x);
888
889   if (child == 0)
890      bcc[bccid[x]].push_back(x);
891
892   if (child == 1)
893      bcc[bccid[x]].push_back(y);
894
895   if (child == 0)
896      bcc[bccid[x]].push_back(y);
897
898   if (child == 1)
899      bcc[bccid[x]].push_back(x);
900
901   if (child == 0)
902      bcc[bccid[x]].push_back(x);
903
904   if (child == 1)
905      bcc[bccid[x]].push_back(y);
906
907   if (child == 0)
908      bcc[bccid[x]].push_back(y);
909
910   if (child == 1)
911      bcc[bccid[x]].push_back(x);
912
913   if (child == 0)
914      bcc[bccid[x]].push_back(x);
915
916   if (child == 1)
917      bcc[bccid[x]].push_back(y);
918
919   if (child == 0)
920      bcc[bccid[x]].push_back(y);
921
922   if (child == 1)
923      bcc[bccid[x]].push_back(x);
924
925   if (child == 0)
926      bcc[bccid[x]].push_back(x);
927
928   if (child == 1)
929      bcc[bccid[x]].push_back(y);
930
931   if (child == 0)
932      bcc[bccid[x]].push_back(y);
933
934   if (child == 1)
935      bcc[bccid[x]].push_back(x);
936
937   if (child == 0)
938      bcc[bccid[x]].push_back(x);
939
940   if (child == 1)
941      bcc[bccid[x]].push_back(y);
942
943   if (child == 0)
944      bcc[bccid[x]].push_back(y);
945
946   if (child == 1)
947      bcc[bccid[x]].push_back(x);
948
949   if (child == 0)
950      bcc[bccid[x]].push_back(x);
951
952   if (child == 1)
953      bcc[bccid[x]].push_back(y);
954
955   if (child == 0)
956      bcc[bccid[x]].push_back(y);
957
958   if (child == 1)
959      bcc[bccid[x]].push_back(x);
960
961   if (child == 0)
962      bcc[bccid[x]].push_back(x);
963
964   if (child == 1)
965      bcc[bccid[x]].push_back(y);
966
967   if (child == 0)
968      bcc[bccid[x]].push_back(y);
969
970   if (child == 1)
971      bcc[bccid[x]].push_back(x);
972
973   if (child == 0)
974      bcc[bccid[x]].push_back(x);
975
976   if (child == 1)
977      bcc[bccid[x]].push_back(y);
978
979   if (child == 0)
980      bcc[bccid[x]].push_back(y);
981
982   if (child == 1)
983      bcc[bccid[x]].push_back(x);
984
985   if (child == 0)
986      bcc[bccid[x]].push_back(x);
987
9
```

```

24     auto pi = stk[stk_cnt--];
25
26     if (bccid[pi.first] != bcc_cnt) {
27         bcc[bcc_cnt].push_back(pi.first);
28         bccid[pi.first] = bcc_cnt;
29     }
30     if (bccid[pi.second] != bcc_cnt) {
31         bcc[bcc_cnt].push_back(pi.second);
32         bccid[pi.second] = bcc_cnt;
33     }
34
35     if (pi.first == x && pi.second ==
36         ~y)
37         break;
38 }
39 }
40 else if (dfn[y] < dfn[x] && y != pr) {
41     stk[++stk_cnt] = make_pair(x, y);
42     low[x] = min(low[x], dfn[y]);
43 }
44 }
45
46 if (!pr && child == 1)
47     iscut[x] = false;
48 }
49
50 void Tarjan(int n) {
51     for (int i = 1; i <= n; i++)
52         if (!dfn[i])
53             dfs(i, 0);
54 }

```

35 }

3.4 欧拉回路

$C[x]$ 是记录每条边对应的编号的.

另外为了保证复杂度需要加当前弧优化.

```

1 vector<int> G[maxn], C[maxn], v[maxn]; // C是边的编号
2 int cur[maxn];
3 bool vis[maxn * 2];
4
5 vector<pair<int, int> > vec;
6
7 int d[maxn];
8
9 void dfs(int x) {
10    bool bad = false;
11
12    while (!bad) {
13        bad = true;
14
15        for (int &i = cur[x]; i < (int)G[x].size(); i+
16            ~+)
17            if (!vis[C[x][i]]) {
18                vis[C[x][i]] = true;
19                vec.emplace_back(x, i);
20                x = G[x][i];
21                bad = false;
22            }
23    }
24 }
25

```

3.3.3 桥边双

```

1 int u[maxe], v[maxe];
2 vector<int> G[maxn]; // 存的是边的编号
3
4 int stk[maxn], top, dfn[maxn], low[maxn], tim, bcc_cnt;
5 vector<int> bcc[maxn];
6
7 bool isbridge[maxe];
8
9 void dfs(int x, int pr) { // 这里pr是入边的编号
10    dfn[x] = low[x] = ++tim;
11    stk[++top] = x;
12
13    for (int i : G[x]) {
14        int y = (u[i] == x ? v[i] : u[i]);
15
16        if (!dfn[y]) {
17            dfs(y, i);
18            low[x] = min(low[x], low[y]);
19
20            if (low[y] > dfn[x])
21                bridge[i] = true;
22        }
23        else if (i != pr)
24            low[x] = min(low[x], dfn[y]);
25    }
26
27    if (dfn[x] == low[x]) {
28        bcc_cnt++;
29        int y;
30        do {
31            y = stk[top--];
32            bcc[bcc_cnt].push_back(y);
33        } while (y != x);
34    }

```

3.5 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.

3.5.1 仙人掌 DP

```

1 struct edge {
2     int to, w, prev;
3 } e[maxn * 2];
4
5 vector<pair<int, int> > v[maxn];
6 vector<long long> d[maxn];
7 stack<int> stk;
8
9 int p[maxn];
10 bool vis[maxn], vise[maxn * 2];
11 int last[maxn], cnte;
12
13 long long f[maxn], g[maxn], sum[maxn];
14 int n, m, cnt;
15
16 void addedge(int x, int y, int w) {
17     v[x].push_back(make_pair(y, w));
18 }
19
20 void dfs(int x) {
21
22     vis[x] = true;
23
24     for (int i = last[x]; ~i; i = e[i].prev) {
25         if (vise[i ^ 1])
26             continue;

```

```

28     int y = e[i].to, w = e[i].w;
29
30     vise[i] = true;
31
32     if (!vis[y]) {
33         stk.push(i);
34         p[y] = x;
35         dfs(y);
36
37         if (!stk.empty() && stk.top() == i) {
38             stk.pop();
39             addedge(x, y, w);
40         }
41     }
42
43     else {
44         cnt++;
45
46         long long tmp = w;
47         while (!stk.empty()) {
48             int i = stk.top();
49             stk.pop();
50
51             int yy = e[i].to, ww = e[i].w;
52
53             addedge(cnt, yy, 0);
54
55             d[cnt].push_back(tmp);
56
57             tmp += ww;
58
59             if (e[i ^ 1].to == y)
60                 break;
61         }
62
63         addedge(y, cnt, 0);
64
65         sum[cnt] = tmp;
66     }
67 }
68
69 void dp(int x) {
70
71     for (auto o : v[x]) {
72         int y = o.first, w = o.second;
73         dp(y);
74     }
75
76
77     if (x <= n) {
78         for (auto o : v[x]) {
79             int y = o.first, w = o.second;
80
81             f[x] += 2 * w + f[y];
82         }
83
84         g[x] = f[x];
85
86         for (auto o : v[x]) {
87             int y = o.first, w = o.second;
88
89             g[x] = min(g[x], f[x] - f[y] - 2 * w + g[y]
90                         → + w);
91         }
92     }
93     else {
94         f[x] = sum[x];
95         for (auto o : v[x]) {
96             int y = o.first;

```

```

97             f[x] += f[y];
98         }
99
100        g[x] = f[x];
101
102        for (int i = 0; i < (int)v[x].size(); i++) {
103            int y = v[x][i].first;
104
105            g[x] = min(g[x], f[x] - f[y] + g[y] +
106                         → min(d[x][i], sum[x] - d[x][i]));
107        }
108    }

```

3.6 二分图

3.6.1 匈牙利

```

1 vector<int> G[maxn];
2
3 int girl[maxn], boy[maxn]; // 男孩在左边, 女孩在右边
4 bool vis[maxn];
5
6 bool dfs(int x) {
7     for (int y : G[x])
8         if (!vis[y]) {
9             vis[y] = true;
10
11             if (!boy[y] || dfs(boy[y])) {
12                 girl[x] = y;
13                 boy[y] = x;
14
15                 return true;
16             }
17         }
18
19     return false;
20 }
21
22 int hungary() {
23     int ans = 0;
24
25     for (int i = 1; i <= n; i++)
26         if (!girl[i]) {
27             memset(vis, 0, sizeof(vis));
28             ans += dfs(i);
29         }
30
31     return ans;
32 }

```

3.6.2 Hopcroft-Karp 二分图匹配

其实长得和 Dinic 差不多, 或者说像匈牙利和 Dinic 的缝合怪.

```

1 vector<int> G[maxn];
2
3 int girl[maxn], boy[maxn]; // girl: 左边匹配右边 boy:
4                         → 右边匹配左边
5 bool vis[maxn]; // 右半的点是否已被访问
6 int dx[maxn], dy[maxn];
7 int q[maxn];
8
9 bool bfs(int n) {
10     memset(dx, -1, sizeof(int) * (n + 1));
11     memset(dy, -1, sizeof(int) * (n + 1));
12

```

```

13 int head = 0, tail = 0;
14 for (int i = 1; i <= n; i++) {
15     if (!girl[i]) {
16         q[tail++] = i;
17         dx[i] = 0;
18     }
19
20     bool flag = false;
21
22     while (head != tail) {
23         int x = q[head++];
24
25         for (auto y : G[x])
26             if (dy[y] == -1) {
27                 dy[y] = dx[x] + 1;
28
29                 if (boy[y]) {
30                     if (dx[boy[y]] == -1) {
31                         dx[boy[y]] = dy[y] + 1;
32                         q[tail++] = boy[y];
33                     }
34                 } else
35                     flag = true;
36             }
37     }
38
39     return flag;
40 }
41
42 bool dfs(int x) {
43     for (int y : G[x])
44         if (!vis[y] && dy[y] == dx[x] + 1) {
45             vis[y] = true;
46
47             if (boy[y] && !dfs(boy[y]))
48                 continue;
49
50             girl[x] = y;
51             boy[y] = x;
52             return true;
53         }
54
55     return false;
56 }
57
58 int hopcroft_karp(int n) {
59     int ans = 0;
60
61     for (int x = 1; x <= n; x++) // 先贪心求出一组初始匹
62     ↪ 配, 当然不写贪心也行
63     for (int y : G[x])
64         if (!boy[y]) {
65             girl[x] = y;
66             boy[y] = x;
67             ans++;
68             break;
69         }
70
71     while (bfs(n)) {
72         memset(vis, 0, sizeof(bool) * (n + 1));
73
74         for (int x = 1; x <= n; x++)
75             if (!girl[x])
76                 ans += dfs(x);
77
78     return ans;
79 }
80

```

3.6.3 KM 二分图最大权匹配

```

1 const long long INF = 0x3f3f3f3f3f3f3f3f;
2
3 long long w[maxn][maxn], lx[maxn], ly[maxn],
4 ↪ slack[maxn];
// 边权 顶标 slack
5 // 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
6 ↪ 边对0取max
7
8 bool visx[maxn], visy[maxn];
9
10 int boy[maxn], girl[maxn], p[maxn], q[maxn], head,
11 ↪ tail; // p : pre
12
13 // 增广
14 bool check(int y) {
15     visy[y] = true;
16
17     if (boy[y]) {
18         visx[boy[y]] = true;
19         q[tail++] = boy[y];
20         return false;
21     }
22
23     while (y) {
24         boy[y] = p[y];
25         swap(y, girl[p[y]]);
26     }
27
28     return true;
29 }
30
31 // bfs每个点
32 void bfs(int x) {
33     memset(q, 0, sizeof(q));
34     head = tail = 0;
35
36     q[tail++] = x;
37     visx[x] = true;
38
39     while (true) {
40         while (head != tail) {
41             int x = q[head++];
42
43             for (int y = 1; y <= N; y++) {
44                 if (!visy[y]) {
45                     long long d = lx[x] + ly[y] - w[x]
46                     ↪ [y];
47
48                     if (d < slack[y]) {
49                         p[y] = x;
50                         slack[y] = d;
51
52                         if (!slack[y] && check(y))
53                             return;
54                     }
55                 }
56             }
57             long long d = INF;
58             for (int i = 1; i <= N; i++)
59                 if (!visy[i])
60                     d = min(d, slack[i]);
61
62             for (int i = 1; i <= N; i++) {

```

```

63     if (visx[i])
64         lx[i] -= d;
65
66     if (visy[i])
67         ly[i] += d;
68     else
69         slack[i] -= d;
70 }
71
72 for (int i = 1; i <= N; i++)
73     if (!visy[i] && !slack[i] && check(i))
74         return;
75 }
76
77 // 主过程
78 long long KM() {
79     for (int i = 1; i <= N; i++) {
80         // lx[i] = 0;
81         ly[i] = -INF;
82         // boy[i] = girl[i] = -1;
83
84         for (int j = 1; j <= N; j++)
85             ly[i] = max(ly[i], w[j][i]);
86     }
87
88     for (int i = 1; i <= N; i++) {
89         memset(slack, 0x3f, sizeof(slack));
90         memset(visx, 0, sizeof(visx));
91         memset(visy, 0, sizeof(visy));
92         bfs(i);
93     }
94
95     long long ans = 0;
96     for (int i = 1; i <= N; i++)
97         ans += w[i][girl[i]];
98     return ans;
99 }
100
101 // 为了方便贴上主函数
102 int main() {
103
104     scanf("%d%d%d", &n, &m, &e);
105     N = max(n, m);
106
107     while (e--) {
108         int x, y, c;
109         scanf("%d%d%d", &x, &y, &c);
110         w[x][y] = max(c, 0);
111     }
112 }
113
114 printf("%lld\n", KM());
115
116 for (int i = 1; i <= n; i++) {
117     if (i > 1)
118         printf(" ");
119     printf("%d", w[i][girl[i]] > 0 ? girl[i] : 0);
120 }
121 printf("\n");
122
123 return 0;
124 }
```

3.6.4 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.

3.7 一般图匹配

3.7.1 高斯消元

```

1 // 这个算法基于Tutte定理和高斯消元, 思维难度相对小一些,
2 // → 也更方便进行可行边的判定
3 // 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
4 // → 是跑不满的
5 // 以及, 根据Tutte定理, 如果求最大匹配的大小的话直接输
6 // → 出Tutte矩阵的秩/2即可
7 // 需要输出方案时才需要再写后面那些乱七八糟的东西
8
9
10 // 复杂度和常数所限, 1s之内500已经是这个算法的极限了
11 const int maxn = 505, p = 1000000007; // p可以是任
12 // → 意10^9以内的质数
13
14 // 全局数组和变量定义
15 int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn],
16 // → id[maxn], a[maxn];
17 bool row[maxn] = {false}, col[maxn] = {false};
18 int n, m, girl[maxn]; // girl是匹配点, 用来输出方案
19
20 // 为了方便使用, 贴上主函数
21 // 需要调用高斯消元和eliminate
22 int main() {
23     srand(19260817);
24
25     scanf("%d%d", &n, &m); // 点数和边数
26     while (m--) {
27         int x, y;
28         scanf("%d%d", &x, &y);
29         A[x][y] = rand() % p;
30         A[y][x] = -A[x][y]; // Tutte矩阵是反对称矩阵
31     }
32
33     for (int i = 1; i <= n; i++)
34         id[i] = i; // 输出方案用的, 因为高斯消元的时候会
35         // → 交换列
36     memcpy(t, A, sizeof(t));
37     Gauss(A, NULL, n);
38
39     m = n;
40     n = 0; // 这里变量复用纯属个人习惯
41
42     for (int i = 1; i <= m; i++)
43         if (A[id[i]][id[i]])
```

```
103
104     if (B)
105         for (int k = 1; k <= n; k++)
106             if (B[i][k])
107                 B[j][k] = (B[j][k] - (long)
108                               → long)t * B[i][k])%p;
109
110 }
111
112 if (B)
113     for (int i = 1; i <= n; i++) {
114         int inv = qpow(A[i][i], p - 2);
115
116         for (int j = 1; j <= n; j++)
117             if (B[i][j])
118                 B[i][j] = (long long)B[i][j] * inv
119                               → % p;
120
121 // 消去一行一列 O(n^2)
122 void eliminate(int r, int c) {
123     row[r] = col[c] = true; // 已经被消掉
124
125     int inv = qpow(B[r][c], p - 2);
126
127     for (int i = 1; i <= n; i++)
128         if (!row[i] && B[i][c]) {
129             int t = (long long)B[i][c] * inv % p;
130
131             for (int j = 1; j <= n; j++)
132                 if (!col[j] && B[r][j])
133                     B[i][j] = (B[i][j] - (long long)t *
134                               → B[r][j]) % p;
135 }
```

3.7.2 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
```

```

30     head = tail = 0;
31     q[tail++] = s;
32     t[s] = 1;
33
34     while (head != tail) {
35         int x = q[head++];
36         for (int y : G[x]) {
37             if (findroot(y) == findroot(x) || t[y] == 2)
38                 continue;
39
40             if (!t[y]) {
41                 t[y] = 2;
42                 p[y] = x;
43
44                 if (!girl[y]) {
45                     for (int u = y, t; u; u = t) {
46                         t = girl[p[u]];
47                         girl[p[u]] = u;
48                         girl[u] = p[u];
49                     }
50                     return true;
51                 }
52
53                 t[girl[y]] = 1;
54                 q[tail++] = girl[y];
55             } else if (t[y] == 1) {
56                 int z = LCA(x, y);
57
58                 shrink(x, y, z);
59                 shrink(y, x, z);
60             }
61         }
62     }
63
64     return false;
65 }
66
//缩奇环 O(n)
67 void shrink(int x, int y, int z) {
68     while (findroot(x) != z) {
69         p[x] = y;
70         y = girl[x];
71
72         if (t[y] == 2) {
73             t[y] = 1;
74             q[tail++] = y;
75         }
76
77         if (findroot(x) == x)
78             f[x] = z;
79         if (findroot(y) == y)
80             f[y] = z;
81
82         x = p[y];
83     }
84 }
85
//暴力找LCA O(n)
86 int LCA(int x, int y) {
87     tim++;
88     while (true) {
89         if (x) {
90             x = findroot(x);
91
92             if (vis[x] == tim)
93                 return x;
94             else {
95
96
97

```

```

98             vis[x] = tim;
99             x = p[girl[x]];
100
101         }
102         swap(x, y);
103     }
104 }
105
106 //并查集的查找 O(1)
107 int findroot(int x) {
108     return x == f[x] ? x : (f[x] = findroot(f[x]));
109 }

```

3.7.3 带权带花树

Forked from the template of Imperisble 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)

```

1 //maximum weight blossom, change g[u][v].w to INF -
2 //→ g[u][v].w when minimum weight blossom is needed
3 //type of ans is long long
4 //replace all int to long long if weight of edge is
5 //→ long long
6
7 struct WeightGraph {
8     static const int INF = INT_MAX;
9     static const int MAXN = 400;
10    struct edge{
11        int u, v, w;
12        edge() {}
13        edge(int u, int v, int w): u(u), v(v), w(w) {}
14    };
15    int n, n_x;
16    edge g[MAXN * 2 + 1][MAXN * 2 + 1];
17    int lab[MAXN * 2 + 1];
18    int match[MAXN * 2 + 1], slack[MAXN * 2 + 1],
19    → st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
20    int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 +
21    → 1], vis[MAXN * 2 + 1];
22    vector<int> flower[MAXN * 2 + 1];
23    queue<int> q;
24    inline int e_delta(const edge &e){ // does not work
25        → inside blossoms
26        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
27    }
28    inline void update_slack(int u, int x){
29        if(!slack[x] || e_delta(g[u][x]) <
30        → e_delta(g[slack[x]][x]))
31            slack[x] = u;
32    }
33    inline void set_slack(int x){
34        slack[x] = 0;
35        for(int u = 1; u <= n; ++u)
36            if(g[u][x].w > 0 && st[u] != x && S[st[u]] ← 0)
37                update_slack(u, x);
38
39    void q_push(int x){
40        if(x <= n)q.push(x);
41        else for(size_t i = 0; i < flower[x].size(); i+
42        → +)
43            q.push(flower[x][i]);
44    }
45    inline void set_st(int x, int b){
46        st[x] = b;
47        if(x > n) for(size_t i = 0; i <
48        → flower[x].size(); ++i)
49            set_st(flower[x][i], b);
50    }

```

```

42 }
43 inline int get_pr(int b, int xr){
44     int pr = find(flower[b].begin(),
45                   flower[b].end(), xr) - flower[b].begin();
46     if(pr % 2 == 1){
47         reverse(flower[b].begin() + 1,
48                 flower[b].end());
49         return (int)flower[b].size() - pr;
50     } else return pr;
51 }
52 inline void set_match(int u, int v){
53     match[u]=g[u][v].v;
54     if(u > n){
55         edge e=g[u][v];
56         int xr = flower_from[u][e.u], pr=get_pr(u,
57             xr);
58         for(int i = 0;i < pr; ++i)
59             set_match(flower[u][i], flower[u][i ^
60                     1]);
61         set_match(xr, v);
62         rotate(flower[u].begin(),
63                 flower[u].begin()+pr, flower[u].end());
64     }
65 }
66 inline void augment(int u, int v){
67     for(; ; ){
68         int xnv=st[match[u]];
69         set_match(u, v);
70         if(!xnv) return;
71         set_match(xnv, st[pa[xnv]]);
72         u=st[pa[xnv]], v=xnv;
73     }
74 }
75 inline int get_lca(int u, int v){
76     static int t=0;
77     for(++t; u || v; swap(u, v)){
78         if(u == 0) continue;
79         if(vis[u] == t) return u;
80         vis[u] = t;
81         u = st[match[u]];
82         if(u) u = st[pa[u]];
83     }
84     return 0;
85 }
86 inline void add_blossom(int u, int lca, int v){
87     int b = n + 1;
88     while(b <= n_x && st[b]) ++b;
89     if(b > n_x) ++n_x;
90     lab[b] = 0, S[b] = 0;
91     match[b] = match[lca];
92     flower[b].clear();
93     flower[b].push_back(lca);
94     for(int x = u, y; x != lca; x = st[pa[y]]) {
95         flower[b].push_back(x),
96         flower[b].push_back(y = st[match[x]]),
97         q_push(y);
98     }
99     reverse(flower[b].begin() + 1,
100            flower[b].end());
101    for(int x = v, y; x != lca; x = st[pa[y]]) {
102        flower[b].push_back(x),
103        flower[b].push_back(y = st[match[x]]),
104        q_push(y);
105    }
106    set_st(b, b);
107    for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x]
108        ↪ [b].w = 0;
109    for(int x = 1; x <= n; ++x) flower_from[b][x] =
110        ↪ 0;

```

```

103    for(size_t i = 0 ; i < flower[b].size(); ++i){
104        int xs = flower[b][i];
105        for(int x = 1; x <= n_x; ++x)
106            if(g[b][x].w == 0 || e_delta(g[xs][x]))
107                ↪ < e_delta(g[b][x]));
108            g[b][x] = g[xs][x], g[x][b] = g[x]
109                ↪ [xs];
110        for(int x = 1;x <= n; ++x)
111            if(flower_from[xs][x]) flower_from[b]
112                ↪ [x] = xs;
113    }
114    set_slack(b);
115 }
116 inline void expand_blossom(int b){ // S[b] == 1
117     for(size_t i = 0; i < flower[b].size(); ++i)
118         set_st(flower[b][i], flower[b][i]);
119     int xr = flower_from[b][g[b][pa[b]].u], pr =
120         ↪ get_pr(b, xr);
121     for(int i = 0; i < pr; i += 2){
122         int xs = flower[b][i], xns = flower[b][i +
123             ↪ 1];
124         pa[xs] = g[xns][xs].u;
125         S[xs] = 1, S[xns] = 0;
126         slack[xs] = 0, set_slack(xns);
127         q_push(xns);
128     }
129     S[xr] = 1, pa[xr] = pa[b];
130     for(size_t i = pr + 1;i < flower[b].size(); + ↪
131         i){
132         int xs = flower[b][i];
133         S[xs] = -1, set_slack(xs);
134     }
135     st[b] = 0;
136 }
137 inline bool on_found_edge(const edge &e){
138     int u = st[e.u], v = st[e.v];
139     if(S[v] == -1){
140         pa[v] = e.u, S[v] = 1;
141         int nu = st[match[v]];
142         slack[v] = slack[nu] = 0;
143         S[nu] = 0, q_push(nu);
144     }else if(S[v] == 0){
145         int lca = get_lca(u, v);
146         if(!lca) return augment(u, v), augment(v,
147             ↪ u), true;
148         else add_blossom(u, lca, v);
149     }
150     return false;
151 }
152 inline bool matching(){
153     memset(S + 1, -1, sizeof(int) * n_x);
154     memset(slack + 1, 0, sizeof(int) * n_x);
155     q = queue<int>();
156     for(int x = 1;x <= n_x; ++x)
157         if(st[x] == x && !match[x]) pa[x]=0,
158             ↪ S[x]=0, q.push(x);
159     if(q.empty())return false;
160     for(;){
161         while(q.size()){
162             int u = q.front();q.pop();
163             if(S[st[u]] == 1) continue;
164             for(int v = 1;v <= n; ++v)
165                 if(g[u][v].w > 0 && st[u] != st[v])
166                     ↪ {
167                         if(e_delta(g[u][v]) == 0){
168                             if(on_found_edge(g[u]
169                                 ↪ [v]))return true;
170                         else update_slack(u, st[v]);
171                     }
172                 }
173             }
174         }
175     }

```

```

161     }
162
163     int d = INF;
164     for(int b = n + 1; b <= n_x; ++b)
165         if(st[b] == b && S[b] == 1)d = min(d,
166             → lab[b]/2);
167     for(int x = 1; x <= n_x; ++x)
168         if(st[x] == x && slack[x]){
169             if(S[x] == -1)d = min(d,
170                 → e_delta(g[slack[x]][x]));
171             else if(S[x] == 0)d = min(d,
172                 → e_delta(g[slack[x]][x])/2);
173         }
174     for(int u = 1; u <= n; ++u){
175         if(S[st[u]] == 0){
176             if(lab[u] <= d) return 0;
177             lab[u] -= d;
178         }else if(S[st[u]] == 1)lab[u] += d;
179     }
180     for(int b = n+1; b <= n_x; ++b)
181         if(st[b] == b){
182             if(S[st[b]] == 0) lab[b] += d * 2;
183             else if(S[st[b]] == 1) lab[b] -= d
184                 → * 2;
185         }
186     q=queue<int>();
187     for(int x = 1; x <= n_x; ++x)
188         if(st[x] == x && slack[x] &&
189             → st[slack[x]] != x &&
190             → e_delta(g[slack[x]][x]) == 0)
191             if(on_found_edge(g[slack[x]]
192                 → [x]))return true;
193     for(int b = n + 1; b <= n_x; ++b)
194         if(st[b] == b && S[b] == 1 && lab[b] ==
195             → 0)expand_blossom(b);
196     }
197     return false;
198 }
199 inline pair<long long, int> solve(){
200     memset(match + 1, 0, sizeof(int) * n);
201     n_x = n;
202     int n_matches = 0;
203     long long tot_weight = 0;
204     for(int u = 0; u <= n; ++u) st[u] = u,
205         → flower[u].clear();
206     int w_max = 0;
207     for(int u = 1; u <= n; ++u)
208         for(int v = 1; v <= n; ++v){
209             flower_from[u][v] = (u == v ? u : 0);
210             w_max = max(w_max, g[u][v].w);
211         }
212     for(int u = 1; u <= n; ++u) lab[u] = w_max;
213     while(matching()) ++n_matches;
214     for(int u = 1; u <= n; ++u)
215         if(match[u] && match[u] < u)
216             tot_weight += g[u][match[u]].w;
217     return make_pair(tot_weight, n_matches);
218 }
219 inline void init(){
220     for(int u = 1; u <= n; ++u)
221         for(int v = 1; v <= n; ++v)
222             g[u][v]=edge(u, v, 0);
223 }
224 };

```

3.7.4 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 $\frac{1}{2}\text{rank}\tilde{A}$.
- $(\tilde{A}^{-1})_{i,j} \neq 0$ 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 $\tilde{A}_{u,v}^{-1} \neq 0$), 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.

3.8 支配树

记得建反图!

```

1 vector<int> G[maxn], R[maxn], son[maxn]; // R是反图,
2                                         → son存的是sdom树上的儿子
3
4 int ufs[maxn];
5
6 int idom[maxn], sdom[maxn], anc[maxn]; // anc:
7                                         → sdom的dfn最小的祖先
8
9 int p[maxn], dfn[maxn], id[maxn], tim;
10
11 int findufs(int x) {
12     if (ufs[x] == x)
13         return x;
14
15     int t = ufs[x];
16     ufs[x] = findufs(ufs[x]);
17
18     if (dfn[sdom[anc[x]]] > dfn[sdom[anc[t]]])
19         anc[x] = anc[t];
20
21     return ufs[x];
22 }
23
24 void dfs(int x) {
25     dfn[x] = ++tim;
26     id[tim] = x;
27     sdom[x] = x;
28
29     for (int y : G[x])
30         if (!dfn[y]) {
31             p[y] = x;
32             dfs(y);
33         }
34 }
35
36 void get_dominator(int n) {
37     for (int i = 1; i <= n; i++)
38         ufs[i] = anc[i] = i;
39
40     dfs(1);
41
42     for (int i = n; i > 1; i--) {
43         int x = id[i];
44
45         for (int y : R[x])
46             if (dfn[y]) {

```

```
45         findufs(y);
46         if (dfn[sdom[x]] > dfn[sdom[anc[y]]])
47             sdom[x] = sdom[anc[y]];
48     }
49
50     son[sdom[x]].push_back(x);
51     ufs[x] = p[x];
52
53     for (int u : son[p[x]]) {
54         findufs(u);
55         idom[u] = (sdm[u] == sdm[anc[u]] ? p[x] :
56                     anc[u]);
57     }
58
59     son[p[x]].clear();
60 }
61
62 for (int i = 2; i <= n; i++) {
63     int x = id[i];
64
65     if (idom[x] != sdom[x])
66         idom[x] = idom[idom[x]];
67
68     son[idom[x]].push_back(x);
69 }
```

```
// 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 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。完成是否有解之后，每个变量只需要取SCC编号较小的那个。

```
1 if (!ok)
2     printf("IMPOSSIBLE\n");
3 else {
4     printf("POSSIBLE\n");
5
6     for (int i = 1; i <= n; i++)
7         printf("%d%c", sccid[i * 2 - 1] > sccid[i * 2],
8                i < n ? ' ' : '\n');
9 }
```

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。

```
1 bool vis[maxn];
2 int stk[maxn], top;
3
4 // 主函数
5 for (int i = 0; i < n; i += 2)
6     if (!vis[i] && !vis[i ^ 1]) {
7         top = 0;
8         if (!dfs(i)) {
9             while (top)
10                 vis[stk[top--]] = false;
11
12         if (!dfs(i + 1))
13             bad = true;
14         break;
15     }
16 }
17
18 // 最后stk中的所有元素就是选中的值
19
```

3.10 最大流

3.10.1 Dinic

```

1 // 注意Dinic适用于二分图或分层图，对于一般稀疏图ISAP更优
2   → 稠密图则HLPP更优
3
4 struct edge {
5     int to, cap, prev;
6 } e[maxe * 2];
7
8 int last[maxn], len, d[maxn], cur[maxn], q[maxn];
9
10 // main函数里要初始化
11 memset(last, -1, sizeof(last));
12
13 void AddEdge(int x, int y, int z) {
14     e[len].to = y;
15     e[len].cap = z;
16     e[len].prev = last[x];
17     last[x] = len++;
18 }
19
20 void addedge(int x, int y, int z) {
21     AddEdge(x, y, z);
22     AddEdge(y, x, 0);
23 }
24
25 void bfs() {
26     int head = 0, tail = 0;
27     memset(d, -1, sizeof(int) * (t + 5));
28     q[tail++] = s;
29     d[s] = 0;
30
31     while (head != tail) {
32         int x = q[head++];
33         for (int i = last[x]; ~i; i = e[i].prev)
34             if (e[i].cap > 0 && d[e[i].to] == -1) {
35                 d[e[i].to] = d[x] + 1;
36                 q[tail++] = e[i].to;
37             }
38     }
39 }
40
41 int dfs(int x, int a) {
42     if (x == t || !a)
43         return a;
44
45     int flow = 0, f;
46     for (int &i = cur[x]; ~i; i = e[i].prev)
47         if (e[i].cap > 0 && d[e[i].to] == d[x] + 1 &&
48             (f = dfs(e[i].to, min(e[i].cap, a)))) {
49             e[i].cap -= f;
50             e[e[i].prev].cap += f;
51             cur[x] = i;
52             flow += f;
53         }
54     return flow;
55 }

```

```

47     e[i].cap -= f;
48     e[i^1].cap += f;
49     flow += f;
50     a -= f;
51
52     if (!a)
53         break;
54 }
55
56 return flow;
57 }
58
59 int Dinic() {
60     int flow = 0;
61     while (bfs(), ~d[t]) {
62         memcpy(cur, last, sizeof(int) * (t + 5));
63         flow += dfs(s, inf);
64     }
65     return flow;
66 }
67

```

3.10.2 ISAP

可能有毒，慎用。

```

1 // 注意ISAP适用于一般稀疏图，对于二分图或分层图情
2 // 况Dinic比较优，稠密图则HLPP更优
3
4 // 边的定义
5 // 这里没有记录起点和反向边，因为反向边即为正向边xor 1,
6 // 起点即为反向边的终点
7 struct edge {
8     int to, cap, prev;
9 } e[maxe * 2];
10
11 // 全局变量和数组定义
12 int last[maxn], cnte = 0, d[maxn], p[maxn], c[maxn],
13 // cur[maxn], q[maxn];
14 int n, m, s, t; // s, t一定要开成全局变量
15
16 void AddEdge(int x, int y, int z) {
17     e[cnte].to = y;
18     e[cnte].cap = z;
19     e[cnte].prev = last[x];
20     last[x] = cnte++;
21 }
22
23 void addedge(int x, int y, int z) {
24     AddEdge(x, y, z);
25     AddEdge(y, x, 0);
26 }
27
28 // 预处理到t的距离标号
29 // 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
30 void bfs() {
31     memset(d, -1, sizeof(d));
32
33     int head = 0, tail = 0;
34     d[t] = 0;
35     q[tail++] = t;
36
37     while (head != tail) {
38         int x = q[head++];
39         c[d[x]]++;
40
41         for (int i = last[x]; ~i; i = e[i].prev)
42             if (e[i ^ 1].cap && d[e[i].to] == -1) {
43                 d[e[i].to] = d[x] + 1;
44                 q[tail++] = e[i].to;
45             }
46     }
47 }
48
49 // augment函数 O(n) 沿增广路增广一次，返回增广的流量
50 int augment() {
51     int a = (~0u) >> 1; // INT_MAX
52
53     for (int x = t; x != s; x = e[p[x] ^ 1].to)
54         e[p[x]].cap -= a;
55         e[p[x] ^ 1].cap += a;
56
57     return a;
58 }
59
60 // 主过程 O(n^2 m)，返回最大流的流量
61 // 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
62 // 变量记录下来并修改代码
63 int ISAP() {
64     bfs();
65
66     memcpy(cur, last, sizeof(cur));
67
68     int x = s, flow = 0;
69
70     while (d[s] < n) {
71         if (x == t) { // 如果走到了t就增广一次，并返
72             // 回s重新找增广路
73             flow += augment();
74             x = s;
75         }
76
77         bool ok = false;
78         for (int &i = cur[x]; ~i; i = e[i].prev)
79             if (e[i].cap && d[x] == d[e[i].to] + 1) {
80                 p[e[i].to] = i;
81                 x = e[i].to;
82
83                 ok = true;
84                 break;
85             }
86
87         if (!ok) { // 修改距离标号
88             int tmp = n - 1;
89             for (int i = last[x]; ~i; i = e[i].prev)
90                 if (e[i].cap)
91                     tmp = min(tmp, d[e[i].to] + 1);
92
93             if (!--c[d[x]])
94                 break; // gap优化，一定要加上
95
96             c[d[x] = tmp]++;
97             cur[x] = last[x];
98
99             if (x != s)
100                 x = e[p[x] ^ 1].to;
101
102     }
103
104     return flow;
105 }
106
107 // 重要！main函数最前面一定要加上如下初始化
108 memset(last, -1, sizeof(last));

```

3.10.3 HLPP 最高标号预流推进

```

1 constexpr int maxn = 1205, maxe = 120005;
2
3 struct edge {
4     int to, cap, prev;
5 } e[maxn * 2];
6
7 int n, m, s, t;
8 int last[maxn], cnte;
9 int h[maxn], gap[maxn * 2];
10 long long ex[maxn]; // 多余流量
11 bool inq[maxn];
12
13 struct cmp {
14     bool operator() (int x, int y) const {
15         return h[x] < h[y];
16     }
17 };
18 priority_queue<int, vector<int>, cmp> heap;
19
20 void adde(int x, int y, int z) {
21     e[cnte].to = y;
22     e[cnte].cap = z;
23     e[cnte].prev = last[x];
24     last[x] = cnte++;
25 }
26
27 void addedge(int x, int y, int z) {
28     adde(x, y, z);
29     adde(y, x, 0);
30 }
31
32
33 bool bfs() {
34     static int q[maxn];
35
36     fill(h, h + n + 1, 2 * n); // 如果没有全局的n, 记得
37     // 改这里
38     int head = 0, tail = 0;
39     q[tail++] = t;
40     h[t] = 0;
41
42     while (head < tail) {
43         int x = q[head++];
44         for (int i = last[x]; ~i; i = e[i].prev)
45             if (e[i ^ 1].cap && h[e[i].to] > h[x] + 1)
46                 {
47                     h[e[i].to] = h[x] + 1;
48                     q[tail++] = e[i].to;
49                 }
50
51     }
52
53     return h[s] < 2 * n;
54 }
55
56 void push(int x) {
57     for (int i = last[x]; ~i; i = e[i].prev)
58         if (e[i].cap && h[x] == h[e[i].to] + 1) {
59             int d = min(ex[x], (long long)e[i].cap);
60
61             e[i].cap -= d;
62             e[i ^ 1].cap += d;
63             ex[x] -= d;
64             ex[e[i].to] += d;
65
66             if (e[i].to != s && e[i].to != t &&
67                 !inq[e[i].to]) {
68                 heap.push(e[i].to);
69                 inq[e[i].to] = true;
70             }
71         }
72
73     void relabel(int x) {
74         h[x] = 2 * n;
75
76         for (int i = last[x]; ~i; i = e[i].prev)
77             if (e[i].cap)
78                 h[x] = min(h[x], h[e[i].to] + 1);
79     }
80
81 long long hlpp() {
82     if (!bfs())
83         return 0;
84
85     // memset(gap, 0, sizeof(int) * 2 * n);
86     h[s] = n;
87
88     for (int i = 1; i <= n; i++)
89         gap[h[i]]++;
90
91     for (int i = last[s]; ~i; i = e[i].prev)
92         if (e[i].cap) {
93             int d = e[i].cap;
94
95             e[i].cap -= d;
96             e[i ^ 1].cap += d;
97             ex[s] -= d;
98             ex[e[i].to] += d;
99
100            if (e[i].to != s && e[i].to != t &&
101                !inq[e[i].to]) {
102                heap.push(e[i].to);
103                inq[e[i].to] = true;
104            }
105
106        while (!heap.empty()) {
107            int x = heap.top();
108            heap.pop();
109            inq[x] = false;
110
111            push(x);
112            if (ex[x]) {
113                if (!--gap[h[x]]) { // gap
114                    for (int i = 1; i <= n; i++)
115                        if (i != s && i != t && h[i] >
116                            h[x])
117                            h[i] = n + 1;
118
119                relabel(x);
120                ++gap[h[x]];
121                heap.push(x);
122                inq[x] = true;
123            }
124
125        return ex[t];
126    }
127
128 //记得初始化
129 memset(last, -1, sizeof(last));
130

```

3.11 费用流

3.11.1 SPFA 费用流

```

1 constexpr int maxn = 20005, maxm = 200005;
2
3 struct edge {
4     int to, prev, cap, w;
5 } e[maxn * 2];
6
7 int last[maxn], cnte, d[maxn], p[maxn]; // 记得把last初始化成-1, 不然会死循环
8 bool inq[maxn];
9
10 void spfa(int s) {
11
12     memset(d, -63, sizeof(d));
13     memset(p, -1, sizeof(p));
14
15     queue<int> q;
16
17     q.push(s);
18     d[s] = 0;
19
20     while (!q.empty()) {
21         int x = q.front();
22         q.pop();
23         inq[x] = false;
24
25         for (int i = last[x]; ~i; i = e[i].prev)
26             if (e[i].cap) {
27                 int y = e[i].to;
28
29                 if (d[x] + e[i].w > d[y]) {
30                     p[y] = i;
31                     d[y] = d[x] + e[i].w;
32                     if (!inq[y]) {
33                         q.push(y);
34                         inq[y] = true;
35                     }
36                 }
37             }
38     }
39 }
40
41 int mcmf(int s, int t) {
42     int ans = 0;
43
44     while (spfa(s), d[t] > 0) {
45         int flow = 0x3f3f3f3f;
46         for (int x = t; x != s; x = e[p[x] ^ 1].to)
47             flow = min(flow, e[p[x]].cap);
48
49         ans += flow * d[t];
50
51         for (int x = t; x != s; x = e[p[x] ^ 1].to) {
52             e[p[x]].cap -= flow;
53             e[p[x] ^ 1].cap += flow;
54         }
55     }
56
57     return ans;
58 }
59
60 void add(int x, int y, int c, int w) {
61     e[cnte].to = y;
62     e[cnte].cap = c;
63     e[cnte].w = w;
64
65     e[cnte].prev = last[x];
66     last[x] = cnte++;
67 }
68

```

```
69 void addedge(int x, int y, int c, int w) {  
70     add(x, y, c, w);  
71     add(y, x, 0, -w);  
72 }
```

3.11.2 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 $u \rightarrow v$ 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。

```

1 struct edge {
2     int to, cap, prev, w;
3 } e[maxe * 2];
4
5 int last[maxn], cnte;
6
7 long long d[maxn], h[maxn];
8 int p[maxn];
9
10 bool vis[maxn];
11 int s, t;
12
13 void Adde(int x, int y, int z, int w) {
14     e[cnte].to = y;
15     e[cnte].cap = z;
16     e[cnte].w = w;
17     e[cnte].prev = last[x];
18     last[x] = cnte++;
19 }
20
21 void addedge(int x, int y, int z, int w) {
22     Adde(x, y, z, w);
23     Adde(y, x, 0, -w);
24 }
25
26 void dijkstra() {
27     memset(d, 63, sizeof(d));
28     memset(vis, 0, sizeof(vis));
29
30     priority_queue<pair<long long, int> > heap;
31
32     d[s] = 0;
33     heap.push(make_pair(0ll, s));
34
35     while (!heap.empty()) {
36         int x = heap.top().second;
37         heap.pop();
38
39         if (vis[x])
40             continue;
41
42         vis[x] = true;
43         for (int i = last[x]; ~i; i = e[i].prev)
44             if (e[i].cap > 0 && d[e[i].to] > d[x] +
45                 → e[i].w + h[x] - h[e[i].to]) {
46                 d[e[i].to] = d[x] + e[i].w + h[x] -
47                 → h[e[i].to];
48                 p[e[i].to] = i;
49                 heap.push(make_pair(-d[e[i].to],
50                                     → e[i].to));
51             }
52     }
53 }

```

```

48     }
49 }
50
51
52 pair<long long, long long> mcmf() {
53     /*
54      spfa();
55      for (int i = 1; i <= t; i++)
56          h[i] = d[i];
57      // 如果初始有负权就像这样跑一遍SPFA预处理
58      */
59
60     long long flow = 0, cost = 0;
61
62     while (dijkstra(), d[t] < 0x3f3f3f3f) {
63         for (int i = 1; i <= t; i++)
64             h[i] += d[i];
65
66         int a = 0x3f3f3f3f;
67
68         for (int x = t; x != s; x = e[p[x] ^ 1].to)
69             a = min(a, e[p[x]].cap);
70
71         flow += a;
72         cost += (long long)a * h[t];
73
74         for (int x = t; x != s; x = e[p[x] ^ 1].to) {
75             e[p[x]].cap -= a;
76             e[p[x] ^ 1].cap += a;
77         }
78     }
79
80     return make_pair(flow, cost);
81 }
82
83 // 记得初始化
84 memset(last, -1, sizeof(last));

```

3.11.3 预流推进费用流 (可处理负环) $O(nm \log C)$

不是很懂什么原理, 待研究.

```

1 // Push-Relabel implementation of the cost-scaling
2     ↪ algorithm
3 // Runs in  $O(\max\_flow * \log(V * \max\_edge\_cost)) = O($ 
4     ↪  $V^3 * \log(V * C))$ 
5 // Really fast in practice, 3e4 edges are fine.
6 // Operates on integers, costs are multiplied by N!!
7
8 #include <bits/stdc++.h>
9 using namespace std;
10
11 // source: unknown
12 template<typename flow_t = int, typename cost_t = int>
13 struct mcSFlow {
14     struct Edge {
15         cost_t c;
16         flow_t f;
17         int to, rev;
18         Edge(int _to, cost_t _c, flow_t _f, int _rev):
19             c(_c), f(_f), to(_to), rev(_rev) {}
20     };
21
22     static constexpr cost_t INF_COST =
23         numeric_limits<cost_t>::max() / 2;
24
25     cost_t eps;
26     int N, S, T;
27     vector<vector<Edge>> G;

```

```

87         }
88     }
89
90     if (++co[h[u]], !--co[hi] && hi
91         <= N)
92         for (int i = 0; i < N; ++i)
93             if (hi < h[i] && h[i] <
94                 N) {
95                 --co[h[i]];
96                 h[i] = N + 1;
97             }
98
99             hi = h[u];
100        }
101    else if (G[u][cur[u]].f && h[u] ==
102             >= h[G[u][cur[u]].to] + 1)
103        add_flow(G[u][cur[u]],
104                 >= min(ex[u], G[u]
105                     >= [cur[u]].f));
106    else
107        ++cur[u];
108
109    while (hi >= 0 && hs[hi].empty())
110        --hi;
111
112    return -ex[S];
113}
114
115 void push(Edge &e, flow_t amt) {
116     if (e.f < amt)
117         amt = e.f;
118
119     e.f -= amt;
120     ex[e.to] += amt;
121     G[e.to][e.rev].f += amt;
122     ex[G[e.to][e.rev].to] -= amt;
123 }
124
125 void relabel(int vertex) {
126     cost_t newHeight = -INFCOST;
127
128     for (unsigned int i = 0; i < G[vertex].size();
129         >= ++i) {
130         Edge const &e = G[vertex][i];
131
132         if (e.f && newHeight < h[e.to] - e.c) {
133             newHeight = h[e.to] - e.c;
134             cur[vertex] = i;
135         }
136     }
137
138     h[vertex] = newHeight - eps;
139 }
140
141 static constexpr int scale = 2;
142
143 pair<flow_t, cost_t> minCostMaxFlow() {
144     cost_t retCost = 0;
145
146     for (int i = 0; i < N; ++i)
147         for (Edge &e : G[i])
148             retCost += e.c * (e.f);
149
150     isq.assign(N, 0);
151     cur.assign(N, 0);
152     queue<int> q;
153
154     for (; eps; eps >= scale) {
155         //refine
156         fill(cur.begin(), cur.end(), 0);
157
158         for (int i = 0; i < N; ++i)
159             for (auto &e : G[i])
160                 if (h[i] + e.c - h[e.to] < 0 &&
161                     >= e.f)
162                     push(e, e.f);
163
164         for (int i = 0; i < N; ++i) {
165             if (ex[i] > 0) {
166                 q.push(i);
167                 isq[i] = 1;
168             }
169
170         // make flow feasible
171         while (!q.empty()) {
172             int u = q.front();
173             q.pop();
174             isq[u] = 0;
175
176             while (ex[u] > 0) {
177                 if (cur[u] == G[u].size())
178                     relabel(u);
179
180                 for (unsigned int &i = cur[u],
181                     >= max_i = G[u].size(); i < max_i;
182                     >= ++i) {
183                     Edge &e = G[u][i];
184
185                     if (h[u] + e.c - h[e.to] < 0) {
186                         push(e, ex[u]);
187
188                         if (ex[e.to] > 0 &&
189                             >= isq[e.to] == 0) {
190                             q.push(e.to);
191                             isq[e.to] = 1;
192                         }
193
194                     }
195
196                 }
197
198             if (eps > 1 && eps >= scale == 0)
199                 eps = 1 << scale;
200         }
201
202         for (int i = 0; i < N; ++i)
203             for (Edge &e : G[i])
204                 retCost -= e.c * (e.f);
205
206         return make_pair(retFlow, retCost / 2 / N);
207     }
208
209     flow_t getFlow(Edge const &e) {
210         return G[e.to][e.rev].f;
211     }
212 }
213
214 int main() {

```

```

215
216     int n, m;
217     scanf("%d%d", &n, &m);
218
219     mcf<long long, long long> mcmf(n, 0, n - 1);
220
221     while (m--) {
222         int x, y, z, w;
223         scanf("%d%d%d%d", &x, &y, &z, &w);
224
225         mcmf.add_edge(x - 1, y - 1, z, w);
226     }
227
228     auto [flow, cost] = mcmf.minCostMaxFlow();
229
230     printf("%lld %lld\n", flow, cost);
231
232     return 0;
233 }
```

3.12 网络流原理

3.12.1 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.)

3.12.2 最小割

首先牢记最小割的定义: 选权值和尽量小的一些边, 使得删除这些边之后 s 无法到达 t .

- 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 .

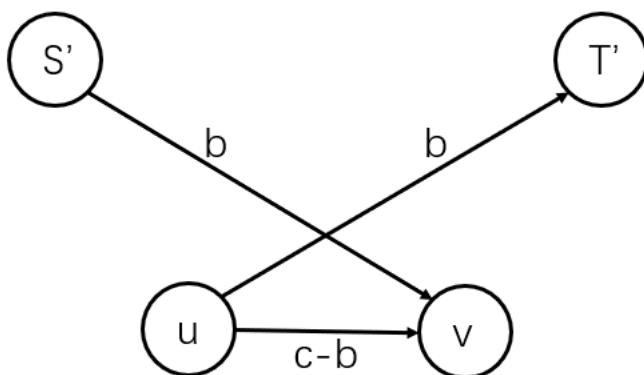
- 字典序最小的最小割

直接按字典序从小到大的顺序依次判断每条边能否在最小割中即可. 如果一条边是可行边, 我们就需要把它删掉, 同时进行退流, $u \rightarrow s$ 和 $t \rightarrow v$ 都退掉等同于这条边容量的流量. 退流用Dinic实现即可.

3.12.3 上下界网络流

无源汇上下界可行流

新建源汇 S', T' , 然后如图所示转化每一条边.



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.

有源汇上下界最大流

如果不需要判断是否有解的话可以直接按照和上面一样的方法转化. 因为附加边实际上算了两次流量, 所以最终答案应该减掉所有下界之和.

(另外这里如果要压缩附加边的话, 不能像无源汇的情况一样对每个点只开一个变量统计溢出的流量, 正确的做法是进出流量各统计一下, 每个点连两条附加边.)

如果需要判有解的话会出一点问题. 这时候就需要转化成无源汇的情况, 验证有解之后撤掉 T 到 S 的那条附加边再从 S 到 T 跑一遍最大流.

```

1 int ex[maxn], id[maxn];
2
3 int main() {
4
5     memset(last, -1, sizeof(last));
6
7     int n, m, src, sink;
8     scanf("%d%d%d%d", &n, &m, &src, &sink);
9     s = n + 1;
10    t = n + 2;
11
12    while (m--) {
13        int x, y, b, c;
14        scanf("%d%d%d%d", &x, &y, &b, &c);
15
16        addedge(x, y, c - b);
17
18        ex[y] += b;
19        ex[x] -= b;
20    }
21
22    for (int i = 1; i <= n; i++) {
23        id[i] = cnte;
24
25        if (ex[i] >= 0)
26            addedge(s, i, ex[i]);
27        else
28            addedge(i, t, -ex[i]);
29    }
30
31    addedge(sink, src, (~0u) >> 1);
32
33    Dinic();
34
35    if (any_of(id + 1, id + n + 1, [] (int i) {return
36        ~e[i].cap;})) {
37        printf("please go home to sleep\n");
38    } else {
39        int flow = e[cte - 1].cap;
40        e[cte - 1].cap = e[cte - 2].cap = 0;
41        s = src;
42        t = sink;
43
44        printf("%d\n", flow + Dinic());
45    }
46
47    return 0;
48 }
```

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, 然后撤掉超级源汇和附加边, 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.

3.12.4 常见建图方法

- 最大流/费用流

流量不是很多的时候可以理解成很多条路径，并且每条边可以经过的次数有限。

- 最小割

常用的模型是**最大权闭合子图**。当然它并不是万能的，因为限制条件可以带权值。

1. 如果某些点全部在 S 集或者 T 集则获得一个正的收益

把这个条件建成一个点，向要求的点连 ∞ 边，然后 s 向它连 ∞ 边。（如果是 T 集就都反过来）

那么如果它在 S 集就一定满足它要求的点都在 S 集，反之如果是 T 集亦然。

2. 如果两个点不在同一集合中则需要付出代价

建双向边，那显然如果它们不在同一集合中就需要割掉中间的边，付出对应的代价。

3. 二分图，如果相邻的两个点在同一集合则需要付出代价

染色后给一半的点反转源汇，就转换成上面的问题了。

3.12.5 例题

- 费用流

1. 序列上选和尽量大的数，但连续 k 个数中最多选 p 个。

费用流建图，先建一条 $n + 1$ 个点的无限容量的链表示不选，然后每个点往后面 k 个位置连边，答案是流量为 p 的最大费用流。因为条件等价于选 p 次并且每次选的所有数间隔都至少是 k 。

2. 还要求连续 k 个数中最少选 q 个。

任选一个位置把图前后切开就会发现通过截面的流量总和恰为 p 。注意到如果走了最开始的链就代表不选，因此要限制至少有 q 的流量不走链，那么只需要把链的容量改成 $p - q$ 就行了。

3.13 Prüfer序列

对一棵有 $n \geq 2$ 个结点的树，它的Prüfer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Prüfer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 。

Prüfer编码的性质

- 每个至少 2 个结点的树都唯一对应一个Prüfer编码。（当然也就叫做无根树哈希。）
- 每个点在Prüfer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Prüfer序列代码：

```

1 // 0-based
2 vector<vector<int>> adj;
3 vector<int> parent;
4
5 void dfs(int v) {
6     for (int u : adj[v]) {
7         if (u != parent[v]) parent[u] = v, dfs(u);
8     }
9 }
10
11 vector<int> pruefer_code() { // pruefer是德语
12     int n = adj.size();
13     parent.resize(n), parent[n - 1] = -1;
14     dfs(n - 1);
15 }
```

```

16     int ptr = -1;
17     vector<int> degree(n);
18     for (int i = 0; i < n; i++) {
19         degree[i] = adj[i].size();
20         if (degree[i] == 1 && ptr == -1) ptr = i;
21     }
22
23     vector<int> code(n - 2);
24     int leaf = ptr;
25     for (int i = 0; i < n - 2; i++) {
26         int next = parent[leaf];
27         code[i] = next;
28         if (--degree[next] == 1 && next < ptr)
29             leaf = next;
30         else {
31             ptr++;
32             while (degree[ptr] != 1)
33                 ptr++;
34             leaf = ptr;
35         }
36     }
37     return code;
38 }
```

线性重构树代码：

```

1 // 0-based
2 vector<pair<int, int>> pruefer_decode(vector<int> const
3     &code) {
4     int n = code.size() + 2;
5     vector<int> degree(n, 1);
6     for (int i : code) degree[i]++;
7
8     int ptr = 0;
9     while (degree[ptr] != 1) ptr++;
10    int leaf = ptr;
11
12    vector<pair<int, int>> edges;
13    for (int v : code) {
14        edges.emplace_back(leaf, v);
15        if (--degree[v] == 1 && v < ptr)
16            leaf = v;
17        else {
18            ptr++;
19            while (degree[ptr] != 1)
20                ptr++;
21            leaf = ptr;
22        }
23    }
24    edges.emplace_back(leaf, n - 1);
25    return edges;
26 }
```

3.14 弦图相关

Forked from the template 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 $\{p_1, p_2, \dots, p_t\}$ ，则 $\{p_1 \cup N(p_1), \dots, p_t \cup N(p_t)\}$ 为最小团覆盖

3.15 其他

3.15.1 Stoer-Wagner 全局最小割

```

1 const int N = 601;
2 int fa[N], siz[N], edge[N][N];
3
4 int find(int x) {
5     return fa[x] == x ? x : fa[x] = find(fa[x]);
6 }
7
8 int dist[N], vis[N], bin[N];
9 int n, m;
10
11 int contract(int& s, int& t) { // Find s, t
12     memset(dist, 0, sizeof(dist));
13     memset(vis, false, sizeof(vis));
14
15     int i, j, k, mincut, maxc;
16
17     for (i = 1; i <= n; i++) {
18         k = -1;
19         maxc = -1;
20
21         for (j = 1; j <= n; j++)
22             if (!bin[j] && !vis[j] && dist[j] > maxc) {
23                 k = j;
24                 maxc = dist[j];
25             }
26
27         if (k == -1)
28             return mincut;
29
30         s = t;
31         t = k;
32         mincut = maxc;
33         vis[k] = true;
34
35         for (j = 1; j <= n; j++)
36             if (!bin[j] && !vis[j]) dist[j] += edge[k]
37                 →[j];
38
39     return mincut;
40 }
41
42 const int inf = 0x3f3f3f3f;
43
44 int Stoer_Wagner() {
45     int mincut, i, j, s, t, ans;
46     for (mincut = inf, i = 1; i < n; i++) {
47         ans = contract(s, t);
48         bin[t] = true;
49
50         if (mincut > ans)
51             mincut = ans;
52         if (mincut == 0)
53             return 0;
54
55         for (j = 1; j <= n; j++)
56             if (!bin[j])
57                 edge[s][j] = (edge[j][s] += edge[j]
58                     →[t]);
59
60     return mincut;
61 }
62
63 int main() {
64     cin >> n >> m;

```

```

65
66     if (m < n - 1) {
67         cout << 0;
68         return 0;
69     }
70
71     for (int i = 1; i <= n; ++i)
72         fa[i] = i, siz[i] = 1;
73
74     for (int i = 1, u, v, w; i <= m; ++i) {
75         cin >> u >> v >> w;
76
77         int fu = find(u), fv = find(v);
78         if (fu != fv) {
79             if (siz[fu] > siz[fv]) swap(fu, fv);
80             fa[fu] = fv, siz[fv] += siz[fu];
81         }
82
83         edge[u][v] += w, edge[v][u] += w;
84     }
85
86     int fr = find(1);
87
88     if (siz[fr] != n) {
89         cout << 0;
90         return 0;
91     }
92
93     cout << Stoer_Wagner();
94
95     return 0;
96 }

```

4 数据结构

4.1 线段树

4.1.1 非递归线段树

- 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

```
1 int query(int l, int r) {
2     l += M - 1;
3     r += M + 1;
4
5     int ans = 0;
6     while (l ^ r != 1) {
7         ans += sum[l ^ 1] + sum[r ^ 1];
8
9         l >>= 1;
10        r >>= 1;
11    }
12
13    return ans;
14 }
```

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

```
1 void update(int l, int r, int d) {
2     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
3     for (l += n - 1, r += n + 1; l ^ r ^ 1; l >= 1, r
4         >>= 1, len <= 1) {
5         tree[l] += cntl * d, tree[r] += cntr * d;
6         if (~l & 1) tree[l ^ 1] += d * len, mark[l ^ 1]
7             += d, cntl += len;
8         if (r & 1) tree[r ^ 1] += d * len, mark[r ^ 1]
9             += d, cntr += len;
10    }
11 }
12
13 int query(int l, int r) {
14     int ans = 0, len = 1, cntl = 0, cntr = 0;
15     for (l += n - 1, r += n + 1; l ^ r ^ 1; l >= 1, r
16         >>= 1, len <= 1) {
17         ans += cntl * mark[l] + cntr * mark[r];
18         if (~l & 1) ans += tree[l ^ 1], cntl += len;
19         if (r & 1) ans += tree[r ^ 1], cntr += len;
20    }
21
22    for (l; l >= 1, r >= 1)
23        ans += cntl * mark[l] + cntr * mark[r];
24
25    return ans;
26 }
```

区间加，区间求最大值

```
1 void update(int l, int r, int d) {
2     for (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1,
3         r >>= 1) {
4         if (l < N) {
5             tree[l] = max(tree[l << 1], tree[l << 1 |
6                 r >> 1]) + mark[l];
```

```

5     |     tree[r] = max(tree[r << 1], tree[r << 1 |
6     |         ↪ 1]) + mark[r];
7 }
8
9     | if (~l & 1) {
10    |     tree[l ^ 1] += d;
11    |     mark[l ^ 1] += d;
12 }
13    | if (r & 1) {
14        tree[r ^ 1] += d;
15        mark[r ^ 1] += d;
16    }
17
18 for (; l; l >= 1, r >= 1)
19     | if (l < N) tree[l] = max(tree[l << 1], tree[l
20     |         ↪ << 1 | 1]) + mark[l],
21     |             tree[r] = max(tree[r << 1], tree[r
22     |         ↪ << 1 | 1]) + mark[r];
23 }
24
25 int query(int l, int r) {
26     int maxl = -INF, maxr = -INF;
27
28     for (l += N - 1, r += N + 1; l ^ r ^ 1; l >= 1, r
29     |         ↪ >= 1) {
30         maxl += mark[l];
31         maxr += mark[r];
32
33         if (~l & 1)
34             maxl = max(maxl, tree[l ^ 1]);
35         if (r & 1)
36             maxr = max(maxr, tree[r ^ 1]);
37     }
38
39     while (l) {
40         maxl += mark[l];
41         maxr += mark[r];
42
43         l >= 1;
44         r >= 1;
45     }
46
47     return max(maxl, maxr);
48 }

```

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。

```
1 constexpr int maxn = 100005, maxm = maxn * 70;
2
3 int lc[maxm], rc[maxm], cover[maxm], sum[maxm], root,
4     ↪ seg_cnt;
5
6 int s, t, d;
7
8 void refresh(int l, int r, int o) {
9     if (cover[o])
10         sum[o] = r - l + 1;
11     else
12         sum[o] = sum[lc[o]] + sum[rc[o]];
13 }
14
15 void modify(int l, int r, int &o) {
16     if (!o)
17         o = ++seg_cnt;
18
19     if (s <= l && t >= r) {
```

```

18     cover[o] += d;
19     refresh(l, r, o);
20
21     return;
22 }
23
24 int mid = (l + r) / 2;
25
26 if (s <= mid)
27     modify(l, mid, lc[o]);
28 if (t > mid)
29     modify(mid + 1, r, rc[o]);
30
31 refresh(l, r, o);
32 }
33
34 struct modi {
35     int x, l, r, d;
36
37     bool operator < (const modi &o) {
38         return x < o.x;
39     }
40 } a[maxn * 2];
41
42 int main() {
43
44     int n;
45     scanf("%d", &n);
46
47     for (int i = 1; i <= n; i++) {
48         int lx, ly, rx, ry;
49         scanf("%d%d%d%d", &lx, &ly, &rx, &ry);
50
51         a[i * 2 - 1] = {lx, ly + 1, ry, 1};
52         a[i * 2] = {rx, ly + 1, ry, -1};
53     }
54
55     sort(a + 1, a + n * 2 + 1);
56
57     int last = -1;
58     long long ans = 0;
59
60     for (int i = 1; i <= n * 2; i++) {
61         if (last != -1)
62             ans += (long long)(a[i].x - last) * sum[1];
63         last = a[i].x;
64
65         s = a[i].l;
66         t = a[i].r;
67         d = a[i].d;
68
69         modify(1, 1e9, root);
70     }
71
72     printf("%lld\n", ans);
73
74     return 0;
75 }

```

4.1.3 历史和

EC-Final2020 G, 原题是询问某个区间有多少子区间, 满足子区间中数的种类数为奇数.

离线之后转化成枚举右端点并用线段树维护左端点, 然后就是一个支持区间反转(0/1互换)和询问历史和的线段树.

“既然标记会复合, 就说明在两个标记中间没有经过任何 pushup 操作

也就是说一个这两个标记对应着相同的 0 的数量以及相同的 1 的数量

那么标记对于答案的影响只能是 $a * 0 + b * 1$
我们维护 a, b 即可”

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = (1 << 20) + 5;
6
7 int cnt[maxn][2], mul[maxn][2];
8 bool rev[maxn];
9 long long sum[maxn];
10
11 int now;
12
13 void build(int l, int r, int o) {
14     cnt[o][0] = r - l + 1;
15
16     if (l == r)
17         return;
18
19     int mid = (l + r) / 2;
20
21     build(l, mid, o * 2);
22     build(mid + 1, r, o * 2 + 1);
23 }
24
25 void apply(int o, bool flip, long long w0, long long
26           ← w1) {
27     sum[o] += w0 * cnt[o][0] + w1 * cnt[o][1];
28
29     if (flip)
30         swap(cnt[o][0], cnt[o][1]);
31
32     if (rev[o])
33         swap(w0, w1);
34
35     mul[o][0] += w0;
36     mul[o][1] += w1;
37     rev[o] ^= flip;
38 }
39
40 void pushdown(int o) {
41     if (!mul[o][0] && !mul[o][1] && !rev[o])
42         return;
43
44     apply(o * 2, rev[o], mul[o][0], mul[o][1]);
45     apply(o * 2 + 1, rev[o], mul[o][0], mul[o][1]);
46
47     mul[o][0] = mul[o][1] = 0;
48     rev[o] = false;
49 }
50
51 void update(int o) {
52     cnt[o][0] = cnt[o * 2][0] + cnt[o * 2 + 1][0];
53     cnt[o][1] = cnt[o * 2][1] + cnt[o * 2 + 1][1];
54
55     sum[o] = sum[o * 2] + sum[o * 2 + 1];
56 }
57
58 int s, t;
59
60 void modify(int l, int r, int o) {
61     if (s <= l && t >= r) {
62         apply(o, true, 0, 0);
63         return;
64     }
65
66     int mid = (l + r) / 2;
67     pushdown(o);

```

```

67     if (s <= mid)
68         modify(l, mid, o * 2);
69     if (t > mid)
70         modify(mid + 1, r, o * 2 + 1);
71
72     update(o);
73 }
74
75
76 long long query(int l, int r, int o) {
77     if (s <= l && t >= r)
78         return sum[o];
79
80     int mid = (l + r) / 2;
81     pushdown(o);
82
83     long long ans = 0;
84     if (s <= mid)
85         ans += query(l, mid, o * 2);
86     if (t > mid)
87         ans += query(mid + 1, r, o * 2 + 1);
88
89     return ans;
90 }
91
92 vector<pair<int, int>> vec[maxn]; // pos, id
93
94 long long ans[maxn];
95 int a[maxn], last[maxn];
96
97 int main() {
98
99     int n;
100    scanf("%d", &n);
101
102    build(1, n, 1);
103
104    for (int i = 1; i <= n; i++)
105        scanf("%d", &a[i]);
106
107    int m;
108    scanf("%d", &m);
109
110    for (int i = 1; i <= m; i++) {
111        int l, r;
112        scanf("%d%d", &l, &r);
113
114        vec[r].emplace_back(l, i);
115    }
116
117    for (int i = 1; i <= n; i++) {
118        s = last[a[i]] + 1;
119        t = now = i;
120
121        modify(1, n, 1);
122        apply(1, false, 0, 1);
123
124        for (auto [l, k] : vec[i]) {
125            s = l;
126            ans[k] = query(1, n, 1);
127        }
128
129        last[a[i]] = i;
130    }
131
132    for (int i = 1; i <= m; i++)
133        printf("%lld\n", ans[i]);
134
135    return 0;
}

```

136 }

4.2 陈丹琦分治

4.2.1 动态图连通性（分治并查集）

```

1 vector<pair<int, int>> seg[(1 << 22) + 5];
2
3 int s, t;
4 pair<int, int> d;
5
6 void add(int l, int r, int o) {
7     if (s > t)
8         return;
9
10    if (s <= l && t >= r) {
11        seg[o].push_back(d);
12        return;
13    }
14
15    int mid = (l + r) / 2;
16
17    if (s <= mid)
18        add(l, mid, o * 2);
19    if (t > mid)
20        add(mid + 1, r, o * 2 + 1);
21 }
22
23 int ufs[maxn], sz[maxn], stk[maxn], top;
24
25 int findufs(int x) {
26     while (ufs[x] != x)
27         x = ufs[x];
28
29     return ufs[x];
30 }
31
32 void link(int x, int y) {
33     x = findufs(x);
34     y = findufs(y);
35
36     if (x == y)
37         return;
38
39     if (sz[x] < sz[y])
40         swap(x, y);
41
42     ufs[y] = x;
43     sz[x] += sz[y];
44     stk[++top] = y;
45 }
46
47 int ans[maxm];
48
49 void solve(int l, int r, int o) {
50     int tmp = top;
51
52     for (auto pi : seg[o])
53         link(pi.first, pi.second);
54
55     if (l == r)
56         ans[l] = top;
57     else {
58         int mid = (l + r) / 2;
59
60         solve(l, mid, o * 2);
61         solve(mid + 1, r, o * 2 + 1);
62     }
63 }

```

```
64     for (int i = top; i > tmp; i--) {
65         int x = stk[i];
66
67         sz[ufs[x]] -= sz[x];
68         ufs[x] = x;
69     }
70
71     top = tmp;
72 }
73
74 map<pair<int, int>, int> mp;
```

```

55
56         t[k++] = b[i++];
57     }
58     else{
59         if (!b[j].ins)
60             ans += query(b[j].z - 1);
61
62         t[k++] = b[j++];
63     }
64 }
65
66 while (i <= mid) {
67     if (b[i].ins)
68         add(b[i].z, 1);
69
70     t[k++] = b[i++];
71 }
72
73 while (j <= r) {
74     if (!b[j].ins)
75         ans += query(b[j].z - 1);
76
77     t[k++] = b[j++];
78 }
79
80 for (i = l; i <= mid; i++)
81     if (b[i].ins)
82         add(b[i].z, -1);
83
84 copy(t + l, t + r + 1, b + l);
85 }

```

4.2.2 四维偏序

```
// 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
    →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
            i++;
        } else {
            if (b[j].ins)
                add(b[j].z, -1); // 树状数组
            j++;
        }
    }

    while (i <= mid) {
        if (b[i].ins)
            add(b[i].z, 1); // 树状数组
        i++;
    }

    while (j <= r) {
        if (b[j].ins)
            add(b[j].z, -1); // 树状数组
        j++;
    }
}
```

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）。

```
1 int op[maxn], ql[maxn], qr[maxn]; // 1: modify 2:
2   ↪ query
3 long long qk[maxn]; // 修改和询问可以一起存
4
5
6 void solve(int l, int r, vector<int> v) { // 如果想卡常
7   ↪ 可以用数组，然后只需要传一个数组的l, r; 递归的时候类
8   ↪ 似归并反过来，开两个辅助数组，处理完再复制回去即可
9
10  if (v.empty())
11    return;
12
13  if (l == r) {
14    for (int i : v)
15      if (op[i] == 2)
16        ans[i] = l;
17
18    return;
19  }
20
21  int mid = (l + r) / 2;
22
23  vector<int> vl, vr;
24
25  for (int i : v) {
26    if (op[i] == 1) {
27      if (qk[i] <= mid)
```

```

25     vl.push_back(i);
26     else {
27         update(ql[i], qr[i], 1); // update是区间
28         ↪加
29         vr.push_back(i);
30     }
31     else {
32         long long tmp = query(ql[i], qr[i]);
33
34         if (qk[i] <= tmp) // 因为是k大数查询
35             vr.push_back(i);
36         else {
37             qk[i] -= tmp;
38             vl.push_back(i);
39         }
40     }
41 }
42
43 for (int i : vr)
44     if (op[i] == 1)
45         update(ql[i], qr[i], -1);
46
v.clear();
48
49 solve(l, mid, vl);
50 solve(mid + 1, r, vr);
51 }
52
53 int main() {
54     int n, m;
55     scanf("%d%d", &n, &m);
56
57     M = 1;
58     while (M < n + 2)
59         M *= 2;
60
61     for (int i = 1; i <= m; i++)
62         scanf("%d%d%d%lld", &op[i], &ql[i], &qr[i],
63               ↪&qk[i]);
64
65     vector<int> v;
66     for (int i = 1; i <= m; i++)
67         v.push_back(i);
68
69     solve(1, 1e9, v);
70
71     for (int i = 1; i <= m; i++)
72         if (op[i] == 2)
73             printf("%d\n", ans[i]);
74
75     return 0;
}

```

4.4 平衡树

pb_ds 平衡树参见 8.11. Public Based DataStructure (PB_DS)
(第 93 页).

4.4.1 Treap

```

1 // 注意: 相同键值可以共存
2
3 struct node { // 结点类定义
4     int key, size, p; // 分别为键值, 子树大小, 优先度
5     node *ch[2]; // 0表示左儿子, 1表示右儿子
6
7     node(int key = 0) : key(key), size(1), p(rand()) {}
8

```

```

9     void refresh() {
10         size = ch[0] -> size + ch[1] -> size + 1;
11         } // 更新子树大小 (和附加信息, 如果有的话)
12     } null[maxn], *root = null, *ptr = null; // 数组名叫
13     ↪做null是为了方便开哨兵节点
14     // 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
15     ↪垃圾回收
16     // 注意: 数组里的元素一定不能delete, 否则会导致RE
17     // 重要! 在主函数最开始一定要加上以下预处理:
18     null -> ch[0] = null -> ch[1] = null;
19     null -> size = 0;
20
21     // 伪构造函数 O(1)
22     // 为了方便, 在结点类外面再定义一个伪构造函数
23     node *newnode(int x) { // 键值为x
24         ++ptr = node(x);
25         ptr -> ch[0] = ptr -> ch[1] = null;
26         return ptr;
27     }
28
29     // 插入键值 期望O(log n)
30     // 需要调用旋转
31     void insert(int x, node *&rt) { // rt为当前结点, 建议调
32         →用时传入root, 下同
33         if (rt == null) {
34             rt = newnode(x);
35             return;
36         }
37
38         int d = x > rt -> key;
39         insert(x, rt -> ch[d]);
40         rt -> refresh();
41
42         if (rt -> ch[d] -> p < rt -> p)
43             rot(rt, d ^ 1);
44
45     // 删除一个键值 期望O(log n)
46     // 要求键值必须存在至少一个, 否则会导致RE
47     // 需要调用旋转
48     void erase(int x, node *&rt) {
49         if (x == rt -> key) {
50             if (rt -> ch[0] != null && rt -> ch[1] != null)
51                 →{
52                     int d = rt -> ch[0] -> p < rt -> ch[1] ->
53                         →p;
54                     rot(rt, d);
55                     erase(x, rt -> ch[d]);
56                 }
57             else
58                 rt = rt -> ch[rt -> ch[0] == null];
59         }
60         else
61             erase(x, rt -> ch[x > rt -> key]);
62
63         if (rt != null)
64             rt -> refresh();
65     }
66
67     // 求元素的排名 (严格小于键值的个数 + 1) 期望O(log n)
68     // 非递归
69     int rank(int x, node *rt) {
70         int ans = 1, d;
71         while (rt != null) {
72             if ((d = x > rt -> key))
73                 ans += rt -> ch[0] -> size + 1;
74             rt = rt -> ch[d];
75         }
76     }

```

```

73     }
74
75     return ans;
76 }
77
78 // 返回排名第k(从1开始)的键值对应的指针 期望O(\log n)
79 // 非递归
80 node *kth(int x, node *rt) {
81     int d;
82     while (rt != null) {
83         if (x == rt -> ch[0] -> size + 1)
84             return rt;
85
86         if ((d = x > rt -> ch[0] -> size))
87             x -= rt -> ch[0] -> size + 1;
88
89         rt = rt -> ch[d];
90     }
91
92     return rt;
93 }
94
95 // 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)
96 // 非递归
97 node *pred(int x, node *rt) {
98     node *y = null;
99     int d;
100
101    while (rt != null) {
102        if ((d = x > rt -> key))
103            y = rt;
104
105        rt = rt -> ch[d];
106    }
107
108    return y;
109 }
110
111 // 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)
112 // 非递归
113 node *succ(int x, node *rt) {
114     node *y = null;
115     int d;
116
117     while (rt != null) {
118         if ((d = x < rt -> key))
119             y = rt;
120
121         rt = rt -> ch[d ^ 1];
122     }
123
124     return y;
125 }
126
127 // 旋转(Treap版本) O(1)
128 // 平衡树基础操作
129 // 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
130 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
131     node *y = x -> ch[d ^ 1];
132
133     x -> ch[d ^ 1] = y -> ch[d];
134     y -> ch[d] = x;
135
136     x -> refresh();
137     (x = y) -> refresh();
138 }

```

4.4.2 无旋 Treap / 可持久化 Treap

```

1 struct node {
2     int val, size;
3     node *ch[2];
4
5     node(int val) : val(val), size(1) {}
6
7     inline void update() {
8         size = ch[0] -> size + ch[1] -> size;
9     }
10
11 } null[maxn];
12
13 node *copied(node *x) { // 如果不用可持久化的话, 直接用
14     // 就行了
15     return new node(*x);
16 }
17
18 node *merge(node *x, node *y) {
19     if (x == null)
20         return y;
21     if (y == null)
22         return x;
23
24     node *z;
25     if (rand() % (x -> size + y -> size) < x -> size) {
26         z = copied(x);
27         z -> ch[1] = merge(x -> ch[1], y);
28     }
29     else {
30         z = copied(y);
31         z -> ch[0] = merge(x, y -> ch[0]);
32     }
33
34     z -> update(); // 因为每次只有一边会递归到儿子, 所以
35     // z 不可能取到 null
36     return z;
37 }
38 pair<node*, node*> split(node *x, int k) { // 左边大小
39     // 为k
40     if (x == null)
41         return make_pair(null, null);
42
43     pair<node*, node*> pi(null, null);
44
45     if (k <= x -> ch[0] -> size) {
46         pi = split(x -> ch[0], k);
47
48         node *z = copied(x);
49         z -> ch[0] = pi.second;
50         z -> update();
51         pi.second = z;
52     }
53     else {
54         pi = split(x -> ch[1], k - x -> ch[0] -> size -
55         // 1);
56
57         node *y = copied(x);
58         y -> ch[1] = pi.first;
59         y -> update();
60         pi.first = y;
61     }
62 }
63

```

```

64 // 记得初始化null
65 int main() {
66     for (int i = 0; i <= n; i++)
67         null[i].ch[0] = null[i].ch[1] = null;
68     null->size = 0;
69
70     // do something
71
72     return 0;
73 }

```

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.

```

1 #define dir(x) ((x) == (x) -> p -> ch[1])
2
3 struct node {
4     int size;
5     bool rev;
6     node *ch[2], *p;
7
8     node() : size(1), rev(false) {}
9
10    void pushdown() {
11        if (!rev)
12            return;
13
14        ch[0]->rev ^= true;
15        ch[1]->rev ^= true;
16        swap(ch[0], ch[1]);
17
18        rev=false;
19    }
20
21    void refresh() {
22        size = ch[0]->size + ch[1]->size + 1;
23    }
24 } null[maxn], *root = null;
25
26 void rot(node **x, int d) {
27     node *y = x->ch[d ^ 1];
28
29     if ((x->ch[d ^ 1] = y->ch[d]) != null)
30         y->ch[d]->p = x;
31     ((y->p = x->p) != null ? x->p->ch[dir(x)] :
32         root) = y;
33     (y->ch[d] = x)->p = y;
34
35     x->refresh();
36     y->refresh();
37 }
38
39 void splay(node *x, node *t) {
40     while (x->p != t) {
41         if (x->p->p == t) {
42             rot(x->p, dir(x) ^ 1);
43             break;
44         }
45
46         if (dir(x) == dir(x->p))
47             rot(x->p->p, dir(x->p) ^ 1);
48         else
49             rot(x->p, dir(x) ^ 1);
50     }
51 }
52
53 node *kth(int k, node *o) {

```

```

54     int d;
55     k++; // 因为最左边有一个哨兵
56
57     while (o != null) {
58         o->pushdown();
59
60         if (k == o->ch[0]->size + 1)
61             return o;
62
63         if ((d = k > o->ch[0]->size)) {
64             k -= o->ch[0]->size + 1;
65             o = o->ch[d];
66         }
67
68         return null;
69     }
70
71 void reverse(int l, int r) {
72     splay(kth(l - 1));
73     splay(kth(r + 1), root);
74
75     root->ch[1]->ch[0]->rev ^= true;
76 }
77
78 int n, m;
79
80 int main() {
81     null->size = 0;
82     null->ch[0] = null->ch[1] = null->p = null;
83
84     scanf("%d%d", &n, &m);
85     root = null + n + 1;
86     root->ch[0] = root->ch[1] = root->p = null;
87
88     for (int i = 1; i <= n; i++) {
89         null[i].ch[1] = null[i].p = null;
90         null[i].ch[0] = root;
91         root->p = null + i;
92         (root = null + i)->refresh();
93     }
94
95     null[n + 2].ch[1] = null[n + 2].p = null;
96     null[n + 2].ch[0] = root; // 这里直接建成一条链的,
97     // 如果想减少常数也可以递归建一个平衡的树
98     root->p = null + n + 2; // 总之记得建两个哨兵, 这
99     // 样splay起来不需要特判
100    (root = null + n + 2)->refresh();
101
102    // Do something
103
104    return 0;
105 }

```

4.5 树链剖分

4.5.1 动态树形 DP (最大权独立集)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 100005, maxm = 262155, inf =
6     ~0x3f3f3f3f;
7
8 struct binary_heap {
9     priority_queue<int> q, t;
10
11     binary_heap() {}

```

```

11 void push(int x) {
12     q.push(x);
13 }
14
15 void erase(int x) {
16     t.push(x);
17 }
18
19 int top() {
20     while (!t.empty() && q.top() == t.top()) {
21         q.pop();
22         t.pop();
23     }
24
25     return q.top();
26 }
27 } heap;
28
29 int pool[maxm][2][2], (*pt)[2][2] = pool;
30
31 void merge(int a[2][2], int b[2][2]) {
32     static int c[2][2];
33     memset(c, 0, sizeof(c));
34
35     for (int i = 0; i < 2; i++)
36         for (int j = 0; j < 2; j++)
37             for (int k = 0; k < 2; k++)
38                 if (!(j && k))
39                     for (int t = 0; t < 2; t++)
40                         c[i][t] = max(c[i][t], a[i][j]
41                                         + b[k][t]);
42
43     memcpy(a, c, sizeof(c));
44 }
45
46 vector<pair<int, int>> tw;
47
48 struct seg_tree {
49     int (*tr)[2][2], n;
50
51     int s, d[2];
52
53     seg_tree() {}
54
55     void update(int o) {
56         memcpy(tr[o], tr[o * 2], sizeof(int) * 4);
57         merge(tr[o], tr[o * 2 + 1]);
58     }
59
60     void build(int l, int r, int o) {
61         if (l == r) {
62             tr[o][0][0] = tw[l - 1].first;
63             tr[o][0][1] = tr[o][1][0] = -inf;
64             tr[o][1][1] = tw[l - 1].second;
65
66             return;
67         }
68
69         int mid = (l + r) / 2;
70
71         build(l, mid, o * 2);
72         build(mid + 1, r, o * 2 + 1);
73
74         update(o);
75     }
76
77     void modify(int l, int r, int o) {
78         if (l == r) {
79             tr[o][0][0] = d[0];
80             tr[o][0][1] = tr[o][1][0] = -inf;
81             tr[o][1][1] = d[1];
82
83             return;
84         }
85
86         int mid = (l + r) / 2;
87
88         if (s <= mid)
89             modify(l, mid, o * 2);
90         else
91             modify(mid + 1, r, o * 2 + 1);
92
93         update(o);
94     }
95
96     int getval() {
97         int ans = 0;
98         for (int i = 0; i < 2; i++)
99             for (int j = 0; j < 2; j++)
100                 ans = max(ans, tr[1][i][j]);
101
102     return ans;
103 }
104
105 pair<int, int> getpair() {
106     int ans[2] = {0};
107     for (int i = 0; i < 2; i++)
108         for (int j = 0; j < 2; j++)
109             ans[i] = max(ans[i], tr[1][i][j]);
110
111     return make_pair(ans[0], ans[1]);
112 }
113
114 void build(int len) {
115     n = len;
116     int N = 1;
117     while (N < n * 2)
118         N *= 2;
119
120     tr = pt;
121     pt += N;
122
123     build(1, n, 1);
124 }
125
126 void modify(int x, int dat[2]) {
127     s = x;
128     for (int i = 0; i < 2; i++)
129         d[i] = dat[i];
130     modify(1, n, 1);
131 }
132 } seg[maxn];
133
134 vector<int> G[maxn];
135
136 int p[maxn], d[maxn], sz[maxn], son[maxn], top[maxn];
137 int dp[maxn][2], dptr[maxn][2], w[maxn];
138
139 void dfs1(int x) {
140     d[x] = d[p[x]] + 1;
141     sz[x] = 1;
142
143     for (int y : G[x])
144         if (y != p[x]) {
145             p[y] = x;
146             dfs1(y);
147
148             if (sz[y] > sz[son[x]])

```

```

149     |     |     son[x] = y;
150
151     |     |     sz[x] += sz[y];
152 }
153
154 void dfs2(int x) {
155     if (x == son[p[x]]) {
156         top[x] = top[p[x]];
157     } else {
158         top[x] = x;
159
160         for (int y : G[x])
161             if (y != p[x])
162                 dfs2(y);
163
164
165         dp[x][1] = w[x];
166         for (int y : G[x])
167             if (y != p[x] && y != son[x]) {
168                 dp[x][1] += dptr[y][0];
169                 dp[x][0] += max(dptr[y][0], dptr[y][1]);
170             }
171
172         if (top[x] == x) {
173             tw.clear();
174
175             for (int u = x; u; u = son[u])
176                 tw.push_back(make_pair(dp[u][0], dp[u]
177                                         → [1]));
178
179             seg[x].build((int)tw.size());
180
181             tie(dptr[x][0], dptr[x][1]) = seg[x].getpair();
182
183         }
184     }
185
186 void modify(int x, int dat) {
187     dp[x][1] -= w[x];
188     dp[x][1] += (w[x] = dat);
189
190     while (x) {
191         if (p[top[x]]) {
192             dp[p[top[x]]][0] -= max(dptr[top[x]][0],
193                                     → dptr[top[x]][1]);
194             dp[p[top[x]]][1] -= dptr[top[x]][0];
195         }
196
197         heap.erase(seg[top[x]].getval());
198         seg[top[x]].modify(d[x] - d[top[x]] + 1,
199                           → dp[x]);
200         heap.push(seg[top[x]].getval());
201
202         tie(dptr[top[x]][0], dptr[top[x]][1]) =
203             → seg[top[x]].getpair();
204
205         if (p[top[x]]) {
206             dp[p[top[x]]][0] += max(dptr[top[x]][0],
207                                     → dptr[top[x]][1]);
208             dp[p[top[x]]][1] += dptr[top[x]][0];
209         }
210
211         x = p[top[x]];
212     }
213
214     int main() {
215         int n, m;
216
217         scanf("%d%d", &n, &m);
218
219         for (int i = 1; i <= n; i++)
220             scanf("%d", &w[i]);
221
222         for (int i = 1; i < n; i++) {
223             int x, y;
224             scanf("%d%d", &x, &y);
225             G[x].push_back(y);
226             G[y].push_back(x);
227         }
228
229         dfs1(1);
230         dfs2(1);
231
232         while (m--) {
233             int x, dat;
234             scanf("%d%d", &x, &dat);
235             modify(x, dat);
236
237             printf("%d\n", heap.top());
238         }
239
240         return 0;
241     }

```

4.6 树分治

4.6.1 动态树分治

```

1 // 为了减小常数, 这里采用bfs写法, 实测预处理比dfs快将近一
2 →半
3 // 以下以维护一个点到每个黑点的距离之和为例
4
5 // 全局数组定义
6 vector<int> G[maxn], W[maxn];
7 int size[maxn], son[maxn], q[maxn];
8 int p[maxn], depth[maxn], id[maxn][20], d[maxn][20]; //
9 → id是对应层所在子树的根
10 int a[maxn], ca[maxn], b[maxn][20], cb[maxn][20]; // 维
11 → 护距离和用的
12 bool vis[maxn], col[maxn];
13
14 // 建树 总计O(n log n)
15 // 需要调用找重心和预处理距离, 同时递归调用自身
16 void build(int x, int k, int s, int pr) { // 结点, 深度,
17   → 连通块大小, 点分树上的父亲
18   x = getcenter(x, s);
19   vis[x] = true;
20   depth[x] = k;
21   p[x] = pr;
22
23   for (int i = 0; i < (int)G[x].size(); i++)
24       if (!vis[G[x][i]]) {
25           d[G[x][i]][k] = W[x][i];
26           p[G[x][i]] = x;
27
28           getdis(G[x][i], k, G[x][i]); // bfs每个子树,
29           → 预处理距离
30       }
31
32   for (int i = 0; i < (int)G[x].size(); i++)
33       if (!vis[G[x][i]])
34           build(G[x][i], k + 1, size[G[x][i]], x); // → 递归建树
35
36 }

```

```

32 // 找重心 O(n)
33 int getcenter(int x, int s) {
34     int head = 0, tail = 0;
35     q[tail++] = x;
36
37     while (head != tail) {
38         x = q[head++];
39         size[x] = 1; // 这里不需要清空，因为以后要用的话
39             → 一定会重新赋值
40         son[x] = 0;
41
42         for (int i = 0; i < (int)G[x].size(); i++) {
43             if (!vis[G[x][i]] && G[x][i] != p[x]) {
44                 p[G[x][i]] = x;
45                 q[tail++] = G[x][i];
46             }
47     }
48
49     for (int i = tail - 1; i; i--) {
50         x = q[i];
51         size[p[x]] += size[x];
52
53         if (size[x] > size[son[p[x]]])
54             son[p[x]] = x;
55     }
56
57     x = q[0];
58     while (son[x] && size[son[x]] * 2 >= s)
59         x = son[x];
60
61     return x;
62 }
63
64 // 预处理距离 O(n)
65 // 方便起见，这里直接用了笨一点的方法，O(n \log n) 全存下来
66 void getdis(int x, int k, int rt) {
67     int head = 0, tail = 0;
68     q[tail++] = x;
69
70     while (head != tail) {
71         x = q[head++];
72         size[x] = 1;
73         id[x][k] = rt;
74
75         for (int i = 0; i < (int)G[x].size(); i++) {
76             if (!vis[G[x][i]] && G[x][i] != p[x]) {
77                 p[G[x][i]] = x;
78                 d[G[x][i]][k] = d[x][k] + w[x][i];
79
80                 q[tail++] = G[x][i];
81             }
82     }
83
84     for (int i = tail - 1; i; i--)
85         size[p[q[i]]] += size[q[i]]; // 后面递归建树要用
85             → 到子问题大小
86 }
87
88 // 修改 O(\log n)
89 void modify(int x) {
90     if (col[x])
91         ca[x]--;
92     else
93         ca[x]++; // 记得先特判自己作为重心的那层
94
95     for (int u = p[x], k = depth[x] - 1; u; u = p[u],
95             → k--) {
96         if (col[x])
96             a[u] -= d[x][k];

```

```

98         ca[u]--;
99
100        b[id[x][k]][k] -= d[x][k];
101        cb[id[x][k]][k]--;
102    }
103    else {
104        a[u] += d[x][k];
105        ca[u]++;
106
107        b[id[x][k]][k] += d[x][k];
108        cb[id[x][k]][k]++;
109    }
110}
111 col[x] ^= true;
112}
113
114 // 询问 O(\log n)
115 int query(int x) {
116     int ans = a[x]; // 特判自己是重心的那层
117
118     for (int u = p[x], k = depth[x] - 1; u; u = p[u],
118         → k--) {
119         ans += a[u] - b[id[x][k]][k] + d[x][k] * (ca[u]
119             → - cb[id[x][k]][k]);
120
121     }
122     return ans;
123 }

```

4.6.2 紫荆花之恋

稍微重构了一下，修改了代码风格。

另外这个是BFS版本，跑得比DFS要快不少。（虽然主要复杂度并不在重构上）

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 100005, maxk = 49;
6 constexpr double alpha = .75;
7
8 mt19937 rnd(23333333);
9
10 struct node {
11     int key, size, p;
12     node *ch[2];
13
14     node() {}
15
16     node(int key) : key(key), size(1), p(rnd()) {}
17
18     inline void update() {
19         size = ch[0] -> size + ch[1] -> size + 1;
20     }
21 } null[maxn * maxk], *pt = null;
22
23 vector<node*> pool;
24
25 node *newnode(int val) {
26     node *x;
27
28     if (!pool.empty()) {
29         x = pool.back();
30         pool.pop_back();
31     }
32     else
33         x = ++pt;
34

```

```

35     *x = node(val);
36     x -> ch[0] = x -> ch[1] = null;
37
38     return x;
39 }
40
41 void rot(node *&x, int d) {
42     node *y = x -> ch[d ^ 1];
43     x -> ch[d ^ 1] = y -> ch[d];
44     y -> ch[d] = x;
45
46     x -> update();
47     (x = y) -> update();
48 }
49
50 void insert(node *&o, int x) {
51     if (o == null) {
52         o = newnode(x);
53         return;
54     }
55
56     int d = (x > o -> key);
57
58     insert(o -> ch[d], x);
59     o -> update();
60
61     if (o -> ch[d] -> p < o -> p)
62         rot(o, d ^ 1);
63 }
64
65 int get_order(node *o, int x) {
66     int ans = 0;
67
68     while (o != null) {
69         int d = (x > o -> key);
70
71         if (d)
72             ans += o -> ch[0] -> size + 1;
73
74         o = o -> ch[d];
75     }
76
77     return ans;
78 }
79
80 void destroy(node *x) {
81     if (x == null)
82         return;
83
84     pool.push_back(x);
85     destroy(x -> ch[0]);
86     destroy(x -> ch[1]);
87 }
88
89 struct my_tree { // 封装了一下，如果不卡内存直接换成PBDS就好了
90     node *rt;
91
92     my_tree() : rt(null) {}
93
94     void clear() {
95         ::destroy(rt);
96         rt = null;
97     }
98
99     void insert(int x) {
100        ::insert(rt, x);
101    }
102
103    int order_of_key(int x) { // less than x
104
105        return ::get_order(rt, x);
106    }
107
108    vector<pair<int, int>> G[maxn];
109
110    int p[maxn], depth[maxn], d[maxn][maxk], rid[maxn]
111        -> [maxk];
112    int sz[maxn], siz[maxn][maxk], q[maxn];
113    bool vis[maxn];
114
115    int w[maxn];
116
117    void destroy(int o) {
118        int head = 0, tail = 0;
119        q[tail++] = o;
120        vis[o] = false;
121
122        while (head != tail) {
123            int x = q[head++];
124            tr[x].clear();
125
126            for (int i = depth[o]; i <= depth[x]; i++) {
127                tre[x][i].clear();
128                d[x][i] = rid[x][i] = siz[x][i] = 0;
129            }
130
131            for (auto pi : G[x]) {
132                int y = pi.first;
133
134                if (vis[y] && depth[y] >= depth[o]) {
135                    vis[y] = false;
136                    q[tail++] = y;
137                }
138            }
139        }
140
141        int getcenter(int o, int s) {
142            int head = 0, tail = 0;
143            q[tail++] = o;
144
145            while (head != tail) {
146                int x = q[head++];
147                sz[x] = 1;
148
149                for (auto pi : G[x]) {
150                    int y = pi.first;
151
152                    if (!vis[y] && y != p[x]) {
153                        p[y] = x;
154                        q[tail++] = y;
155                    }
156                }
157            }
158
159            for (int i = s - 1; i; i--)
160                sz[p[q[i]]] += sz[q[i]];
161
162            int x = o;
163            while (true) {
164                bool ok = false;
165
166                for (auto pi : G[x]) {
167                    int y = pi.first;
168
169                    if (!vis[y] && y != p[x] && sz[y] * 2 > s)
170                        -> {
171                            x = y;
172                            ok = true;
173
174                        if (ok)
175                            break;
176
177                    }
178
179                }
180            }
181
182        }
183
184        return x;
185    }
186
187    void print() {
188        for (int i = 0; i < maxn; i++)
189            cout << "rt[" << i << "] = ";
190
191        for (int i = 0; i < maxn; i++)
192            cout << "tr[" << i << "] = ";
193
194        for (int i = 0; i < maxn; i++)
195            cout << "d[" << i << "] = ";
196
197        for (int i = 0; i < maxn; i++)
198            cout << "rid[" << i << "] = ";
199
200        for (int i = 0; i < maxn; i++)
201            cout << "sz[" << i << "] = ";
202
203        for (int i = 0; i < maxn; i++)
204            cout << "siz[" << i << "] = ";
205
206        for (int i = 0; i < maxn; i++)
207            cout << "q[" << i << "] = ";
208
209        for (int i = 0; i < maxn; i++)
210            cout << "vis[" << i << "] = ";
211
212        cout << endl;
213
214        for (int i = 0; i < maxn; i++)
215            cout << "p[" << i << "] = ";
216
217        for (int i = 0; i < maxn; i++)
218            cout << "w[" << i << "] = ";
219
220        cout << endl;
221
222        for (int i = 0; i < maxn; i++)
223            cout << "G[" << i << "] = ";
224
225        cout << endl;
226
227        for (int i = 0; i < maxn; i++)
228            cout << "depth[" << i << "] = ";
229
230        cout << endl;
231
232        for (int i = 0; i < maxn; i++)
233            cout << "d[" << i << "][0] = ";
234
235        cout << endl;
236
237        for (int i = 0; i < maxn; i++)
238            cout << "d[" << i << "][1] = ";
239
240        cout << endl;
241
242        for (int i = 0; i < maxn; i++)
243            cout << "d[" << i << "][2] = ";
244
245        cout << endl;
246
247        for (int i = 0; i < maxn; i++)
248            cout << "d[" << i << "][3] = ";
249
250        cout << endl;
251
252        for (int i = 0; i < maxn; i++)
253            cout << "d[" << i << "][4] = ";
254
255        cout << endl;
256
257        for (int i = 0; i < maxn; i++)
258            cout << "d[" << i << "][5] = ";
259
260        cout << endl;
261
262        for (int i = 0; i < maxn; i++)
263            cout << "d[" << i << "][6] = ";
264
265        cout << endl;
266
267        for (int i = 0; i < maxn; i++)
268            cout << "d[" << i << "][7] = ";
269
270        cout << endl;
271
272        for (int i = 0; i < maxn; i++)
273            cout << "d[" << i << "][8] = ";
274
275        cout << endl;
276
277        for (int i = 0; i < maxn; i++)
278            cout << "d[" << i << "][9] = ";
279
280        cout << endl;
281
282        for (int i = 0; i < maxn; i++)
283            cout << "d[" << i << "][10] = ";
284
285        cout << endl;
286
287        for (int i = 0; i < maxn; i++)
288            cout << "d[" << i << "][11] = ";
289
290        cout << endl;
291
292        for (int i = 0; i < maxn; i++)
293            cout << "d[" << i << "][12] = ";
294
295        cout << endl;
296
297        for (int i = 0; i < maxn; i++)
298            cout << "d[" << i << "][13] = ";
299
300        cout << endl;
301
302        for (int i = 0; i < maxn; i++)
303            cout << "d[" << i << "][14] = ";
304
305        cout << endl;
306
307        for (int i = 0; i < maxn; i++)
308            cout << "d[" << i << "][15] = ";
309
310        cout << endl;
311
312        for (int i = 0; i < maxn; i++)
313            cout << "d[" << i << "][16] = ";
314
315        cout << endl;
316
317        for (int i = 0; i < maxn; i++)
318            cout << "d[" << i << "][17] = ";
319
320        cout << endl;
321
322        for (int i = 0; i < maxn; i++)
323            cout << "d[" << i << "][18] = ";
324
325        cout << endl;
326
327        for (int i = 0; i < maxn; i++)
328            cout << "d[" << i << "][19] = ";
329
330        cout << endl;
331
332        for (int i = 0; i < maxn; i++)
333            cout << "d[" << i << "][20] = ";
334
335        cout << endl;
336
337        for (int i = 0; i < maxn; i++)
338            cout << "d[" << i << "][21] = ";
339
340        cout << endl;
341
342        for (int i = 0; i < maxn; i++)
343            cout << "d[" << i << "][22] = ";
344
345        cout << endl;
346
347        for (int i = 0; i < maxn; i++)
348            cout << "d[" << i << "][23] = ";
349
350        cout << endl;
351
352        for (int i = 0; i < maxn; i++)
353            cout << "d[" << i << "][24] = ";
354
355        cout << endl;
356
357        for (int i = 0; i < maxn; i++)
358            cout << "d[" << i << "][25] = ";
359
360        cout << endl;
361
362        for (int i = 0; i < maxn; i++)
363            cout << "d[" << i << "][26] = ";
364
365        cout << endl;
366
367        for (int i = 0; i < maxn; i++)
368            cout << "d[" << i << "][27] = ";
369
370        cout << endl;
371
372        for (int i = 0; i < maxn; i++)
373            cout << "d[" << i << "][28] = ";
374
375        cout << endl;
376
377        for (int i = 0; i < maxn; i++)
378            cout << "d[" << i << "][29] = ";
379
380        cout << endl;
381
382        for (int i = 0; i < maxn; i++)
383            cout << "d[" << i << "][30] = ";
384
385        cout << endl;
386
387        for (int i = 0; i < maxn; i++)
388            cout << "d[" << i << "][31] = ";
389
390        cout << endl;
391
392        for (int i = 0; i < maxn; i++)
393            cout << "d[" << i << "][32] = ";
394
395        cout << endl;
396
397        for (int i = 0; i < maxn; i++)
398            cout << "d[" << i << "][33] = ";
399
400        cout << endl;
401
402        for (int i = 0; i < maxn; i++)
403            cout << "d[" << i << "][34] = ";
404
405        cout << endl;
406
407        for (int i = 0; i < maxn; i++)
408            cout << "d[" << i << "][35] = ";
409
410        cout << endl;
411
412        for (int i = 0; i < maxn; i++)
413            cout << "d[" << i << "][36] = ";
414
415        cout << endl;
416
417        for (int i = 0; i < maxn; i++)
418            cout << "d[" << i << "][37] = ";
419
420        cout << endl;
421
422        for (int i = 0; i < maxn; i++)
423            cout << "d[" << i << "][38] = ";
424
425        cout << endl;
426
427        for (int i = 0; i < maxn; i++)
428            cout << "d[" << i << "][39] = ";
429
430        cout << endl;
431
432        for (int i = 0; i < maxn; i++)
433            cout << "d[" << i << "][40] = ";
434
435        cout << endl;
436
437        for (int i = 0; i < maxn; i++)
438            cout << "d[" << i << "][41] = ";
439
440        cout << endl;
441
442        for (int i = 0; i < maxn; i++)
443            cout << "d[" << i << "][42] = ";
444
445        cout << endl;
446
447        for (int i = 0; i < maxn; i++)
448            cout << "d[" << i << "][43] = ";
449
450        cout << endl;
451
452        for (int i = 0; i < maxn; i++)
453            cout << "d[" << i << "][44] = ";
454
455        cout << endl;
456
457        for (int i = 0; i < maxn; i++)
458            cout << "d[" << i << "][45] = ";
459
460        cout << endl;
461
462        for (int i = 0; i < maxn; i++)
463            cout << "d[" << i << "][46] = ";
464
465        cout << endl;
466
467        for (int i = 0; i < maxn; i++)
468            cout << "d[" << i << "][47] = ";
469
470        cout << endl;
471
472        for (int i = 0; i < maxn; i++)
473            cout << "d[" << i << "][48] = ";
474
475        cout << endl;
476
477        for (int i = 0; i < maxn; i++)
478            cout << "d[" << i << "][49] = ";
479
480        cout << endl;
481
482        for (int i = 0; i < maxn; i++)
483            cout << "d[" << i << "][50] = ";
484
485        cout << endl;
486
487        for (int i = 0; i < maxn; i++)
488            cout << "d[" << i << "][51] = ";
489
490        cout << endl;
491
492        for (int i = 0; i < maxn; i++)
493            cout << "d[" << i << "][52] = ";
494
495        cout << endl;
496
497        for (int i = 0; i < maxn; i++)
498            cout << "d[" << i << "][53] = ";
499
500        cout << endl;
501
502        for (int i = 0; i < maxn; i++)
503            cout << "d[" << i << "][54] = ";
504
505        cout << endl;
506
507        for (int i = 0; i < maxn; i++)
508            cout << "d[" << i << "][55] = ";
509
510        cout << endl;
511
512        for (int i = 0; i < maxn; i++)
513            cout << "d[" << i << "][56] = ";
514
515        cout << endl;
516
517        for (int i = 0; i < maxn; i++)
518            cout << "d[" << i << "][57] = ";
519
520        cout << endl;
521
522        for (int i = 0; i < maxn; i++)
523            cout << "d[" << i << "][58] = ";
524
525        cout << endl;
526
527        for (int i = 0; i < maxn; i++)
528            cout << "d[" << i << "][59] = ";
529
530        cout << endl;
531
532        for (int i = 0; i < maxn; i++)
533            cout << "d[" << i << "][60] = ";
534
535        cout << endl;
536
537        for (int i = 0; i < maxn; i++)
538            cout << "d[" << i << "][61] = ";
539
540        cout << endl;
541
542        for (int i = 0; i < maxn; i++)
543            cout << "d[" << i << "][62] = ";
544
545        cout << endl;
546
547        for (int i = 0; i < maxn; i++)
548            cout << "d[" << i << "][63] = ";
549
550        cout << endl;
551
552        for (int i = 0; i < maxn; i++)
553            cout << "d[" << i << "][64] = ";
554
555        cout << endl;
556
557        for (int i = 0; i < maxn; i++)
558            cout << "d[" << i << "][65] = ";
559
560        cout << endl;
561
562        for (int i = 0; i < maxn; i++)
563            cout << "d[" << i << "][66] = ";
564
565        cout << endl;
566
567        for (int i = 0; i < maxn; i++)
568            cout << "d[" << i << "][67] = ";
569
570        cout << endl;
571
572        for (int i = 0; i < maxn; i++)
573            cout << "d[" << i << "][68] = ";
574
575        cout << endl;
576
577        for (int i = 0; i < maxn; i++)
578            cout << "d[" << i << "][69] = ";
579
580        cout << endl;
581
582        for (int i = 0; i < maxn; i++)
583            cout << "d[" << i << "][70] = ";
584
585        cout << endl;
586
587        for (int i = 0; i < maxn; i++)
588            cout << "d[" << i << "][71] = ";
589
590        cout << endl;
591
592        for (int i = 0; i < maxn; i++)
593            cout << "d[" << i << "][72] = ";
594
595        cout << endl;
596
597        for (int i = 0; i < maxn; i++)
598            cout << "d[" << i << "][73] = ";
599
600        cout << endl;
601
602        for (int i = 0; i < maxn; i++)
603            cout << "d[" << i << "][74] = ";
604
605        cout << endl;
606
607        for (int i = 0; i < maxn; i++)
608            cout << "d[" << i << "][75] = ";
609
610        cout << endl;
611
612        for (int i = 0; i < maxn; i++)
613            cout << "d[" << i << "][76] = ";
614
615        cout << endl;
616
617        for (int i = 0; i < maxn; i++)
618            cout << "d[" << i << "][77] = ";
619
620        cout << endl;
621
622        for (int i = 0; i < maxn; i++)
623            cout << "d[" << i << "][78] = ";
624
625        cout << endl;
626
627        for (int i = 0; i < maxn; i++)
628            cout << "d[" << i << "][79] = ";
629
630        cout << endl;
631
632        for (int i = 0; i < maxn; i++)
633            cout << "d[" << i << "][80] = ";
634
635        cout << endl;
636
637        for (int i = 0; i < maxn; i++)
638            cout << "d[" << i << "][81] = ";
639
640        cout << endl;
641
642        for (int i = 0; i < maxn; i++)
643            cout << "d[" << i << "][82] = ";
644
645        cout << endl;
646
647        for (int i = 0; i < maxn; i++)
648            cout << "d[" << i << "][83] = ";
649
650        cout << endl;
651
652        for (int i = 0; i < maxn; i++)
653            cout << "d[" << i << "][84] = ";
654
655        cout << endl;
656
657        for (int i = 0; i < maxn; i++)
658            cout << "d[" << i << "][85] = ";
659
660        cout << endl;
661
662        for (int i = 0; i < maxn; i++)
663            cout << "d[" << i << "][86] = ";
664
665        cout << endl;
666
667        for (int i = 0; i < maxn; i++)
668            cout << "d[" << i << "][87] = ";
669
670        cout << endl;
671
672        for (int i = 0; i < maxn; i++)
673            cout << "d[" << i << "][88] = ";
674
675        cout << endl;
676
677        for (int i = 0; i < maxn; i++)
678            cout << "d[" << i << "][89] = ";
679
680        cout << endl;
681
682        for (int i = 0; i < maxn; i++)
683            cout << "d[" << i << "][90] = ";
684
685        cout << endl;
686
687        for (int i = 0; i < maxn; i++)
688            cout << "d[" << i << "][91] = ";
689
690        cout << endl;
691
692        for (int i = 0; i < maxn; i++)
693            cout << "d[" << i << "][92] = ";
694
695        cout << endl;
696
697        for (int i = 0; i < maxn; i++)
698            cout << "d[" << i << "][93] = ";
699
700        cout << endl;
701
702        for (int i = 0; i < maxn; i++)
703            cout << "d[" << i << "][94] = ";
704
705        cout << endl;
706
707        for (int i = 0; i < maxn; i++)
708            cout << "d[" << i << "][95] = ";
709
710        cout << endl;
711
712        for (int i = 0; i < maxn; i++)
713            cout << "d[" << i << "][96] = ";
714
715        cout << endl;
716
717        for (int i = 0; i < maxn; i++)
718            cout << "d[" << i << "][97] = ";
719
720        cout << endl;
721
722        for (int i = 0; i < maxn; i++)
723            cout << "d[" << i << "][98] = ";
724
725        cout << endl;
726
727        for (int i = 0; i < maxn; i++)
728            cout << "d[" << i << "][99] = ";
729
730        cout << endl;
731
732        for (int i = 0; i < maxn; i++)
733            cout << "d[" << i << "][100] = ";
734
735        cout << endl;
736
737        for (int i = 0; i < maxn; i++)
738            cout << "d[" << i << "][101] = ";
739
740        cout << endl;
741
742        for (int i = 0; i < maxn; i++)
743            cout << "d[" << i << "][102] = ";
744
745        cout << endl;
746
747        for (int i = 0; i < maxn; i++)
748            cout << "d[" << i << "][103] = ";
749
750        cout << endl;
751
752        for (int i = 0; i < maxn; i++)
753            cout << "d[" << i << "][104] = ";
754
755        cout << endl;
756
757        for (int i = 0; i < maxn; i++)
758            cout << "d[" << i << "][105] = ";
759
760        cout << endl;
761
762        for (int i = 0; i < maxn; i++)
763            cout << "d[" << i << "][106] = ";
764
765        cout << endl;
766
767        for (int i = 0; i < maxn; i++)
768            cout << "d[" << i << "][107] = ";
769
770        cout << endl;
771
772        for (int i = 0; i < maxn; i++)
773            cout << "d[" << i << "][108] = ";
774
775        cout << endl;
776
777        for (int i = 0; i < maxn; i++)
778            cout << "d[" << i << "][109] = ";
779
780        cout << endl;
781
782        for (int i = 0; i < maxn; i++)
783            cout << "d[" << i << "][110] = ";
784
785        cout << endl;
786
787        for (int i = 0; i < maxn; i++)
788            cout << "d[" << i << "][111] = ";
789
790        cout << endl;
791
792        for (int i = 0; i < maxn; i++)
793            cout << "d[" << i << "][112] = ";
794
795        cout << endl;
796
797        for (int i = 0; i < maxn; i++)
798            cout << "d[" << i << "][113] = ";
799
800        cout << endl;
801
802        for (int i = 0; i < maxn; i++)
803            cout << "d[" << i << "][114] = ";
804
805        cout << endl;
806
807        for (int i = 0; i < maxn; i++)
808            cout << "d[" << i << "][115] = ";
809
810        cout << endl;
811
812        for (int i = 0; i < maxn; i++)
813            cout << "d[" << i << "][116] = ";
814
815        cout << endl;
816
817        for (int i = 0; i < maxn; i++)
818            cout << "d[" << i << "][117] = ";
819
820        cout << endl;
821
822        for (int i = 0; i < maxn; i++)
823            cout << "d[" << i << "][118] = ";
824
825        cout << endl;
826
827        for (int i = 0; i < maxn; i++)
828            cout << "d[" << i << "][119] = ";
829
830        cout << endl;
831
832        for (int i = 0; i < maxn; i++)
833            cout << "d[" << i << "][120] = ";
834
835        cout << endl;
836
837        for (int i = 0; i < maxn; i++)
838            cout << "d[" << i << "][121] = ";
839
840        cout << endl;
841
842        for (int i = 0; i < maxn; i++)
843            cout << "d[" << i << "][122] = ";
844
845        cout << endl;
846
847        for (int i = 0; i < maxn; i++)
848            cout << "d[" << i << "][123] = ";
849
850        cout << endl;
851
852        for (int i = 0; i < maxn; i++)
853            cout << "d[" << i << "][124] = ";
854
855        cout <&lt
```

```

171     break;
172 }
173
174 if (!ok)
175     break;
176 }
177
178 return x;
179 }
180
181 void getdis(int st, int o, int k) {
182     int head = 0, tail = 0;
183     q[tail++] = st;
184
185     while (head != tail) {
186         int x = q[head++];
187         sz[x] = 1;
188         rid[x][k] = st;
189
190         tr[o].insert(d[x][k] - w[x]);
191         tre[st][k].insert(d[x][k] - w[x]);
192
193         for (auto pi : G[x]) {
194             int y = pi.first, val = pi.second;
195
196             if (!vis[y] && y != p[x]) {
197                 p[y] = x;
198                 d[y][k] = d[x][k] + val;
199                 q[tail++] = y;
200             }
201         }
202     }
203
204     for (int i = tail - 1; i; i--)
205         sz[p[q[i]]] += sz[q[i]];
206
207     siz[st][k] = sz[st];
208 }
209
210 void rebuild(int x, int s, int pr) {
211     x = getcenter(x, s);
212     vis[x] = true;
213     p[x] = pr;
214     depth[x] = depth[pr] + 1;
215     sz[x] = s;
216
217     tr[x].insert(-w[x]);
218
219     for (auto pi : G[x]) {
220         int y = pi.first, val = pi.second;
221
222         if (!vis[y]) {
223             p[y] = x;
224             d[y][depth[x]] = val;
225             getdis(y, x, depth[x]);
226         }
227     }
228
229     for (auto pi : G[x]) {
230         int y = pi.first;
231
232         if (!vis[y])
233             rebuild(y, sz[y], x);
234     }
235 }
236
237 long long add_node(int x, int nw) { // nw是边权
238     depth[x] = depth[p[x]] + 1;
239
240     sz[x] = 1;
241     vis[x] = true;
242
243     tr[x].insert(-w[x]);
244
245     long long tmp = 0;
246     int goat = 0; // 替罪羊
247
248     for (int u = p[x], k = depth[x] - 1; u; u = p[u],
249          k--) {
250         d[x][k] = d[p[x]][k] + nw;
251         rid[x][k] = (rid[p[x]][k] ? rid[p[x]][k] : x);
252
253         tmp += tr[u].order_of_key(w[x] - d[x][k] + 1);
254         tmp -= tre[rid[x][k]][k].order_of_key(w[x] -
255                                         d[x][k] + 1);
256
257         tr[u].insert(d[x][k] - w[x]);
258         tre[rid[x][k]][k].insert(d[x][k] - w[x]);
259
260         sz[u]++;
261         siz[rid[x][k]][k]++;
262
263         if (siz[rid[x][k]][k] > sz[u] * alpha + 5)
264             goat = u;
265     }
266
267     if (goat) {
268         destroy(goat);
269         rebuild(goat, sz[goat], p[goat]);
270     }
271 }
272
273 int main() {
274
275     null->ch[0] = null->ch[1] = null;
276     null->size = 0;
277
278     int n;
279     scanf("%*d%d", &n);
280
281     scanf("%*d%d%d", &w[1]);
282     vis[1] = true;
283     sz[1] = 1;
284     tr[1].insert(-w[1]);
285
286     printf("0\n");
287
288     long long ans = 0;
289
290     for (int i = 2; i <= n; i++) {
291         int nw;
292         scanf("%d%d%d", &p[i], &nw, &w[i]);
293
294         p[i] ^= (ans % 1000000000);
295
296         G[i].push_back(make_pair(p[i], nw));
297         G[p[i]].push_back(make_pair(i, nw));
298
299         ans += add_node(i, nw);
300
301         printf("%lld\n", ans);
302     }
303
304     return 0;
305 }

```

4.7 LCT动态树

4.7.1 不换根 (弹飞绵羊)

```

1 #define isroot(x) ((x) != (x) -> p -> ch[0] && (x) !=  
2     ↪ (x) -> p -> ch[1]) // 判断是不是Splay的根  
3 #define dir(x) ((x) == (x) -> p -> ch[1]) // 判断它是它  
4     ↪ 父亲的左 / 右儿子  
5  
6 struct node { // 结点类定义  
7     int size; // Splay的子树大小  
8     node *ch[2], *p;  
9  
10    node() : size(1) {}  
11    void refresh() {  
12        size = ch[0] -> size + ch[1] -> size + 1;  
13    } // 附加信息维护  
14 } null[maxn];  
15  
16 // 在主函数开头加上这句初始化  
17 null -> size = 0;  
18  
19 // 初始化结点  
20 void initialize(node *x) {  
21     x -> ch[0] = x -> ch[1] = x -> p = null;  
22 }  
23  
24 // Access 均摊O(log n)  
25 // LCT核心操作，把结点到根的路径打通，顺便把与重儿子的连  
26     ↪ 边变成轻边  
27 // 需要调用splay  
28 node *access(node *x) {  
29     node *y = null;  
30  
31     while (x != null) {  
32         splay(x);  
33  
34         x -> ch[1] = y;  
35         (y = x) -> refresh();  
36  
37         x = x -> p;  
38     }  
39  
40     return y;  
41 }  
42  
43 // Link 均摊O(log n)  
44 // 把x的父亲设为y  
45 // 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  
46     ↪ 问题  
47 // 需要调用splay  
48 void link(node *x, node *y) {  
49     splay(x);  
50     x -> p = y;  
51 }  
52  
53 // Cut 均摊O(log n)  
54 // 把x与其父亲的连边断掉  
55 // x可以是所在树的根节点，这时此操作没有任何实质效果  
56 // 需要调用access和splay  
57 void cut(node *x) {  
58     access(x);  
59     splay(x);  
60  
61     x -> ch[0] -> p = null;  
62     x -> ch[0] = null;  
63  
64     x -> refresh();  
65 }  
66  
67 // Splay 均摊O(log n)

```

```

64 // 需要调用旋转
65 void splay(node **x) {
66     while (!isroot(*x)) {
67         if (isroot(*x -> p)) {
68             rot(*x -> p, dir(*x) ^ 1);
69             break;
70         }
71
72         if (dir(*x) == dir(*x -> p))
73             rot(*x -> p -> p, dir(*x -> p) ^ 1);
74         else
75             rot(*x -> p, dir(*x) ^ 1);
76         rot(*x -> p, dir(*x) ^ 1);
77     }
78 }
79
80 // 旋转 (LCT版本) O(1)
81 // 平衡树基本操作
82 // 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
83 void rot(node **x, int d) {
84     node *y = *x -> ch[d ^ 1];
85
86     y -> p = *x -> p;
87     if (!isroot(*x))
88         *x -> p -> ch[dir(*x)] = y;
89
90     if ((*x -> ch[d ^ 1] = y -> ch[d]) != null)
91         y -> ch[d] -> p = *x;
92     (*y -> ch[d] = *x) -> p = y;
93
94     *x -> refresh();
95     y -> refresh();
96 }

```

4.7.2 换根/维护生成树

```

1 #define isroot(x) ((x) -> p == null || ((x) -> p ->  
2     ↪ ch[0] != (x) && (x) -> p -> ch[1] != (x)))
3 #define dir(x) ((x) == (x) -> p -> ch[1])
4
5 using namespace std;
6
7 const int maxn = 200005;
8
9 struct node{
10     int key, mx, pos;
11     bool rev;
12     node *ch[2], *p;
13
14     node(int key = 0): key(key), mx(key), pos(-1),
15           ↪ rev(false) {}
16
17     void pushdown() {
18         if (!rev)
19             return;
20
21         ch[0] -> rev ^= true;
22         ch[1] -> rev ^= true;
23         swap(ch[0], ch[1]);
24
25         if (pos != -1)
26             pos ^= 1;
27
28         rev = false;
29     }
30
31     void refresh() {
32         mx = key;
33         pos = -1;
34     }
35
36     void rotate(int d) {
37         if (d == 0)
38             rot(ch[1], 1);
39         else
40             rot(ch[0], 0);
41
42         if (pos != -1)
43             pos ^= 1;
44
45         rev = !rev;
46     }
47
48     void update() {
49         if (pos != -1)
50             pos = -1;
51
52         if (rev)
53             rev = false;
54
55         if (key != mx)
56             mx = key;
57
58         if (pos != -1)
59             pos = -1;
60
61         if (rev)
62             rev = false;
63
64         if (key != mx)
65             mx = key;
66
67         if (pos != -1)
68             pos = -1;
69
70         if (rev)
71             rev = false;
72
73         if (key != mx)
74             mx = key;
75
76         if (pos != -1)
77             pos = -1;
78
79         if (rev)
80             rev = false;
81
82         if (key != mx)
83             mx = key;
84
85         if (pos != -1)
86             pos = -1;
87
88         if (rev)
89             rev = false;
90
91         if (key != mx)
92             mx = key;
93
94         if (pos != -1)
95             pos = -1;
96
97         if (rev)
98             rev = false;
99
100        if (key != mx)
101            mx = key;
102
103        if (pos != -1)
104            pos = -1;
105
106        if (rev)
107            rev = false;
108
109        if (key != mx)
110            mx = key;
111
112        if (pos != -1)
113            pos = -1;
114
115        if (rev)
116            rev = false;
117
118        if (key != mx)
119            mx = key;
120
121        if (pos != -1)
122            pos = -1;
123
124        if (rev)
125            rev = false;
126
127        if (key != mx)
128            mx = key;
129
130        if (pos != -1)
131            pos = -1;
132
133        if (rev)
134            rev = false;
135
136        if (key != mx)
137            mx = key;
138
139        if (pos != -1)
140            pos = -1;
141
142        if (rev)
143            rev = false;
144
145        if (key != mx)
146            mx = key;
147
148        if (pos != -1)
149            pos = -1;
150
151        if (rev)
152            rev = false;
153
154        if (key != mx)
155            mx = key;
156
157        if (pos != -1)
158            pos = -1;
159
160        if (rev)
161            rev = false;
162
163        if (key != mx)
164            mx = key;
165
166        if (pos != -1)
167            pos = -1;
168
169        if (rev)
170            rev = false;
171
172        if (key != mx)
173            mx = key;
174
175        if (pos != -1)
176            pos = -1;
177
178        if (rev)
179            rev = false;
180
181        if (key != mx)
182            mx = key;
183
184        if (pos != -1)
185            pos = -1;
186
187        if (rev)
188            rev = false;
189
190        if (key != mx)
191            mx = key;
192
193        if (pos != -1)
194            pos = -1;
195
196        if (rev)
197            rev = false;
198
199        if (key != mx)
200            mx = key;
201
202        if (pos != -1)
203            pos = -1;
204
205        if (rev)
206            rev = false;
207
208        if (key != mx)
209            mx = key;
210
211        if (pos != -1)
212            pos = -1;
213
214        if (rev)
215            rev = false;
216
217        if (key != mx)
218            mx = key;
219
220        if (pos != -1)
221            pos = -1;
222
223        if (rev)
224            rev = false;
225
226        if (key != mx)
227            mx = key;
228
229        if (pos != -1)
230            pos = -1;
231
232        if (rev)
233            rev = false;
234
235        if (key != mx)
236            mx = key;
237
238        if (pos != -1)
239            pos = -1;
240
241        if (rev)
242            rev = false;
243
244        if (key != mx)
245            mx = key;
246
247        if (pos != -1)
248            pos = -1;
249
250        if (rev)
251            rev = false;
252
253        if (key != mx)
254            mx = key;
255
256        if (pos != -1)
257            pos = -1;
258
259        if (rev)
260            rev = false;
261
262        if (key != mx)
263            mx = key;
264
265        if (pos != -1)
266            pos = -1;
267
268        if (rev)
269            rev = false;
270
271        if (key != mx)
272            mx = key;
273
274        if (pos != -1)
275            pos = -1;
276
277        if (rev)
278            rev = false;
279
280        if (key != mx)
281            mx = key;
282
283        if (pos != -1)
284            pos = -1;
285
286        if (rev)
287            rev = false;
288
289        if (key != mx)
290            mx = key;
291
292        if (pos != -1)
293            pos = -1;
294
295        if (rev)
296            rev = false;
297
298        if (key != mx)
299            mx = key;
300
301        if (pos != -1)
302            pos = -1;
303
304        if (rev)
305            rev = false;
306
307        if (key != mx)
308            mx = key;
309
310        if (pos != -1)
311            pos = -1;
312
313        if (rev)
314            rev = false;
315
316        if (key != mx)
317            mx = key;
318
319        if (pos != -1)
320            pos = -1;
321
322        if (rev)
323            rev = false;
324
325        if (key != mx)
326            mx = key;
327
328        if (pos != -1)
329            pos = -1;
330
331        if (rev)
332            rev = false;
333
334        if (key != mx)
335            mx = key;
336
337        if (pos != -1)
338            pos = -1;
339
340        if (rev)
341            rev = false;
342
343        if (key != mx)
344            mx = key;
345
346        if (pos != -1)
347            pos = -1;
348
349        if (rev)
350            rev = false;
351
352        if (key != mx)
353            mx = key;
354
355        if (pos != -1)
356            pos = -1;
357
358        if (rev)
359            rev = false;
360
361        if (key != mx)
362            mx = key;
363
364        if (pos != -1)
365            pos = -1;
366
367        if (rev)
368            rev = false;
369
370        if (key != mx)
371            mx = key;
372
373        if (pos != -1)
374            pos = -1;
375
376        if (rev)
377            rev = false;
378
379        if (key != mx)
380            mx = key;
381
382        if (pos != -1)
383            pos = -1;
384
385        if (rev)
386            rev = false;
387
388        if (key != mx)
389            mx = key;
390
391        if (pos != -1)
392            pos = -1;
393
394        if (rev)
395            rev = false;
396
397        if (key != mx)
398            mx = key;
399
400        if (pos != -1)
401            pos = -1;
402
403        if (rev)
404            rev = false;
405
406        if (key != mx)
407            mx = key;
408
409        if (pos != -1)
410            pos = -1;
411
412        if (rev)
413            rev = false;
414
415        if (key != mx)
416            mx = key;
417
418        if (pos != -1)
419            pos = -1;
420
421        if (rev)
422            rev = false;
423
424        if (key != mx)
425            mx = key;
426
427        if (pos != -1)
428            pos = -1;
429
430        if (rev)
431            rev = false;
432
433        if (key != mx)
434            mx = key;
435
436        if (pos != -1)
437            pos = -1;
438
439        if (rev)
440            rev = false;
441
442        if (key != mx)
443            mx = key;
444
445        if (pos != -1)
446            pos = -1;
447
448        if (rev)
449            rev = false;
450
451        if (key != mx)
452            mx = key;
453
454        if (pos != -1)
455            pos = -1;
456
457        if (rev)
458            rev = false;
459
460        if (key != mx)
461            mx = key;
462
463        if (pos != -1)
464            pos = -1;
465
466        if (rev)
467            rev = false;
468
469        if (key != mx)
470            mx = key;
471
472        if (pos != -1)
473            pos = -1;
474
475        if (rev)
476            rev = false;
477
478        if (key != mx)
479            mx = key;
480
481        if (pos != -1)
482            pos = -1;
483
484        if (rev)
485            rev = false;
486
487        if (key != mx)
488            mx = key;
489
490        if (pos != -1)
491            pos = -1;
492
493        if (rev)
494            rev = false;
495
496        if (key != mx)
497            mx = key;
498
499        if (pos != -1)
500            pos = -1;
501
502        if (rev)
503            rev = false;
504
505        if (key != mx)
506            mx = key;
507
508        if (pos != -1)
509            pos = -1;
510
511        if (rev)
512            rev = false;
513
514        if (key != mx)
515            mx = key;
516
517        if (pos != -1)
518            pos = -1;
519
520        if (rev)
521            rev = false;
522
523        if (key != mx)
524            mx = key;
525
526        if (pos != -1)
527            pos = -1;
528
529        if (rev)
530            rev = false;
531
532        if (key != mx)
533            mx = key;
534
535        if (pos != -1)
536            pos = -1;
537
538        if (rev)
539            rev = false;
540
541        if (key != mx)
542            mx = key;
543
544        if (pos != -1)
545            pos = -1;
546
547        if (rev)
548            rev = false;
549
550        if (key != mx)
551            mx = key;
552
553        if (pos != -1)
554            pos = -1;
555
556        if (rev)
557            rev = false;
558
559        if (key != mx)
560            mx = key;
561
562        if (pos != -1)
563            pos = -1;
564
565        if (rev)
566            rev = false;
567
568        if (key != mx)
569            mx = key;
570
571        if (pos != -1)
572            pos = -1;
573
574        if (rev)
575            rev = false;
576
577        if (key != mx)
578            mx = key;
579
580        if (pos != -1)
581            pos = -1;
582
583        if (rev)
584            rev = false;
585
586        if (key != mx)
587            mx = key;
588
589        if (pos != -1)
590            pos = -1;
591
592        if (rev)
593            rev = false;
594
595        if (key != mx)
596            mx = key;
597
598        if (pos != -1)
599            pos = -1;
600
601        if (rev)
602            rev = false;
603
604        if (key != mx)
605            mx = key;
606
607        if (pos != -1)
608            pos = -1;
609
610        if (rev)
611            rev = false;
612
613        if (key != mx)
614            mx = key;
615
616        if (pos != -1)
617            pos = -1;
618
619        if (rev)
620            rev = false;
621
622        if (key != mx)
623            mx = key;
624
625        if (pos != -1)
626            pos = -1;
627
628        if (rev)
629            rev = false;
630
631        if (key != mx)
632            mx = key;
633
634        if (pos != -1)
635            pos = -1;
636
637        if (rev)
638            rev = false;
639
640        if (key != mx)
641            mx = key;
642
643        if (pos != -1)
644            pos = -1;
645
646        if (rev)
647            rev = false;
648
649        if (key != mx)
650            mx = key;
651
652        if (pos != -1)
653            pos = -1;
654
655        if (rev)
656            rev = false;
657
658        if (key != mx)
659            mx = key;
660
661        if (pos != -1)
662            pos = -1;
663
664        if (rev)
665            rev = false;
666
667        if (key != mx)
668            mx = key;
669
670        if (pos != -1)
671            pos = -1;
672
673        if (rev)
674            rev = false;
675
676        if (key != mx)
677            mx = key;
678
679        if (pos != -1)
680            pos = -1;
681
682        if (rev)
683            rev = false;
684
685        if (key != mx)
686            mx = key;
687
688        if (pos != -1)
689            pos = -1;
690
691        if (rev)
692            rev = false;
693
694        if (key != mx)
695            mx = key;
696
697        if (pos != -1)
698            pos = -1;
699
700        if (rev)
701            rev = false;
702
703        if (key != mx)
704            mx = key;
705
706        if (pos != -1)
707            pos = -1;
708
709        if (rev)
710            rev = false;
711
712        if (key != mx)
713            mx = key;
714
715        if (pos != -1)
716            pos = -1;
717
718        if (rev)
719            rev = false;
720
721        if (key != mx)
722            mx = key;
723
724        if (pos != -1)
725            pos = -1;
726
727        if (rev)
728            rev = false;
729
730        if (key != mx)
731            mx = key;
732
733        if (pos != -1)
734            pos = -1;
735
736        if (rev)
737            rev = false;
738
739        if (key != mx)
740            mx = key;
741
742        if (pos != -1)
743            pos = -1;
744
745        if (rev)
746            rev = false;
747
748        if (key != mx)
749            mx = key;
750
751        if (pos != -1)
752            pos = -1;
753
754        if (rev)
755            rev = false;
756
757        if (key != mx)
758            mx = key;
759
760        if (pos != -1)
761            pos = -1;
762
763        if (rev)
764            rev = false;
765
766        if (key != mx)
767            mx = key;
768
769        if (pos != -1)
770            pos = -1;
771
772        if (rev)
773            rev = false;
774
775        if (key != mx)
776            mx = key;
777
778        if (pos != -1)
779            pos = -1;
780
781        if (rev)
782            rev = false;
783
784        if (key != mx)
785            mx = key;
786
787        if (pos != -1)
788            pos = -1;
789
790        if (rev)
791            rev = false;
792
793        if (key != mx)
794            mx = key;
795
796        if (pos != -1)
797            pos = -1;
798
799        if (rev)
800            rev = false;
801
802        if (key != mx)
803            mx = key;
804
805        if (pos != -1)
806            pos = -1;
807
808        if (rev)
809            rev = false;
810
811        if (key != mx)
812            mx = key;
813
814        if (pos != -1)
815            pos = -1;
816
817        if (rev)
818            rev = false;
819
820        if (key != mx)
821            mx = key;
822
823        if (pos != -1)
824            pos = -1;
825
826        if (rev)
827            rev = false;
828
829        if (key != mx)
830            mx = key;
831
832        if (pos != -1)
833            pos = -1;
834
835        if (rev)
836            rev = false;
837
838        if (key != mx)
839            mx = key;
840
841        if (pos != -1)
842            pos = -1;
843
844        if (rev)
845            rev = false;
846
847        if (key != mx)
848            mx = key;
849
850        if (pos != -1)
851            pos = -1;
852
853        if (rev)
854            rev = false;
855
856        if (key != mx)
857            mx = key;
858
859        if (pos != -1)
860            pos = -1;
861
862        if (rev)
863            rev = false;
864
865        if (key != mx)
866            mx = key;
867
868        if (pos != -1)
869            pos = -1;
870
871        if (rev)
872            rev = false;
873
874        if (key != mx)
875            mx = key;
876
877        if (pos != -1)
878            pos = -1;
879
880        if (rev)
881            rev = false;
882
883        if (key != mx)
884            mx = key;
885
886        if (pos != -1)
887            pos = -1;
888
889        if (rev)
890            rev = false;
891
892        if (key != mx)
893            mx = key;
894
895        if (pos != -1)
896            pos = -1;
897
898        if (rev)
899            rev = false;
900
901        if (key != mx)
902            mx = key;
903
904        if (pos != -1)
905            pos = -1;
906
907        if (rev)
908            rev = false;
909
910        if (key != mx)
911            mx = key;
912
913        if (pos != -1)
914            pos = -1;
915
916        if (rev)
917            rev = false;
918
919        if (key != mx)
920            mx = key;
921
922        if (pos != -1)
923            pos = -1;
924
925        if (rev)
926            rev = false;
927
928        if (key != mx)
929            mx = key;
930
931        if (pos != -1)
932            pos = -1;
933
934        if (rev)
935            rev = false;
936
937        if (key != mx)
938            mx = key;
939
940        if (pos != -1)
941            pos = -1;
942
943        if (rev)
944            rev = false;
945
946        if (key != mx)
947            mx = key;
948
949        if (pos != -1)
950            pos = -1;
951
952        if (rev)
953            rev = false;
954
955        if (key != mx)
956            mx = key;
957
958        if (pos != -1)
959            pos = -1;
960
961        if (rev)
962            rev = false;
963
964        if (key != mx)
965            mx = key;
966
967        if (pos != -1)
968            pos = -1;
969
970        if (rev)
971            rev = false;
972
973        if (key != mx)
974            mx = key;
975
976        if (pos != -1)
977            pos = -1;
978
979        if (rev)
980            rev = false;
981
982        if (key != mx)
983            mx = key;
984
985        if (pos != -1)
986            pos = -1;
987
988        if (rev)
989            rev = false;
990
991        if (key != mx)
992            mx = key;
993
994        if (pos != -1)
995            pos = -1;
996
997        if (rev)
998            rev = false;
999
1000        if (key != mx)
1001            mx = key;
1002
1003        if (pos != -1)
1004            pos = -1;
1005
1006        if (rev)
1007            rev = false;
1008
1009        if (key != mx)
1010            mx = key;
1011
1012        if (pos != -1)
1013            pos = -1;
1014
1015        if (rev)
1016            rev = false;
1017
1018        if (key != mx)
1019            mx = key;
1020
1021        if (pos != -1)
1022            pos = -1;
1023
1024        if (rev)
1025            rev = false;
1026
1027        if (key != mx)
1028            mx = key;
1029
1030        if (pos != -1)
1031            pos = -1;
1032
1033        if (rev)
1034            rev = false;
1035
1036        if (key != mx)
1037            mx = key;
1038
1039        if (pos != -1)
1040            pos = -1;
1041
1042        if (rev)
1043            rev = false;
1044
1045        if (key != mx)
1046            mx = key;
1047
1048        if (pos != -1)
1049            pos = -1;
1050
1051        if (rev)
1052            rev = false;
1053
1054        if (key != mx)
1055            mx = key;
1056
1057        if (pos != -1)
1058            pos = -1;
1059
1060        if (rev)
1061            rev = false;
1062
1063        if (key != mx)
1064            mx = key;
1065
1066        if (pos != -1)
1067            pos = -1;
1068
1069        if (rev)
1070            rev = false;
1071
1072        if (key != mx)
1073            mx = key;
1074
1075        if (pos != -1)
1076            pos = -1;
1077
1078        if (rev)
1079            rev = false;
1080
1081        if (key != mx)
1082            mx = key;
1083
1084        if (pos != -1)
1085            pos = -1;
1086
1087        if (rev)
1088            rev = false;
1089
1090        if (key != mx)
1091            mx = key;
1092
1093        if (pos != -1)
1094            pos = -1;
1095
1096        if (rev)
1097            rev = false;
1098
1099        if (key != mx)
1100            mx = key;
1101
1102        if (pos != -1)
1103            pos = -1;
1104
1105        if (rev)
1106            rev = false;
1107
1108        if (key != mx)
1109            mx = key;
1110
1111        if (pos != -1)
1112            pos = -1;
1113
1114        if (rev)
1115            rev = false;
1116
1117        if (key != mx)
1118            mx = key;
1119
1120        if (pos != -1)
1121            pos = -1;
1122
1123        if (rev)
1124            rev = false;
1125
1126        if (key != mx)
1127            mx = key;
1128
1129        if (pos != -1)
1130            pos = -1;
1131
1132        if (rev)
1133            rev = false;
1134
1135        if (key != mx)
1136            mx = key;
1137
1138        if (pos != -1)
1139            pos = -1;
1140
1141        if (rev)
1142            rev = false;
1143
1144        if (key != mx)
1145            mx = key;
1146
1147        if (pos != -1)
1148            pos = -1;
1149
1150        if (rev)
1151            rev = false;
1152
1153        if (key != mx)
1154            mx = key;
1155
1156        if (pos != -1)
1157            pos = -1;
1158
1159        if (rev)
1160            rev = false;
1161
1162        if (key != mx)
1163            mx = key;
1164
1165        if (pos != -1)
1166            pos = -1;
1167
1168        if (rev)
1169            rev = false;
1170
1171        if (key != mx)
1172            mx = key;
1173
1174        if (pos != -1)
1175            pos = -1;
1176
1177        if (rev)
1178            rev = false;
1179
1180        if (key != mx)
1181            mx = key;
1182
1183        if (pos != -1)
1184            pos = -1;
1185
1186        if (rev)
1187            rev = false;
1188
1189        if (key != mx)
1190            mx = key;
1191
1192        if (pos != -1)
1193            pos = -1;
1194
1195        if (rev)
1196            rev = false;
1197
1198        if (key != mx)
1199            mx = key;
1200
1201        if (pos != -1)
1202            pos = -1;
1203
1204        if (rev)
1205            rev = false;
1206
1207        if (key != mx)
1208            mx = key;
1209
1210        if (pos != -1)
1211            pos = -1;
1212
1213        if (rev)
1214            rev = false;
1215
1216        if (key != mx)
1217            mx = key;
1218
1219        if (pos != -1)
1220            pos = -1;
1221
1222        if (rev)
1223            rev = false;
1224
1225        if (key != mx)
1226            mx = key;
1227
1228        if (pos != -1)
1229            pos = -1;
1230
1231        if (rev)
1232            rev = false;
1233
1234        if (key != mx)
1235            mx = key;
1236
1237        if (pos != -1)
1238            pos = -1;
1239
1240        if (rev)
1241            rev = false;
1242
1243        if (key != mx)
1244            mx = key;
1245
1246        if (pos != -1)
1247            pos = -1;
1248
1249        if (rev)
1250            rev = false;
1251
1252        if (key != mx)
1253            mx = key;
1254
1255        if (pos != -1)
1256            pos = -1;
1257
1258        if (rev)
1259            rev = false;
1260
1261        if (key != mx)
1262            mx = key;
1263
1264        if (pos != -1)
1265            pos = -1;
1266
1267        if (rev)
1268            rev = false;
1269
1270        if (key != mx)
1271           
```

```

32     if (ch[0] -> mx > mx) {
33         mx = ch[0] -> mx;
34         pos = 0;
35     }
36     if (ch[1] -> mx > mx) {
37         mx = ch[1] -> mx;
38         pos = 1;
39     }
40 }
41 } null[maxn * 2];
42
43 void init(node *x, int k) {
44     x -> ch[0] = x -> ch[1] = x -> p = null;
45     x -> key = x -> mx = k;
46 }
47
48 void rot(node *x, int d) {
49     node *y = x -> ch[d ^ 1];
50     if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
51         y -> ch[d] -> p = x;
52
53     y -> p = x -> p;
54     if (!isroot(x))
55         x -> p -> ch[dir(x)] = y;
56
57     (y -> ch[d] = x) -> p = y;
58
59     x -> refresh();
60     y -> refresh();
61 }
62
63 void splay(node *x) {
64     x -> pushdown();
65
66     while (!isroot(x)) {
67         if (!isroot(x -> p))
68             x -> p -> p -> pushdown();
69         x -> p -> pushdown();
70         x -> pushdown();
71
72         if (isroot(x -> p)) {
73             rot(x -> p, dir(x) ^ 1);
74             break;
75         }
76
77         if (dir(x) == dir(x -> p))
78             rot(x -> p -> p, dir(x -> p) ^ 1);
79         else
80             rot(x -> p, dir(x) ^ 1);
81
82         rot(x -> p, dir(x) ^ 1);
83     }
84 }
85
86 node *access(node *x) {
87     node *y = null;
88
89     while (x != null) {
90         splay(x);
91
92         x -> ch[1] = y;
93         (y = x) -> refresh();
94
95         x = x -> p;
96     }
97
98     return y;
99 }
100
101 void makeroott(node **x) {
102     access(x);
103     splay(x);
104     x -> rev ^= true;
105 }
106
107 void link(node *x, node *y) {
108     makeroott(x);
109     x -> p = y;
110 }
111
112 void cut(node *x, node *y) {
113     makeroott(x);
114     access(y);
115     splay(y);
116
117     y -> ch[0] -> p = null;
118     y -> ch[0] = null;
119     y -> refresh();
120 }
121
122 node *getroot(node *x) {
123     x = access(x);
124     while (x -> pushdown(), x -> ch[0] != null)
125         x = x -> ch[0];
126     splay(x);
127     return x;
128 }
129
130 node *getmax(node *x, node *y) {
131     makeroott(x);
132     x = access(y);
133
134     while (x -> pushdown(), x -> pos != -1)
135         x = x -> ch[x -> pos];
136     splay(x);
137
138     return x;
139 }
140
141 // 以下为主函数示例
142 for (int i = 1; i <= m; i++) {
143     init(null + n + i, w[i]);
144     if (getroot(null + u[i]) != getroot(null + v[i])) {
145         ans[q + 1] -= k;
146         ans[q + 1] += w[i];
147
148         link(null + u[i], null + n + i);
149         link(null + v[i], null + n + i);
150         vis[i] = true;
151     }
152     else {
153         int ii = getmax(null + u[i], null + v[i]) -
154             >> null - n;
155         if (w[i] >= w[ii])
156             continue;
157
158         cut(null + u[ii], null + n + ii);
159         cut(null + v[ii], null + n + ii);
160
161         link(null + u[i], null + n + i);
162         link(null + v[i], null + n + i);
163
164         ans[q + 1] -= w[ii];
165         ans[q + 1] += w[i];
166     }
167 }

```

4.7.3 维护子树信息

```

1 // 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，  

2 // → 没必要的时候就不要用  

3 // 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  

4 // → 会变成 $O(n \log^2 n)$   

5 // 注意由于这道题与边权有关，需要边权拆点变点权  

6  

7 // 宏定义  

8 #define isroot(x) ((x) -> p == null || ((x) != (x) -> p  

9 // → -> ch[0] && (x) != (x) -> p -> ch[1]))  

10 #define dir(x) ((x) == (x) -> p -> ch[1])  

11  

12 // 节点类定义  

13 struct node { // 以维护子树中黑点到根距离和为例  

14     int w, chain_cnt, tree_cnt;  

15     long long sum, suml, sumr, tree_sum; // 由于换根需要  

16     // → 子树反转，需要维护两个方向的信息  

17     bool rev, col;  

18     node *ch[2], *p;  

19  

20     node() : w(0), chain_cnt(0),  

21         // → tree_cnt(0), sum(0), suml(0), sumr(0),  

22         // → tree_sum(0), rev(false), col(false) {}  

23  

24     inline void pushdown() {  

25         if(!rev)  

26             return;  

27  

28         ch[0]->rev ^= true;  

29         ch[1]->rev ^= true;  

30         swap(ch[0], ch[1]);  

31         swap(suml, sumr);  

32  

33         rev = false;  

34     }  

35  

36     inline void refresh() { // 如果不想这样特判  

37         // → 就pushdown一下  

38         // pushdown();  

39  

40         sum = ch[0] -> sum + ch[1] -> sum + w;  

41         suml = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->  

42             // → suml) + (ch[1] -> rev ? ch[1] -> sumr :  

43             // → ch[1] -> suml) + (tree_cnt + ch[1] ->  

44             // → chain_cnt) * (ch[0] -> sum + w) + tree_sum;  

45         sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->  

46             // → sumr) + (ch[1] -> rev ? ch[1] -> suml :  

47             // → ch[1] -> sumr) + (tree_cnt + ch[0] ->  

48             // → chain_cnt) * (ch[1] -> sum + w) + tree_sum;  

49         chain_cnt = ch[0] -> chain_cnt + ch[1] ->  

50             // → chain_cnt + tree_cnt;  

51     }  

52     null[maxn * 2]; // 如果没有边权变点权就不用乘2了  

53  

54 // 封装构造函数  

55 node *newnode(int w) {  

56     node *x = nodes.front(); // 因为有删边加边，可以用一  

57     // → 个队列维护可用结点  

58     nodes.pop();  

59     initialize(x);  

60     x -> w = w;  

61     x -> refresh();  

62     return x;  

63 }  

64  

65 // 封装初始化函数  

66 // 记得在进行操作之前对所有结点调用一遍  

67 inline void initialize(node *x) {  

68     *x = node();  

69     x -> ch[0] = x -> ch[1] = x -> p = null;  

70  

71     }  

72  

73 // 注意一下在Access的同时更新子树信息的方法  

74 node *access(node *x) {  

75     node *y = null;  

76  

77     while (x != null) {  

78         splay(x);  

79  

80         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->  

81             // → chain_cnt;  

82         x -> tree_sum += (x -> ch[1] -> rev ? x ->  

83             // → ch[1] -> sumr : x -> ch[1] -> suml) - y ->  

84                 // → suml;  

85         x -> ch[1] = y;  

86  

87         (y = x) -> refresh();  

88         x = x -> p;  

89     }  

90  

91     return y;  

92 }  

93  

94 // 找到一个点所在连通块的根  

95 // 对比原版没有变化  

96 node *getroot(node *x) {  

97     x = access(x);  

98  

99     while (x -> pushdown(), x -> ch[0] != null)  

100         x = x -> ch[0];  

101     splay(x);  

102  

103     return x;  

104 }  

105  

106 // 换根，同样没有变化  

107 void makeroott(node *x) {  

108     access(x);  

109     splay(x);  

110     x -> rev ^= true;  

111     x -> pushdown();  

112 }  

113  

114 // 连接两个点  

115 // !!! 注意这里必须把两者都变成根，因为只能修改根结点  

116 void link(node *x, node *y) {  

117     makeroott(x);  

118     makeroott(y);  

119  

120     x -> p = y;  

121     y -> tree_cnt += x -> chain_cnt;  

122     y -> tree_sum += x -> suml;  

123     y -> refresh();  

124 }  

125  

126 // 删除一条边  

127 // 对比原版没有变化  

128 void cut(node *x, node *y) {  

129     makeroott(x);  

130     access(y);  

131     splay(y);  

132  

133     y -> ch[0] -> p = null;  

134     y -> ch[0] = null;  

135     y -> refresh();  

136 }  

137  

138 // 修改/询问一个点，这里以询问为例  

139 // 如果是修改就在换根之后搞一些操作

```

```

122 long long query(node *x) {
123     makeroot(x);
124     return x->suml;
125 }
126
127 // Splay函数
128 // 对比原版没有变化
129 void splay(node *x) {
130     x->pushdown();
131
132     while (!isroot(x)) {
133         if (!isroot(x->p))
134             x->p->p->pushdown();
135         x->p->pushdown();
136         x->pushdown();
137
138         if (isroot(x->p)) {
139             rot(x->p, dir(x) ^ 1);
140             break;
141         }
142
143         if (dir(x) == dir(x->p))
144             rot(x->p->p, dir(x->p) ^ 1);
145         else
146             rot(x->p, dir(x) ^ 1);
147
148         rot(x->p, dir(x) ^ 1);
149     }
150 }
151
152 // 旋转函数
153 // 对比原版没有变化
154 void rot(node *x, int d) {
155     node *y = x->ch[d ^ 1];
156
157     if ((x->ch[d ^ 1] = y->ch[d]) != null)
158         y->ch[d]->p = x;
159
160     y->p = x->p;
161     if (!isroot(x))
162         x->p->ch[dir(x)] = y;
163
164     (y->ch[d] = x)->p = y;
165
166     x->refresh();
167     y->refresh();
168 }

```

```

19     void push(long long x) {
20         if (x > (-INF) >> 2)
21             q1.push(x);
22     }
23
24     void erase(long long x) {
25         if (x > (-INF) >> 2)
26             q2.push(x);
27     }
28
29     long long top() {
30         if (empty())
31             return -INF;
32
33         while (!q2.empty() && q1.top() == q2.top()) {
34             q1.pop();
35             q2.pop();
36         }
37
38         return q1.top();
39     }
40
41     long long top2() {
42         if (size() < 2)
43             return -INF;
44
45         long long a = top();
46         erase(a);
47         long long b = top();
48         push(a);
49         return a + b;
50     }
51
52     int size() {
53         return q1.size() - q2.size();
54     }
55
56     bool empty() {
57         return q1.size() == q2.size();
58     }
59 } heap; // 全局堆维护每条链的最大子段和
60
61 struct node {
62     long long sum, maxsum, prefix, suffix;
63     int key;
64     binary_heap heap; // 每个点的堆存的是它的子树中到它的
65     // 最远距离, 如果它是黑点的话还会包括自己
66     node *ch[2], *p;
67     bool rev;
68     node(int k = 0): sum(k), maxsum(-INF),
69     // prefix(-INF),
70     // suffix(-INF), key(k), rev(false) {}
71
72     inline void pushdown() {
73         if (!rev)
74             return;
75
76         ch[0]->rev ^= true;
77         ch[1]->rev ^= true;
78         swap(ch[0], ch[1]);
79         swap(prefix, suffix);
80         rev = false;
81     }
82
83     inline void refresh() {
84         pushdown();
85         ch[0]->pushdown();
86         ch[1]->pushdown();
87         sum = ch[0]->sum + ch[1]->sum + key;
88         prefix = max(ch[0]->prefix,
89                      ch[0]->sum + key + ch[1]->
90                      // prefix);
91         suffix = max(ch[1]->suffix,
92

```

4.7.4 模板题: 动态 QTREE4 (询问树上相距最远点)

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4 #include <ext/pb_ds/priority_queue.hpp>
5
6 #define isroot(x) ((x)->p == null || ((x) != (x)->p
6     ->ch[0] && (x) != (x)->p->ch[1]))
7 #define dir(x) ((x) == (x)->p->ch[1])
8
9 using namespace std;
10 using namespace __gnu_pbds;
11
12 constexpr int maxn = 100005;
13 constexpr long long INF = 1000000000000000000ll;
14
15 struct binary_heap {
16     __gnu_pbds::priority_queue<long long, less<long
17     ->long>, binary_heap_tag> q1, q2;
18     binary_heap() {}

```

```

87         ch[1] -> sum + key + ch[0] ->
88             ↪ suffix);
89     maxsum = max(max(ch[0] -> maxsum, ch[1] ->
90             ↪ maxsum),
91             ch[0] -> suffix + key + ch[1] ->
92                 ↪ prefix);
93
94     if (!heap.empty()) {
95         prefix = max(prefix,
96             ch[0] -> sum + key +
97                 ↪ heap.top());
98         suffix = max(suffix,
99             ch[1] -> sum + key +
100                 ↪ heap.top());
101        maxsum = max(maxsum, max(ch[0] -> suffix,
102            ch[1] -> prefix) +
103                ↪ key +
104                    ↪ heap.top());
105
106        if (heap.size() > 1) {
107            maxsum = max(maxsum, heap.top2() +
108                ↪ key);
109        }
110    }
111
112    } null[maxn << 1], *ptr = null;
113
114 void addedge(int, int, int);
115 void deledge(int, int);
116 void modify(int, int, int);
117 void modify_color(int);
118 node *newnode(int);
119 node *access(node *);
120 void makeroott(node *);
121 void link(node *, node *);
122 void cut(node *, node *);
123 void splay(node *);
124 void rot(node *, int);
125
126 queue<node *> freenodes;
127 tree<pair<int, int>, node *> mp;
128
129 bool col[maxn] = {false};
130 char c;
131 int n, m, k, x, y, z;
132
133 int main() {
134     null -> ch[0] = null -> ch[1] = null -> p = null;
135     scanf("%d%d%d", &n, &m, &k);
136
137     for (int i = 1; i <= n; i++)
138         newnode(0);
139
140     heap.push(0);
141
142     while (k--) {
143         scanf("%d", &x);
144
145         col[x] = true;
146         null[x].heap.push(0);
147     }
148
149     for (int i = 1; i < n; i++) {
150         scanf("%d%d%d", &x, &y, &z);
151
152         if (x > y)
153             swap(x, y);
154         addedge(x, y, z);
155
156         while (m--) {
157             scanf(" %c%d", &c, &x);
158
159             if (c == 'A') {
160                 scanf("%d", &y);
161
162                 if (x > y)
163                     swap(x, y);
164                 deledge(x, y);
165
166             else if (c == 'B') {
167                 scanf("%d%d", &y, &z);
168
169                 if (x > y)
170                     swap(x, y);
171                 addedge(x, y, z);
172
173             else if (c == 'C') {
174                 scanf("%d%d", &y, &z);
175
176                 if (x > y)
177                     swap(x, y);
178                 modify(x, y, z);
179
180             else
181                 modify_color(x);
182
183             printf("%lld\n", (heap.top() > 0 ? heap.top() :
184                             -1));
185
186             return 0;
187         }
188
189         void addedge(int x, int y, int z) {
190             node *tmp;
191             if (freenodes.empty())
192                 tmp = newnode(z);
193             else {
194                 tmp = freenodes.front();
195                 freenodes.pop();
196                 *tmp = node(z);
197             }
198
199             tmp -> ch[0] = tmp -> ch[1] = tmp -> p = null;
200
201             heap.push(tmp -> maxsum);
202             link(tmp, null + x);
203             link(tmp, null + y);
204             mp[make_pair(x, y)] = tmp;
205
206         void deledge(int x, int y) {
207             node *tmp = mp[make_pair(x, y)];
208
209             cut(tmp, null + x);
210             cut(tmp, null + y);
211
212             freenodes.push(tmp);
213             heap.erase(tmp -> maxsum);
214             mp.erase(make_pair(x, y));
215
216         void modify(int x, int y, int z) {
217             node *tmp = mp[make_pair(x, y)];
218             makeroott(tmp);
219             tmp -> pushdown();
220
221             heap.erase(tmp -> maxsum);
222             tmp -> key = z;
223             tmp -> refresh();
224             heap.push(tmp -> maxsum);
225
226         }
227
228     }
229
230     cout << "OK" << endl;
231
232     return 0;
233 }

```

```

222 void modify_color(int x) {
223     makeroot(null + x);
224     col[x] ^= true;
225
226     if (col[x])
227         null[x].heap.push(0);
228     else
229         null[x].heap.erase(0);
230
231     heap.erase(null[x].maxsum);
232     null[x].refresh();
233     heap.push(null[x].maxsum);
234 }
235
236 node *newnode(int k) {
237     *(++ptr) = node(k);
238     ptr -> ch[0] = ptr -> ch[1] = ptr -> p = null;
239     return ptr;
240 }
241
242 node *access(node *x) {
243     splay(x);
244     heap.erase(x -> maxsum);
245     x -> refresh();
246
247     if (x -> ch[1] != null) {
248         x -> ch[1] -> pushdown();
249         x -> heap.push(x -> ch[1] -> prefix);
250         x -> refresh();
251         heap.push(x -> ch[1] -> maxsum);
252     }
253
254     x -> ch[1] = null;
255     x -> refresh();
256     node *y = x;
257     x = x -> p;
258
259     while (x != null) {
260         splay(x);
261         heap.erase(x -> maxsum);
262
263         if (x -> ch[1] != null) {
264             x -> ch[1] -> pushdown();
265             x -> heap.push(x -> ch[1] -> prefix);
266             heap.push(x -> ch[1] -> maxsum);
267         }
268
269         x -> heap.erase(y -> prefix);
270         x -> ch[1] = y;
271         (y = x) -> refresh();
272         x = x -> p;
273     }
274
275     heap.push(y -> maxsum);
276     return y;
277 }
278
279 void makeroot(node **x) {
280     access(x);
281     splay(x);
282     x -> rev ^= true;
283 }
284
285 void link(node *x, node *y) { // 新添一条虚边, 维护y对应的堆
286     makeroot(x);
287     makeroot(y);
288
289     x -> pushdown();
290     x -> p = y;
291     heap.erase(y -> maxsum);
292     y -> heap.push(x -> prefix);
293
294     y -> refresh();
295     heap.push(y -> maxsum);
296 }
297
298 void cut(node **x, node *y) { // 断开一条实边, 一条链变成
299     // 两条链, 需要维护全局堆
300     makeroot(x);
301     access(y);
302     splay(y);
303
304     heap.erase(y -> maxsum);
305     heap.push(y -> ch[0] -> maxsum);
306     y -> ch[0] -> p = null;
307     y -> ch[0] = null;
308     y -> refresh();
309     heap.push(y -> maxsum);
310
311     void splay(node *x) {
312         x -> pushdown();
313
314         while (!isroot(x)) {
315             if (!isroot(x -> p))
316                 x -> p -> p -> pushdown();
317
318             x -> p -> pushdown();
319
320             if (isroot(x -> p)) {
321                 rot(x -> p, dir(x) ^ 1);
322                 break;
323             }
324
325             if (dir(x) == dir(x -> p))
326                 rot(x -> p -> p, dir(x -> p) ^ 1);
327             else
328                 rot(x -> p, dir(x) ^ 1);
329
330             rot(x -> p, dir(x) ^ 1);
331         }
332     }
333
334 void rot(node **x, int d) {
335     node *y = x -> ch[d ^ 1];
336
337     if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
338         y -> ch[d] -> p = x;
339
340     y -> p = x -> p;
341
342     if (!isroot(x))
343         x -> p -> ch[dir(x)] = y;
344
345     (y -> ch[d] = x) -> p = y;
346
347     x -> refresh();
348     y -> refresh();
349 }

```

4.8 K-D树

4.8.1 动态 K-D 树 (定期重构)

```

1 int l[2], r[2], x[B + 10][2], w[B + 10];
2 int n, op, ans = 0, cnt = 0, tmp = 0;
3 int d;
4
5 struct node {
6     int x[2], l[2], r[2], w, sum;
7     node *ch[2];
8

```

```

9  bool operator < (const node &a) const {
10     return x[d] < a.x[d];
11 }
12
13 void refresh() {
14     sum = ch[0] -> sum + ch[1] -> sum + w;
15     l[0] = min(x[0], min(ch[0] -> l[0], ch[1] ->
16                 -> l[0]));
17     l[1] = min(x[1], min(ch[0] -> l[1], ch[1] ->
18                 -> l[1]));
19     r[0] = max(x[0], max(ch[0] -> r[0], ch[1] ->
20                 -> r[0]));
21     r[1] = max(x[1], max(ch[0] -> r[1], ch[1] ->
22                 -> r[1]));
23 }
24 } null[maxn], *root = null;
25
26 void build(int l, int r, int k, node *&rt) {
27     if (l > r) {
28         rt = null;
29         return;
30     }
31
32     int mid = (l + r) / 2;
33
34     d = k;
35     nth_element(null + l, null + mid, null + r + 1);
36
37     rt = null + mid;
38     build(l, mid - 1, k ^ 1, rt -> ch[0]);
39     build(mid + 1, r, k ^ 1, rt -> ch[1]);
40
41     rt -> refresh();
42 }
43
44 void query(node *rt) {
45     if (l[0] <= rt -> l[0] && l[1] <= rt -> l[1] && rt
46         ->-> r[0] <= r[0] && rt -> r[1] <= r[1]) {
47         ans += rt -> sum;
48         return;
49     }
50     else if (l[0] > rt -> r[0] || l[1] > rt -> r[1] ||
51             -> r[0] < rt -> l[0] || r[1] < rt -> l[1])
52         return;
53
54     if (l[0] <= rt -> x[0] && l[1] <= rt -> x[1] && rt
55         ->-> x[0] <= r[0] && rt -> x[1] <= r[1])
56         ans += rt -> w;
57
58     query(rt -> ch[0]);
59     query(rt -> ch[1]);
60 }
61
62 int main() {
63
64     null -> l[0] = null -> l[1] = 100000000;
65     null -> r[0] = null -> r[1] = -100000000;
66     null -> sum = 0;
67     null -> ch[0] = null -> ch[1] = null;
68     scanf("%*d");
69
70     while (scanf("%d", &op) == 1 && op != 3) {
71         if (op == 1) {
72             tmp++;
73             scanf("%d%d%d", &x[tmp][0], &x[tmp][1],
74                   -> &w[tmp]);
75             x[tmp][0] ^= ans;
76             x[tmp][1] ^= ans;
77             w[tmp] ^= ans;
78
79             if (tmp == B) {
80                 for (int i = 1; i <= tmp; i++) {
81                     null[cnt + i].x[0] = x[i][0];
82                     null[cnt + i].x[1] = x[i][1];
83                     null[cnt + i].w = w[i];
84                 }
85                 build(1, cnt += tmp, 0, root);
86                 tmp = 0;
87             }
88             else {
89                 scanf("%d%d%d", &l[0], &l[1], &r[0],
90                       -> &r[1]);
91                 l[0] ^= ans;
92                 l[1] ^= ans;
93                 r[0] ^= ans;
94                 r[1] ^= ans;
95                 ans = 0;
96
97                 for (int i = 1; i <= tmp; i++) {
98                     if (l[0] <= x[i][0] && l[1] <= x[i][1]
99                         ->&& x[i][0] <= r[0] && x[i][1] <=
100                             -> r[1])
101                         ans += w[i];
102
103                 query(root);
104                 printf("%d\n", ans);
105             }
106         }
107     }
108
109     return 0;
110 }

```

```

71
72     if (tmp == B) {
73         for (int i = 1; i <= tmp; i++) {
74             null[cnt + i].x[0] = x[i][0];
75             null[cnt + i].x[1] = x[i][1];
76             null[cnt + i].w = w[i];
77         }
78         build(1, cnt += tmp, 0, root);
79         tmp = 0;
80     }
81     else {
82         scanf("%d%d%d", &l[0], &l[1], &r[0],
83               -> &r[1]);
84         l[0] ^= ans;
85         l[1] ^= ans;
86         r[0] ^= ans;
87         r[1] ^= ans;
88         ans = 0;
89
90         for (int i = 1; i <= tmp; i++) {
91             if (l[0] <= x[i][0] && l[1] <= x[i][1]
92                 ->&& x[i][0] <= r[0] && x[i][1] <=
93                     -> r[1])
94                 ans += w[i];
95
96         query(root);
97         printf("%d\n", ans);
98     }
99 }
100
101 return 0;
102

```

4.9 LCA 最近公共祖先

4.9.1 Tarjan LCA $O(n + m)$

```

1 vector<pair<int, int>> q[maxn];
2 int lca[maxn];
3
4 void dfs(int x) {
5     dfn[x] = ++tim; // 其实求LCA是用不到DFS序的，但是一般其他步骤要用
6     ufs[x] = x;
7
8     for (auto pi : q[x]) {
9         int y = pi.first, i = pi.second;
10        if (dfn[y])
11            lca[i] = findufs(y);
12    }
13
14    for (int y : G[x]) {
15        if (y != p[x]) {
16            p[y] = x;
17            dfs(y);
18        }
19    }
20
21    ufs[x] = p[x];
22 }

```

4.10 虚树

```

1 struct Tree {
2     vector<int> G[maxn], W[maxn];
3     int p[maxn], d[maxn], size[maxn], mn[maxn],
4          mx[maxn];

```

```

4   bool col[maxn];
5   long long ans_sum;
6   int ans_min, ans_max;
7
8   void add(int x, int y, int z) {
9     G[x].push_back(y);
10    W[x].push_back(z);
11  }
12
13  void dfs(int x) {
14    size[x] = col[x];
15    mx[x] = (col[x] ? d[x] : -inf);
16    mn[x] = (col[x] ? d[x] : inf);
17
18    for (int i = 0; i < (int)G[x].size(); i++) {
19      d[G[x][i]] = d[x] + W[x][i];
20      dfs(G[x][i]);
21      ans_sum += (long long)size[x] * size[G[x]
22        ↪ [i]] * d[x];
23      ans_max = max(ans_max, mx[x] + mx[G[x][i]]
24        ↪ - (d[x] << 1));
25      ans_min = min(ans_min, mn[x] + mn[G[x][i]]
26        ↪ - (d[x] << 1));
27      size[x] += size[G[x][i]];
28      mx[x] = max(mx[x], mx[G[x][i]]);
29      mn[x] = min(mn[x], mn[G[x][i]]);
30    }
31  }
32
33  void clear(int x) {
34    G[x].clear();
35    W[x].clear();
36    col[x] = false;
37  }
38
39  void solve(int rt) {
40    ans_sum = 0;
41    ans_max = -inf;
42    ans_min = inf;
43    dfs(rt);
44    ans_sum <= 1;
45  }
46  virtree;
47
48  void dfs(int);
49  int LCA(int, int);
50
51  vector<int> G[maxn];
52  int f[maxn][20], dfn[maxn], dfn[maxn], tim = 0;
53
54  bool cmp(int x, int y) {
55    return dfn[x] < dfn[y];
56  }
57
58  int n, m, lgn = 0, a[maxn], s[maxn], v[maxn];
59
60  int main() {
61    scanf("%d", &n);
62
63    for (int i = 1, x, y; i < n; i++) {
64      scanf("%d%d", &x, &y);
65      G[x].push_back(y);
66      G[y].push_back(x);
67    }
68
69    G[n + 1].push_back(1);
70    dfs(n + 1);
71
72    lgn--;
73
74    for (int j = 1; j <= lgn; j++)
75      for (int i = 1; i <= n; i++)
76        f[i][j] = f[f[i][j - 1]][j - 1];
77
78    scanf("%d", &m);
79
80    while (m--) {
81      int k;
82      scanf("%d", &k);
83
84      for (int i = 1; i <= k; i++)
85        scanf("%d", &a[i]);
86
87      sort(a + 1, a + k + 1, cmp);
88      int top = 0, cnt = 0;
89      s[++top] = v[++cnt] = n + 1;
90      long long ans = 0;
91
92      for (int i = 1; i <= k; i++) {
93        virtree.col[a[i]] = true;
94        ans += d[a[i]] - 1;
95        int u = LCA(a[i], s[top]);
96
97        if (s[top] != u) {
98          while (top > 1 && d[s[top - 1]] >=
99            ↪ d[u]) {
100            virtree.add(s[top - 1], s[top],
101              ↪ d[s[top]] - d[s[top - 1]]);
102            top--;
103          }
104
105          if (s[top] != u) {
106            virtree.add(u, s[top], d[s[top]] -
107              ↪ d[u]);
108            s[top] = v[++cnt] = u;
109          }
110        }
111
112        s[++top] = a[i];
113      }
114
115      virtree.add(s[i], s[i + 1], d[s[i + 1]] -
116        ↪ d[s[i]]);
117
118      virtree.solve(n + 1);
119      ans *= k - 1;
120      printf("%lld %d %d\n", ans - virtree.ans_sum,
121        ↪ virtree.ans_min, virtree.ans_max);
122
123      for (int i = 1; i <= k; i++)
124        virtree.clear(a[i]);
125      for (int i = 1; i <= cnt; i++)
126        virtree.clear(v[i]);
127
128      return 0;
129    }
130
131    void dfs(int x) {
132      dfn[x] = ++tim;
133      d[x] = d[f[x][0]] + 1;
134
135      while ((1 << lgn) < d[x])
136        lgn++;
137
138      for (int i = 0; i < (int)G[x].size(); i++)
139        if (G[x][i] != f[x][0]) {
140          f[G[x][i]][0] = x;
141          dfs(G[x][i]);
142        }
143    }
144  }
145
146  if (G[x][i] != f[x][0]) {
147    f[G[x][i]][0] = x;
148    dfs(G[x][i]);
149  }
150}

```

```

140     }
141 }
142
143 int LCA(int x, int y) {
144     if (d[x] != d[y]) {
145         if (d[x] < d[y])
146             swap(x, y);
147
148         for (int i = lgn; i >= 0; i--)
149             if (((d[x] - d[y]) >> i) & 1)
150                 x = f[x][i];
151     }
152
153     if (x == y)
154         return x;
155
156     for (int i = lgn; i >= 0; i--)
157         if (f[x][i] != f[y][i]) {
158             x = f[x][i];
159             y = f[y][i];
160         }
161
162     return f[x][0];
163 }
```

```

42     v[x][h[x] - j - 1] += v[y][h[y] - j];
43
44     int t = v[x][h[x] - j - 1];
45     if (t > mx || (t == mx && h[x] - j - 1
46         > ans[x])) {
47         mx = t;
48         ans[x] = h[x] - j - 1;
49     }
50
51     v[y].clear();
52 }
53 }
```

4.11 长链剖分

```

1 // 顾名思义，长链剖分是取最深的儿子作为重儿子
2
3 // O(n) 维护以深度为下标的子树信息
4 vector<int> G[maxn], v[maxn];
5 int n, p[maxn], h[maxn], son[maxn], ans[maxn];
6
7 // 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
8 //   → 取最大深度
9 // 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
10 // 或者开一个结构体维护倒过来的vector
11 void dfs(int x) {
12     h[x] = 1;
13
14     for (int y : G[x])
15         if (y != p[x]){
16             p[y] = x;
17             dfs(y);
18
19             if (h[y] > h[son[x]])
20                 son[x] = y;
21
22     if (!son[x]) {
23         v[x].push_back(1);
24         ans[x] = 0;
25         return;
26     }
27
28     h[x] = h[son[x]] + 1;
29     swap(v[x], v[son[x]]);
30
31     if (v[x][ans[son[x]]] == 1)
32         ans[x] = h[x] - 1;
33     else
34         ans[x] = ans[son[x]];
35
36     v[x].push_back(1);
37
38     int mx = v[x][ans[x]];
39     for (int y : G[x])
40         if (y != p[x] && y != son[x]) {
41             for (int j = 1; j <= h[y]; j++) {
```

4.11.1 梯子剖分

```

1 // 在线求一个点的第k祖先 O(n \log n)-O(1)
2 // 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
3
4 // 全局数组定义
5 vector<int> G[maxn], v[maxn];
6 int d[maxn], mxd[maxn], son[maxn], top[maxn],
7     len[maxn];
8 int f[19][maxn], log_tbl[maxn];
9
10 // 在主函数中两遍dfs之后加上如下预处理
11 log_tbl[0] = -1;
12 for (int i = 1; i <= n; i++)
13     log_tbl[i] = log_tbl[i / 2] + 1;
14 for (int j = 1; (1 << j) < n; j++)
15     for (int i = 1; i <= n; i++)
16         f[j][i] = f[j - 1][f[j - 1][i]];
17
18 // 第一遍dfs，用于计算深度和找出重儿子
19 void dfs1(int x) {
20     mxd[x] = d[x];
21
22     for (int y : G[x])
23         if (y != f[0][x]){
24             f[0][y] = x;
25             d[y] = d[x] + 1;
26
27             dfs1(y);
28
29             mxd[x] = max(mxd[x], mxd[y]);
30             if (mxd[y] > mxd[son[x]])
31                 son[x] = y;
32         }
33
34 // 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
35 void dfs2(int x) {
36     top[x] = (x == son[f[0][x]] ? top[f[0][x]] : x);
37
38     for (int y : G[x])
39         if (y != f[0][x])
40             dfs2(y);
41
42     if (top[x] == x) {
43         int u = x;
44         while (top[son[u]] == x)
45             u = son[u];
46
47         len[x] = d[u] - d[x];
48         for (int i = 0; i < len[x]; i++, u = f[0][u])
49             v[x].push_back(u);
50         u = x;
51     }
```

```

52     for (int i = 0; i < len[x] && u; i++, u = f[0]
53         ↪ [u])
54         v[x].push_back(u);
55     }
56
57 // 在线询问x的k级祖先 O(1)
58 // 不存在时返回0
59 int query(int x, int k) {
60     if (!k)
61         return x;
62     if (k > d[x])
63         return 0;
64
65     x = f[log_tbl[k]][x];
66     k ^= 1 << log_tbl[k];
67     return v[top[x]][d[top[x]] + len[top[x]] - d[x] +
68         ↪ k];
}

```

4.12 堆

4.12.1 左偏树

参见3.2.3.k短路 (第 30 页).

4.12.2 二叉堆

```

1 struct my_binary_heap {
2     static constexpr int maxn = 100005;
3
4     int a[maxn], size;
5
6     my_binary_heap() : size(0) {}
7
8     void push(int val) {
9         a[++size] = val;
10
11        for (int x = size; x > 1; x /= 2) {
12            if (a[x] < a[x / 2])
13                swap(a[x], a[x / 2]);
14            else
15                break;
16        }
17    }
18
19    int &top() {
20        return a[1];
21    }
22
23    int pop() {
24        int res = a[1];
25        a[1] = a[size--];
26
27        for (int x = 1, son; ; x = son) {
28            if (x * 2 == size)
29                son = x * 2;
30            else if (x * 2 > size)
31                break;
32            else if (a[x * 2] < a[x * 2 + 1])
33                son = x * 2;
34            else
35                son = x * 2 + 1;
36
37            if (a[son] < a[x])
38                swap(a[x], a[son]);
39            else
40                break;
41        }
}

```

```

42
43         ↪
44     }
45 }

```

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 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)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 100005, B = 314;
6
7 struct Q {
8     int l, r, d, id;
9
10    Q() = default;
11
12    Q(int l, int r, int d, int id) : l(l), r(r), d(d),
13        ↪ id(id) {}
14
15    friend bool operator < (const Q &a, const Q &b) {
16        if (a.d != b.d)
17            ↪ return a.d < b.d;
18
19        return a.r < b.r;
20    }
21 } q[maxn]; // 结构体可以复用, d既可以作为左端点块编号, 也
22             ↪ 可以作为二次离线处理的倍数
23
24 int global_n, bid[maxn], L[maxn], R[maxn], cntb;
25
26 int sa[maxn], sb[maxn];
27
28 void addp(int x) { // sqrt(n) 修改 O(1) 查询
29     for (int k = bid[x]; k <= cntb; k++)
30         sb[k]++;
31
32     for (int i = x; i <= R[bid[x]]; i++)
33         sa[i]++;
34 }
35
36 int queryp(int x) {
37     if (!x)
38         ↪ return 0;
39
40     int res = 0;
41
42     for (int i = 1; i <= L[bid[x]]; i++)
43         res += sb[i];
44
45     for (int i = 1; i <= R[bid[x]] - L[bid[x]] + 1; i++)
46         res -= sa[i];
47
48     for (int i = bid[x] + 1; i <= cntb; i++)
49         res += sb[i];
50
51     return res;
52 }

```

```

38     return sa[x] + sb[bid[x] - 1];
39 }
40
41 void adds(int x) {
42     for (int k = 1; k <= bid[x]; k++)
43         sb[k]++;
44
45     for (int i = L[bid[x]]; i <= x; i++)
46         sa[i]++;
47 }
48
49 int querys(int x) {
50     if (x > global_n)
51         return 0; // 为了防止越界就判一下
52     return sa[x] + sb[bid[x] + 1];
53 }
54
55 vector<Q> vp[maxn], vs[maxn]; // prefix, suffix
56
57 long long fp[maxn], fs[maxn]; // prefix, suffix
58
59 int a[maxn], b[maxn];
60
61 long long ta[maxn], ans[maxn];
62
63 int main() {
64
65     int n, m;
66     scanf("%d%d", &n, &m);
67
68     global_n = n;
69
70     for (int i = 1; i <= n; i++)
71         scanf("%d", &a[i]);
72
73     memcpy(b, a, sizeof(int) * (n + 1));
74     sort(b + 1, b + n + 1);
75
76     for (int i = 1; i <= n; i++)
77         a[i] = lower_bound(b + 1, b + n + 1, a[i]) - b;
78
79     for (int i = 1; i <= n; i++) {
80         bid[i] = (i - 1) / B + 1;
81
82         if (!L[bid[i]])
83             L[bid[i]] = i;
84
85         R[bid[i]] = i;
86         cntb = bid[i];
87     }
88
89     for (int i = 1; i <= m; i++) {
90         scanf("%d%d", &q[i].l, &q[i].r);
91
92         q[i].d = bid[q[i].l];
93         q[i].id = i;
94     }
95
96     sort(q + 1, q + m + 1);
97
98     int l = 2, r = 1; // l, r是上一个查询的端点
99
100    for (int i = 1; i <= m; i++) {
101        int s = q[i].l, t = q[i].r; // s, t是当前要调整
102        // 到的端点
103
104        if (s < l)
105            vs[r + 1].push_back(Q(s, l - 1, 1, i));
106        else if (s > l)
107            vs[r + 1].push_back(Q(l, s - 1, -1, i));

```

```

107
108     l = s;
109
110     if (t > r)
111         vp[l - 1].push_back(Q(r + 1, t, 1, i));
112     else if (t < r)
113         vp[l - 1].push_back(Q(t + 1, r, -1, i));
114
115     r = t;
116 }
117
118 for (int i = 1; i <= n; i++) { // 第一遍正着处理, 解
119     // 决关于前缀的询问
120     fp[i] = fp[i - 1] + querys(a[i] + 1);
121     adds(a[i]);
122
123     for (auto q : vp[i]) {
124         long long tmp = 0;
125         for (int k = q.l; k <= q.r; k++)
126             tmp += querys(a[k] + 1);
127
128         ta[q.id] -= q.d * tmp;
129     }
130 }
131
132 memset(sa, 0, sizeof(sa));
133 memset(sb, 0, sizeof(sb));
134
135 for (int i = n; i; i--) { // 第二遍倒着处理, 解决关
136     // 于后缀的询问
137     fs[i] = fs[i + 1] + queryp(a[i] - 1);
138     addp(a[i]);
139
140     for (auto q : vs[i]) {
141         long long tmp = 0;
142         for (int k = q.l; k <= q.r; k++)
143             tmp += queryp(a[k] - 1);
144
145         ta[q.id] -= q.d * tmp;
146     }
147 }
148
149 l = 2;
150 r = 1;
151
152 for (int i = 1; i <= m; i++) { // 求出fs和fp之后再加
153     // 上这部分的贡献
154     int s = q[i].l, t = q[i].r;
155
156     ta[i] += fs[s] - fs[l];
157     ta[i] += fp[t] - fp[r];
158
159     l = s;
160     r = t;
161
162     ta[i] += ta[i - 1]; // 因为算出来的是相邻两个询
163     // 问之间的贡献, 所以要前缀和
164     ans[q[i].id] = ta[i];
165 }
166
167 for (int i = 1; i <= m; i++)
168     printf("%lld\n", ans[i]);
169 }

```

4.13.2 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。

4.13.3 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。）

```

1 constexpr int maxn = 100005, B = 315, maxb = maxn / B +
2   ↪ 5;
3
4 int n, bid[maxn], L[maxb], R[maxb], cntb;
5
6 struct DS { // O(sqrt(n)) 修改 O(1) 查询
7     int total;
8     int sa[maxn], sb[maxb];
9
10    void init(const DS &o) {
11        total = o.total;
12        memcpy(sa, o.sa, sizeof(int) * (n + 1));
13        memcpy(sb, o.sb, sizeof(int) * (cntb + 1));
14    }
15
16    void add(int x) {
17        total++;
18        for (int k = 1; k <= bid[x]; k++)
19            sb[k]++;
20        for (int i = L[bid[x]]; i <= x; i++)
21            sa[i]++;
22    }
23
24    int querys(int x) {
25        if (x > n)
26            return 0;
27
28        return sb[bid[x] + 1] + sa[x];
29    }
30
31    int queryp(int x) {
32        return total - querys(x + 1);
33    }
34 } pr[maxb];
35
36 int c[maxn]; // 树状数组
37
38 void addc(int x, int d) {
39     while (x) {
40         c[x] += d;
41         x -= x & -x;
42     }
43
44 int queryc(int x) {
45     int ans = 0;
46     while (x <= n) {
47         ans += c[x];
48         x += x & -x;
49     }
50
51     return ans;
52 }
53
54 long long fp[maxn], fs[maxn];
55
56 int rnk[maxn], val[maxn][B + 5];
57
58 long long dat[maxb][maxb];
59
60 int a[maxn];
61
62 int main() {
63
64     int m;
65     cin >> n >> m;
66
67     for (int i = 1; i <= n; i++) {
68         cin >> a[i];
69
70         bid[i] = (i - 1) / B + 1;
71         if (!L[bid[i]])
72             L[bid[i]] = i;
73         R[bid[i]] = i;
74         cntb = bid[i];
75
76         rnk[i] = i;
77     }
78
79     for (int k = 1; k <= cntb; k++)
80         sort(rnk + L[k], rnk + R[k] + 1, [](int x, int
81           ↪ y) {return a[x] < a[y];}); // 每个块排序
82
83     for (int i = n; i; i--) {
84         for (int j = 2; i + j - 1 <= R[bid[i]]; j++) {
85             val[i][j] = val[i + 1][j - 1] + val[i][j -
86               ↪ 1] - val[i + 1][j - 2];
87             if (a[i] > a[i + j - 1])
88                 val[i][j]++; // 块内用二维前缀和预处理
89     }
90
91     for (int k = 1; k <= cntb; k++) {
92         for (int i = L[k]; i <= R[k]; i++) {
93             dat[k][k] += queryc(a[i] + 1); // 单块内的逆
94             ↪ 序对直接用树状数组预处理
95             addc(a[i], 1);
96     }
97
98     for (int i = L[k]; i <= R[k]; i++) {
99         addc(a[i], -1);
100
101     for (int i = 1; i <= n; i++) {
102         if (i > 1 && i == L[bid[i]])
103             pr[bid[i]].init(pr[bid[i] - 1]);
104
105         fp[i] = fp[i - 1] + pr[bid[i]].querys(a[i] +
106           ↪ 1);
107
108         pr[bid[i]].add(a[i]);
109     }
110
111     for (int i = n; i; i--) {
112         fs[i] = fs[i + 1] + (n - i - queryc(a[i] + 1));
113     }
114 }
115 }
```

```

109     addc(a[i], 1);
110 }
111
112 for (int s = 1; s <= cntb; s++) {
113     for (int t = s + 1; t <= cntb; t++) {
114         dat[s][t] = dat[s][t - 1] + dat[t][t];
115
116         for (int i = L[t]; i <= R[t]; i++) // 块间的
117             // 逆序对用刚才处理的分块求出
118             dat[s][t] += pr[t - 1].querys(a[i] + 1)
119             - pr[s - 1].querys(a[i] + 1);
120     }
121
122 long long ans = 0;
123
124 while (m--) {
125     long long s, t;
126     cin >> s >> t;
127
128     int l = s ^ ans, r = t ^ ans;
129
130     if (bid[l] == bid[r])
131         ans = val[l][r - l + 1];
132     else {
133         ans = dat[bid[l] + 1][bid[r] - 1];
134
135         ans += fp[r] - fp[L[bid[r]] - 1];
136         for (int i = L[bid[r]]; i <= r; i++)
137             ans -= pr[bid[l]].querys(a[i] + 1);
138
139         ans += fs[l] - fs[R[bid[l]] + 1];
140         for (int i = l; i <= R[bid[l]]; i++)
141             ans -= (a[i] - 1) - pr[bid[r] - 1].queryp(a[i] - 1);
142
143         int i = L[bid[l]], j = L[bid[r]], w = 0; // 手写归并
144
145         while (true) {
146             while (i <= R[bid[l]] && rnk[i] < l)
147                 i++;
148             while (j <= R[bid[r]] && rnk[j] > r)
149                 j++;
150
151             if (i > R[bid[l]] && j > R[bid[r]])
152                 break;
153
154             int x = (i <= R[bid[l]] ? a[rnk[i]] : (int)1e9),
155                 y = (j <= R[bid[r]] ? a[rnk[j]] : (int)1e9);
156
157             if (x < y) {
158                 ans += w;
159                 i++;
160             } else {
161                 j++;
162                 w++;
163             }
164         }
165         cout << ans << '\n';
166     }
167
168     return 0;
169 }
```

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

```

1 int fail[maxn];
2
3 void kmp(const char *s, int n) {
4     fail[0] = fail[1] = 0;
5
6     for (int i = 1; i < n; i++) {
7         int j = fail[i];
8
9         while (j && s[i + 1] != s[j + 1])
10            j = fail[j];
11
12         if (s[i + 1] == s[j + 1])
13             fail[i + 1] = j + 1;
14         else
15             fail[i + 1] = 0;
16     }
17 }
```

5.1.1 ex-KMP

```

1 void exkmp(const char *s, int *a, int n) {
2     int l = 0, r = 0;
3     a[0] = n;
4
5     for (int i = 1; i <= n; i++) {
6         a[i] = i > r ? 0 : min(r - i + 1, a[i - l]);
7
8         while (i + a[i] < n && s[a[i]] == s[i + a[i]])
9             a[i]++;
10
11         if (i + a[i] - 1 > r) {
12             l = i;
13             r = i + a[i] - 1;
14         }
15     }
16 }
```

5.2 AC 自动机

```

1 int ch[maxm][26], f[maxm][26], q[maxm], sum[maxm], cnt
2     → = 0;
3
4 // 在字典树中插入一个字符串 O(n)
5 int insert(const char *c) {
6     int x = 0;
7     while (*c) {
8         if (!ch[x][*c - 'a'])
9             ch[x][*c - 'a'] = ++cnt;
10        x = ch[x][*c++ - 'a'];
11    }
12    return x;
13 }
14
15 // 建AC自动机 O(n * sigma)
16 void getfail() {
17     int x, head = 0, tail = 0;
18
19     for (int c = 0; c < 26; c++)
20         if (ch[0][c])
21             q[tail++] = ch[0][c]; // 把根节点的儿子加入
22                         → 队列
23
24     while (head != tail) {
```

```

23         x = q[head++];
24
25         G[f[x][0]].push_back(x);
26         fill(f[x] + 1, f[x] + 26, cnt + 1);
27
28         for (int c = 0; c < 26; c++) {
29             if (ch[x][c]) {
30                 int y = f[x][0];
31
32                 f[ch[x][c]][0] = ch[y][c];
33                 q[tail++] = ch[x][c];
34             } else
35                 ch[x][c] = ch[f[x][0]][c];
36         }
37     }
38
39     fill(f[0], f[0] + 26, cnt + 1);
40 }
```

5.3 后缀数组

5.3.1 倍增

```

1 constexpr int MAXN = 100005;
2
3 // 清空的话全部都要清空到 max(n, m) + 1
4 void get_sa(char *s, int n, int *sa, int *rnk, int
5     → *height) { // 1-base
6     static int buc[MAXN], id[MAXN], p[MAXN], t[MAXN *
7         → 2];
8
9     int m = 300;
10
11    for (int i = 1; i <= n; i++)
12        buc[rnk[i]] = s[i]++;
13    for (int i = 1; i <= m; i++)
14        buc[i] += buc[i - 1];
15    for (int i = n; i; i--)
16        sa[buc[rnk[i]]--] = i;
17
18    memset(buc, 0, sizeof(int) * (m + 1));
19
20    for (int k = 1, cnt = 0; cnt != n; k *= 2, m = cnt)
21        → {
22        cnt = 0;
23        for (int i = n; i > n - k; i--)
24            id[++cnt] = i;
25
26        for (int i = 1; i <= n; i++)
27            if (sa[i] > k)
28                id[++cnt] = sa[i] - k;
29
30        for (int i = 1; i <= n; i++)
31            buc[p[i]] = rnk[id[i]]++;
32        for (int i = 1; i <= m; i++)
33            buc[i] += buc[i - 1];
34        for (int i = n; i; i--)
35            sa[buc[p[i]]--] = id[i];
36
37        memset(buc, 0, sizeof(int) * (m + 1));
38
39        memcpy(t, rnk, sizeof(int) * (n + 1));
40
41        cnt = 0;
42        for (int i = 1; i <= n; i++) {
43            if (t[sa[i]] != t[sa[i - 1]] || t[sa[i] +
44                → k] != t[sa[i - 1] + k])
45                cnt++;
46        }
47    }
48 }
```

```

43     rnk[sa[i]] = cnt;
44 }
45 }
46
47 for (int i = 1; i <= n; i++)
48     sa[rnk[i]] = i;
49
50 for (int i = 1, k = 0; i <= n; i++) { // 顺便
   ↪ 求height
51     if (k)
52         k--;
53
54     if (rnk[i] > 1)
55         while (sa[rnk[i] - 1] + k <= n && s[i + k]
   ↪ == s[sa[rnk[i] - 1] + k])
56         k++;
57
58     height[rnk[i]] = k; // height[i] = lcp(sa[i],
   ↪ sa[i - 1])
59 }
60
61 char s[MAXN];
62 int sa[MAXN], rnk[MAXN], height[MAXN];
63
64 int main() {
65     cin >> (s + 1);
66
67     int n = strlen(s + 1);
68
69     get_sa(s, n, sa, rnk, height);
70
71     for (int i = 1; i <= n; i++)
72         cout << sa[i] << (i < n ? ' ' : '\n');
73
74     for (int i = 2; i <= n; i++)
75         cout << height[i] << (i < n ? ' ' : '\n');
76
77
78     return 0;
79 }
```

5.3.2 SA-IS

```

1 // 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
2
3 constexpr int maxn = 100005, l_type = 0, s_type = 1;
4
5 // 判断一个字符是否为LMS字符
6 bool is_lms(int *tp, int x) {
7     return x > 0 && tp[x] == s_type && tp[x - 1] ==
   ↪ l_type;
8 }
9
10 // 判断两个LMS子串是否相同
11 bool equal_substr(int *s, int x, int y, int *tp) {
12     do {
13         if (s[x] != s[y])
14             return false;
15         x++;
16         y++;
17     } while (!is_lms(tp, x) && !is_lms(tp, y));
18
19     return s[x] == s[y];
20 }
21
22 // 诱导排序 (从*型诱导到L型, 从L型诱导到S型)
23 // 调用之前应将*型按要求放入SA中
24 void induced_sort(int *s, int *sa, int *tp, int *buc,
   ↪ int *lbuc, int *sbuc, int n, int m) {

```

```

25     for (int i = 0; i <= n; i++)
26         if (sa[i] > 0 && tp[sa[i] - 1] == l_type)
27             sa[lbuc[s[sa[i] - 1]]++] = sa[i] - 1;
28
29     for (int i = 1; i <= m; i++)
30         sbuc[i] = buc[i] - 1;
31
32     for (int i = n; ~i; i--)
33         if (sa[i] > 0 && tp[sa[i] - 1] == s_type)
34             sa[sbuc[s[sa[i] - 1]]--] = sa[i] - 1;
35 }
36
37 // s是输入字符串, n是字符串的长度, m是字符集的大小
38 int *sa_is(int *s, int len, int m) {
39     int n = len - 1;
40
41     int *tp = new int[n + 1];
42     int *pos = new int[n + 1];
43     int *name = new int[n + 1];
44     int *sa = new int[n + 1];
45     int *buc = new int[m + 1];
46     int *lbuc = new int[m + 1];
47     int *sbuc = new int[m + 1];
48
49     memset(buc, 0, sizeof(int) * (m + 1));
50     memset(lbuc, 0, sizeof(int) * (m + 1));
51     memset(sbuc, 0, sizeof(int) * (m + 1));
52
53     for (int i = 0; i <= n; i++)
54         buc[s[i]]++;
55
56     for (int i = 1; i <= m; i++) {
57         buc[i] += buc[i - 1];
58
59         lbuc[i] = buc[i - 1];
60         sbuc[i] = buc[i] - 1;
61     }
62
63     tp[n] = s_type;
64     for (int i = n - 1; ~i; i--) {
65         if (s[i] < s[i + 1])
66             tp[i] = s_type;
67         else if (s[i] > s[i + 1])
68             tp[i] = l_type;
69         else
70             tp[i] = tp[i + 1];
71     }
72
73     int cnt = 0;
74     for (int i = 1; i <= n; i++) {
75         if (tp[i] == s_type && tp[i - 1] == l_type)
76             pos[cnt++] = i;
77
78     memset(sa, -1, sizeof(int) * (n + 1));
79     for (int i = 0; i < cnt; i++)
80         sa[sbuc[s[pos[i]]]--] = pos[i];
81     induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);
82
83     memset(name, -1, sizeof(int) * (n + 1));
84     int lastx = -1, namecnt = 1;
85     bool flag = false;
86
87     for (int i = 1; i <= n; i++) {
88         int x = sa[i];
89
90         if (is_lms(tp, x)) {
91             if (lastx >= 0 && !equal_substr(s, x,
   ↪ lastx, tp))

```

```

92     namecnt++;
93
94     if (lastx >= 0 && namecnt == name[lastx])
95         flag = true;
96
97     name[x] = namecnt;
98     lastx = x;
99 }
100}
101name[n] = 0;
102
103int *t = new int[cnt];
104int p = 0;
105for (int i = 0; i <= n; i++)
106    if (name[i] >= 0)
107        t[p++] = name[i];
108
109int *tsa;
110if (!flag) {
111    tsa = new int[cnt];
112
113    for (int i = 0; i < cnt; i++)
114        tsa[t[i]] = i;
115}
116else
117    tsa = sais(t, cnt, namecnt);
118
119lbuc[0] = sbuc[0] = 0;
120for (int i = 1; i <= m; i++) {
121    lbuc[i] = buc[i - 1];
122    sbuc[i] = buc[i] - 1;
123}
124
125memset(sa, -1, sizeof(int) * (n + 1));
126for (int i = cnt - 1; ~i; i--)
127    sa[sbuc[s[pos[tsa[i]]]]--] = pos[tsa[i]];
128induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);
129
130// 多组数据的时候最好 delete 掉
131delete[] tp;
132delete[] pos;
133delete[] name;
134delete[] buc;
135delete[] lbuc;
136delete[] sbuc;
137delete[] t;
138delete[] tsa;
139
140return sa;
141}
142
143// 封装好的函数, 1-based
144void get_sa(char *s, int n, int *sa, int *rnk, int
→ *height) {
145    static int a[maxn];
146
147    for (int i = 1; i <= n; i++)
148        a[i - 1] = s[i];
149
150    a[n] = '$';
151
152    int *t = sais(a, n + 1, 256);
153    memcpy(sa, t, sizeof(int) * (n + 1));
154    delete[] t;
155
156    sa[0] = 0;
157    for (int i = 1; i <= n; i++)
158        rnk[sa[i]] = i;
159

```

```

160    for (int i = 1, k = 0; i <= n; i++) { // 求height
161        if (k)
162            k--;
163
164        while (s[i + k] == s[sa[rnk[i] - 1] + k])
165            k++;
166
167        height[rnk[i]] = k; // height[i] = lcp(sa[i],
→ sa[i - 1])
168    }
169}

```

5.3.3 SAMSA

```

1 bool vis[maxn * 2];
2 char s[maxn];
3 int n, id[maxn * 2], ch[maxn * 2][26], height[maxn],
→ tim = 0;
4
5 void dfs(int x) {
6     if (id[x]) {
7         height[tim++] = val[last];
8         sa[tim] = id[x];
9
10        last = x;
11    }
12
13    for (int c = 0; c < 26; c++)
14        if (ch[x][c])
15            dfs(ch[x][c]);
16
17    last = par[x];
18}
19
20int main() {
21    last = ++cnt;
22
23    scanf("%s", s + 1);
24    n = strlen(s + 1);
25
26    for (int i = n; i; i--) {
27        expand(s[i] - 'a');
28        id[last] = i;
29    }
30
31    vis[1] = true;
32    for (int i = 1; i <= cnt; i++) {
33        if (id[i])
34            for (int x = i, pos = n; x && !vis[x]; x =
→ par[x]) {
35                vis[x] = true;
36                pos -= val[x] - val[par[x]];
37                ch[par[x]][s[pos + 1] - 'a'] = x;
38            }
39
40    dfs(1);
41
42    for (int i = 1; i <= n; i++) {
43        if (i > 1)
44            printf(" ");
45        printf("%d", sa[i]); // 1-based
46    }
47    printf("\n");
48
49    for (int i = 1; i < n; i++) {
50        if (i > 1)
51            printf(" ");
52        printf("%d", height[i]);
53    }

```

```

54     printf("\n");
55
56     return 0;
57 }
```

```

50
51     last = np;
52 }
```

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

```

1 // 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
2 // →者哈希表替换
3 // 注意!!! 结点数要开成串长的两倍
4
5 // 全局变量与数组定义
6 int last, val[maxn], par[maxn], go[maxn][26], sam_cnt;
7 int c[maxn], q[maxn]; // 用来桶排序
8
9 // 在主函数开头加上这句初始化
10 last = sam_cnt = 1;
11
12 // 以下是按val进行桶排序的代码
13 for (int i = 1; i <= sam_cnt; i++)
14     c[val[i] + 1]++;
15 for (int i = 1; i <= n; i++)
16     c[i] += c[i - 1]; // 这里n是串长
17 for (int i = 1; i <= sam_cnt; i++)
18     q[++c[val[i]]] = i;
19
20 // 加入一个字符 均摊O(1)
21 void extend(int c) {
22     int p = last, np = ++sam_cnt;
23     val[np] = val[p] + 1;
24
25     while (p && !go[p][c]) {
26         go[p][c] = np;
27         p = par[p];
28     }
29
30     if (!p)
31         par[np] = 1;
32     else {
33         int q = go[p][c];
34
35         if (val[q] == val[p] + 1)
36             par[np] = q;
37         else {
38             int nq = ++sam_cnt;
39             val[nq] = val[p] + 1;
40             memcpy(go[nq], go[q], sizeof(go[q]));
41
42             par[nq] = par[q];
43             par[np] = par[q] = nq;
44
45             while (p && go[p][c] == q)
46                 go[p][c] = nq;
47                 p = par[p];
48             }
49     }
50 }
```

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。

```

1 int extend(int p, int c) {
2     int np = 0;
3
4     if (!go[p][c]) {
5         np = ++sam_cnt;
6         val[np] = val[p] + 1;
7         while (p && !go[p][c]) {
8             go[p][c] = np;
9             p = par[p];
10        }
11    }
12
13    if (!p)
14        par[np] = 1;
15    else {
16        int q = go[p][c];
17
18        if (val[q] == val[p] + 1) {
19            if (np)
20                par[np] = q;
21            else
22                return q;
23        }
24        else {
25            int nq = ++sam_cnt;
26            val[nq] = val[p] + 1;
27            memcpy(go[nq], go[q], sizeof(go[q]));
28
29            par[nq] = par[q];
30            par[q] = nq;
31            if (np)
32                par[np] = nq;
33
34            while (p && go[p][c] == q){
35                go[p][c] = nq;
36                p = par[p];
37            }
38
39            if (!np)
40                return nq;
41        }
42    }
43
44    return np;
45 }
46 // 调用的时候直接last = 1然后一路调用last = extend(last,
47 // →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.

```

1 int tim; // tim实际上就是当前的右端点
2
3 node *access(node **x) {
4     node *y = null;
5
6     while (*x != null) {
7         splay(*x);
8
9         (*x -> ch[1]) = null;
10        (*x -> refresh());
11
12        if ((*x -> val) // val记录的是上次访问时间, 也就
13            ↪是right集合最大值
14        | update(*x -> val - val[*x -> r] + 1, *x -> val
15            ↪- val[par[*x -> l]], -1);
16
17        *x -> val = tim;
18        *x -> lazy = true;
19
20        update(*x -> val - val[*x -> r] + 1, *x -> val -
21            ↪val[par[*x -> l]], 1);
22
23        *x -> ch[1] = y;
24
25        (*y = *x) -> refresh();
26
27        x = *x -> p;
28    }
29
30    return y;
31}
32
33 // 以下是main函数中的用法
34 for (int i = 1; i <= n; i++) {
35     tim++;
36     access(null + id[i]);
37
38     if (i >= m) // 例题询问长度是固定的, 如果不固定的话就
39         ↪按照右端点离线即可
40     | ans[i - m + 1] = query(i - m + 1, i);
41 }

```

```

mx[o][0] = max(mx[u][0], t);
mx[o][1] = max(mx[u][1], d);

if (l == r)
    return;

lc[o] = lc[u];
rc[o] = rc[u];

int mid = (l + r) / 2;
if (s <= mid)
    modify_seg(l, mid, lc[o]);
else
    modify_seg(mid + 1, r, rc[o]);

query_seg(int l, int r, int o, int k) {
if (s <= l && t >= r)
    return mx[o][k];

int mid = (l + r) / 2, ans = 0;

if (s <= mid)
    ans = max(ans, query_seg(l, mid, lc[o], k));
if (t > mid)
    ans = max(ans, query_seg(mid + 1, r, rc[o],
        → k));

return ans;

N;

d modify(int pos, int u, int v, int &rt) {
s = pos;
t = u;
d = v;

modify_seg(1, N, rt);

query(int l, int r, int rt) {
s = l;
t = r;
int ans = query_seg(1, N, rt, 0);

s = 1;
t = l;
return max(ans, query_seg(1, N, rt, 1) - l);
}

```

还有一份完整的代码，因为写起来确实细节挺多的。这份代码支持在尾部加一个字符或者询问区间有多少子串至少出现了两次，并且强制在线。

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 200005, maxm = maxn * 17 * 15;
6
7 int mx[maxm][2], lc[maxm], rc[maxm], seg_cnt;
8 int root[maxn];
9
10 int s, t, d;
11
12 void modify_seg(int l, int r, int &o) {
13     int u = o;
14     o = ++seg_cnt;
```

```
uct node {
    int size, l, r, id, tim;
    node *ch[2], *p;
    bool tag;

    node() = default;

    void apply(int v) {
        tim = v;
        tag = true;
    }

    void pushdown() {
        if (tag) {
            ch[0] -> tim = ch[1] -> tim = tim;
            ch[0] -> tag = ch[1] -> tag = true;

            tag = false;
        }
    }
}
```

```

84     }
85 }
86
87 void update() {
88     size = ch[0] -> size + ch[1] -> size + 1;
89     l = (ch[0] -> l ? ch[0] -> l : id);
90     r = (ch[1] -> r ? ch[1] -> r : id);
91 }
92 } null[maxn];
93
94 inline bool isroot(node *x) {
95     return x != x -> p -> ch[0] && x != x -> p ->
96         -> ch[1];
97 }
98
99 inline bool dir(node *x) {
100    return x == x -> p -> ch[1];
101 }
102
103 void init(node *x, int i) {
104     *x = node();
105     x -> ch[0] = x -> ch[1] = x -> p = null;
106     x -> size = 1;
107     x -> id = x -> l = x -> r = i;
108 }
109
110 void rot(node **x, int d) {
111     node *y = x -> ch[d ^ 1];
112
113     y -> p = x -> p;
114     if (!isroot(x))
115         x -> p -> ch[dir(x)] = y;
116
117     if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
118         y -> ch[d] -> p = x;
119     (y -> ch[d] = x) -> p = y;
120
121     x -> update();
122     y -> update();
123 }
124
125 void splay(node *x) {
126     x -> pushdown();
127
128     while (!isroot(x)) {
129         if (!isroot(x -> p))
130             x -> p -> p -> pushdown();
131         x -> p -> pushdown();
132         x -> pushdown();
133
134         if (isroot(x -> p)) {
135             rot(x -> p, dir(x) ^ 1);
136             break;
137         }
138
139         if (dir(x) == dir(x -> p))
140             rot(x -> p -> p, dir(x -> p) ^ 1);
141         else
142             rot(x -> p, dir(x) ^ 1);
143
144         rot(x -> p, dir(x) ^ 1);
145     }
146
147 void splay(node *x, node *rt) {
148     x -> pushdown();
149
150     while (x -> p != rt) {
151         if (x -> p -> p != rt)
152             x -> p -> p -> pushdown();
153
154         x -> pushdown();
155
156         if (x -> p -> p == rt) {
157             rot(x -> p, dir(x) ^ 1);
158             break;
159         }
160
161         if (dir(x) == dir(x -> p))
162             rot(x -> p -> p, dir(x -> p) ^ 1);
163         else
164             rot(x -> p, dir(x) ^ 1);
165
166         rot(x -> p, dir(x) ^ 1);
167     }
168 }
169
170 int val[maxn], par[maxn], go[maxn][26], sam_cnt,
171     -> sam_last;
172
173 node *access(node *x, int r) {
174     root[r] = root[r - 1];
175
176     node *y = null;
177
178     while (x != null) {
179         splay(x);
180
181         x -> pushdown();
182
183         x -> ch[1] = null;
184         x -> update();
185
186         if (x -> tim && val[x -> r]) { // last time
187             -> visited
188             int right = x -> tim, left = right - val[x
189                 -> -> r] + 1;
190             modify(left, val[x -> r], right + 1,
191                 -> root[r]);
192
193         }
194
195         x -> apply(r);
196         x -> pushdown();
197
198         x -> ch[1] = y;
199         (y = x) -> update();
200
201         x = x -> p;
202
203     }
204
205     return y;
206 }
207
208 void new_leaf(node *x, node *par) {
209     x -> p = par;
210 }
211
212 void new_node(node *x, node *y, node *par) {
213     splay(y);
214
215     if (isroot(y) && y -> p == par) {
216         assert(y -> ch[0] == null);
217
218         y -> ch[0] = x;
219         x -> p = y;
220         y -> update();
221
222     } else {
223         splay(par, y);
224     }
225 }

```

```

218     assert(y -> ch[0] == par);
219     assert(par -> ch[1] == null);
220     par -> ch[1] = x;
221     x -> p = par;
222
223     par -> update();
224     y -> update();
225 }
226
227 x -> tim = y -> tim;
228 }
229
230 void extend(int c) {
231     int p = sam_last, np = ++sam_cnt;
232     val[np] = val[p] + 1;
233
234     init(null + np, np);
235
236     while (p && !go[p][c]) {
237         go[p][c] = np;
238         p = par[p];
239     }
240
241     if (!p) {
242         par[np] = 1;
243         new_leaf(null + np, null + par[np]);
244     }
245     else {
246         int q = go[p][c];
247
248         if (val[q] == val[p] + 1) {
249             par[np] = q;
250             new_leaf(null + np, null + par[np]);
251         }
252         else {
253             int nq = ++sam_cnt;
254             val[nq] = val[p] + 1;
255             memcpy(go[nq], go[q], sizeof(go[q]));
256
257             init(null + nq, nq);
258
259             new_node(null + nq, null + q, null +
260                     -> par[q]);
261             new_leaf(null + np, null + nq);
262
263             par[nq] = par[q];
264             par[np] = par[q] = nq;
265
266             while (p && go[p][c] == q) {
267                 go[p][c] = nq;
268                 p = par[p];
269             }
270         }
271     }
272
273     sam_last = np;
274 }
275
276 char str[maxn];
277
278 int main() {
279     init(null, 0);
280
281     sam_last = sam_cnt = 1;
282     init(null + 1, 1);
283
284     int n, m;
285     scanf("%s%d", str + 1, &m);
286
287     n = strlen(str + 1);
288     N = n + m;
289
290     for (int i = 1; i <= n; i++) {
291         extend(str[i] - 'a');
292         access(null + sam_last, i);
293     }
294
295     int tmp = 0;
296
297     while (m--) {
298         int op;
299         scanf("%d", &op);
300
301         if (op == 1) {
302             scanf(" %c", &str[++n]);
303
304             str[n] = (str[n] - 'a' + tmp) % 26 + 'a';
305
306             extend(str[n] - 'a');
307             access(null + sam_last, n);
308         }
309         else {
310             int l, r;
311             scanf("%d%d", &l, &r);
312
313             l = (l - 1 + tmp) % n + 1;
314             r = (r - 1 + tmp) % n + 1;
315
316             printf("%d\n", tmp = query(l, r, root[r]));
317         }
318     }
319
320     return 0;
321 }

```

5.6 回文树

```

1 // 定理：一个字符串本质不同的回文子串个数是O(n) 的
2 // 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
3 // → 的bfs序
4
5 // 全局数组定义
6 int val[maxn], par[maxn], go[maxn][26], last, cnt;
7 char s[maxn];
8
9 // 重要！在主函数最前面一定要加上以下初始化
10 par[0] = cnt = 1;
11 val[1] = -1;
12 // 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
13 // → 树就不要乱搞了
14
15 // extend函数 均摊O(1)
16 // 向后扩展一个字符
17 // 传入对应下标
18 void extend(int n) {
19     int p = last, c = s[n] - 'a';
20     while (s[n - val[p] - 1] != s[n])
21         p = par[p];
22
23     if (!go[p][c]) {
24         int q = ++cnt, now = p;
25         val[q] = val[p] + 2;
26
27         do
28             p = par[p];
29         while (s[n - val[p] - 1] != s[n]);
30     }
31 }
32
33 // 从根节点开始遍历所有子节点
34 // 从左到右遍历子节点
35 void dfs(int p) {
36     for (int i = 0; i < 26; i++)
37         if (go[p][i])
38             dfs(go[p][i]);
39
40     for (int i = 0; i < 26; i++)
41         if (go[p][i])
42             val[go[p][i]] = val[p] + 1;
43
44     for (int i = 0; i < 26; i++)
45         if (go[p][i])
46             par[go[p][i]] = p;
47
48     for (int i = 0; i < 26; i++)
49         if (go[p][i])
50             go[p][i] = 0;
51 }
52
53 // 打印所有回文子串
54 void print() {
55     for (int i = 1; i <= n; i++)
56         if (val[i] > 1)
57             cout << s[i - val[i] + 1] << " ";
58 }
59
60 // 求以p为根的子树中所有回文子串的个数
61 int query(int l, int r, int p) {
62     if (l > r)
63         return 0;
64
65     if (l == r)
66         return 1;
67
68     if (p == 0)
69         return query(l, r, go[0][s[l] - 'a']);
70
71     if (p == 1)
72         return query(l, r, go[1][s[l] - 'a']);
73
74     int ans = 1;
75     for (int i = l; i <= r; i++)
76         if (s[i] == s[l])
77             ans += query(i, r, go[p][s[i] - 'a']);
78
79     return ans;
80 }
81
82 // 求以p为根的子树中所有回文子串的总长度
83 int sum(int l, int r, int p) {
84     if (l > r)
85         return 0;
86
87     if (l == r)
88         return 1;
89
90     if (p == 0)
91         return sum(l, r, go[0][s[l] - 'a']);
92
93     if (p == 1)
94         return sum(l, r, go[1][s[l] - 'a']);
95
96     int ans = 1;
97     for (int i = l; i <= r; i++)
98         if (s[i] == s[l])
99             ans += sum(i, r, go[p][s[i] - 'a']);
100
101    return ans;
102 }
103
104 // 求以p为根的子树中所有回文子串的总长度
105 int query2(int l, int r, int p) {
106     if (l > r)
107         return 0;
108
109     if (l == r)
110         return 1;
111
112     if (p == 0)
113         return query2(l, r, go[0][s[l] - 'a']);
114
115     if (p == 1)
116         return query2(l, r, go[1][s[l] - 'a']);
117
118     int ans = 1;
119     for (int i = l; i <= r; i++)
120         if (s[i] == s[l])
121             ans += query2(i, r, go[p][s[i] - 'a']);
122
123     return ans;
124 }
125
126 // 求以p为根的子树中所有回文子串的总长度
127 int sum2(int l, int r, int p) {
128     if (l > r)
129         return 0;
130
131     if (l == r)
132         return 1;
133
134     if (p == 0)
135         return sum2(l, r, go[0][s[l] - 'a']);
136
137     if (p == 1)
138         return sum2(l, r, go[1][s[l] - 'a']);
139
140     int ans = 1;
141     for (int i = l; i <= r; i++)
142         if (s[i] == s[l])
143             ans += sum2(i, r, go[p][s[i] - 'a']);
144
145     return ans;
146 }
147
148 // 求以p为根的子树中所有回文子串的总长度
149 int query3(int l, int r, int p) {
150     if (l > r)
151         return 0;
152
153     if (l == r)
154         return 1;
155
156     if (p == 0)
157         return query3(l, r, go[0][s[l] - 'a']);
158
159     if (p == 1)
160         return query3(l, r, go[1][s[l] - 'a']);
161
162     int ans = 1;
163     for (int i = l; i <= r; i++)
164         if (s[i] == s[l])
165             ans += query3(i, r, go[p][s[i] - 'a']);
166
167     return ans;
168 }
169
170 // 求以p为根的子树中所有回文子串的总长度
171 int sum3(int l, int r, int p) {
172     if (l > r)
173         return 0;
174
175     if (l == r)
176         return 1;
177
178     if (p == 0)
179         return sum3(l, r, go[0][s[l] - 'a']);
180
181     if (p == 1)
182         return sum3(l, r, go[1][s[l] - 'a']);
183
184     int ans = 1;
185     for (int i = l; i <= r; i++)
186         if (s[i] == s[l])
187             ans += sum3(i, r, go[p][s[i] - 'a']);
188
189     return ans;
190 }
191
192 // 求以p为根的子树中所有回文子串的总长度
193 int query4(int l, int r, int p) {
194     if (l > r)
195         return 0;
196
197     if (l == r)
198         return 1;
199
200     if (p == 0)
201         return query4(l, r, go[0][s[l] - 'a']);
202
203     if (p == 1)
204         return query4(l, r, go[1][s[l] - 'a']);
205
206     int ans = 1;
207     for (int i = l; i <= r; i++)
208         if (s[i] == s[l])
209             ans += query4(i, r, go[p][s[i] - 'a']);
210
211     return ans;
212 }
213
214 // 求以p为根的子树中所有回文子串的总长度
215 int sum4(int l, int r, int p) {
216     if (l > r)
217         return 0;
218
219     if (l == r)
220         return 1;
221
222     if (p == 0)
223         return sum4(l, r, go[0][s[l] - 'a']);
224
225     if (p == 1)
226         return sum4(l, r, go[1][s[l] - 'a']);
227
228     int ans = 1;
229     for (int i = l; i <= r; i++)
230         if (s[i] == s[l])
231             ans += sum4(i, r, go[p][s[i] - 'a']);
232
233     return ans;
234 }
235
236 // 求以p为根的子树中所有回文子串的总长度
237 int query5(int l, int r, int p) {
238     if (l > r)
239         return 0;
240
241     if (l == r)
242         return 1;
243
244     if (p == 0)
245         return query5(l, r, go[0][s[l] - 'a']);
246
247     if (p == 1)
248         return query5(l, r, go[1][s[l] - 'a']);
249
250     int ans = 1;
251     for (int i = l; i <= r; i++)
252         if (s[i] == s[l])
253             ans += query5(i, r, go[p][s[i] - 'a']);
254
255     return ans;
256 }
257
258 // 求以p为根的子树中所有回文子串的总长度
259 int sum5(int l, int r, int p) {
260     if (l > r)
261         return 0;
262
263     if (l == r)
264         return 1;
265
266     if (p == 0)
267         return sum5(l, r, go[0][s[l] - 'a']);
268
269     if (p == 1)
270         return sum5(l, r, go[1][s[l] - 'a']);
271
272     int ans = 1;
273     for (int i = l; i <= r; i++)
274         if (s[i] == s[l])
275             ans += sum5(i, r, go[p][s[i] - 'a']);
276
277     return ans;
278 }
279
280 // 求以p为根的子树中所有回文子串的总长度
281 int query6(int l, int r, int p) {
282     if (l > r)
283         return 0;
284
285     if (l == r)
286         return 1;
287
288     if (p == 0)
289         return query6(l, r, go[0][s[l] - 'a']);
290
291     if (p == 1)
292         return query6(l, r, go[1][s[l] - 'a']);
293
294     int ans = 1;
295     for (int i = l; i <= r; i++)
296         if (s[i] == s[l])
297             ans += query6(i, r, go[p][s[i] - 'a']);
298
299     return ans;
300 }
301
302 // 求以p为根的子树中所有回文子串的总长度
303 int sum6(int l, int r, int p) {
304     if (l > r)
305         return 0;
306
307     if (l == r)
308         return 1;
309
310     if (p == 0)
311         return sum6(l, r, go[0][s[l] - 'a']);
312
313     if (p == 1)
314         return sum6(l, r, go[1][s[l] - 'a']);
315
316     int ans = 1;
317     for (int i = l; i <= r; i++)
318         if (s[i] == s[l])
319             ans += sum6(i, r, go[p][s[i] - 'a']);
320
321     return ans;
322 }
323
324 // 求以p为根的子树中所有回文子串的总长度
325 int query7(int l, int r, int p) {
326     if (l > r)
327         return 0;
328
329     if (l == r)
330         return 1;
331
332     if (p == 0)
333         return query7(l, r, go[0][s[l] - 'a']);
334
335     if (p == 1)
336         return query7(l, r, go[1][s[l] - 'a']);
337
338     int ans = 1;
339     for (int i = l; i <= r; i++)
340         if (s[i] == s[l])
341             ans += query7(i, r, go[p][s[i] - 'a']);
342
343     return ans;
344 }
345
346 // 求以p为根的子树中所有回文子串的总长度
347 int sum7(int l, int r, int p) {
348     if (l > r)
349         return 0;
350
351     if (l == r)
352         return 1;
353
354     if (p == 0)
355         return sum7(l, r, go[0][s[l] - 'a']);
356
357     if (p == 1)
358         return sum7(l, r, go[1][s[l] - 'a']);
359
360     int ans = 1;
361     for (int i = l; i <= r; i++)
362         if (s[i] == s[l])
363             ans += sum7(i, r, go[p][s[i] - 'a']);
364
365     return ans;
366 }
367
368 // 求以p为根的子树中所有回文子串的总长度
369 int query8(int l, int r, int p) {
370     if (l > r)
371         return 0;
372
373     if (l == r)
374         return 1;
375
376     if (p == 0)
377         return query8(l, r, go[0][s[l] - 'a']);
378
379     if (p == 1)
380         return query8(l, r, go[1][s[l] - 'a']);
381
382     int ans = 1;
383     for (int i = l; i <= r; i++)
384         if (s[i] == s[l])
385             ans += query8(i, r, go[p][s[i] - 'a']);
386
387     return ans;
388 }
389
390 // 求以p为根的子树中所有回文子串的总长度
391 int sum8(int l, int r, int p) {
392     if (l > r)
393         return 0;
394
395     if (l == r)
396         return 1;
397
398     if (p == 0)
399         return sum8(l, r, go[0][s[l] - 'a']);
400
401     if (p == 1)
402         return sum8(l, r, go[1][s[l] - 'a']);
403
404     int ans = 1;
405     for (int i = l; i <= r; i++)
406         if (s[i] == s[l])
407             ans += sum8(i, r, go[p][s[i] - 'a']);
408
409     return ans;
410 }
411
412 // 求以p为根的子树中所有回文子串的总长度
413 int query9(int l, int r, int p) {
414     if (l > r)
415         return 0;
416
417     if (l == r)
418         return 1;
419
420     if (p == 0)
421         return query9(l, r, go[0][s[l] - 'a']);
422
423     if (p == 1)
424         return query9(l, r, go[1][s[l] - 'a']);
425
426     int ans = 1;
427     for (int i = l; i <= r; i++)
428         if (s[i] == s[l])
429             ans += query9(i, r, go[p][s[i] - 'a']);
430
431     return ans;
432 }
433
434 // 求以p为根的子树中所有回文子串的总长度
435 int sum9(int l, int r, int p) {
436     if (l > r)
437         return 0;
438
439     if (l == r)
440         return 1;
441
442     if (p == 0)
443         return sum9(l, r, go[0][s[l] - 'a']);
444
445     if (p == 1)
446         return sum9(l, r, go[1][s[l] - 'a']);
447
448     int ans = 1;
449     for (int i = l; i <= r; i++)
450         if (s[i] == s[l])
451             ans += sum9(i, r, go[p][s[i] - 'a']);
452
453     return ans;
454 }
455
456 // 求以p为根的子树中所有回文子串的总长度
457 int query10(int l, int r, int p) {
458     if (l > r)
459         return 0;
460
461     if (l == r)
462         return 1;
463
464     if (p == 0)
465         return query10(l, r, go[0][s[l] - 'a']);
466
467     if (p == 1)
468         return query10(l, r, go[1][s[l] - 'a']);
469
470     int ans = 1;
471     for (int i = l; i <= r; i++)
472         if (s[i] == s[l])
473             ans += query10(i, r, go[p][s[i] - 'a']);
474
475     return ans;
476 }
477
478 // 求以p为根的子树中所有回文子串的总长度
479 int sum10(int l, int r, int p) {
480     if (l > r)
481         return 0;
482
483     if (l == r)
484         return 1;
485
486     if (p == 0)
487         return sum10(l, r, go[0][s[l] - 'a']);
488
489     if (p == 1)
490         return sum10(l, r, go[1][s[l] - 'a']);
491
492     int ans = 1;
493     for (int i = l; i <= r; i++)
494         if (s[i] == s[l])
495             ans += sum10(i, r, go[p][s[i] - 'a']);
496
497     return ans;
498 }
499
500 // 求以p为根的子树中所有回文子串的总长度
501 int query11(int l, int r, int p) {
502     if (l > r)
503         return 0;
504
505     if (l == r)
506         return 1;
507
508     if (p == 0)
509         return query11(l, r, go[0][s[l] - 'a']);
510
511     if (p == 1)
512         return query11(l, r, go[1][s[l] - 'a']);
513
514     int ans = 1;
515     for (int i = l; i <= r; i++)
516         if (s[i] == s[l])
517             ans += query11(i, r, go[p][s[i] - 'a']);
518
519     return ans;
520 }
521
522 // 求以p为根的子树中所有回文子串的总长度
523 int sum11(int l, int r, int p) {
524     if (l > r)
525         return 0;
526
527     if (l == r)
528         return 1;
529
530     if (p == 0)
531         return sum11(l, r, go[0][s[l] - 'a']);
532
533     if (p == 1)
534         return sum11(l, r, go[1][s[l] - 'a']);
535
536     int ans = 1;
537     for (int i = l; i <= r; i++)
538         if (s[i] == s[l])
539             ans += sum11(i, r, go[p][s[i] - 'a']);
540
541     return ans;
542 }
543
544 // 求以p为根的子树中所有回文子串的总长度
545 int query12(int l, int r, int p) {
546     if (l > r)
547         return 0;
548
549     if (l == r)
550         return 1;
551
552     if (p == 0)
553         return query12(l, r, go[0][s[l] - 'a']);
554
555     if (p == 1)
556         return query12(l, r, go[1][s[l] - 'a']);
557
558     int ans = 1;
559     for (int i = l; i <= r; i++)
560         if (s[i] == s[l])
561             ans += query12(i, r, go[p][s[i] - 'a']);
562
563     return ans;
564 }
565
566 // 求以p为根的子树中所有回文子串的总长度
567 int sum12(int l, int r, int p) {
568     if (l > r)
569         return 0;
570
571     if (l == r)
572         return 1;
573
574     if (p == 0)
575         return sum12(l, r, go[0][s[l] - 'a']);
576
577     if (p == 1)
578         return sum12(l, r, go[1][s[l] - 'a']);
579
580     int ans = 1;
581     for (int i = l; i <= r; i++)
582         if (s[i] == s[l])
583             ans += sum12(i, r, go[p][s[i] - 'a']);
584
585     return ans;
586 }
587
588 // 求以p为根的子树中所有回文子串的总长度
589 int query13(int l, int r, int p) {
590     if (l > r)
591         return 0;
592
593     if (l == r)
594         return 1;
595
596     if (p == 0)
597         return query13(l, r, go[0][s[l] - 'a']);
598
599     if (p == 1)
600         return query13(l, r, go[1][s[l] - 'a']);
601
602     int ans = 1;
603     for (int i = l; i <= r; i++)
604         if (s[i] == s[l])
605             ans += query13(i, r, go[p][s[i] - 'a']);
606
607     return ans;
608 }
609
610 // 求以p为根的子树中所有回文子串的总长度
611 int sum13(int l, int r, int p) {
612     if (l > r)
613         return 0;
614
615     if (l == r)
616         return 1;
617
618     if (p == 0)
619         return sum13(l, r, go[0][s[l] - 'a']);
620
621     if (p == 1)
622         return sum13(l, r, go[1][s[l] - 'a']);
623
624     int ans = 1;
625     for (int i = l; i <= r; i++)
626         if (s[i] == s[l])
627             ans += sum13(i, r, go[p][s[i] - 'a']);
628
629     return ans;
630 }
631
632 // 求以p为根的子树中所有回文子串的总长度
633 int query14(int l, int r, int p) {
634     if (l > r)
635         return 0;
636
637     if (l == r)
638         return 1;
639
640     if (p == 0)
641         return query14(l, r, go[0][s[l] - 'a']);
642
643     if (p == 1)
644         return query14(l, r, go[1][s[l] - 'a']);
645
646     int ans = 1;
647     for (int i = l; i <= r; i++)
648         if (s[i] == s[l])
649             ans += query14(i, r, go[p][s[i] - 'a']);
650
651     return ans;
652 }
653
654 // 求以p为根的子树中所有回文子串的总长度
655 int sum14(int l, int r, int p) {
656     if (l > r)
657         return 0;
658
659     if (l == r)
660         return 1;
661
662     if (p == 0)
663         return sum14(l, r, go[0][s[l] - 'a']);
664
665     if (p == 1)
666         return sum14(l, r, go[1][s[l] - 'a']);
667
668     int ans = 1;
669     for (int i = l; i <= r; i++)
670         if (s[i] == s[l])
671             ans += sum14(i, r, go[p][s[i] - 'a']);
672
673     return ans;
674 }
675
676 // 求以p为根的子树中所有回文子串的总长度
677 int query15(int l, int r, int p) {
678     if (l > r)
679         return 0;
680
681     if (l == r)
682         return 1;
683
684     if (p == 0)
685         return query15(l, r, go[0][s[l] - 'a']);
686
687     if (p == 1)
688         return query15(l, r, go[1][s[l] - 'a']);
689
690     int ans = 1;
691     for (int i = l; i <= r; i++)
692         if (s[i] == s[l])
693             ans += query15(i, r, go[p][s[i] - 'a']);
694
695     return ans;
696 }
697
698 // 求以p为根的子树中所有回文子串的总长度
699 int sum15(int l, int r, int p) {
700     if (l > r)
701         return 0;
702
703     if (l == r)
704         return 1;
705
706     if (p == 0)
707         return sum15(l, r, go[0][s[l] - 'a']);
708
709     if (p == 1)
710         return sum15(l, r, go[1][s[l] - 'a']);
711
712     int ans = 1;
713     for (int i = l; i <= r; i++)
714         if (s[i] == s[l])
715             ans += sum15(i, r, go[p][s[i] - 'a']);
716
717     return ans;
718 }
719
720 // 求以p为根的子树中所有回文子串的总长度
721 int query16(int l, int r, int p) {
722     if (l > r)
723         return 0;
724
725     if (l == r)
726         return 1;
727
728     if (p == 0)
729         return query16(l, r, go[0][s[l] - 'a']);
730
731     if (p == 1)
732         return query16(l, r, go[1][s[l] - 'a']);
733
734     int ans = 1;
735     for (int i = l; i <= r; i++)
736         if (s[i] == s[l])
737             ans += query16(i, r, go[p][s[i] - 'a']);
738
739     return ans;
740 }
741
742 // 求以p为根的子树中所有回文子串的总长度
743 int sum16(int l, int r, int p) {
744     if (l > r)
745         return 0;
746
747     if (l == r)
748         return 1;
749
750     if (p == 0)
751         return sum16(l, r, go[0][s[l] - 'a']);
752
753     if (p == 1)
754         return sum16(l, r, go[1][s[l] - 'a']);
755
756     int ans = 1;
757     for (int i = l; i <= r; i++)
758         if (s[i] == s[l])
759             ans += sum16(i, r, go[p][s[i] - 'a']);
760
761     return ans;
762 }
763
764 // 求以p为根的子树中所有回文子串的总长度
765 int query17(int l, int r, int p) {
766     if (l > r)
767         return 0;
768
769     if (l == r)
770         return 1;
771
772     if (p == 0)
773         return query17(l, r, go[0][s[l] - 'a']);
774
775     if (p == 1)
776         return query17(l, r, go[1][s[l] - 'a']);
777
778     int ans = 1;
779     for (int i = l; i <= r; i++)
780         if (s[i] == s[l])
781             ans += query17(i, r, go[p][s[i] - 'a']);
782
783     return ans;
784 }
785
786 // 求以p为根的子树中所有回文子串的总长度
787 int sum17(int l, int r, int p) {
788     if (l > r)
789         return 0;
790
791     if (l == r)
792         return 1;
793
794     if (p == 0)
795         return sum17(l, r, go[0][s[l] - 'a']);
796
797     if (p == 1)
798         return sum17(l, r, go[1][s[l] - 'a']);
799
800     int ans = 1;
801     for (int i = l; i <= r; i++)
802         if (s[i] == s[l])
803             ans += sum17(i, r, go[p][s[i] - 'a']);
804
805     return ans;
806 }
807
808 // 求以p为根的子树中所有回文子串的总长度
809 int query18(int l, int r, int p) {
810     if (l > r)
811         return 0;
812
813     if (l == r)
814         return 1;
815
816     if (p == 0)
817         return query18(l, r, go[0][s[l] - 'a']);
818
819     if (p == 1)
820         return query18(l, r, go[1][s[l] - 'a']);
821
822     int ans = 1;
823     for (int i = l; i <= r; i++)
824         if (s[i] == s[l])
825             ans += query18(i, r, go[p][s[i] - 'a']);
826
827     return ans;
828 }
829
830 // 求以p为根的子树中所有回文子串的总长度
831 int sum18(int l, int r, int p) {
832     if (l > r)
833         return 0;
834
835     if (l == r)
836         return 1;
837
838     if (p == 0)
839         return sum18(l, r, go[0][s[l] - 'a']);
840
841     if (p == 1)
842         return sum18(l, r, go[1][s[l] - 'a']);
843
844     int ans = 1;
845     for (int i = l; i <= r; i++)
846         if (s[i] == s[l])
847             ans += sum18(i, r, go[p][s[i] - 'a']);
848
849     return ans;
850 }
851
852 // 求以p为根的子树中所有回文子串的总长度
853 int query19(int l, int r, int p) {
854     if (l > r)
855         return 0;
856
857     if (l == r)
858         return 1;
859
860     if (p == 0)
861         return query19(l, r, go[0][s[l] - 'a']);
862
863     if (p == 1)
864         return query19(l, r, go[1][s[l] - 'a']);
865
866     int ans = 1;
867     for (int i = l; i <= r; i++)
868         if (s[i] == s[l])
869             ans += query19(i, r, go[p][s[i] - 'a']);
870
871     return ans;
872 }
873
874 // 求以p为根的子树中所有回文子串的总长度
875 int sum19(int l, int r, int p) {
876     if (l > r)
877         return 0;
878
879     if (l == r)
880         return 1;
881
882     if (p == 0)
883         return sum19(l, r, go[0][s[l] - 'a']);
884
885     if (p == 1)
886         return sum19(l, r, go[1][s[l] - 'a']);
887
888     int ans = 1;
889     for (int i = l; i <= r; i++)
890         if (s[i] == s[l])
891             ans += sum19(i, r, go[p][s[i] - 'a']);
892
893     return ans;
894 }
895
896 // 求以p为根的子树中所有回文子串的总长度
```

```

29     par[q] = go[p][c];
30     last = go[now][c] = q;
31 }
32 else
33     last = go[p][c];
34
35 // a[last]++;
36 }
```

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)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 1000005, mod = 1000000007;
6
7 int val[maxn], par[maxn], go[maxn][26], fail[maxn][26],
8     → pam_last[maxn], pam_cnt;
9 int weight[maxn], pow_26[maxn];
10
11 int trie[maxn][26], trie_cnt, d[maxn], mxd[maxn],
12     → son[maxn], top[maxn], len[maxn], sum[maxn];
13 char chr[maxn];
14 int f[25][maxn], log_tbl[maxn];
15 vector<int> v[maxn];
16
17 vector<int> queries[maxn];
18
19 char str[maxn];
20 int n, m, ans[maxn];
21
22 int add(int x, int c) {
23     if (!trie[x][c]) {
24         trie[x][c] = ++trie_cnt;
25         f[0][trie[x][c]] = x;
26         chr[trie[x][c]] = c + 'a';
27     }
28
29     return trie[x][c];
30 }
31
32 int del(int x) {
33     return f[0][x];
34 }
35
36 void dfs1(int x) {
37     mxd[x] = d[x] = d[f[0][x]] + 1;
38
39     for (int i = 0; i < 26; i++) {
40         if (trie[x][i]) {
41             int y = trie[x][i];
42
43             mxd[x] = max(mxd[x], mxd[y]);
44             if (mxd[y] > mxd[son[x]])
45                 son[x] = y;
46         }
47     }
48
49 void dfs2(int x) {
50     if (x == son[f[0][x]])
51         top[x] = top[f[0][x]];
52     else
53         top[x] = x;
54
55     for (int i = 0; i < 26; i++)
56         if (trie[x][i]) {
```

```

57         int y = trie[x][i];
58         dfs2(y);
59     }
60
61     if (top[x] == x) {
62         int u = x;
63         while (top[son[u]] == x)
64             u = son[u];
65
66         len[x] = d[u] - d[x];
67
68         for (int i = 0; i < len[x]; i++) {
69             v[x].push_back(u);
70             u = f[0][u];
71         }
72
73         u = x;
74         for (int i = 0; i < len[x]; i++) { // 梯子剖分,
75             → 要延长一倍
76             v[x].push_back(u);
77             u = f[0][u];
78         }
79     }
80
81     int get_anc(int x, int k) {
82         if (!k)
83             return x;
84         if (k > d[x])
85             return 0;
86
87         x = f[log_tbl[k]][x];
88         k ^= 1 << log_tbl[k];
89
90         return v[top[x]][d[top[x]] + len[top[x]] - d[x] +
91             → k];
92     }
93
94     char get_char(int x, int k) { // 查询x前面k个的字符是哪
95         → 个
96         return chr[get_anc(x, k)];
97     }
98
99     int getfail(int x, int p) {
100         if (get_char(x, val[p] + 1) == chr[x])
101             return p;
102         return fail[p][chr[x] - 'a'];
103     }
104
105     int extend(int x) {
106
107         int p = pam_last[f[0][x]], c = chr[x] - 'a';
108
109         p = getfail(x, p);
110
111         if (!go[p][c]) {
112             int q = ++pam_cnt, now = p;
113             val[q] = val[p] + 2;
114
115             p = getfail(x, par[p]);
116
117             par[q] = go[p][c];
118             new_last = go[now][c] = q;
119
120             for (int i = 0; i < 26; i++)
121                 fail[q][i] = fail[par[q]][i];
122
123             if (get_char(x, val[par[q]]) >= 'a')
124                 fail[q][get_char(x, val[par[q]]) - 'a'] =
125                     → par[q];
126         }
127     }
128 }
```

```

125
126     if (val[q] <= n)
127         weight[q] = (weight[par[q]] + (Long long)(n
128             - val[q] + 1) * pow_26[n - val[q]]) %
129             mod;
130     else
131         weight[q] = weight[par[q]];
132 }
133 else
134     new_last = go[p][c];
135
136 pam_last[x] = new_last;
137 }
138
139 void bfs() {
140
141     queue<int> q;
142
143     q.push(1);
144
145     while (!q.empty()) {
146         int x = q.front();
147         q.pop();
148
149         sum[x] = sum[f[0][x]];
150         if (x > 1)
151             sum[x] = (sum[x] + extend(x)) % mod;
152
153         for (int i : queries[x])
154             ans[i] = sum[x];
155
156         for (int i = 0; i < 26; i++)
157             if (trie[x][i])
158                 q.push(trie[x][i]);
159     }
160 }
161
162 int main() {
163
164     pow_26[0] = 1;
165     log_tbl[0] = -1;
166
167     for (int i = 1; i <= 1000000; i++) {
168         pow_26[i] = 26ll * pow_26[i - 1] % mod;
169         log_tbl[i] = log_tbl[i / 2] + 1;
170     }
171
172     int T;
173     scanf("%d", &T);
174
175     while (T--) {
176         scanf("%d%d%s", &n, &m, str);
177
178         trie_cnt = 1;
179         chr[1] = '#';
180
181         int last = 1;
182         for (char *c = str; *c; c++)
183             last = add(last, *c - 'a');
184
185         queries[last].push_back(0);
186
187         for (int i = 1; i <= m; i++) {
188             int op;
189             scanf("%d", &op);
190
191             if (op == 1) {
192                 char c;
193                 scanf(" %c", &c);
194
195                     last = add(last, c - 'a');
196             }
197             else
198                 last = del(last);
199
200             queries[last].push_back(i);
201         }
202
203         dfs1(1);
204         dfs2(1);
205
206         for (int j = 1; j <= log_tbl[trie_cnt]; j++)
207             for (int i = 1; i <= trie_cnt; i++)
208                 f[j][i] = f[j - 1][f[j - 1][i]];
209
210         par[0] = pam_cnt = 1;
211
212         for (int i = 0; i < 26; i++)
213             fail[0][i] = fail[1][i] = 1;
214
215         val[1] = -1;
216         pam_last[1] = 1;
217
218         bfs();
219
220         for (int i = 0; i <= m; i++)
221             printf("%d\n", ans[i]);
222
223         for (int j = 0; j <= log_tbl[trie_cnt]; j++)
224             memset(f[j], 0, sizeof(f[j]));
225
226         for (int i = 1; i <= trie_cnt; i++) {
227             chr[i] = 0;
228             d[i] = mxd[i] = son[i] = top[i] = len[i] =
229                 ->pam_last[i] = sum[i] = 0;
230             v[i].clear();
231             queries[i].clear();
232
233             memset(trie[i], 0, sizeof(trie[i]));
234         }
235         trie_cnt = 0;
236
237         for (int i = 0; i <= pam_cnt; i++) {
238             val[i] = par[i] = weight[i];
239
240             memset(go[i], 0, sizeof(go[i]));
241             memset(fail[i], 0, sizeof(fail[i]));
242         }
243         pam_cnt = 0;
244
245     }
246
247     return 0;
248 }
249 }
```

5.7 Manacher 马拉车

```

1 // n为串长，回文半径输出到p数组中
2 // 数组要开串长的两倍
3 void manacher(const char *t, int n) {
4     static char s[maxn * 2];
5
6     for (int i = n; i--)
7         s[i * 2] = t[i];
8     for (int i = 0; i <= n; i++)
9         s[i * 2 + 1] = '#';
10
11    s[0] = '$';
12    s[(n + 1) * 2] = '\0';

```

```

13     n = n * 2 + 1;
14
15     int mx = 0, j = 0;
16
17     for (int i = 1; i <= n; i++) {
18         p[i] = (mx > i ? min(p[j * 2 - i], mx - i) :
19             → 1);
20         while (s[i - p[i]] == s[i + p[i]])
21             p[i]++;
22
23         if (i + p[i] > mx) {
24             mx = i + p[i];
25             j = i;
26         }
27     }

```

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 \geq 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 $O(n \log n)$

```

1 int a[maxn], q[maxn], p[maxn], g[maxn]; // 存左端点, 右
2   端点就是下一个左端点 - 1
3
4 long long f[maxn], s[maxn];
5
6 int n, m;
7
8 long long calc(int l, int r) {
9     if (r < l)
10        return 0;
11
12    int mid = (l + r) / 2;
13    if ((r - l + 1) % 2 == 0)
14        return (s[r] - s[mid]) - (s[mid] - s[l - 1]);
15    else
16        return (s[r] - s[mid]) - (s[mid - 1] - s[l -
17           - 1]);
18
19 int solve(long long tmp) {
20     memset(f, 63, sizeof(f));
21     f[0] = 0;
22
23     int head = 1, tail = 0;
24
25     for (int i = 1; i <= n; i++) {
26         f[i] = calc(1, i);
27         g[i] = 1;
28
29         while (head < tail && p[head + 1] <= i)
30             head++;
31         if (head <= tail) {
32             if (f[q[head]] + calc(q[head] + 1, i) <
33                 -> f[i]) {
34                 f[i] = f[q[head]] + calc(q[head] + 1,
35                   -> i);
36                 g[i] = g[q[head]] + 1;
37             }
38             while (head < tail && p[head + 1] <= i + 1)
39                 head++;
40             if (head <= tail)
41                 p[head] = i + 1;
42         }
43         f[i] += tmp;
44
45         int r = n;
46
47         while (head <= tail) {
48             if (f[q[tail]] + calc(q[tail] + 1, p[tail])
49                 -> > f[i] + calc(i + 1, p[tail])) {
50                 r = p[tail] - 1;
51                 tail--;
52             }
53             else if (f[q[tail]] + calc(q[tail] + 1, r)
54                 -> <= f[i] + calc(i + 1, r)) {
55                 if (r < n) {
56                     q[++tail] = i;
57                     p[tail] = r + 1;
58                 }
59                 break;
60             }
61             else {
62                 int L = p[tail], R = r;
63                 while (L < R) {
64                     int M = (L + R) / 2;
65
66                     if (f[q[tail]] + calc(q[tail] + 1,
67                         -> M) <= f[i] + calc(i + 1, M))
68                         L = M + 1;
69                     else
70                         R = M;
71                 }
72             }
73             if (head > tail) {
74                 q[++tail] = i;
75                 p[tail] = i + 1;
76             }
77         }
78
79         return g[n];
80     }

```

```

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

6.2.1 103388A Assigning Prizes 容斥

题意 给定一个长为 n 的序列 a_i , 要求构造非严格递减序列 b_i , 满足 $a_i \leq b_i \leq R$, 求方案数. $n \leq 5 \times 10^3, R, a_i \leq 10^9$.

做法 a_i 的范围太大了, 不能简单地记录上一位的值.

考虑使用容斥. 方便起见把 a_i 直接变成 $R - a_i + 1$, 条件就变成了 $b_i \leq a_i$ 且 $b_i \geq b_{i-1}$.

这里有两个限制条件, 可以固定 $b_i \leq a_i$ 是必须满足的条件, 只对 $b_i \geq b_{i-1}$ 使用容斥, 枚举哪些位置是比上一位小的 (违反限制), 其他位置随意.

枚举后的形态一定是有若干个区间是严格递减的, 其他位置随意. 考虑如果一个区间 $[l, r]$ 是严格递减的, 显然所有的数都 $< a_l$, 所以这段区间的方案数就是 $\binom{a_l}{r-l+1}$. 另外实际上 b_l 是没有违反限制的, 所以这里对系数的贡献是 $(-1)^{r-l}$.

考虑令 dp_i 表示只考虑前 i 个位置的答案, 转移时自然就是枚举一个 j , 然后计算 dp_{j-1} 乘上区间 $[j, i]$ 严格递减的方案数. 另外还有一种情况是 b_i 没有违反限制, 这时显然直接在 dp_{i-1} 的基础上乘上一个 a_i 就好了. (转移时还要注意, 由于枚举的是严格递减区间, 自然就不能枚举只有一个数的区间.)

```

1 constexpr int maxn = 5005, p = (int)1e9 + 7;
2
3 int inv[maxn];
4 int a[maxn], f[maxn][maxn], dp[maxn];
5
6 int main() {
7
8     int n, m;
9     scanf("%d%d", &n, &m);
10
11    inv[1] = 1;
12    for (int i = 2; i <= n; i++)
13        inv[i] = (long long)(p - p / i) * inv[p % i] %
14            -> p;
15
16    for (int i = 1; i <= n; i++) {
17        scanf("%d", &a[i]);
18        a[i] = m - a[i] + 1;
19    }
20
21    if (any_of(a + 1, a + n + 1, [] (int x) {return x
22        -> <= 0;})) {
23        printf("0\n");
24        return 0;
25    }
26
27    int ans = 0;
28
29    for (int i = 0; i < n; i++) {
30        for (int j = i + 1; j <= n; j++) {
31            int sum = 0;
32            for (int k = i; k < j; k++)
33                sum += a[k];
34
35            if (sum <= m) {
36                int val = 1;
37                for (int l = i + 1; l < j; l++)
38                    val *= inv[l];
39
40                ans += val;
41            }
42        }
43    }
44
45    cout << ans;
46
47    return 0;
48}

```

```
23 }
24
25 for (int i = n - 1; i; i--)
|   a[i] = min(a[i], a[i + 1]);
26
27 // b_i >= b_{i - 1} && b_i <= a_i
28 // 我们可以假设 b_i <= a_i 是必定被满足的, 然后对 bi
|   ↪ 非严格递增的条件进行容斥, 枚举某一段是严格递减的
29 // 如果 [j, i] 严格递减, 显然它们都 <= a_j, 所以这个
|   ↪ 区间的方案数是 {a_j \choose i - j + 1}
30 // 如果 i 是合法的, 直接一个个转移即可, 因为这一部分的
|   ↪ 转移和区间长度没有关系
31
32 for (int i = 1; i <= n; i++) {
|   f[i][0] = 1;
33
34     for (int j = 1; j <= n - i + 1 && j <= a[i]; j+
|       ↪ +)
|       |   f[i][j] = (long long)f[i][j - 1] * (a[i] -
|         ↪ j + 1) % p * inv[j] % p;
35   }
36
37 dp[0] = 1;
38
39 for (int i = 1; i <= n; i++) {
40   dp[i] = (long long)dp[i - 1] * a[i] % p;
41
42     for (int j = 1; j < i; j++) {
43       int tmp = (long long)dp[j - 1] * f[j][i - j
|         ↪ + 1] % p;
44
45       if ((i - j) % 2)
|         tmp = p - tmp;
46
47       dp[i] = (dp[i] + tmp) % p;
48     }
49   }
50
51 print("%d\n", dp[n]);
52
53 return 0;
54
55 }
```

7 计算几何

7.1 Delaunay 三角剖分

只要两个点同在某个三角形上，它们就互为一对最近点。注意返回的三角形似乎不保证顺序，所以要加边的话还是要加双向边。

如果要建 V 图的话求出每个三角形的外心就行了，每个点控制的区域就是所在三角形的外心连起来。

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 500005;
6
7 using ll = long long;
8
9 constexpr int INF = 0x3f3f3f3f;
10 constexpr ll LINF = 0x3f3f3f3f3f3f3f3fll;
11 constexpr double eps = 1e-8;
12
13 template <class T>
14 int sgn(T x) {
15     return x > 0 ? 1 : x < 0 ? -1 : 0;
16 }
17
18 struct point {
19     ll x, y;
20
21     point() = default;
22
23     point(ll x, ll y) : x(x), y(y) {}
24
25     point operator - (const point &p) const {
26         return point(x - p.x, y - p.y);
27     }
28
29     ll cross(const point &p) const {
30         return x * p.y - y * p.x;
31     }
32
33     ll cross(const point &a, const point &b) const {
34         return (a - *this).cross(b - *this);
35     }
36
37     ll dot(const point &p) const {
38         return x * p.x + y * p.y;
39     }
40
41     ll dot(const point &a, const point &b) const {
42         return (a - *this).dot(b - *this);
43     }
44
45     ll abs2() const {
46         return this -> dot(*this);
47     }
48
49     bool operator == (const point &p) const {
50         return x == p.x && y == p.y;
51     }
52
53     bool operator < (const point &p) const {
54         if (x != p.x) return x < p.x;
55         return y < p.y;
56     }
57 };
58
59 const point inf_point = point(1e18, 1e18);

```

```

62
63     struct quad_edge {
64         point origin;
65         quad_edge *rot = nullptr;
66         quad_edge *onext = nullptr;
67         bool used = false;
68
69         quad_edge *rev() const {
70             return rot -> rot;
71         }
72         quad_edge *lnext() const {
73             return rot -> rev() -> onext -> rot;
74         }
75         quad_edge *oprev() const {
76             return rot -> onext -> rot;
77         }
78         point dest() const {
79             return rev() -> origin;
80         }
81     };
82
83     quad_edge *make_edge(point from, point to) {
84         quad_edge *e1 = new quad_edge;
85         quad_edge *e2 = new quad_edge;
86         quad_edge *e3 = new quad_edge;
87         quad_edge *e4 = new quad_edge;
88
89         e1 -> origin = from;
90         e2 -> origin = to;
91         e3 -> origin = e4 -> origin = inf_point;
92
93         e1 -> rot = e3;
94         e2 -> rot = e4;
95         e3 -> rot = e2;
96         e4 -> rot = e1;
97
98         e1 -> onext = e1;
99         e2 -> onext = e2;
100        e3 -> onext = e4;
101        e4 -> onext = e3;
102
103        return e1;
104    }
105
106    void splice(quad_edge *a, quad_edge *b) { // 拼接
107        swap(a -> onext -> rot -> onext, b -> onext -> rot
108            -> onext);
109        swap(a -> onext, b -> onext);
110    }
111
112    void delete_edge(quad_edge *e) {
113        splice(e, e -> oprev());
114        splice(e -> rev(), e -> rev() -> oprev());
115
116        delete e -> rev() -> rot;
117        delete e -> rev();
118        delete e -> rot;
119        delete e;
120    }
121
122    quad_edge *connect(quad_edge *a, quad_edge *b) {
123        quad_edge *e = make_edge(a -> dest(), b -> origin);
124
125        splice(e, a -> lnext());
126        splice(e -> rev(), b);
127
128        return e;
129    }
130
131    bool left_of(point p, quad_edge *e) {

```

```

131     return p.cross(e -> origin, e -> dest()) > 0;
132 }
133
134 bool right_of(point p, quad_edge *e) {
135     return p.cross(e -> origin, e -> dest()) < 0;
136 }
137
138 template <class T>
139 T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2,
140    → T c3) {
141     return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 -
142        → c1 * b3) +
143            a3 * (b1 * c2 - c1 * b2);
144 }
145
146 bool in_circle(point a, point b, point c, point d) { // 如果有__int128就直接计算行列式, 否则算角度
147 #if defined(__LP64__) || defined(__WIN64__)
148     __int128 det = -det3<__int128>(b.x, b.y, b.abs2(),
149        → c.x, c.y, c.abs2(), d.x, d.y, d.abs2());
150     det += det3<__int128>(a.x, a.y, a.abs2(), c.x, c.y
151        → c.abs2(), d.x, d.y, d.abs2());
152     det -= det3<__int128>(a.x, a.y, a.abs2(), b.x, b.y
153        → b.abs2(), d.x, d.y, d.abs2());
154     det += det3<__int128>(a.x, a.y, a.abs2(), b.x, b.y
155        → b.abs2(), c.x, c.y, c.abs2());
156
157     return det > 0;
158 #else
159     auto ang = [] (point l, point mid, point r) {
160         ll x = mid.dot(l, r);
161         ll y = mid.cross(l, r);
162         long double res = atan2((long double)x, (long
163             → double)y);
164         return res;
165     };
166
167     long double kek = ang(a, b, c) + ang(c, d, a) -
168        → ang(b, c, d) - ang(d, a, b);
169
170     return kek > 1e-10;
171 #endif
172 }
173
174 pair<quad_edge*, quad_edge*> divide_and_conquer(int l,
175    → int r, vector<point> &p) {
176     if (r - l + 1 == 2) {
177         quad_edge *res = make_edge(p[l], p[r]);
178         return make_pair(res, res -> rev());
179     }
180
181     if (r - l + 1 == 3) {
182         quad_edge *a = make_edge(p[l], p[l + 1]), *b =
183             → make_edge(p[l + 1], p[r]);
184         splice(a -> rev(), b);
185
186         int sg = sgn(p[l].cross(p[l + 1], p[r]));
187
188         if (sg == 0)
189             return make_pair(a, b -> rev());
190
191         quad_edge *c = connect(b, a);
192
193         if (sg == 1)
194             return make_pair(a, b -> rev());
195         else
196             return make_pair(c -> rev(), c);
197     }
198
199     int mid = (l + r) / 2;

```

```

190
191     quad_edge *ldo, *ldi, *rdo, *rdi;
192     tie(ldo, ldi) = divide_and_conquer(l, mid, p);
193     tie(rdi, rdo) = divide_and_conquer(mid + 1, r, p);
194
195     while (true) {
196         if (left_of(rdi -> origin, ldi))
197             ldi = ldi -> lnext();
198
199         else if (right_of(ldi -> origin, rdi))
200             rdi = rdi -> rev() -> onext;
201
202         else
203             break;
204     }
205
206     quad_edge *basel = connect(rdi -> rev(), ldi);
207     auto is_valid = [&basel] (quad_edge *e) {
208         return right_of(e -> dest(), basel);
209     };
210
211     if (ldi -> origin == ldo -> origin)
212         ldo = basel -> rev();
213     if (rdi -> origin == rdo -> origin)
214         rdo = basel;
215
216     while (true) {
217         quad_edge *lcand = basel -> rev() -> onext;
218         if (is_valid(lcand)) {
219             while (in_circle(basel -> dest(), basel ->
220                               -> origin, lcand -> dest(), lcand -> onext
221                               ->-> dest())) {
222                 quad_edge *t = lcand -> onext;
223                 delete_edge(lcand);
224                 lcand = t;
225             }
226         }
227
228         quad_edge *rcand = basel -> oprev();
229         if (is_valid(rcand)) {
230             while (in_circle(basel -> dest(), basel ->
231                               -> origin, rcand -> dest(), rcand ->
232                               -> oprev() -> dest())) {
233                 quad_edge *t = rcand -> oprev();
234                 delete_edge(rcand);
235                 rcand = t;
236             }
237
238         if (!is_valid(lcand) && !is_valid(rcand))
239             break;
240
241         if (!is_valid(lcand) || (is_valid(rcand) &&
242                               -> in_circle(lcand -> dest(), lcand -> origin,
243                               -> rcand -> origin, rcand -> dest())))
244             basel = connect(rcand, basel -> rev());
245         else
246             basel = connect(basel -> rev(), lcand ->
247                             -> rev());
248     }
249
250     return make_pair(ldo, rdo);
251 }
252
253 vector<tuple<point, point, point> >
254     delaunay(vector<point> p) { // Delaunay 三角剖分
255     sort(p.begin(), p.end(), [] (const point &a, const
256                               point &b) {

```

```

249     |     return a.x < b.x || (a.x == b.x && a.y < b.y);
250     |     ↪ // 实际上已经重载小于了，只是为了清晰
251     |
252     auto res = divide_and_conquer(0, (int)p.size() - 1,
253                                    ↪ p);
254
255     quad_edge *e = res.first;
256     vector<quad_edge*> edges = {e};
257
258     while (e->onext->dest().cross(e->dest(), e->
259                                      ↪ origin) < 0)
260         e = e->onext;
261
262     auto add = [&p, &e, &edges] () { // 修改 p, e, edges
263         quad_edge *cur = e;
264         do {
265             cur->used = true;
266             p.push_back(cur->origin);
267             edges.push_back(cur->rev());
268
269             cur = cur->lnext();
270         } while (cur != e);
271     };
272
273     add();
274     p.clear();
275
276     int kek = 0;
277     while (kek < (int)edges.size())
278         if (!(*e = edges[kek++])->used)
279             add();
280
281     vector<tuple<point, point, point>> ans;
282     for (int i = 0; i < (int)p.size(); i += 3)
283         ans.push_back(make_tuple(p[i], p[i + 1], p[i +
284                                   ↪ 2]));
285
286     #define sq(x) ((x) * (ll)(x)) // 平方
287
288     ll dist(point p, point q) { // 两点间距离的平方
289         return (p - q).abs2();
290     }
291
292     ll sarea2(point p, point q, point r) { // 三角形面积的两
293         ↪ 倍(叉积)
294         return (q - p).cross(r - q);
295     }
296
297     point v[maxn];
298
299     int main() {
300         int n;
301         cin >> n;
302
303         // read the points, v[1 ~ n]
304
305         bool col_linear = true; // 如果给出的所有点都共线则
306         ↪ 需要特判
307         for (int i = 3; i <= n; i++)
308             if (sarea2(v[1], v[2], v[i]))
309                 col_linear = false;
310
311         if (col_linear) {
312             // do something
313             return 0;
314         }
315
316         auto triangles = delaunay(vector<point>(v + 1, v +
317                                         ↪ n + 1));
318
319         // do something
320
321     }
322 }
```

7.2 最近点对

首先分治的做法是众所周知的.

有期望 $O(n)$ 的随机增量法: 首先将所有点随机打乱, 然后每次增加一个点, 更新答案.

假设当前最近点对距离为 s , 则把平面划分成 $s \times s$ 的方格, 用哈希表存储每个方格有哪些点.

加入一个新点时只需要枚举自身和周围共计 9 个方格中的点, 显然枚举到的点最多 16 个. 如果加入之后答案变小了就 $O(n)$ 暴力重构. 前 i 个点中 i 是最近点对中的点的概率至多为 $\frac{2}{i}$, 所以每个点的期望贡献都是 $O(1)$, 总的复杂度就是期望 $O(n)$.

如果对每个点都要求出距离最近的点的话, 也有随机化的 $O(n)$ 做法:

一个真的随机算法:

A simple randomized sieve algorithm for the closest-pair problem (<https://www.cs.umd.edu/~samir/grant/cp.pdf>)

1. 循环直到删完所有点:

- 随机选一个点, 计算它到所有点的最短距离 d .
- 将所有点划分到 $l = d/3$ 的网格里, 比如 $(\lfloor \frac{x}{l} \rfloor, \lfloor \frac{y}{l} \rfloor)$.
- 将九宫格内孤立的点删除, 这意味着这些点的最近点对距离不小于 $\frac{2\sqrt{2}}{3}d$, 其中 $\frac{2\sqrt{2}}{3} < 1$.

2. 取最后一个 d , 将所有点划分到 $(\lfloor \frac{x}{d} \rfloor, \lfloor \frac{y}{d} \rfloor)$ 的网格里, 暴力计算九宫格内的答案.

第一部分每次期望会删掉至少一半的点, 因为有 $\geq 1/2$ 概率碰到一个最近点距离在中位数以下的点, 因此第一部分的复杂度是 $O(n)$ 的.

第二部分分析类似分治做法, 周围只有常数个点.

所以总复杂度是 $O(n)$ 的.

8 杂项

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.

```

1 // long double 快速乘
2 // 在两数直接相乘会爆long long时才有必要使用
3 // 常数比直接long long乘法 + 取模大很多，非必要时不建议使用
4 long long mul(long long a, long long b, long long p) {
5     a %= p;
6     b %= p;
7     return ((a * b - p * (long long)((long double)a / p
8         - * b + 0.5)) % p + p) % p;
9 }
10
11 // 指令集快速乘
12 // 试机记得测试能不能过编译
13 inline long long mul(const long long a, const long long
14     -> b, const long long p) {
15     long long ans;
16     __asm__ __volatile__ ("\\tmulq %%rbx\\n\\tdivq
17     -> %%rcx\\n" : "=d"(ans) : "a"(a), "b"(b), "c"(p));
18     return ans;
19 }
20
21 // int乘法取模，大概比直接做快一倍
22 inline int mul_mod(int a, int b, int p) {
23     int ans;
24     __asm__ __volatile__ ("\\tmull %%ebx\\n\\tdivl
25     -> %%ecx\\n" : "=d"(ans) : "a"(a), "b"(b), "c"(p));
26     return ans;
27 }
```

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.

```

1 double kahanSum(vector<double> vec) {
2     double sum = 0, c = 0;
3     for (auto x : vec) {
4         double y = x - c;
5         double t = sum + y;
6         c = (t - sum) - y;
7         sum = t;
8     }
9     return sum;
10 }
```

8.3 Python Decimal

```

1 import decimal
2
3 decimal.getcontext().prec = 1234 # 有效数字位数
4
5 x = decimal.Decimal(2)
6 x = decimal.Decimal('50.5679') # 不要用float，因为float本身就不准确
7
8 x = decimal.Decimal('50.5679'). \
9     quantize(decimal.Decimal('0.00')) # 保留两位小数,
10    -> 50.57
11 x = decimal.Decimal('50.5679'). \
12     quantize(decimal.Decimal('0.00'),
13     -> decimal.ROUND_HALF_UP) # 四舍五入
# 第二个参数可选如下:
# ROUND_HALF_UP 四舍五入
```

```

14 # ROUND_HALF_DOWN 五舍六入
15 # ROUND_HALF_EVEN 银行家舍入法，舍入到最近的偶数
16 # ROUND_UP 向绝对值大的取整
17 # ROUND_DOWN 向绝对值小的取整
18 # ROUND_CEILING 向正无穷取整
19 # ROUND_FLOOR 向负无穷取整
20 # ROUND_05UP (away from zero if last digit after
21     -> rounding towards zero would have been 0 or 5;
22     -> otherwise towards zero)
23
24 print('%f' % x) # 这样做只有float的精度
25 s = str(x)
26
27 decimal.is_finite(x) # x是否有穷 (NaN也算)
28 decimal.is_infinite(x)
29 decimal.is_nan(x)
30 decimal.is_normal(x) # x是否正常
31 decimal.is_signed(x) # 是否为负数
32
33 decimal.fma(a, b, c) # a * b + c, 精度更高
34
35 x.exp(), x.ln(), x.sqrt(), x.log10()
# 可以转复数，前提是import complex
```

8.4 $O(n^2)$ 高精度

```

1 // 注意如果只需要正数运算的话
2 // 可以只抄英文名的运算函数
3 // 按需自取
4 // 乘法O(n ^ 2), 除法O(10 * n ^ 2)
5
6 constexpr int maxn = 1005;
7
8 struct big_decimal {
9     int a[maxn];
10    -> bool negative;
11
12    big_decimal() {
13        memset(a, 0, sizeof(a));
14        negative = false;
15    }
16
17    big_decimal(long long x) {
18        memset(a, 0, sizeof(a));
19        negative = false;
20
21        if (x < 0) {
22            negative = true;
23            x = -x;
24        }
25
26        while (x) {
27            a[++a[0]] = x % 10;
28            x /= 10;
29        }
30    }
31
32    big_decimal(string s) {
33        memset(a, 0, sizeof(a));
34        negative = false;
35
36        if (s == "") {
37            return;
38        }
39        if (s[0] == '-') {
40            negative = true;
41            s = s.substr(1);
42        }
43        a[0] = s.size();
```

```

44     for (int i = 1; i <= a[0]; i++)
45         a[i] = s[a[0] - i] - '0';
46
47     while (a[0] && !a[a[0]])
48         a[0]--;
49 }
50
51 void input() {
52     string s;
53     cin >> s;
54     *this = s;
55 }
56
57 string str() const {
58     if (!a[0])
59         return "0";
60
61     string s;
62     if (negative)
63         s = "-";
64
65     for (int i = a[0]; i; i--)
66         s.push_back('0' + a[i]);
67
68     return s;
69 }
70
71 operator string () const {
72     return str();
73 }
74
75 big_decimal operator - () const {
76     big_decimal o = *this;
77     if (a[0])
78         o.negative ^= true;
79
80     return o;
81 }
82
83 friend big_decimal abs(const big_decimal &u) {
84     big_decimal o = u;
85     o.negative = false;
86     return o;
87 }
88
89 big_decimal &operator <= (int k) {
90     a[0] += k;
91
92     for (int i = a[0]; i > k; i--)
93         a[i] = a[i - k];
94
95     for (int i = k; i; i--)
96         a[i] = 0;
97
98     return *this;
99 }
100
101 friend big_decimal operator << (const big_decimal
102     &u, int k) {
103     big_decimal o = u;
104     return o <= k;
105 }
106
107 big_decimal &operator >= (int k) {
108     if (a[0] < k)
109         return *this = big_decimal(0);
110
111     a[0] -= k;
112     for (int i = 1; i <= a[0]; i++)
113         a[i] = a[i + k];
114
115     for (int i = a[0] + 1; i <= a[0] + k; i++)
116         a[i] = 0;
117
118     return *this;
119 }
120
121 friend big_decimal operator >> (const big_decimal
122     &u, int k) {
123     big_decimal o = u;
124     return o >= k;
125 }
126
127 friend int cmp(const big_decimal &u, const
128     &big_decimal &v) {
129     if (u.negative || v.negative) {
130         if (u.negative && v.negative)
131             return -cmp(-u, -v);
132
133         if (u.negative)
134             return -1;
135
136         if (v.negative)
137             return 1;
138
139     if (u.a[0] != v.a[0])
140         return u.a[0] < v.a[0] ? -1 : 1;
141
142     for (int i = u.a[0]; i; i--)
143         if (u.a[i] != v.a[i])
144             return u.a[i] < v.a[i] ? -1 : 1;
145
146     return 0;
147 }
148
149 friend bool operator < (const big_decimal &u, const
150     &big_decimal &v) {
151     return cmp(u, v) == -1;
152 }
153
154 friend bool operator > (const big_decimal &u, const
155     &big_decimal &v) {
156     return cmp(u, v) == 1;
157 }
158
159 friend bool operator == (const big_decimal &u,
160     &const big_decimal &v) {
161     return cmp(u, v) == 0;
162 }
163
164 friend bool operator <= (const big_decimal &u,
165     &const big_decimal &v) {
166     return cmp(u, v) <= 0;
167 }
168
169 friend bool operator >= (const big_decimal &u,
170     &const big_decimal &v) {
171     return cmp(u, v) >= 0;
172 }
173
174 friend big_decimal decimal_plus(const big_decimal
175     &u, const big_decimal &v) { // 保证u, v均为正数
176     big_decimal o;
177
178     o.a[0] = max(u.a[0], v.a[0]);
179 }
```

```

172     for (int i = 1; i <= u.a[0] || i <= v.a[0]; i++)
173         o.a[i] += u.a[i] + v.a[i];
174
175     if (o.a[i] >= 10) {
176         o.a[i + 1]++;
177         o.a[i] -= 10;
178     }
179 }
180
181 if (o.a[o.a[0] + 1])
182     o.a[0]++;
183
184 return o;
185 }

186 friend big_decimal decimal_minus(const big_decimal
187     &u, const big_decimal &v) { // 保证u, v均为正数
188     // 的话可以直接调用
189     int k = cmp(u, v);
190
191     if (k == -1)
192         return -decimal_minus(v, u);
193     else if (k == 0)
194         return big_decimal(0);
195
196     big_decimal o;
197
198     o.a[0] = u.a[0];
199
200     for (int i = 1; i <= u.a[0]; i++) {
201         o.a[i] += u.a[i] - v.a[i];
202
203         if (o.a[i] < 0) {
204             o.a[i] += 10;
205             o.a[i + 1]--;
206         }
207     }
208
209     while (o.a[0] && !o.a[o.a[0]])
210         o.a[0]--;
211
212     return o;
213 }

214 friend big_decimal decimal_multi(const big_decimal
215     &u, const big_decimal &v) {
216     big_decimal o;
217
218     o.a[0] = u.a[0] + v.a[0] - 1;
219
220     for (int i = 1; i <= u.a[0]; i++)
221         for (int j = 1; j <= v.a[0]; j++)
222             o.a[i + j - 1] += u.a[i] * v.a[j];
223
224     for (int i = 1; i <= o.a[0]; i++)
225         if (o.a[i] >= 10) {
226             o.a[i + 1] += o.a[i] / 10;
227             o.a[i] %= 10;
228         }
229
230         if (o.a[o.a[0] + 1])
231             o.a[0]++;
232
233     return o;
234 }

235 friend pair<big_decimal, big_decimal>
236     decimal_divide(big_decimal u, big_decimal v) {
237     // 整除
238
239     if (v > u)
240         return make_pair(big_decimal(0), u);
241
242     big_decimal o;
243     o.a[0] = u.a[0] - v.a[0] + 1;
244
245     int m = v.a[0];
246     v <= u.a[0] - m;
247
248     for (int i = u.a[0]; i >= m; i--) {
249         while (u >= v) {
250             u = u - v;
251             o.a[i - m + 1]++;
252         }
253
254         v >>= 1;
255     }
256
257     while (o.a[0] && !o.a[o.a[0]])
258         o.a[0]--;
259
260     return make_pair(o, u);
261 }

262 friend big_decimal operator + (const big_decimal
263     &u, const big_decimal &v) {
264     if (u.negative || v.negative) {
265         if (u.negative && v.negative)
266             return -decimal_plus(-u, -v);
267
268         if (u.negative)
269             return v - (-u);
270     }
271
272     return decimal_plus(u, v);
273 }

274 friend big_decimal operator - (const big_decimal
275     &u, const big_decimal &v) {
276     if (u.negative || v.negative) {
277         if (u.negative && v.negative)
278             return -decimal_minus(-u, -v);
279
280         if (u.negative)
281             return -decimal_plus(-u, v);
282
283         if (v.negative)
284             return decimal_plus(u, -v);
285     }
286
287     return decimal_minus(u, v);
288 }

289 friend big_decimal operator * (const big_decimal
290     &u, const big_decimal &v) {
291     if (u.negative || v.negative) {
292         big_decimal o = decimal_multi(abs(u),
293             abs(v));
294
295         if (u.negative ^ v.negative)
296             return -o;
297     }
298
299     return decimal_multi(u, v);
300 }
```

```

300 }
301
302 big_decimal operator * (long long x) const {
303     if (x >= 10)
304         return *this * big_decimal(x);
305
306     if (negative)
307         return -(*this * x);
308
309     big_decimal o;
310
311     o.a[0] = a[0];
312
313     for (int i = 1; i <= a[0]; i++) {
314         o.a[i] += a[i] * x;
315
316         if (o.a[i] >= 10) {
317             o.a[i + 1] += o.a[i] / 10;
318             o.a[i] %= 10;
319         }
320     }
321
322     if (o.a[a[0] + 1])
323         o.a[0]++;
324
325     return o;
326 }
327
328 friend pair<big_decimal, big_decimal>
329     decimal_div(const big_decimal &u, const
330     big_decimal &v) {
331     if (u.negative || v.negative) {
332         pair<big_decimal, big_decimal> o =
333             decimal_div(abs(u), abs(v));
334
335         if (u.negative ^ v.negative)
336             return make_pair(-o.first, -o.second);
337         return o;
338     }
339
340     return decimal_divide(u, v);
341
342 friend big_decimal operator / (const big_decimal
343     &u, const big_decimal &v) { // v不能是0
344     if (u.negative || v.negative) {
345         big_decimal o = abs(u) / abs(v);
346
347         if (u.negative ^ v.negative)
348             return -o;
349         return o;
350     }
351
352     return decimal_divide(u, v).first;
353
354 friend big_decimal operator % (const big_decimal
355     &u, const big_decimal &v) {
356     if (u.negative || v.negative) {
357         big_decimal o = abs(u) % abs(v);
358
359         if (u.negative ^ v.negative)
360             return -o;
361         return o;
362     }
363
364     return decimal_divide(u, v).second;
365 }

```

8.5 笛卡尔树

```

1 int s[maxn], root, lc[maxn], rc[maxn];
2
3 int top = 0;
4 s[++top] = root = 1;
5 for (int i = 2; i <= n; i++) {
6     s[top + 1] = 0;
7     while (top && a[i] < a[s[top]]) // 小根笛卡尔树
8         top--;
9
10    if (top)
11        rc[s[top]] = i;
12    else
13        root = i;
14
15    lc[i] = s[top + 1];
16    s[++top] = i;
17 }

```

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 $a_{k-1} \leq a_{k+1}$ 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 $a_j \geq a_k + a_{k-1}$ (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
194555039024054273	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

```

1 ull k1, k2;
2 const int mod = 100000000;
3 ull xorShift128Plus() {
4     ull k3 = k1, k4 = k2;
5     k1 = k4;
6     k3 ^= (k3 << 23);
7     k2 = k3 ^ k4 ^ (k3 >> 17) ^ (k4 >> 26);
8     return k2 + k4;
9 }
10 void gen(ull _k1, ull _k2) {
11     k1 = _k1, k2 = _k2;
12     int x = xorShift128Plus() % threshold + 1;
13     // do sth
14 }

```

```

15
16
17 uint32_t xor128(void) {
18     static uint32_t x = 123456789;
19     static uint32_t y = 362436069;
20     static uint32_t z = 521288629;
21     static uint32_t w = 88675123;
22     uint32_t t;
23
24     t = x ^ (x << 11);
25     x = y; y = z; z = w;
26     return w = w ^ (w >> 19) ^ (t ^ (t >> 8));
27 }

```

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)

```

1 for (int t = s; t; (--t) &= s) {
2     // do something
3 }

```

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
- void emplace_back(Args&& ... args)**: 自动构造并push_back一个元素, 例如对一个存储pair的vector可以 v.emplace_back(x, y)

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/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.

8.10.4 自定义 Hash

```

1 struct fuck_hash {
2     fuck_hash() = default;
3
4     size_t operator() (const fuck &f) const {
5         return (size_t)f[0] ^ ((size_t)f[1] << 7) ^
6             ((size_t)f[2] << 15) ^ ((size_t)f[3] <<
7                 23);
8     }
9 };
10
11 unordered_map<fuck, int, fuck_hash> cnt, sum;

```

8.11 Public Based DataStructure (PB_DS)

8.11.1 哈希表

```

1 #include<ext/pb_ds/assoc_container.hpp>
2 #include<ext/pb_ds/hash_policy.hpp>
3 using namespace __gnu_pbds;
4
5 cc_hash_table<string, int> mp1; // 拉链法
6 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.

```

1 #include<ext/pb_ds/priority_queue.hpp>
2 using namespace __gnu_pbds;
3
4 __gnu_pbds::priority_queue<int> q;
5 __gnu_pbds::priority_queue<int, greater<int>,
6     → 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 $\Theta(\log n)$; 如果只有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

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

```

1 #include <ext/pb_ds/tree_policy.hpp>
2 #include <ext/pb_ds/assoc_container.hpp>
3 using namespace __gnu_pbds;
4
5 tree<int, null_type, less<int>, rb_tree_tag,
6     → tree_order_statistics_node_update> t;
7
// 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 \geq 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

```

1 #include <ext/rope>
2 using namespace __gnu_cxx;
3
4 push_back(x); // 在末尾添加x
5 insert(pos, x); // 在pos插入x, 自然支持整个char数组的一
   → 次插入
6 erase(pos, x); // 从pos开始删除x个, 不要只传一个参数, 有
   → 痛
7 copy(pos, len, x); // 从pos开始到pos + len为止的部分, 赋
   → 值给x
8 replace(pos, x); // 从pos开始换成x
9 substr(pos, x); // 提取pos开始x个
10 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))`. 但是要注意返回的是引用, 所以不能直接用来交换 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 德州扑克

一般来说德扑里Ace都是最大的, 所以把Ace的点数规定为14会好写许多.

附一个高低奥马哈的参考代码, 除了有四张底牌和需要比低之外和德扑区别不大.

```

1 struct Card {
2     int suit, value; // Ace is treated as 14
3
4     Card(string s) {
5         char a = s[0];
6
7         if (isdigit(a))
8             value = a - '0';
9         else if (a == 'T')
10            value = 10;
11        else if (a == 'A')
12            value = 14;
13        else if (a == 'J')
14            value = 11;
15        else if (a == 'Q')
16            value = 12;
17        else if (a == 'K')
18            value = 13;
19        else
20            value = 5;
21    }
22
23    friend bool operator<(const Card & a, const Card & b) {
24        if (a.suit < b.suit)
25            return true;
26        else if (a.suit > b.suit)
27            return false;
28        else
29            return a.value < b.value;
30    }
31
32    friend ostream & operator<< (ostream & os, const Card & c) {
33        os << c.suit << " " << c.value;
34        return os;
35    }
36}

```

```

81     vector<int> kind[5];
82
83     for (int i = 2; i <= 14; i++)
84         if (c[i] > 1)
85             kind[c[i]].push_back(i);
86
87     if (!kind[4].empty()) { // 四条
88         type = FourofaKind;
89
90         for (int i = 0; i < 4; i++)
91             if (v[i].value != kind[4].front())
92                 swap(v[i], v.back());
93             break;
94         }
95
96
97         return;
98     }
99
100    if (!kind[3].empty() && !kind[2].empty()) {
101        type = FullHouse;
102
103        sort(v.begin(), v.end(), [&] (const Card
104            &a, const Card &b) {
105            bool ta = (a.value == kind[3].front());
106            &tb = (b.value == kind[3].front());
107
108            return ta > tb;
109        });
110
111        return;
112    }
113
114    if (flush) {
115        type = Flush;
116        sort(v.begin(), v.end());
117        reverse(v.begin(), v.end());
118
119        return;
120    }
121
122    if (straight) {
123        type = Straight;
124        reverse(v.begin(), v.end());
125        return;
126    }
127
128    if (!kind[3].empty()) {
129        type = ThreeofaKind;
130
131        sort(v.begin(), v.end(), [&] (const Card
132            &a, const Card &b) {
133            bool ta = (a.value == kind[3].front());
134            &tb = (b.value == kind[3].front());
135
136            return ta > tb;
137        });
138
139        if (v[3] < v[4])
140            swap(v[3], v[4]);
141
142        return;
143    }
144
145    if ((int)kind[2].size() == 2) {
146        type = TwoPairs;
147
148        sort(v.begin(), v.end(), [&] (const Card
149            &a, const Card &b) {

```

```

145     |     |     |     bool ta = (c[a.value] == 2), tb =
146     |     |     |     ↪ (c[b.value] == 2);
147     |     |     |     if (ta != tb)
148     |     |     |     return ta > tb;
149     |     |     |     return a.value > b.value;
150     |     |     | );
151     |     |     return;
152     |     | );
153     |     |     return;
154     |     | );
155
156     if ((int)kind[2].size() == 1) {
157         type = Pair;
158
159         sort(v.begin(), v.end(), [&] (const Card
160             ↪ &a, const Card &b) {
161             bool ta = (c[a.value] == 2), tb =
162                 ↪ (c[b.value] == 2);
163
164             if (ta != tb)
165                 return ta > tb;
166
167             return a.value > b.value;
168         });
169
170         return;
171
172         type = Highcard;
173
174         sort(v.begin(), v.end());
175         reverse(v.begin(), v.end());
176     }
177
178     void init_low() {
179         for (auto &o : v)
180             if (o.value == 14)
181                 o.value = 1;
182
183         sort(v.begin(), v.end());
184         reverse(v.begin(), v.end());
185     }
186
187     friend int cmp_high(const Hand &a, const Hand &b) {
188         if (a.type != b.type)
189             return a.type < b.type ? -1 : 1;
190
191         if (a.v != b.v)
192             return a.v < b.v ? -1 : 1;
193
194         return 0;
195     }
196
197     friend bool small_high(const Hand &a, const Hand
198         ↪ &b) {
199         return cmp_high(a, b) < 0;
200     }
201
202     friend int cmp_low(const Hand &a, const Hand &b) {
203         for (int i = 0; i < 5; i++)
204             if (a.v[i].value != b.v[i].value)
205                 return a.v[i] < b.v[i] ? 1 : -1;
206
207         return 0;
208     }
209
210     friend bool small_low(const Hand &a, const Hand &b)
211         ↪ {
212         return cmp_low(a, b) < 0;
213
214     }
215
216     bool operator ! () const {
217         return v.empty();
218     }
219
220     string str() const {
221         stringstream ss;
222         for (auto &o : v)
223             ss << o.value << ' ';
224     }
225
226 Hand get_max_high(vector<Card> u, vector<Card> v) { // //
227     ↪ private, public
228     Hand ans;
229
230     for (int i = 0; i < 4; i++)
231         for (int j = i + 1; j < 4; j++)
232             for (int k = 0; k < 5; k++)
233                 for (int p = k + 1; p < 5; p++)
234                     for (int q = p + 1; q < 5; q++) {
235                         Hand tmp({u[i], u[j], v[k],
236                             ↪ v[p], v[q]});
237                         tmp.init_high();
238                         if (!ans || cmp_high(tmp, ans)
239                             ↪ > 0)
240                             ans = tmp;
241
242     return ans;
243 }
244
245 Hand get_max_low(vector<Card> tu, vector<Card> tv) {
246     vector<Card> u, v;
247
248     for (auto o : tu)
249         if (o.value == 14 || o.value <= 8)
250             u.push_back(o);
251
252     for (auto o : tv)
253         if (o.value == 14 || o.value <= 8)
254             v.push_back(o);
255
256     Hand ans;
257
258     for (int i = 0; i < (int)u.size(); i++)
259         for (int j = i + 1; j < (int)u.size(); j++)
260             for (int k = 0; k < (int)v.size(); k++)
261                 for (int p = k + 1; p < (int)v.size();
262                     ↪ p++)
263                     for (int q = p + 1; q <
264                         ↪ (int)v.size(); q++) {
265                         vector<Card> vec = {u[i], u[j],
266                             ↪ v[k], v[p], v[q]};
267                         bool bad = false;
268
269                         for (int a = 0; a < 5; a++)
270                             for (int b = a + 1; b < 5;
271                                 ↪ b++)
272                                 if (vec[a].value ==
273                                     ↪ vec[b].value)
274                                     bad = true;
275
276     }
277
278 }
```

```

271
272     if (bad)
273         continue;
274
275     Hand tmp(vec);
276
277     tmp.init_low();
278
279     if (!ans || cmp_low(tmp, ans) >
280         ↪ 0)
281         ans = tmp;
282
283     return ans;
284 }
285
286 int main() {
287
288     ios::sync_with_stdio(false);
289
290     int T;
291     cin >> T;
292
293     while (T--) {
294         int p;
295         cin >> p;
296
297         vector<Card> alice, bob, pub;
298
299         for (int i = 0; i < 4; i++) {
300             string s;
301             cin >> s;
302             alice.push_back(Card(s));
303         }
304
305         for (int i = 0; i < 4; i++) {
306             string s;
307             cin >> s;
308             bob.push_back(Card(s));
309         }
310
311         for (int i = 0; i < 5; i++) {
312             string s;
313             cin >> s;
314             pub.push_back(Card(s));
315         }
316
317         Hand alice_high = get_max_high(alice, pub),
318             ↪ bob_high = get_max_high(bob, pub);
319         Hand alice_low = get_max_low(alice, pub),
320             ↪ bob_low = get_max_low(bob, pub);
321
322         int dh = cmp_high(alice_high, bob_high);
323         int ans[2] = {0};
324
325         if (!alice_low && !bob_low) {
326             if (!dh) {
327                 ans[0] = p - p / 2;
328                 ans[1] = p / 2;
329             }
330             else
331                 ans[dh == -1] = p;
332
333             else if (!alice_low || !bob_low) {
334                 ans[!alice_low] += p / 2;
335
336                 if (!dh) {
337                     ans[0] += p - p / 2 - (p - p / 2) / 2;
338                     ans[1] += (p - p / 2) / 2;
339
340                 }
341             }
342             else {
343                 int dl = cmp_low(alice_low, bob_low);
344
345                 if (!dl) {
346                     ans[0] += p / 2 - p / 2 / 2;
347                     ans[1] += p / 2 / 2;
348                 }
349                 else
350                     ans[dl == -1] += p / 2;
351
352                 if (!dh) {
353                     ans[0] += p - p / 2 - (p - p / 2) / 2;
354                     ans[1] += (p - p / 2) / 2;
355                 }
356                 else
357                     ans[dh == -1] += p - p / 2;
358
359                 cout << ans[0] << ' ' << ans[1] << '\n';
360             }
361
362         }
363     }
364 }
```

8.14.2 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1.10.7.卡特兰数, 施罗德数, 默慈金数(17页).

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1.10.7.卡特兰数, 施罗德数, 默慈金数(17页).

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, ...

$$n! - (n-1)! + (n-2)! - \dots 1! = \sum_{i=0}^{n-1} (-1)^i (n-i)!$$

$a_0 = 0$, $a_n = n! - a_{n-1}$.

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, 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 $a^{n-1} \equiv 1 \pmod{n}$ 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 1.10.2. 伯努利数, 自然数幂次和(16页).

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质是 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++17`: 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/ftrapv)`: 检查数组越界/有符号整数溢出 (算 ub)
 - 调试神器，在遇到错误时会输出信息.
 - 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.
- `#define debug(x) cout << #x << " = " << x << endl`

8.17 附录: VScode 相关

8.17.1 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)

8.17.2 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”)
- Terminal → Integrated: Cursor Style (修改终端光标形状)
- Terminal → Integrated: Cursor Blinking (终端光标闪烁)
- 字体改为Cascadia Code/Mono SemiLight (Windows可用)

8.17.3 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

更多快捷键参见最后两页, 分别是Windows和Linux下的快捷键列表.

8.18 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，尤其是那一种怒怒的感情。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。俗语少用，因为俗语一览无遗，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！

8.19 附录：Cheat Sheet

见后面几页。

Theoretical Computer Science Cheat Sheet

Definitions		Series
$f(n) = O(g(n))$	iff \exists positive c, n_0 such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0$.	$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$. In general: $\sum_{i=1}^n i^m = \frac{1}{m+1} \left[(n+1)^{m+1} - 1 - \sum_{i=1}^n ((i+1)^{m+1} - i^{m+1} - (m+1)i^m) \right]$ $\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}.$
$f(n) = \Omega(g(n))$	iff \exists positive c, n_0 such that $f(n) \geq cg(n) \geq 0 \forall n \geq n_0$.	
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.	
$f(n) = o(g(n))$	iff $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$.	
$\lim_{n \rightarrow \infty} a_n = a$	iff $\forall \epsilon > 0, \exists n_0$ such that $ a_n - a < \epsilon, \forall n \geq n_0$.	
$\sup S$	least $b \in \mathbb{R}$ such that $b \geq s, \forall s \in S$.	Geometric series: $\sum_{i=0}^n c^i = \frac{c^{n+1} - 1}{c - 1}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1 - c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1 - c}, \quad c < 1,$ $\sum_{i=0}^n i c^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} i c^i = \frac{c}{(1-c)^2}, \quad c < 1.$
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \leq s, \forall s \in S$.	
$\liminf_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \inf \{a_i \mid i \geq n, i \in \mathbb{N}\}$.	Harmonic series: $H_n = \sum_{i=1}^n \frac{1}{i}, \quad \sum_{i=1}^n i H_i = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}.$
$\limsup_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \sup \{a_i \mid i \geq n, i \in \mathbb{N}\}$.	$\sum_{i=1}^n H_i = (n+1)H_n - n, \quad \sum_{i=1}^n \binom{i}{m} H_i = \binom{n+1}{m+1} \left(H_{n+1} - \frac{1}{m+1} \right).$
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.	
$\begin{bmatrix} n \\ k \end{bmatrix}$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.	1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad 2. \sum_{k=0}^n \binom{n}{k} = 2^n, \quad 3. \binom{n}{k} = \binom{n}{n-k},$
$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.	4. $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad 5. \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$
$\left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle$	1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \dots \pi_n$ on $\{1, 2, \dots, n\}$ with k ascents.	6. $\binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}, \quad 7. \sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n},$
$\left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle$	2nd order Eulerian numbers.	8. $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}, \quad 9. \sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n},$
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.	10. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad 11. \left\{ \begin{smallmatrix} n \\ 1 \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} n \\ n \end{smallmatrix} \right\} = 1,$
14. $\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!$	15. $\begin{bmatrix} n \\ 2 \end{bmatrix} = (n-1)!H_{n-1},$	12. $\left\{ \begin{smallmatrix} n \\ 2 \end{smallmatrix} \right\} = 2^{n-1} - 1, \quad 13. \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = k \left\{ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\} + \left\{ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\},$
18. $\begin{bmatrix} n \\ k \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix},$	19. $\left\{ \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \right\} = \left[\begin{smallmatrix} n \\ n-1 \end{smallmatrix} \right] = \binom{n}{2},$	16. $\begin{bmatrix} n \\ n \end{bmatrix} = 1, \quad 17. \begin{bmatrix} n \\ k \end{bmatrix} \geq \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\},$
22. $\left\langle \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right\rangle = \left\langle \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \right\rangle = 1,$	23. $\left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle = \left\langle \begin{smallmatrix} n \\ n-1-k \end{smallmatrix} \right\rangle,$	20. $\sum_{k=0}^n \begin{bmatrix} n \\ k \end{bmatrix} = n!, \quad 21. C_n = \frac{1}{n+1} \binom{2n}{n},$
25. $\left\langle \begin{smallmatrix} 0 \\ k \end{smallmatrix} \right\rangle = \begin{cases} 1 & \text{if } k=0, \\ 0 & \text{otherwise} \end{cases}$	26. $\left\langle \begin{smallmatrix} n \\ 1 \end{smallmatrix} \right\rangle = 2^n - n - 1,$	24. $\left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle = (k+1) \left\langle \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\rangle + (n-k) \left\langle \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\rangle,$
28. $x^n = \sum_{k=0}^n \left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle \binom{x+k}{n},$	29. $\left\langle \begin{smallmatrix} n \\ m \end{smallmatrix} \right\rangle = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k,$	27. $\left\langle \begin{smallmatrix} n \\ 2 \end{smallmatrix} \right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2},$
31. $\left\langle \begin{smallmatrix} n \\ m \end{smallmatrix} \right\rangle = \sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} \binom{n-k}{m} (-1)^{n-k-m} k!,$	32. $\left\langle \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right\rangle = 1,$	30. $m! \left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\} = \sum_{k=0}^n \left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle \binom{k}{n-m},$
34. $\left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle = (k+1) \left\langle \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\rangle + (2n-1-k) \left\langle \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\rangle,$		33. $\left\langle \begin{smallmatrix} n \\ n \end{smallmatrix} \right\rangle = 0 \quad \text{for } n \neq 0,$
36. $\left\{ \begin{smallmatrix} x \\ x-n \end{smallmatrix} \right\} = \sum_{k=0}^n \left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle \binom{x+n-1-k}{2n},$	37. $\left\{ \begin{smallmatrix} n+1 \\ m+1 \end{smallmatrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{smallmatrix} k \\ m \end{smallmatrix} \right\} = \sum_{k=0}^n \left\{ \begin{smallmatrix} k \\ m \end{smallmatrix} \right\} (m+1)^{n-k},$	35. $\sum_{k=0}^n \left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle = \frac{(2n)^n}{2^n},$

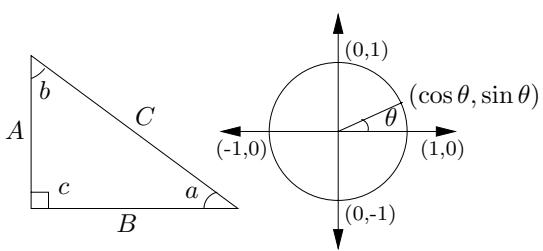
Theoretical Computer Science Cheat Sheet		
Identities Cont.		Trees
38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \sum_{k=0}^n \begin{bmatrix} k \\ m \end{bmatrix} n^{n-k} = n! \sum_{k=0}^n \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix}$,	39. $\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{x+k}{2n}$,	Every tree with n vertices has $n-1$ edges.
40. $\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \begin{Bmatrix} k+1 \\ m+1 \end{Bmatrix} (-1)^{n-k}$,	41. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k}$,	Kraft inequality: If the depths of the leaves of a binary tree are d_1, \dots, d_n : $\sum_{i=1}^n 2^{-d_i} \leq 1,$
42. $\begin{Bmatrix} m+n+1 \\ m \end{Bmatrix} = \sum_{k=0}^m k \begin{Bmatrix} n+k \\ k \end{Bmatrix}$,	43. $\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^m k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix}$,	and equality holds only if every internal node has 2 sons.
44. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \binom{k}{m} (-1)^{m-k}$,	45. $(n-m)! \begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{Bmatrix} \begin{Bmatrix} k \\ m \end{Bmatrix} (-1)^{m-k}$, for $n \geq m$,	
46. $\begin{Bmatrix} n \\ n-m \end{Bmatrix} = \sum_k \begin{bmatrix} m-n \\ m+k \end{bmatrix} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix}$,	47. $\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \begin{bmatrix} m-n \\ m+k \end{bmatrix} \binom{m+n}{n+k} \begin{Bmatrix} m+k \\ k \end{Bmatrix}$,	
48. $\begin{Bmatrix} n \\ \ell+m \end{Bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{Bmatrix} k \\ \ell \end{Bmatrix} \begin{Bmatrix} n-k \\ m \end{Bmatrix} \binom{n}{k}$,	49. $\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}$.	
Recurrences		
<p>Master method: $T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$</p> <p>If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$.</p> <p>If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log_2 n)$.</p> <p>If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n, then $T(n) = \Theta(f(n))$.</p> <p>Substitution (example): Consider the following recurrence $T_{i+1} = 2^{2^i} \cdot T_i^2$, $T_1 = 2$.</p> <p>Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have $t_{i+1} = 2^i + 2t_i$, $t_1 = 1$.</p> <p>Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get $\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}$.</p> <p>Substituting we find $u_{i+1} = \frac{1}{2} + u_i$, $u_1 = \frac{1}{2}$,</p> <p>which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.</p> <p>Summing factors (example): Consider the following recurrence $T(n) = 3T(n/2) + n$, $T(1) = 1$.</p> <p>Rewrite so that all terms involving T are on the left side $T(n) - 3T(n/2) = n$.</p> <p>Now expand the recurrence, and choose a factor which makes the left side “telescope”</p>	$1(T(n) - 3T(n/2) = n)$ $3(T(n/2) - 3T(n/4) = n/2)$ $\vdots \quad \vdots \quad \vdots$ $3^{\log_2 n-1}(T(2) - 3T(1) = 2)$ <p>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</p> $\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\frac{3}{2}\right)^i.$ <p>Let $c = \frac{3}{2}$. Then we have</p> $\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n \left(\frac{c^m - 1}{c - 1} \right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_c n} - 1) \\ &= 2n^k - 2n, \end{aligned}$ <p>and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider</p> $T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$ <p>Note that</p> $T_{i+1} = 1 + \sum_{j=0}^i T_j.$ <p>Subtracting we find</p> $\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^i T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$ <p>And so $T_{i+1} = 2T_i = 2^{i+1}$.</p>	<p>Generating functions:</p> <ol style="list-style-type: none"> Multiply both sides of the equation by x^i. Sum both sides over all i for which the equation is valid. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$. Rewrite the equation in terms of the generating function $G(x)$. Solve for $G(x)$. The coefficient of x^i in $G(x)$ is g_i. <p>Example: $g_{i+1} = 2g_i + 1$, $g_0 = 0$.</p> <p>Multiply and sum: $\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$</p> <p>We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:</p> $\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$ <p>Simplify: $\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$</p> <p>Solve for $G(x)$:</p> $G(x) = \frac{x}{(1-x)(1-2x)}.$ <p>Expand this using partial fractions:</p> $\begin{aligned} G(x) &= x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right) \\ &= x \left(2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right) \\ &= \sum_{i \geq 0} (2^{i+1} - 1)x^{i+1}. \end{aligned}$ <p>So $g_i = 2^i - 1$.</p>

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	General	Probability
1	2	2	Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$): $B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30}, B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}.$	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 $P(a) = \int_{-\infty}^a p(x) dx.$
2	4	3		
3	8	5		
4	16	7	Change of base, quadratic formula: $\log_b x = \frac{\log_a x}{\log_a b}, \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$	
5	32	11		
6	64	13		
7	128	17	Euler's number e : $e = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots$ $\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$	Expectation: If X is discrete $E[g(X)] = \sum_x g(x) \Pr[X = x].$
8	256	19		If X continuous then $E[g(X)] = \int_{-\infty}^{\infty} g(x)p(x) dx = \int_{-\infty}^{\infty} g(x) dP(x).$
9	512	23		Variance, standard deviation: $\text{VAR}[X] = E[X^2] - E[X]^2,$ $\sigma = \sqrt{\text{VAR}[X]}.$
10	1,024	29		For events A and B : $\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$
11	2,048	31		$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$ iff A and B are independent.
12	4,096	37		$\Pr[A B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$
13	8,192	41	Harmonic numbers: $1, \frac{3}{2}, \frac{11}{6}, \frac{25}{12}, \frac{137}{60}, \frac{49}{20}, \frac{363}{140}, \frac{761}{280}, \frac{7129}{2520}, \dots$	For random variables X and Y : $E[X \cdot Y] = E[X] \cdot E[Y],$ if X and Y are independent.
14	16,384	43		$E[X + Y] = E[X] + E[Y],$ $E[cX] = cE[X].$
15	32,768	47		Bayes' theorem:
16	65,536	53		$\Pr[A_i B] = \frac{\Pr[B A_i] \Pr[A_i]}{\sum_{j=1}^n \Pr[A_j] \Pr[B A_j]}.$
17	131,072	59		Inclusion-exclusion:
18	262,144	61		$\Pr\left[\bigvee_{i=1}^n X_i\right] = \sum_{i=1}^n \Pr[X_i] + \sum_{k=2}^n (-1)^{k+1} \sum_{i_1 < \dots < i_k} \Pr\left[\bigwedge_{j=1}^k X_{i_j}\right].$
19	524,288	67	Factorial, Stirling's approximation: $1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \dots$	Moment inequalities:
20	1,048,576	71		$\Pr[X \geq \lambda E[X]] \leq \frac{1}{\lambda},$ $\Pr[X - E[X] \geq \lambda \cdot \sigma] \leq \frac{1}{\lambda^2}.$
21	2,097,152	73		Geometric distribution:
22	4,194,304	79		$\Pr[X = k] = pq^{k-1}, \quad q = 1 - p,$ $E[X] = \sum_{k=1}^{\infty} k \Pr[X = k] = \frac{p}{1-q}.$
23	8,388,608	83	Ackermann's function and inverse: $a(i, j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$ $\alpha(i) = \min\{j \mid a(j, j) \geq i\}.$	
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	Binomial distribution: $\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p,$ $E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.$	
29	536,870,912	109		
30	1,073,741,824	113	Poisson distribution: $\Pr[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}, \quad E[X] = \lambda.$	
31	2,147,483,648	127	Normal (Gaussian) distribution: $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad E[X] = \mu.$	
32	4,294,967,296	131	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 collect all n types is $nH_n.$	
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			

Theoretical Computer Science Cheat Sheet

Trigonometry



Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$

$$\csc a = C/A, \quad \sec a = C/B,$$

$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\frac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x},$$

$$\cos x = \frac{1}{\sec x},$$

$$\tan x = \frac{1}{\cot x},$$

$$\sin^2 x + \cos^2 x = 1,$$

$$1 + \tan^2 x = \sec^2 x,$$

$$1 + \cot^2 x = \csc^2 x,$$

$$\sin x = \cos(\frac{\pi}{2} - x),$$

$$\sin x = \sin(\pi - x),$$

$$\cos x = -\cos(\pi - x),$$

$$\tan x = \cot(\frac{\pi}{2} - x),$$

$$\cot x = -\cot(\pi - x),$$

$$\csc x = \cot \frac{x}{2} - \cot x,$$

$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$

$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$

$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$

$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$

$$\sin 2x = 2 \sin x \cos x, \quad \sin 2x = \frac{2 \tan x}{1 + \tan^2 x},$$

$$\cos 2x = \cos^2 x - \sin^2 x, \quad \cos 2x = 2 \cos^2 x - 1,$$

$$\cos 2x = 1 - 2 \sin^2 x, \quad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$

$$\tan 2x = \frac{2 \tan x}{1 - \tan^2 x}, \quad \cot 2x = \frac{\cot^2 x - 1}{2 \cot x},$$

$$\sin(x+y) \sin(x-y) = \sin^2 x - \sin^2 y,$$

$$\cos(x+y) \cos(x-y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i \sin x, \quad e^{i\pi} = -1.$$

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.

$$\det A \cdot B = \det A \cdot \det B,$$

$$\det A = \sum_{\pi} \prod_{i=1}^n \text{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$

$$= aei + bfg + cdh - ceg - fha - ibd.$$

Permanents:

$$\text{perm } A = \sum_{\pi} \prod_{i=1}^n a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \quad \cosh x = \frac{e^x + e^{-x}}{2},$$

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \operatorname{csch} x = \frac{1}{\sinh x},$$

$$\operatorname{sech} x = \frac{1}{\cosh x}, \quad \coth x = \frac{1}{\tanh x}.$$

Identities:

$$\cosh^2 x - \sinh^2 x = 1, \quad \tanh^2 x + \operatorname{sech}^2 x = 1,$$

$$\coth^2 x - \operatorname{csch}^2 x = 1, \quad \sinh(-x) = -\sinh x,$$

$$\cosh(-x) = \cosh x, \quad \tanh(-x) = -\tanh x,$$

$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$

$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$

$$\sinh 2x = 2 \sinh x \cosh x,$$

$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$

$$\cosh x + \sinh x = e^x, \quad \cosh x - \sinh x = e^{-x},$$

$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$

$$2 \sinh^2 \frac{x}{2} = \cosh x - 1, \quad 2 \cosh^2 \frac{x}{2} = \cosh x + 1.$$

$$\begin{array}{cccc} \theta & \sin \theta & \cos \theta & \tan \theta \end{array}$$

$$\begin{array}{cccc} 0 & 0 & 1 & 0 \end{array}$$

$$\begin{array}{cccc} \frac{\pi}{6} & \frac{1}{2} & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{3} \end{array}$$

$$\begin{array}{cccc} \frac{\pi}{4} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 1 \end{array}$$

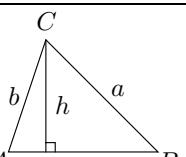
$$\begin{array}{cccc} \frac{\pi}{3} & \frac{\sqrt{3}}{2} & \frac{1}{2} & \sqrt{3} \end{array}$$

$$\begin{array}{cccc} \frac{\pi}{2} & 1 & 0 & \infty \end{array}$$

... 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:

$$\begin{aligned} A &= \frac{1}{2}hc, \\ &= \frac{1}{2}ab \sin C, \\ &= \frac{c^2 \sin A \sin B}{2 \sin C}. \end{aligned}$$

Heron's formula:

$$\begin{aligned} A &= \sqrt{s \cdot s_a \cdot s_b \cdot s_c}, \\ s &= \frac{1}{2}(a+b+c), \\ s_a &= s-a, \\ s_b &= s-b, \\ s_c &= s-c. \end{aligned}$$

More identities:

$$\sin \frac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$

$$\cos \frac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$

$$\tan \frac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$

$$= \frac{1 - \cos x}{\sin x},$$

$$= \frac{\sin x}{1 + \cos x},$$

$$\cot \frac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$

$$= \frac{1 + \cos x}{\sin x},$$

$$= \frac{\sin x}{1 - \cos x},$$

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$

$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$

$$\tan x = -i \frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$

$$= -i \frac{e^{2ix} - 1}{e^{2ix} + 1},$$

$$\sinh ix = \frac{e^{ix} - e^{-ix}}{i},$$

$$\cos x = \cosh ix,$$

$$\tan x = \frac{\tanh ix}{i}.$$

Theoretical Computer Science Cheat Sheet

Theoretical Computer Science Cheat Sheet									
Number Theory	Graph Theory								
<p>The Chinese remainder theorem: There exists a number C such that:</p> $C \equiv r_1 \pmod{m_1}$ $\vdots \vdots \vdots$ $C \equiv r_n \pmod{m_n}$ <p>if m_i and m_j are relatively prime for $i \neq j$.</p> <p>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</p> $\phi(x) = \prod_{i=1}^n p_i^{e_i-1} (p_i - 1).$ <p>Euler's theorem: If a and b are relatively prime then</p> $1 \equiv a^{\phi(b)} \pmod{b}.$ <p>Fermat's theorem:</p> $1 \equiv a^{p-1} \pmod{p}.$ <p>The Euclidean algorithm: if $a > b$ are integers then</p> $\gcd(a, b) = \gcd(a \bmod b, b).$ <p>If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then</p> $S(x) = \sum_{d x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$ <p>Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.</p> <p>Wilson's theorem: n is a prime iff</p> $(n-1)! \equiv -1 \pmod{n}.$ <p>Möbius inversion:</p> $\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$ <p>If</p> $G(a) = \sum_{d a} F(d),$ <p>then</p> $F(a) = \sum_{d a} \mu(d) G\left(\frac{a}{d}\right).$ <p>Prime numbers:</p> $p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$ $\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$	<p>Definitions:</p> <ul style="list-style-type: none"> <i>Loop</i>: An edge connecting a vertex to itself. <i>Directed</i>: Each edge has a direction. <i>Simple</i>: Graph with no loops or multi-edges. <i>Walk</i>: A sequence $v_0 e_1 v_1 \dots e_\ell v_\ell$. <i>Trail</i>: A walk with distinct edges. <i>Path</i>: A trail with distinct vertices. <i>Connected</i>: A graph where there exists a path between any two vertices. <i>Component</i>: A maximal connected subgraph. <i>Tree</i>: A connected acyclic graph. <i>Free tree</i>: A tree with no root. <i>DAG</i>: Directed acyclic graph. <i>Eulerian</i>: Graph with a trail visiting each edge exactly once. <i>Hamiltonian</i>: Graph with a cycle visiting each vertex exactly once. <i>Cut</i>: A set of edges whose removal increases the number of components. <i>Cut-set</i>: A minimal cut. <i>Cut edge</i>: A size 1 cut. <i>k-Connected</i>: A graph connected with the removal of any $k-1$ vertices. <i>k-Tough</i>: $\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G-S) \leq S$. <i>k-Regular</i>: A graph where all vertices have degree k. <i>k-Factor</i>: A k-regular spanning subgraph. <i>Matching</i>: A set of edges, no two of which are adjacent. <i>Clique</i>: A set of vertices, all of which are adjacent. <i>Ind. set</i>: A set of vertices, none of which are adjacent. <i>Vertex cover</i>: A set of vertices which cover all edges. <i>Planar graph</i>: A graph which can be embedded in the plane. <i>Plane graph</i>: An embedding of a planar graph. <p style="text-align: right;">$\sum_{v \in V} \deg(v) = 2m.$</p>	<p>Notation:</p> <ul style="list-style-type: none"> $E(G)$: 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 $r(k, \ell)$: Ramsey number 							
		Geometry							
<p>Projective coordinates: triples (x, y, z), not all x, y and z zero.</p> $(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$ <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Cartesian</td><td style="width: 50%;">Projective</td></tr> <tr> <td>(x, y)</td><td>$(x, y, 1)$</td></tr> <tr> <td>$y = mx + b$</td><td>$(m, -1, b)$</td></tr> <tr> <td>$x = c$</td><td>$(1, 0, -c)$</td></tr> </table> <p>Distance formula, L_p and L_∞ metric:</p> $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$ $[x_1 - x_0 ^p + y_1 - y_0 ^p]^{1/p},$ $\lim_{p \rightarrow \infty} [x_1 - x_0 ^p + y_1 - y_0 ^p]^{1/p}.$ <p>Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):</p> $\frac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$ <p>Angle formed by three points:</p> $\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$ <p>Line through two points (x_0, y_0) and (x_1, y_1):</p> $\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$ <p>Area of circle, volume of sphere:</p> $A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$		Cartesian	Projective	(x, y)	$(x, y, 1)$	$y = mx + b$	$(m, -1, b)$	$x = c$	$(1, 0, -c)$
Cartesian	Projective								
(x, y)	$(x, y, 1)$								
$y = mx + b$	$(m, -1, b)$								
$x = c$	$(1, 0, -c)$								
<p>If I have seen farther than others, it is because I have stood on the shoulders of giants. – Issac Newton</p>									

Theoretical Computer Science Cheat Sheet

π	Calculus
<p>Wallis' identity:</p> $\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$ <p>Brouncker's continued fraction expansion:</p> $\frac{\pi}{4} = 1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{\cdots}}}}$ <p>Gregory's series:</p> $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$ <p>Newton's series:</p> $\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$ <p>Sharp's series:</p> $\frac{\pi}{6} = \frac{1}{\sqrt{3}} \left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots \right)$ <p>Euler's series:</p> $\begin{aligned}\frac{\pi^2}{6} &= \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots \\ \frac{\pi^2}{8} &= \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots \\ \frac{\pi^2}{12} &= \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots\end{aligned}$	<p>Derivatives:</p> <ol style="list-style-type: none"> 1. $\frac{d(cu)}{dx} = c \frac{du}{dx},$ 2. $\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx},$ 3. $\frac{d(uv)}{dx} = u \frac{dv}{dx} + v \frac{du}{dx},$ 4. $\frac{d(u^n)}{dx} = nu^{n-1} \frac{du}{dx},$ 5. $\frac{d(u/v)}{dx} = \frac{v(\frac{du}{dx}) - u(\frac{dv}{dx})}{v^2},$ 6. $\frac{d(e^{cu})}{dx} = ce^{cu} \frac{du}{dx},$ 7. $\frac{d(c^u)}{dx} = (\ln c)c^u \frac{du}{dx},$ 8. $\frac{d(\ln u)}{dx} = \frac{1}{u} \frac{du}{dx},$ 10. $\frac{d(\cos u)}{dx} = -\sin u \frac{du}{dx},$ 12. $\frac{d(\cot u)}{dx} = \csc^2 u \frac{du}{dx},$ 14. $\frac{d(\csc u)}{dx} = -\cot u \csc u \frac{du}{dx},$ 16. $\frac{d(\arccos u)}{dx} = \frac{-1}{\sqrt{1-u^2}} \frac{du}{dx},$ 18. $\frac{d(\operatorname{arccot} u)}{dx} = \frac{-1}{1+u^2} \frac{du}{dx},$ 20. $\frac{d(\operatorname{arccsc} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}} \frac{du}{dx},$ 22. $\frac{d(\cosh u)}{dx} = \sinh u \frac{du}{dx},$ 24. $\frac{d(\coth u)}{dx} = -\operatorname{csch}^2 u \frac{du}{dx},$ 26. $\frac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u \frac{du}{dx},$ 28. $\frac{d(\operatorname{arccosh} u)}{dx} = \frac{1}{\sqrt{u^2-1}} \frac{du}{dx},$ 30. $\frac{d(\operatorname{arccoth} u)}{dx} = \frac{1}{u^2-1} \frac{du}{dx},$ 32. $\frac{d(\operatorname{arccsch} u)}{dx} = \frac{-1}{ u \sqrt{1+u^2}} \frac{du}{dx}.$ <p>Integrals:</p> <ol style="list-style-type: none"> 1. $\int cu \, dx = c \int u \, dx,$ 2. $\int (u+v) \, dx = \int u \, dx + \int v \, dx,$ 3. $\int x^n \, dx = \frac{1}{n+1} x^{n+1}, \quad n \neq -1,$ 4. $\int \frac{1}{x} \, dx = \ln x,$ 5. $\int e^x \, dx = e^x,$ 6. $\int \frac{dx}{1+x^2} = \arctan x,$ 7. $\int u \frac{dv}{dx} \, dx = uv - \int v \frac{du}{dx} \, dx,$ 8. $\int \sin x \, dx = -\cos x,$ 9. $\int \cos x \, dx = \sin x,$ 10. $\int \tan x \, dx = -\ln \cos x ,$ 11. $\int \cot x \, dx = \ln \cos x ,$ 12. $\int \sec x \, dx = \ln \sec x + \tan x ,$ 13. $\int \csc x \, dx = \ln \csc x + \cot x ,$ 14. $\int \arcsin \frac{x}{a} \, dx = \arcsin \frac{x}{a} + \sqrt{a^2 - x^2}, \quad a > 0,$
<p>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</p>	

Theoretical Computer Science Cheat Sheet

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. $\int \sin^2(ax) dx = \frac{1}{2a} (ax - \sin(ax) \cos(ax)),$
18. $\int \cos^2(ax) dx = \frac{1}{2a} (ax + \sin(ax) \cos(ax)),$
19. $\int \sec^2 x dx = \tan x,$
20. $\int \csc^2 x dx = -\cot x,$
21. $\int \sin^n x dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x dx,$
22. $\int \cos^n x dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x dx,$
23. $\int \tan^n x dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x dx, \quad n \neq 1,$
24. $\int \cot^n x dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x dx, \quad n \neq 1,$
25. $\int \sec^n x dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x dx, \quad n \neq 1,$
26. $\int \csc^n x dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x dx, \quad n \neq 1,$
27. $\int \sinh x dx = \cosh x, \quad 28. \int \cosh x dx = \sinh x,$
29. $\int \tanh x dx = \ln |\cosh x|, \quad 30. \int \coth x dx = \ln |\sinh x|, \quad 31. \int \operatorname{sech} x dx = \arctan \sinh x, \quad 32. \int \operatorname{csch} x dx = \ln |\tanh \frac{x}{2}|,$
33. $\int \sinh^2 x dx = \frac{1}{4} \sinh(2x) - \frac{1}{2}x, \quad 34. \int \cosh^2 x dx = \frac{1}{4} \sinh(2x) + \frac{1}{2}x, \quad 35. \int \operatorname{sech}^2 x dx = \tanh x,$
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. $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$
39. $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln \left(x + \sqrt{a^2 + x^2} \right), \quad a > 0,$
40. $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$
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. $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$
44. $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left| \frac{a+x}{a-x} \right|, \quad 45. \int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2 \sqrt{a^2 - x^2}},$
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. $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left| x + \sqrt{x^2 - a^2} \right|, \quad a > 0,$
48. $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left| \frac{x}{a+bx} \right|,$
49. $\int x \sqrt{a+bx} dx = \frac{2(3bx - 2a)(a+bx)^{3/2}}{15b^2},$
50. $\int \frac{\sqrt{a+bx}}{x} dx = 2\sqrt{a+bx} + a \int \frac{1}{x\sqrt{a+bx}} dx,$
51. $\int \frac{x}{\sqrt{a+bx}} dx = \frac{1}{\sqrt{2}} \ln \left| \frac{\sqrt{a+bx} - \sqrt{a}}{\sqrt{a+bx} + \sqrt{a}} \right|, \quad a > 0,$
52. $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
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. $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
56. $\int \frac{x dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$
57. $\int \frac{x^2 dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
58. $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left| \frac{a + \sqrt{a^2 + x^2}}{x} \right|,$
59. $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$
60. $\int x \sqrt{x^2 \pm a^2} dx = \frac{1}{3} (x^2 \pm a^2)^{3/2},$
61. $\int \frac{dx}{x\sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left| \frac{x}{a + \sqrt{a^2 + x^2}} \right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

- 62.** $\int \frac{dx}{x\sqrt{x^2 - a^2}} = \frac{1}{a} \arccos \frac{a}{|x|}, \quad a > 0,$ **63.** $\int \frac{dx}{x^2\sqrt{x^2 \pm a^2}} = \mp \frac{\sqrt{x^2 \pm a^2}}{a^2 x},$
64. $\int \frac{x dx}{\sqrt{x^2 \pm a^2}} = \sqrt{x^2 \pm a^2},$ **65.** $\int \frac{\sqrt{x^2 \pm a^2}}{x^4} dx = \mp \frac{(x^2 + a^2)^{3/2}}{3a^2 x^3},$
66. $\int \frac{dx}{ax^2 + bx + c} = \begin{cases} \frac{1}{\sqrt{b^2 - 4ac}} \ln \left| \frac{2ax + b - \sqrt{b^2 - 4ac}}{2ax + b + \sqrt{b^2 - 4ac}} \right|, & \text{if } b^2 > 4ac, \\ \frac{2}{\sqrt{4ac - b^2}} \arctan \frac{2ax + b}{\sqrt{4ac - b^2}}, & \text{if } b^2 < 4ac, \end{cases}$
67. $\int \frac{dx}{\sqrt{ax^2 + bx + c}} = \begin{cases} \frac{1}{\sqrt{a}} \ln \left| 2ax + b + 2\sqrt{a}\sqrt{ax^2 + bx + c} \right|, & \text{if } a > 0, \\ \frac{1}{\sqrt{-a}} \arcsin \frac{-2ax - b}{\sqrt{b^2 - 4ac}}, & \text{if } a < 0, \end{cases}$
68. $\int \sqrt{ax^2 + bx + c} dx = \frac{2ax + b}{4a} \sqrt{ax^2 + bx + c} + \frac{4ax - b^2}{8a} \int \frac{dx}{\sqrt{ax^2 + bx + c}},$
69. $\int \frac{x dx}{\sqrt{ax^2 + bx + c}} = \frac{\sqrt{ax^2 + bx + c}}{a} - \frac{b}{2a} \int \frac{dx}{\sqrt{ax^2 + bx + c}},$
70. $\int \frac{dx}{x\sqrt{ax^2 + bx + c}} = \begin{cases} \frac{-1}{\sqrt{c}} \ln \left| \frac{2\sqrt{c}\sqrt{ax^2 + bx + c} + bx + 2c}{x} \right|, & \text{if } c > 0, \\ \frac{1}{\sqrt{-c}} \arcsin \frac{bx + 2c}{|x|\sqrt{b^2 - 4ac}}, & \text{if } c < 0, \end{cases}$
71. $\int x^3 \sqrt{x^2 + a^2} dx = (\frac{1}{3}x^2 - \frac{2}{15}a^2)(x^2 + a^2)^{3/2},$
72. $\int x^n \sin(ax) dx = -\frac{1}{a} x^n \cos(ax) + \frac{n}{a} \int x^{n-1} \cos(ax) dx,$
73. $\int x^n \cos(ax) dx = \frac{1}{a} x^n \sin(ax) - \frac{n}{a} \int x^{n-1} \sin(ax) dx,$
74. $\int x^n e^{ax} dx = \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx,$
75. $\int x^n \ln(ax) dx = x^{n+1} \left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2} \right),$
76. $\int x^n (\ln ax)^m dx = \frac{x^{n+1}}{n+1} (\ln ax)^m - \frac{m}{n+1} \int x^n (\ln ax)^{m-1} dx.$

$$\begin{array}{llll}
x^1 & x^1 & = & x^{\bar{1}} \\
x^2 & x^2 + x^1 & = & x^{\bar{2}} - x^{\bar{1}} \\
x^3 & x^3 + 3x^2 + x^1 & = & x^{\bar{3}} - 3x^{\bar{2}} + x^{\bar{1}} \\
x^4 & x^4 + 6x^3 + 7x^2 + x^1 & = & x^{\bar{4}} - 6x^{\bar{3}} + 7x^{\bar{2}} - x^{\bar{1}} \\
x^5 & x^5 + 15x^4 + 25x^3 + 10x^2 + x^1 & = & x^{\bar{5}} - 15x^{\bar{4}} + 25x^{\bar{3}} - 10x^{\bar{2}} + x^{\bar{1}} \\
x^{\bar{1}} & x^1 & x^{\bar{1}} & x^1 \\
x^{\bar{2}} & x^2 + x^1 & x^{\bar{2}} & x^2 - x^1 \\
x^{\bar{3}} & x^3 + 3x^2 + 2x^1 & x^{\bar{3}} & x^3 - 3x^2 + 2x^1 \\
x^{\bar{4}} & x^4 + 6x^3 + 11x^2 + 6x^1 & x^{\bar{4}} & x^4 - 6x^3 + 11x^2 - 6x^1 \\
x^{\bar{5}} & x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1 & x^{\bar{5}} & x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1
\end{array}$$

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:

$$\Delta(cu) = c\Delta u, \quad \Delta(u+v) = \Delta u + \Delta v,$$

$$\Delta(uv) = u\Delta v + \mathrm{E} v \Delta u,$$

$$\Delta(x^n) = nx^{n-1},$$

$$\Delta(H_x) = x^{-1}, \quad \Delta(2^x) = 2^x,$$

$$\Delta(c^x) = (c-1)c^x, \quad \Delta(\binom{x}{m}) = \binom{x}{m-1}.$$

Sums:

$$\sum cu \delta x = c \sum u \delta x,$$

$$\sum(u+v) \delta x = \sum u \delta x + \sum v \delta x,$$

$$\sum u \Delta v \delta x = uv - \sum \mathrm{E} v \Delta u \delta x,$$

$$\sum x^n \delta x = \frac{x^{n+1}}{n+1}, \quad \sum x^{-1} \delta x = H_x,$$

$$\sum c^x \delta x = \frac{c^x}{c-1}, \quad \sum \binom{x}{m} \delta x = \binom{x}{m+1}.$$

Falling Factorial Powers:

$$x^{\underline{n}} = x(x-1)\cdots(x-n+1), \quad n > 0,$$

$$x^{\underline{0}} = 1,$$

$$x^{\bar{n}} = \frac{1}{(x+1)\cdots(x+|n|)}, \quad n < 0,$$

$$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$$

Rising Factorial Powers:

$$x^{\overline{n}} = x(x+1)\cdots(x+n-1), \quad n > 0,$$

$$x^{\overline{0}} = 1,$$

$$x^{\overline{n}} = \frac{1}{(x-1)\cdots(x-|n|)}, \quad n < 0,$$

$$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$$

Conversion:

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

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

$$x^n = \sum_{k=1}^n \binom{n}{k} x^k = \sum_{k=1}^n \binom{n}{k} (-1)^{n-k} x^{\bar{k}},$$

$$x^{\underline{n}} = \sum_{k=1}^n \binom{n}{k} (-1)^{n-k} x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \binom{n}{k} x^k.$$

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \dots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$\frac{1}{1-x}$	$= 1 + x + x^2 + x^3 + x^4 + \dots$	$= \sum_{i=0}^{\infty} x^i,$
$\frac{1}{1-cx}$	$= 1 + cx + c^2x^2 + c^3x^3 + \dots$	$= \sum_{i=0}^{\infty} c^i x^i,$
$\frac{1}{1-x^n}$	$= 1 + x^n + x^{2n} + x^{3n} + \dots$	$= \sum_{i=0}^{\infty} x^{ni},$
$\frac{x}{(1-x)^2}$	$= x + 2x^2 + 3x^3 + 4x^4 + \dots$	$= \sum_{i=0}^{\infty} ix^i,$
$x^k \frac{d^n}{dx^n} \left(\frac{1}{1-x} \right)$	$= x + 2^nx^2 + 3^nx^3 + 4^nx^4 + \dots$	$= \sum_{i=0}^{\infty} i^n x^i,$
e^x	$= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots$	$= \sum_{i=0}^{\infty} \frac{x^i}{i!},$
$\ln(1+x)$	$= x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 - \dots$	$= \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i},$
$\ln \frac{1}{1-x}$	$= x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \frac{1}{4}x^4 + \dots$	$= \sum_{i=1}^{\infty} \frac{x^i}{i},$
$\sin x$	$= x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots$	$= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$
$\cos x$	$= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \dots$	$= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},$
$\tan^{-1} x$	$= x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \dots$	$= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},$
$(1+x)^n$	$= 1 + nx + \frac{n(n-1)}{2}x^2 + \dots$	$= \sum_{i=0}^{\infty} \binom{n}{i} x^i,$
$\frac{1}{(1-x)^{n+1}}$	$= 1 + (n+1)x + \binom{n+2}{2}x^2 + \dots$	$= \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,$
$\frac{x}{e^x - 1}$	$= 1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{1}{720}x^4 + \dots$	$= \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},$
$\frac{1}{2x}(1 - \sqrt{1-4x})$	$= 1 + x + 2x^2 + 5x^3 + \dots$	$= \sum_{i=0}^{\infty} \frac{1}{i+1} \binom{2i}{i} x^i,$
$\frac{1}{\sqrt{1-4x}}$	$= 1 + x + 2x^2 + 6x^3 + \dots$	$= \sum_{i=0}^{\infty} \binom{2i}{i} x^i,$
$\frac{1}{\sqrt{1-4x}} \left(\frac{1 - \sqrt{1-4x}}{2x} \right)^n$	$= 1 + (2+n)x + \binom{4+n}{2}x^2 + \dots$	$= \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,$
$\frac{1}{1-x} \ln \frac{1}{1-x}$	$= x + \frac{3}{2}x^2 + \frac{11}{6}x^3 + \frac{25}{12}x^4 + \dots$	$= \sum_{i=1}^{\infty} H_i x^i,$
$\frac{1}{2} \left(\ln \frac{1}{1-x} \right)^2$	$= \frac{1}{2}x^2 + \frac{3}{4}x^3 + \frac{11}{24}x^4 + \dots$	$= \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i},$
$\frac{x}{1-x-x^2}$	$= x + x^2 + 2x^3 + 3x^4 + \dots$	$= \sum_{i=0}^{\infty} F_i x^i,$
$\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2}$	$= F_n x + F_{2n} x^2 + F_{3n} x^3 + \dots$	$= \sum_{i=0}^{\infty} F_{ni} x^i.$

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 $b_i = \sum_{j=0}^i a_j$ then

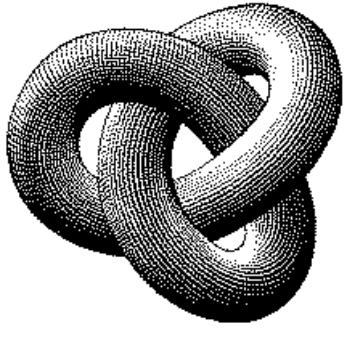
$$B(x) = \frac{1}{1-x} A(x).$$

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	Escher's Knot																																																																																																				
<p>Expansions:</p> $\frac{1}{(1-x)^{n+1}} \ln \frac{1}{1-x} = \sum_{i=0}^{\infty} (H_{n+i} - H_n) \binom{n+i}{i} x^i,$ $x^{\frac{n}{k}} = \sum_{i=0}^{\infty} \binom{n}{i} x^i,$ $\left(\ln \frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty} \binom{i}{n} \frac{n! x^i}{i!},$ $\tan x = \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!},$ $\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty} \frac{\mu(i)}{i^x},$ $\zeta(x) = \prod_p \frac{1}{1-p^{-x}},$ $\zeta^2(x) = \sum_{i=1}^{\infty} \frac{d(i)}{i^x} \quad \text{where } d(n) = \sum_{d n} 1,$ $\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty} \frac{S(i)}{i^x} \quad \text{where } S(n) = \sum_{d n} d,$ $\zeta(2n) = \frac{2^{2n-1} B_{2n} }{(2n)!} \pi^{2n}, \quad n \in \mathbb{N},$ $\frac{x}{\sin x} = \sum_{i=0}^{\infty} (-1)^{i-1} \frac{(4^i - 2)B_{2i}x^{2i}}{(2i)!},$ $\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty} \frac{n(2i+n-1)!}{i!(n+i)!} x^i,$ $e^x \sin x = \sum_{i=1}^{\infty} \frac{2^{i/2} \sin \frac{i\pi}{4}}{i!} x^i,$ $\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty} \frac{(4i)!}{16^i \sqrt{2}(2i)!(2i+1)!} x^i,$ $\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i+1)(2i+1)!} x^{2i}.$																																																																																																					
	Stieltjes Integration																																																																																																				
	<p>If G is continuous in the interval $[a, b]$ and F is nondecreasing then</p> $\int_a^b G(x) dF(x)$ <p>exists. If $a \leq b \leq c$ then</p> $\int_a^c G(x) dF(x) = \int_a^b G(x) dF(x) + \int_b^c G(x) dF(x).$ <p>If the integrals involved exist</p> $\int_a^b (G(x) + H(x)) dF(x) = \int_a^b G(x) dF(x) + \int_a^b H(x) dF(x),$ $\int_a^b G(x) d(F(x) + H(x)) = \int_a^b G(x) dF(x) + \int_a^b G(x) dH(x),$ $\int_a^b c \cdot G(x) dF(x) = \int_a^b G(x) d(c \cdot F(x)) = c \int_a^b G(x) dF(x),$ $\int_a^b G(x) dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x) dG(x).$																																																																																																				
<p>Cramer's Rule</p> <p>If we have equations:</p> $a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1$ $a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2$ $\vdots \quad \vdots \quad \vdots$ $a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$ <p>Let $A = (a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A \neq 0$. Let A_i be A with column i replaced by B. Then</p> $x_i = \frac{\det A_i}{\det A}.$	<p>Fibonacci Numbers</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>00</td><td>47</td><td>18</td><td>76</td><td>29</td><td>93</td><td>85</td><td>34</td><td>61</td><td>52</td></tr> <tr><td>86</td><td>11</td><td>57</td><td>28</td><td>70</td><td>39</td><td>94</td><td>45</td><td>02</td><td>63</td></tr> <tr><td>95</td><td>80</td><td>22</td><td>67</td><td>38</td><td>71</td><td>49</td><td>56</td><td>13</td><td>04</td></tr> <tr><td>59</td><td>96</td><td>81</td><td>33</td><td>07</td><td>48</td><td>72</td><td>60</td><td>24</td><td>15</td></tr> <tr><td>73</td><td>69</td><td>90</td><td>82</td><td>44</td><td>17</td><td>58</td><td>01</td><td>35</td><td>26</td></tr> <tr><td>68</td><td>74</td><td>09</td><td>91</td><td>83</td><td>55</td><td>27</td><td>12</td><td>46</td><td>30</td></tr> <tr><td>37</td><td>08</td><td>75</td><td>19</td><td>92</td><td>84</td><td>66</td><td>23</td><td>50</td><td>41</td></tr> <tr><td>14</td><td>25</td><td>36</td><td>40</td><td>51</td><td>62</td><td>03</td><td>77</td><td>88</td><td>99</td></tr> <tr><td>21</td><td>32</td><td>43</td><td>54</td><td>65</td><td>06</td><td>10</td><td>89</td><td>97</td><td>78</td></tr> <tr><td>42</td><td>53</td><td>64</td><td>05</td><td>16</td><td>20</td><td>31</td><td>98</td><td>79</td><td>87</td></tr> </table> <p>Definitions:</p> $F_i = F_{i-1} + F_{i-2}, \quad F_0 = F_1 = 1,$ $F_{-i} = (-1)^{i-1} F_i,$ $F_i = \frac{1}{\sqrt{5}} (\phi^i - \hat{\phi}^i),$ <p>Cassini's identity: for $i > 0$:</p> $F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$ <p>Additive rule:</p> $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n,$ $F_{2n} = F_n F_{n+1} + F_{n-1} F_n.$ <p>Calculation by matrices:</p> $\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$	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
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																																																																																												
<p>Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius. – William Blake (The Marriage of Heaven and Hell)</p>	<p>The Fibonacci number system: Every integer n has a unique representation</p> $n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$ <p>where $k_i \geq k_{i+1} + 2$ for all i, $1 \leq i < m$ and $k_m \geq 2$.</p>																																																																																																				



File management

Ctrl+M Toggle Tab moves focus

Search and replace

Ctrl+N	New File
Ctrl+O	Open File...
Ctrl+S	Save
Ctrl+Shift+S	Save As...
Ctrl+K S	Save All
Ctrl+F4	Close
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K Enter	Keep preview mode editor open
Ctrl+Tab	Open next
Ctrl+Shift+Tab	Open previous
Ctrl+P	Copy path of active file
Ctrl+K R	Copy active file in Explorer
Ctrl+K O	Show active file in new window/instance

Display

F11	Toggle full screen
Shift+Alt+0	Toggle editor layout (horizontal/vertical)
Ctrl+= - / -	Zoom in/out
Ctrl+B	Toggle Sidebar visibility
Ctrl+Shift+E	Show Explorer / Toggle focus
Ctrl+Shift+F	Show Search
Ctrl+Shift+G	Show Source Control
Ctrl+Shift+D	Show Debug
Ctrl+Shift+X	Show Extensions
Ctrl+Shift+H	Replace in files
Ctrl+Shift+J	Toggle Search details
Ctrl+Shift+U	Show Output panel
Ctrl+Shift+V	Open Markdown preview
Ctrl+K V	Open Markdown preview to the side
Ctrl+K Z	Zen Mode (Esc Esc to exit)

Multi-cursor and selection

Alt+Click	Insert cursor
Ctrl+Alt+ 1 / 1	Insert cursor above / below
Ctrl+U	Undo last cursor operation
Shift+Alt+I	Insert cursor at end of each line selected
Ctrl+L	Select current line
Ctrl+Shift+L	Select all occurrences of current selection
Ctrl+F2	Select all occurrences of current word
Shift+Alt+→	Expand selection
Shift+Alt+←	Shrink selection
Shift+Alt+ (drag mouse)	Column (box) selection
Ctrl+Shift+Alt + (arrow key)	Column (box) selection
Ctrl+Shift+Alt +PgUp/PgDn	Column (box) selection page up/down

Rich languages editing

Ctrl+Space, Ctrl+I	Trigger suggestion
Ctrl+Shift+Space	Trigger parameter hints
Shift+Alt+F	Format document
Ctrl+K Ctrl+F	Format selection
F12	Go to Definition
Alt+F12	Peek Definition
Ctrl+K F12	Open Definition to the side
Ctrl+.	Quick Fix
Shift+F12	Show References
F2	Rename Symbol
Ctrl+K Ctrl+X	Trim trailing whitespace
Ctrl+K M	Change file language

Navigation

Ctrl+T	Show all Symbols
Ctrl+G	Go to Line...
Ctrl+P	Go to File...
Ctrl+Shift+O	Go to Symbol...
Ctrl+Shift+M	Show Problems panel
F8	Go to next error or warning
Shift+F8	Navigate editor group history
Ctrl+Shift+Tab	Move active editor group
Alt+ ← / →	Go back / forward

Other operating systems' keyboard shortcuts and additional unassigned shortcuts available at aka.ms/vscodetkeybindings



Multi-cursor and selection

Editor management

General

Basic editing	
Ctrl+Shift+P	Show Command Palette
Ctrl+P	Quick Open, Go to File...
Ctrl+Shift+N	New window/instance
Ctrl+W	Close window/instance
Ctrl+,	User Settings
Ctrl+K Ctrl+S	Keyboard Shortcuts
Rich languages editing	
Ctrl+Space, Ctrl+I	Trigger suggestion
Ctrl+Shift+Space	Trigger parameter hints
Ctrl+Shift+I	Format document
Ctrl+K Ctrl+F	Format selection
F12	Go to Definition
Ctrl+Shift+F10	Peek Definition
Ctrl+K F12	Open Definition to the side
Ctrl+,	Quick Fix
Shift+F12	Show References
F2	Rename Symbol
Ctrl+K Ctrl+X	Trim trailing whitespace
Ctrl+K M	Change file language

Display

F11	Shift+Alt+O	Toggle full screen
Ctrl+ / -	Zoom in/out	Toggle editor layout (horizontal/vertical)
Ctrl+B	Toggle Sidebar visibility	Show Explorer / Toggle focus
Ctrl+Shift+E	Show Search	Show Search
Ctrl+Shift+F	Show Source Control	Show Source Control
Ctrl+Shift+G	Show Debug	Show Debug
Ctrl+Shift+D	Show Extensions	Show Extensions
Ctrl+Shift+X	Replace in files	Replace in files
Ctrl+Shift+H	Toggle Search details	Toggle Search details
Ctrl+Shift+J	Open new command prompt/terminal	Open new command prompt/terminal
Ctrl+Shift+C	Show Output panel	Show Output panel
Ctrl+K Ctrl+H	Open Markdown preview	Open Markdown preview
Ctrl+Shift+V	Open Markdown preview to the side	Open Markdown preview to the side
Ctrl+K V	Zen Mode (Esc Esc to exit)	Zen Mode (Esc Esc to exit)
Ctrl+K Z		
Search and replace		
Ctrl+F	Find	Find
Ctrl+H	Replace	Replace
F3 / Shift+F3	Find next/previous	Find next/previous
Alt+Enter	Select all occurrences of Find match	Select all occurrences of Find match
Ctrl+D	Add selection to next Find match	Add selection to next Find match
Ctrl+K Ctrl+D	Move last selection to next Find match	Move last selection to next Find match
Navigation		
Ctrl+T	Show all Symbols	Show all Symbols
Ctrl+G	Go to Line...	Go to Line...
Ctrl+P	Go to File...	Go to File...
Ctrl+Shift+O	Go to Symbol...	Go to Symbol...
Ctrl+Shift+M	Show Problems panel	Show Problems panel
F8	Go to next error or warning	Go to next error or warning
Shift+F8	Go to previous error or warning	Go to previous error or warning
Ctrl+Shift+Tab	Navigate editor group history	Navigate editor group history
Ctrl+Alt+-	Go back	Go back
Ctrl+Shift+-	Go forward	Go forward
Ctrl+M	Toggle Tab moves focus	Toggle Tab moves focus

File management

Ctrl+N	New File
Ctrl+O	Open File...
Ctrl+S	Save
Ctrl+Shift+S	Save As...
Ctrl+W	Close
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K Enter	Keep preview mode editor open
Ctrl+Tab	Open next
Ctrl+Shift+Tab	Open previous
Ctrl+K P	Copy path of active file
Ctrl+K R	Reveal active file in Explorer
Ctrl+K O	Show active file in new window-instance
<hr/>	
Debug	
F9	Toggle breakpoint
F5	Start / Continue
F11 / Shift+F11	Step into/out
F10	Step over
Shift+F5	Stop
Ctrl+K Ctrl+I	Show hover
<hr/>	
Integrated terminal	
Ctrl+`	Show integrated terminal
Ctrl+Shift+`	Create new terminal
Ctrl+Shift+C	Copy selection
Ctrl+Shift+V	Paste into active terminal
Ctrl+Shift+↑ / ↓	Scroll up/down
Shift+ PgUp / PgDn	Scroll page up/down
Shift+ Home / End	Scroll to top/bottom

* The Alt+Click gesture may not work on some Linux distributions. You can change the modifier key for the Insert cursor command to Ctrl+Click with the “`editor.multiCursorModifier`” setting.



