



Standard Code Library

antileaf

Last Compiled: 08:02 Jul 02, 2024

目录

1 数学

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)	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 $O(k \log k \log n)$	5
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	10
1.4.1 矩阵乘法	10
1.4.2 高斯消元	10
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	11
1.5 $O(k^2 \log n)$ 齐次线性递推	11
1.6 Berlekamp-Massey 最小递推式	12
1.6.1 优化矩阵快速幂DP	12
1.6.2 求矩阵最小多项式	12
1.6.3 求稀疏矩阵的行列式	12
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	14
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	15
1.10.3 分拆数	15
1.10.4 斯特林数	16
1.10.5 贝尔数	16
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	17
1.11.1 方差	17
1.11.2 min-max 反演	17
1.11.3 单位根反演 (展开整除条件 $[n k]$)	17
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

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 扩展欧几里得 exgcd	23
2.10.1 求通解的方法	23
2.10.2 类欧几里得算法 (直线下整点个数)	23
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

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	39
3.10 最大流	40
3.10.1 Dinic	40
3.10.2 ISAP	40
3.10.3 HLPP 最高标号预流推进	41
3.11 费用流	42
3.11.1 SPFAs 费用流	42
3.11.2 Dijkstra 费用流	43
3.11.3 预流推进费用流 (可处理负环) $O(nm \log C)$	43
3.12 网络流原理	45
3.12.1 最大流	45
3.12.2 最小割	45
3.12.3 上下界网络流	45
3.12.4 常见建图方法	46
3.12.5 例题	46
3.13 Prüfer 序列	46
3.14 弦图相关	47
3.15 其他	47
3.15.1 Stoer-Wagner 全局最小割	47

4 数据结构	49	5.6.1 广义回文树	80
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	82
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	6 动态规划	83
4.2.2 四维偏序	52	6.1 决策单调性 $O(n \log n)$	83
4.3 整体二分	52	6.2 例题	83
4.4 平衡树	53	6.2.1 103388A Assigning Prizes 容斥	83
4.4.1 Treap	53		
4.4.2 无旋 Treap / 可持久化 Treap	54		
4.4.3 Splay	54		
4.5 树链剖分	55		
4.5.1 动态树形 DP (最大权独立集)	55	7 计算几何	85
4.6 树分治	57	7.1 Delaunay 三角剖分	85
4.6.1 动态树分治	57	7.2 最近点对	87
4.6.2 紫荆花之恋	58		
4.7 LCT 动态树	60		
4.7.1 不换根 (弹飞绵羊)	60	8 杂项	88
4.7.2 换根/维护生成树	61	8.1 $O(1)$ 快速乘	88
4.7.3 维护子树信息	62	8.2 Kahan 求和算法 (减少浮点数累加的误差)	88
4.7.4 模板题: 动态QTREE4	63	8.3 Python Decimal	88
4.8 K-D 树	66	8.4 $O(n^2)$ 高精度	88
4.8.1 动态 K-D 树 (定期重构)	66	8.5 笛卡尔树	91
4.9 LCA 最近公共祖先	67	8.6 GarsiaWachs 算法 ($O(n \log n)$ 合并石子)	91
4.9.1 Tarjan LCA $O(n + m)$	67	8.7 常用 NTT 素数及原根	91
4.10 虚树	67	8.8 xorshift	91
4.11 长链剖分	68	8.9 枚举子集	91
4.11.1 梯子剖分	69	8.10 STL	92
4.12 堆	69	8.10.1 vector	92
4.12.1 左偏树	69	8.10.2 list	92
4.12.2 二叉堆	69	8.10.3 unordered_set/map	92
4.13 莫队	70	8.10.4 自定义 Hash	92
4.13.1 莫队二次离线	70	8.11 Public Based DataStructure (PB_DS)	92
4.13.2 带修莫队在线化 $O(n^{\frac{5}{3}})$	71	8.11.1 哈希表	92
4.13.3 莫队二次离线在线化 $O((n + m)\sqrt{n})$	71	8.11.2 堆	92
4.14 常见根号思路	72	8.11.3 平衡树	92
5 字符串	74	8.12 rope	93
5.1 KMP	74	8.13 其他 C++ 相关	93
5.1.1 ex-KMP	74	8.13.1 cmath	93
5.2 AC 自动机	74	8.13.2 algorithm	93
5.3 后缀数组 SA	74	8.13.3 std::tuple	93
5.3.1 倍增	74	8.13.4 complex	93
5.3.2 SA-IS	75	8.14 一些游戏	93
5.3.3 SAMSA	76	8.14.1 德州扑克	93
5.4 后缀平衡树	76	8.14.2 炉石传说	96
5.5 后缀自动机 SAM	76	8.15 OEIS	96
5.5.1 广义后缀自动机	77	8.15.1 计数相关	96
5.5.2 区间本质不同子串计数	77	8.15.2 线性递推数列	96
5.6 回文树 PAM	80	8.15.3 数论相关	97
		8.15.4 其他	97
		8.16 编译选项	97
		8.17 附录: VScode 相关	97
		8.17.1 插件	97
		8.17.2 设置选项	97
		8.17.3 快捷键	97
		8.18 附录: 骂人的艺术—梁实秋	98
		8.19 附录: Cheat Sheet	98



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}

```

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 = (ll)tmp * wn % p;
19         }
20
21         inv[1] = 1;
22         for (int i = 2; i < n; i++)
23             inv[i] = (ll)(p - p / i) * inv[p % i] % p;
24}

```

```

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         else {
60             int inv = qpow(n, p - 2);
61             for (int i = 0; i < n; i++)
62                 c[i] = a[i] * inv % p;
63         }
64     }
65 }

```

1.1.3 任意模数卷积 (MTT)

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     fft(c, n, 1);
6     for (int i = 0; i < n; i++) {
7         int j = (n - i) & (n - 1);
8         a[i] = (c[i] + conj(c[j])) * 0.5;
9         b[i] = (c[i] - conj(c[j])) * -0.5i;
10    }
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 }
```

74 }

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 using poly = vector<int>;
8
9 // u, v 长度要相同, 返回长度是两倍
10 poly poly_calc(const poly& u, const poly& v,
11 → function<int(int, int)> op) {
12     static int a[MAXN], b[MAXN], c[MAXN];
13
14     int n = (int)u.size();
15
16     memcpy(a, u.data(), sizeof(int) * n);
17     fill(a + n, a + n * 2, 0);
18     memcpy(b, v.data(), sizeof(int) * n);
19     fill(b + n, b + n * 2, 0);
20
21     ntt(a, n * 2, 1);
22     ntt(b, n * 2, 1);
23
24     for (int i = 0; i < n * 2; i++)
25         c[i] = op(a[i], b[i]);
26
27     ntt(c, n * 2, -1);
28
29     return poly(c, c + n * 2);
30 }

31
32 // 乘法, 返回长度是两倍
33 poly poly_mul(const poly& u, const poly& v) {
34     return poly_calc(u, v, [](int a, int b) { return
35         → (ll)a * b % p; });
36 }
37
38 // 求逆, 返回长度不变
39 poly poly_inv(const poly& a) {
40     poly c{qpow(a[0], p - 2)}; // 常数项一般都是 1
41
42     for (int k = 2; k <= (int)a.size(); k *= 2) {
43         c.resize(k);
44
45         poly b(a.begin(), a.begin() + k);
46         c = poly_calc(b, c, [](int bi, int ci) {
47             → return ((2 - (ll)bi * ci) % p + p) * ci %
48             → p;
49         });
50         memset(c.data() + k, 0, sizeof(int) * k);
51
52         c.resize(a.size());
53     }
54
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 }
```

```

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
123 // → 理一下
124 poly poly_pow(const poly& a, int k) {
125 |     poly c = poly_ln(a);
126 |     for (int i = 0; i < (int)c.size(); i++)
127 |     |     c[i] = (ll)c[i] * k % p;
128 |     return poly_exp(c);
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 |         |         int mid = (l + r) / 2;

```

```

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

```

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

问题 求 $n!$ ($\text{mod } p$), $n < p$, 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
17    pretreat2(l, mid, k + 1);
18    pretreat2(mid + 1, r, k + 1);
19
20    int N = 1;
21    while (N <= r - l + 1)
22        N *= 2;
23
24    int *hl = th[k + 1] + l * 2, *hr = th[k + 1] + (mid
25      ↪ + 1) * 2;
26
27    memset(A, 0, sizeof(int) * N);
28    memset(B, 0, sizeof(int) * N);
29
30    memcpy(A, hl, sizeof(int) * (mid - l + 2));

```

```

29 |     memcpy(B, hr, sizeof(int) * (r - mid + 1));
30 |
31 |     NTT(A, N, 1);
32 |     NTT(B, N, 1);
33 |
34 |     for (int i = 0; i < N; i++)
35 |         A[i] = (long long)A[i] * B[i] % p;
36 |
37 |     NTT(A, N, -1);
38 |
39 |     for (int i = 0; i <= r - l + 1; i++)
40 |         h[i] = A[i];
41 }
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     NTT(B, N, 1);
82
83     for (int i = 0; i < N; i++)
84         t[i] = (t[i] + (long long)A[i] * B[i]) % p;
85
86     NTT(t, N, -1);
87
88     memcpy(ansf + l, t, sizeof(int) * (r - l + 1));
89 }
90
91 // 主过程
92 // 如果x, y传nullptr表示询问已经存在了qx, qy里
93 void interpolation(int *x, int *y, int n, int *f =
94     nullptr) {
95     static int d[maxn];
96
97     if (x)
98         memcpy(qx, x, sizeof(int) * n);
99     if (y)
100        memcpy(qy, y, sizeof(int) * n);
101
102    NTT(A, N, 1);
103    NTT(B, N, 1);
104
105    for (int i = 0; i < N; i++)
106        A[i] = (long long)A[i] * B[i] % p;
107
108    NTT(A, N, -1);
109
110    for (int i = 0; i < n; i++)
111        ansf[i] = (long long)qy[i] * qpow(ans[i], p -
112            2) % p;
113
114    solve2(0, n - 1, 0);
115
116    if (f)
117        memcpy(f, ansf, sizeof(int) * n);
118 }
```

1.1.7 Bostan-Mori

问题 已知两个 n 次多项式 $f(x)$ 与 $g(x)$, 求 $\frac{f(x)}{g(x)}$ 的第 k 项系数。

做法 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 $O(n \log n \log k)$ 。

```

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        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;
8     |     a = poly_mod(poly_auto_mul(a, a), m).first;
9     |     k /= 2;
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 << ↳ 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项初始值
36         assert(a.size() == f.size()); int ans = 0;
37         for (int i = 0; i < (int)a.size(); i++)
38             ans = (ans + (ll)f[i] * a[i]) % p;
39         return ans;
40     }
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 }

```

```

22     |     |     a[i][j] = t[i];
23 }
24
25 // 对二元多项式做 DFT, 只保留前 n 行中 filter = true 的行
26 void idft_2d(vector<poly>& a, int n,
27   ↪ function<bool(int)> filter) {
28     int ntt_n = (int)a.size(), m = (int)a[0].size();
29
30     for (int j = 0; j < m; j++) {
31       poly t(ntt_n);
32       for (int i = 0; i < ntt_n; i++)
33         t[i] = a[i][j];
34
35       ntt(t.data(), ntt_n, -1);
36
37       for (int i = 0; i < n; i++)
38         a[i][j] = t[i];
39   }
40
41   a.resize(n);
42   for (int i = 0; i < n; i++)
43     if (filter(i))
44       ntt(a[i].data(), m, -1);
45 }
46
47 // 对所有 k ∈ [1, n], 求 [x^n]f^k(x)
48 // 注意这里 n 是最高次数, 即 f 的长度为 n+1
49 poly bostan_mori(const poly& f) {
50   int n = (int)f.size() - 1;
51   vector<poly> a(n + 1), b(n + 1);
52
53   for (int i = 0; i <= n; i++) {
54     a[i] = {0};
55     b[i] = {(p - f[i]) % p};
56   }
57
58   bool a00 = true;
59   int m = 1;
60   while (n) {
61     vector<poly> ac(a), bc(b);
62     for (int i = 0; i <= n; i++) {
63       for (int j = 0; j < m; j++) {
64         if (a00) {
65           int d = i % 2 ? p - b[i][j] : b[i]
66             ↪ [j];
67           (a[i][j] += d) %= p;
68
69         if (i % 2 == 0)
70           b[i][j] = b[i][j] * 2 % p;
71         else
72           b[i][j] = 0;
73       }
74
75     int ntt_n = get_ntt_n(n * 2 + 1);
76
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 - 1)][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
92     bc[i][j] = (ll)bc[i][j] * bc[i + ntt_n
93       ↪ / 2][j] % p;
94
95     bc.resize(ntt_n / 2);
96
97     idft_2d(ac, n + 1, [n](int i) { return i % 2 ==
98       ↪ n % 2; });
99     idft_2d(bc, n / 2 + 1, [](int i) { return true;
100    ↪ });
101
102    for (int i = 0; i <= n; i++) {
103      if (i % 2 == n % 2) {
104        a[i].resize(m * 2);
105
106        for (int j = 1; j < m * 2; j++)
107          (a[i][j] += ac[i][j - 1]) %= p;
108
109        a[i / 2].swap(a[i]);
110
111      }
112
113      for (int i = 0; i <= n / 2; i++) {
114        b[i].swap(b[i * 2]);
115        b[i].resize(m * 2);
116
117        for (int j = 1; j < m * 2; j++)
118          (b[i][j] += bc[i][j - 1]) %= p;
119
120      }
121
122      a00 &= !(n % 2);
123      n /= 2;
124      m *= 2;
125
126      a.resize(n + 1);
127      b.resize(n + 1);
128    }
129
130    for (int i = (int)f.size() - 1; i; i--) {
131      a[0][i] = a[0][i - 1];
132      b[0][i] = b[0][i - 1];
133    }
134    a[0][0] = b[0][0] = 1;
135
136    poly res = poly_mul(a[0], poly_inv(b[0])); // 因为 m
137    ↪ 每次都翻倍, 所以此时 FFT 的长度一定就是 m
138    res.resize(f.size());
139    return res;
140
141
142 // 返回长度和 g 相同
143 // 因为做 DFT 的时候没有二维转一维, 所以还是只需要初始化
144 // → get_ntt_n(n + 1) * 2 就行了
145 poly poly_composite_inversion(const poly& g) {
146   assert(!g[0] && g[1]);
147
148   int n = (int)g.size() - 1;
149
150   int g1_inv = qpow(g[1], p - 2);
151   poly t(n + 1);
152   for (int i = 1; i <= n; i++)
153     t[i] = (ll)g[i] * g1_inv % p;
154
155   poly res = bostan_mori(t);
156
157   for (int i = 0, pw = 1; i < n; i++) {
158     t[i] = (ll)n * inv[n - i] % p * res[n - i] % p
159       ↪ * pw % p;
160     pw = (ll)pw * g1_inv % p;
161
162   }
163
164   int ntt_n = 1;
165   while (ntt_n <= n)
166     ntt_n *= 2;

```

```

152 |     t.resize(ntt_n);
153 |
154 |     t = poly_pow(t, p - inv[n]);
155 |
156 |     poly f(n + 1);
157 |     for (int i = 1; i <= n; i++)
158 |     |     f[i] = (ll)t[i - 1] * g1_inv % p;
159 |     return f;
160 }

```

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 $\frac{R(y)}{Q(x, y)} \equiv R(y)Q(0, y)^{-1} \pmod{x^1}$ 。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, 假设当前层 $Q(-x, y)$ 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(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 - 1)];
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++) {
40        for (int j = m * 2 - 1; j; j--)
41            |     b[i][j] = b[i][j - 1];
42        b[i][0] = 0;
43
44        for (int j = 0; j < m; j++)
45            |     b[i][j] = (b[i][j] + q[i * 2][j] * 2 % p) %
46                → p;
47    }
48
49    b.resize(n / 2 + 1);
50
51    vector<poly> res = bostan_mori_comp(n / 2, a,
52        → move(b));
53
54    vector<poly> t = res;
55    dft_2d(t, ntt_n / 2, m * 2);
56
57    for (int i = 0; i < ntt_n; i++) // 因为 V(x^2, y) 只
58        → 有 x^2k 项, 所以只需做一半
59        for (int j = 0; j < m * 2; j++) // DFT 的后一半
60            → 一定和前一半一样
61            |     c[i][j] = (ll)t[i & (ntt_n / 2 - 1)][j] *
62                → c[i][j] % p;
63
64    idft_2d(c, n + 1, [] (int i) { return true; });
65
66    t = vector<poly>(n + 1);
67    for (int i = 0; i <= n; i++) {
68        t[i].resize(m);
69
70        for (int j = 0; j < m; j++)
71            |     t[i][j] = (ll)c[i][j] * q[i][m - j] % p;
72    }
73
74    return t;
75}

```

```

64 |     |     t[i][j] = c[i][j + m - 1];
65 |
66 |     if (i % 2 == 0)
67 |         for (int j = 0; j < m; j++)
68 |             (t[i][j] += res[i / 2][j + m]) %= p;
69 |
70 |
71 |     return t;
72 }
73
74 // 求 F(G(x)), 长度要相同
75 poly poly_composition(const poly& f, const poly& g) {
76     int n = (int)f.size() - 1;
77
78     poly a(n + 1);
79     for (int i = 0; i <= n; i++)
80         a[i] = f[n - i];
81
82     vector<poly> b(n + 1);
83     for (int i = 0; i <= n; i++)
84         b[i] = {(p - g[i]) % p};
85
86     vector<poly> t = bostan_mori_comp(n, a, b);
87
88     poly res(n + 1);
89     for (int i = 0; i <= n; i++)
90         res[i] = t[i][0];
91
92     return res;
}

```

```

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      int mid = (l + r) / 2;
18
19      solve(l, mid);
20
21      if (l == 0) {
22          for (int i = 1; i < mid; i++) {
23              A[i] = f[i];
24              B[i] = (c[i] + g[i]) % p;
25          }
26          NTT(A, r, 1);
27          NTT(B, r, 1);
28          for (int i = 0; i < r; i++)
29              A[i] = (long long)A[i] * B[i] % p;
30          NTT(A, r, -1);
31
32          for (int i = mid; i < r; i++)
33              f[i] = (f[i] + A[i]) % p;
34      }
35      else {
36          for (int i = 0; i < r - l; i++) {
37              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 }

```

1.1.11 分治 FFT

```

1 void solve(int l, int r) {
2     if (l == r)
3         return;
4
5     int mid = (l + r) / 2;
6
7     solve(l, mid);
8
9     int N = 1;
10    while (N <= r - l + 1)
11        N *= 2;
12
13    for (int i = l; i <= mid; i++)
14        B[i - l] = (long long)A[i] * fac_inv[i] % p;
15    fill(B + mid - l + 1, B + N, 0);
16    for (int i = 0; i < N; i++)
17        C[i] = fac_inv[i];
18
19    NTT(B, N, 1);
20    NTT(C, N, 1);
21
22    for (int i = 0; i < N; i++)
23        B[i] = (long long)B[i] * C[i] % p;
24
25    NTT(B, N, -1);
26
27    for (int i = mid + 1; i <= r; i++)
28        A[i] = (A[i] + B[i - l] * 2 % p * (long
29            long)fac[i] % p) % p;
30
31    solve(mid + 1, r);
}

```

1.1.12 半在线卷积

```

1 void solve(int l, int r) {
2     if (r <= m)
3         return;
4

```

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 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
   ↳ 正/逆变换
3
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                                     ↳ 2] + A[i + j]) % p;
12                 else
13                     A[i + j + k / 2] = (A[i + j + k / 2] -
14                                     ↳ 2] - A[i + j] + p) % p;
15             }
16
17 // 按位异或版本
18 void FWT_xor(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] +
24                                     ↳ 2]) % p;
25                 else
26                     A[i + j] = (A[i + j] - A[i + j + k / 2] +
27                                     ↳ 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
            ↳ 取模时需要用每层除以 2 代替
40         for (int i = 0; i < n; i++)
41             A[i] = A[i] * inv % p;
42     }
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    }
12
13 // and自然就是or反过来
14 void fwt_and(int *a, int n, int tp) {
15     for (int j = 0; (1 << j) < n; j++) {
16         for (int i = 0; i < n; i++) {
17             if (!(i >> j & 1)) {
18                 if (tp > 0)
19                     a[i] += a[i | (1 << j)];
20                 else
21                     a[i] -= a[i | (1 << j)];
22         }
23    }
24
25 // 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 12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     if (ii != i) // 这里没有判是否无解, 如果有可能无
8         → 解的话要判一下
9         for (int j = i; j <= n + 1; j++)
10            swap(A[i][j], A[ii][j]);
11
12    for (int j = 1; j <= n; j++)
13        if (j != i) // 消成对角
14            for (int k = n + 1; k >= i; k--)
15                A[j][k] -= A[j][i] / A[i][i] * A[i]
16                → [k];
17
}

```

```

16
17
18
19
20

```

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                if (!A[i][i])
19                    return 0;
20
21                det = (long long)det * A[i][i] % p;
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];
}
}

```

1.4.5 线性代数知识

行列式:

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

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;
}
}

```

```

14     for (int i = n * 2; i > n; i--) {
15         for (int j = 0; j < n; j++)
16             c[i - 1 - j] = (c[i - 1 - j] + (long
17             ↪ long)c[i] * trans[j]) % p;
18
19         c[i] = 0;
20     }
21
22     c.resize(n + 1);
23     return c;
24 }
25
26 LinearRecurrence(vector<int> &first, vector<int>
27   ↪ &trans) : first(first), trans(trans) {
28     int n = (int)first.size();
29
30     vector<int> a(n + 1);
31     a[1] = 1;
32     bin.push_back(a);
33
34     for (int i = 1; i < 64; i++)
35         bin.push_back(multi(bin[i - 1], bin[i -
36           ↪ 1]));
37
38     int calc(long long k) {
39       int n = (int)first.size();
40
41       vector<int> a(n + 1);
42       a[0] = 1;
43
44       for (int i = 0; i < 64; i++)
45           if (k >> i & 1)
46               a = multi(a, bin[i]);
47
48       int ans = 0;
49       for (int i = 0; i < n; i++)
50           ans = (ans + (long long)a[i + 1] *
51             ↪ first[i]) % p;
52
53     return ans;
54 }
55 }
```

```

17     v = vector<int>(i + 1);
18
19     continue;
20 }
21
22 vector<int> u = v;
23 int val = (long long)(a[i] - tmp + p) *
24   ↪ qpow(delta, p - 2) % p;
25
26 if (v.size() < last.size() + i - k)
27     v.resize(last.size() + i - k);
28
29 (v[i - k - 1] += val) %= p;
30
31 for (int j = 0; j < (int)last.size(); j++) {
32     v[i - k + j] = (v[i - k + j] - (long
33       ↪ long)val * last[j]) % p;
34     if (v[i - k + j] < 0)
35         v[i - k + j] += p;
36 }
37
38 if ((int)u.size() - i < (int)last.size() - k) {
39     last = u;
40     k = i;
41     delta = a[i] - tmp;
42     if (delta < 0)
43         delta += p;
44 }
45
46 for (auto &x : v) // 一般是需要最小递推式的, 所以处理
47   ↪ 一下
48     x = (p - x) % p;
49     v.insert(v.begin(), 1);
50 }
```

1.6 Berlekamp-Massey 最小递推式

如果要求出一个次数为 k 的递推式, 则输入的数列需要至少有 $2k$ 项. 返回的内容满足 $\sum_{j=0}^{m-1} a_{i-j} c_j = 0$, 并且 $c_0 = 1$. 称为最小递推式. 如果不加最后的处理的话, 代码返回的结果会变成 $a_i = \sum_{j=0}^{m-1} c_{j-1} a_{i-j}$, 有时候这样会方便接着跑递推, 需要的话就删掉最后的处理.

(实际上Berlekamp-Massey是对每个前缀都求出了递推式, 但一般没啥用.)

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 (5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.

```

1 vector<int> berlekamp_massey(const vector<int> &a) {
2     vector<int> v, last; // v is the answer, 0-based
3     int k = -1, delta = 0;
4
5     for (int i = 0; i < (int)a.size(); i++) {
6
7         int tmp = 0;
8         for (int j = 0; j < (int)v.size(); j++)
9             tmp = (tmp + (long long)a[i - j - 1] *
10               ↪ v[j]) % p;
11
12         if (a[i] == tmp)
13             continue;
14
15         if (k < 0) {
16             k = i;
17             delta = (a[i] - tmp + p) % p;
18 }
```

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 }
38
39 // 初始化
40 // 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     | pivot(l, e);
89 }
90
91 //转轴操作，本质是在凸包上沿着一条棱移动
92 void pivot(int l, int e) {
93     swap(id[e], id[n + l]);
94     double t = A[l][e];
95     A[l][e] = 1.0;
96
97     for (int i = 0; i <= n; i++)
98     | A[l][i] /= t;
99
100    for (int i = 0; i <= m; i++)
101    | if (i != l) {
102    |     t = A[i][e];
103    |     A[i][e] = 0.0;
104    |     for (int j = 0; j <= n; j++)
105    |         | A[i][j] -= t * A[l][j];
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:

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

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 (14页)。

1.8.2 纳什均衡

纯策略 混合策略 纯策略是指一定会选择某个选项，混合策略是指对每个选项都有一个概率分布 p_i ，以相应的概率选择这个选项。

考虑这样的游戏：有几个人（当然可以是两个）各自独立地做决定，然后同时公布每个人的决定，而每个人的收益和所有人的选择有关。那么纳什均衡就是每个人都决定一个混合策略，使得在其他人都是纯策略的情况下，这个人最坏情况下（也就是说其他人的纯策略最针对他的时候）的收益是最大的。也就是说，收益函数对这个人的混合策略求一个偏导，结果是 0（因为是极大值）。

纳什均衡点可能存在多个，不过在一个双人零和游戏中，纳什均衡点一定唯一存在。

1.8.3 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。

（实际上前一条可以看做 $k = \infty$ 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。（未验证）

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

Copy-and-pasted from the template of Nemesis.

```

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 96 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: $B(x) = \sum_{i \geq 0} \frac{B_i x^i}{i!} = \frac{x}{e^x - 1}$

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

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

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

$B_0 = 1, B_1 = -\frac{1}{2}, B_4 = -\frac{1}{30}, B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, \dots$
(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 long long a[100010];
15 long long p[50005]; // 欧拉五边形数定理
16
17 int main() {
18   p[0] = 1;
19   p[1] = 1;
20 }
```

```

21 | p[2] = 2;
22 | int i;
23 | for (i = 1; i < 50005; i++) { // 递推式系
24 |   → 数 1, 2, 5, 7, 12, 15, 22, 26 ... i*(3*i-1)/2, i*(3*i+1)/2
25 |   a[2 * i] = i * (i * 3 - 1) / 2; // 五边形数
26 |   → 为 1, 5, 12, 22 ... i*(3*i-1)/2
27 |   a[2 * i + 1] = i * (i * 3 + 1) / 2;
28 |
29 | for (i = 3; i < 50005; i++) { //
30 |   → p[n]=p[n-1]+p[n-2]-p[n-5]-p[n-7]+p[12]+p[15]-...
31 |   → +p[n-i*[3i-1]/2]+p[n-i*[3i+1]/2]
32 |   p[i] = 0;
33 |   int j;
34 |   for (j = 2; a[j] <= i; j++) { // 可能为负数, 式
35 |     → 中加 1000007
36 |     if (j & 2)
37 |       p[i] = (p[i] + p[i - a[j]] + 1000007) %
38 |         → 1000007;
39 |     else
40 |       p[i] = (p[i] - p[i - a[j]] + 1000007) %
         → 1000007;
}
| int n;
| while (~scanf("%d", &n))
|   printf("%lld\n", p[n]);
}

```

1.10.4 斯特林数

1. 第一类斯特林数

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

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

求同一行: 分治 FFT $O(n \log^2 n)$, 或者倍增 $O(n \log n)$ (每次都是 $f(x) = g(x)g(x+d)$ 的形式, 可以用 $g(x)$ 反转之后做一个卷积求出后者).

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

求同一列: 用一个轮换的指数生成函数做 k 次幂

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

2. 第二类斯特林数

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

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

求一个: 容斥, 狗都会做

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

求同一行: FFT, 狗都会做

求同一列: 指数生成函数

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

普通生成函数

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

3. 斯特林反演

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

4. 幂的转换

上升幂与普通幂的转换

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

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

下降幂与普通幂的转换

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

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

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

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.

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

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

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

$$B_0 = 1, B_1 = 1, B_2 = 2, B_3 = 5,$$

$$B_4 = 15, B_5 = 52, B_6 = 203, \dots$$

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

递推式:

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

指数生成函数:

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

Touchard 同余

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

1.10.6 欧拉数 (Eulerian Number)

1. 欧拉数

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

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

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

$$\sum_{k=0}^{n-1} \langle \begin{matrix} n \\ k \end{matrix} \rangle = n!$$

$$x^n = \sum_{k=0}^{n-1} \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{x+k}{n}$$

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

2. 二阶欧拉数

$\langle \begin{matrix} n \\ k \end{matrix} \rangle$: 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.

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

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

3. 二阶欧拉数与斯特林数的关系

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

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

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

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

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

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

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

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

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

(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

```

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

```

2.2 线性筛

```

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

4 int prime[maxn / 10], sigma_one[maxn], f[maxn],
5     g[maxn];
// f: 除掉最小质因子后剩下的部分
6 // g: 最小质因子的幂次, 在f(p^k) 比较复杂时很有用,
7 // → 但f(p^k) 可以递推时就可以省略了
8 // 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
9 // → 会被它最小的质因子筛掉
10 bool notp[maxn]; // 顾名思义

11 void get_table(int n) {
12     sigma_one[1] = 1; // 积性函数必有f(1) = 1
13     for (int i = 2; i <= n; i++) {
14         if (!notp[i]) { // 质数情况
15             prime[++prime[0]] = i;
16             sigma_one[i] = i + 1;
17             f[i] = g[i] = 1;
18         }
19
20         for (int j = 1; j <= prime[0] && i * prime[j]
21             <= n; j++) {
22             notp[i * prime[j]] = true;
23
24             if (i % prime[j]) { // 加入一个新的质因子,
25                 // → 这种情况很简单
26                 sigma_one[i * prime[j]] = (long
27                     → long)sigma_one[i] * (prime[j] + 1)
28                     % p;
29                 f[i * prime[j]] = i;
30                 g[i * prime[j]] = 1;
31             } else { // 再加入一次最小质因子, 需要再进行分
32                 // → 类讨论
33                 f[i * prime[j]] = f[i];
34                 g[i * prime[j]] = g[i] + 1;
35                 // 对于f(p^k) 可以直接递推的函数, 这里的
36                 // → 判断可以改成
37                 // i / prime[j] % prime[j] != 0, 这样可
38                 // → 以省下f[] 的空间,
39                 // 但常数很可能会稍大一些
40
41                 if (f[i] == 1) // 质数的幂次, 这
42                     // → 里\sigma_1可以递推
43                     sigma_one[i * prime[j]] =
44                         (sigma_one[i] + i * prime[j]) %
45                         p;
46
47                 // 对于更一般的情况, 可以借助g[] 计
48                 // → 算f(p^k)
49                 else sigma_one[i * prime[j]] = // 否则直
50                     // → 接利用积性, 两半乘起来
51                     (long long)sigma_one[i * prime[j] /
52                         f[i]] * sigma_one[f[i]] % p;
53             }
54         }
55     }
56 }

```

```

41     }
42     }
43     }
44 }

```

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 <= 10 ^ 9, 这时就没必要开long long了, 但
4 // → 记得乘法时强转
5
6 // 常量/全局变量/数组定义
7 const int maxn = 5000005, table_size = 5000000, p =
8     → 1000000007, inv_2 = (p + 1) / 2;
9 bool notp[maxn];
10 int prime[maxn / 20], phi[maxn], tbl[100005];
11 // tbl用来顶替哈希表, 其实开到n ^ {1 / 3} 就够了, 不过保
12 // → 险起见开成\sqrt n比较好
13 long long N;
14
15 // 主函数前面加上这么一句
16 memset(tbl, -1, sizeof(tbl));
17
18 // 线性筛预处理部分略去
19
20 // 杜教筛主过程 总计O(n ^ {2 / 3})
21 // 递归调用自身
22 // 递推式还需具体情况具体分析, 这里以求欧拉函数前缀和
23 // → (mod 10 ^ 9 + 7) 为例
24 int S(long long n) {
25     if (n <= table_size)
26         return phi[n];
27     else if (~tbl[N / n])
28         return tbl[N / n];
29     // 原理: n除以所有可能的数的结果一定互不相同
30
31     int ans = 0;
32     for (long long i = 2, last; i <= n; i = last + 1) {
33         last = n / (n / i);
34         ans = (ans + (last - i + 1) % p * S(n / i)) %
35             p; // 如果n是int范围的话记得强转
36     }
37
38     ans = (n % p * ((n + 1) % p) % p * inv_2 - ans + p)
39     // → % p; // 同上
40     return tbl[N / n] = ans;
41 }

```

2.4 Powerful Number 筛

注意 Powerful Number 筛只能求 积性函数的前缀和.

本质上就是构造一个方便求前缀和的函数, 然后做类似杜教筛的操作.
定义 Powerful Number 表示每个质因子幂次都大于 1 的数, 显然最多有 \sqrt{n} 个.

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

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

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

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

求 $h(d)$ 时要先预处理 $h(p^k)$, 显然有

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

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

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

边界显然就是自然数幂次和, 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 i = 1; i <= sqrt; i++)
69             ans = (ans + (long long)f[i] * (g[1][getid(N / i)] - g[0][getid(N / i)])) % p;
70
71         if (ans < 0)
72             ans += p;
73     }
74 }
```

```

73     return ans;
74 }
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
101 return (long long)(dp[val[0]] + pr[r[val[0]]] -
102     → pr[l[val[0]]] + p) % p;
103
104 int main() {
105     // ios::sync_with_stdio(false);
106     cin >> N;
107
108     sqrtN = (int)sqrt(N);
109
110     get_table(sqrtN);
111
112     for (int i = 1; i <= sqrtN; i++)
113         val[++val[0]] = i;
114
115     for (int i = 1; i <= sqrtN; i++)
116         val[++val[0]] = N / i;
117
118     sort(val + 1, val + val[0] + 1);
119
120     val[0] = unique(val + 1, val + val[0] + 1) - val -
121         → 1;
122
123     int li = 0, ri = 0;
124     for (int j = 1; j <= val[0]; j++) {
125         while (ri < prime_cnt && prime[ri + 1] <=
126             → val[j])
127             ri++;
128
129         while (li <= prime_cnt && (long long)prime[li]
130             → * prime[li] <= val[j])
131             li++;
132
133         l[j] = li - 1;
134         r[j] = ri;
135     }
136
137     cout << (calc1(N) + calc2(N)) % p << endl;
138
139     return 0;
140 }
```

2.6 min25 筛

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
11 void get_val(long long n) {
12     for (int i = 1; i <= sqrtN; i++)
13         val[++val[0]] = i;
14
15     for (int i = sqrtN; i; i--)
16         val[++val[0]] = n / i;
17
18     assert(is_sorted(val + 1, val + val[0] + 1));
19 }
```

```

19 |     val[0] = unique(val + 1, val + val[0] + 1) - val -
20 |     ↪ 1;
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 | int f(long long n) {
27 |     n %= p;
28 |     return n * (n - 1) % p;
29 |
30 |
31 | int g[maxn];
32 | int dp[maxn], prime_sum[maxn];
33 |
34 | void get_dp(function<int(long long)> pw,
35 |     ↪ function<int(long long)> sum) {
36 |     memset(dp, 0, sizeof(dp));
37 |     memset(prime_sum, 0, sizeof(prime_sum));
38 |
39 |     for (int i = 1; i <= prime_cnt; i++)
40 |         prime_sum[i] = (prime_sum[i - 1] +
41 |             ↪ pw(prime[i])) % p;
42 |
43 |     for (int i = 1; i <= val[0]; i++)
44 |         dp[i] = (sum(val[i]) + p - 1) % p;
45 |
46 |     for (int i = 1; i <= prime_cnt; i++) {
47 |         int pi = prime[i];
48 |
49 |         for (int j = val[0]; (long long)pi * pi <=
50 |             ↪ val[j]; j--) {
51 |             int k = getid(val[j] / pi);
52 |
53 |             dp[j] = (dp[j] + (long long)pw(pi) *
54 |                 ↪ (prime_sum[i - 1] - dp[k])) % p;
55 |             if (dp[j] < 0)
56 |                 dp[j] += p;
57 |         }
58 |     }
59 |
60 | void calc(long long n) {
61 |     get_val(n);
62 |
63 |     get_dp([] (long long x) {
64 |         x %= p;
65 |         return x * x % p;
66 |     }, [] (long long n) {
67 |         n %= p;
68 |         return n * (n + 1) % p * (2 * n + 1) % p * ((p
69 |             ↪ + 1) / 6) % p;
70 |     }); // x ^ 2
71 |
72 |     for (int i = 1; i <= val[0]; i++)
73 |         g[i] = dp[i];
74 |
75 |     get_dp([] (long long x) {
76 |         return x % p;
77 |     }, [] (long long n) {
78 |         n %= p;
79 |         return n * (n + 1) / 2 % p;
80 |     }); // x
81 |
82 |     for (int i = 1; i <= val[0]; i++)
83 |         g[i] = (g[i] - dp[i] + p) % p;
84 |
85 |     memset(prime_sum, 0, sizeof(prime_sum));
86 |     for (int i = 1; i <= prime_cnt; i++)
87 |         prime_sum[i] = (prime_sum[i - 1] + f(prime[i]))
88 |             ↪ % p;
89 |
90 |     int S(int i, long long n) {
91 |         if (prime[i] > n || n < 1)
92 |             return 0;
93 |
94 |         int sq = sqrt(n + 0.5);
95 |         int tmp = (g[getid(n)] - prime_sum[i - 1] + p) % p;
96 |
97 |         for (int k = i; k <= prime_cnt && prime[k] <= sq;
98 |             ↪ k++) {
99 |             int pk = prime[k];
100 |             long long pw = pk;
101 |
102 |             for (int c = 1; pw * pk <= n; c++, pw *= pk)
103 |                 tmp = (tmp + (long long)S(k + 1, n / pw) *
104 |                     ↪ f(pw) + f(pw * pk)) % p;
105 |
106 |         }
107 |
108 |         return tmp;
109 |
110 |     }
111 |
112 |     int main() {
113 |         long long n;
114 |         cin >> n;
115 |
116 |         sqrtn = 1; // sqrtn 是全局的
117 |         while ((long long)(sqrtn + 1) * (sqrtn + 1) <= n)
118 |             sqrtn++;
119 |
120 |         get_table(sqrtn);
121 |
122 |         calc(n);
123 |
124 |         int ans = (S(1, n) + 1) % p;
125 |
126 |         cout << ans << endl;
127 |
128 |         return 0;
129 |     }
130 |
131 | }
```

```

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 |         return tmp;
105 |
106 |     }
107 |
108 |     int main() {
109 |         long long n;
110 |         cin >> n;
111 |
112 |         sqrtn = 1; // sqrtn 是全局的
113 |         while ((long long)(sqrtn + 1) * (sqrtn + 1) <= n)
114 |             sqrtn++;
115 |
116 |         get_table(sqrtn);
117 |
118 |         calc(n);
119 |
120 |         int ans = (S(1, n) + 1) % p;
121 |
122 |         cout << ans << endl;
123 |
124 |         return 0;
125 |     }
126 |
127 | }
```

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 }
```

```

25 }
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;

```

```

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
7 // 封装好的函数体
8 // 需要调用solve
9 void factorize(long long n, vector<long long> &v) { // → v用于存分解出来的质因子, 重复的会放多个
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;
27     do
28         p = Pollards_Rho(n);
29     while (!p); // p是任意一个非平凡因子
30
31     if (p == n) {
32         v.push_back(p); // 说明n本身就是质数
33         return;
34     }
35
36     solve(p, v); // 递归分解两半
37     solve(n / p, v);
38
39 // Pollard's Rho主过程

```

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,
   |     |     → b);
}

```

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
15 pi qpow(pi a, int b) {
16     pi ans(1, 0);
17
18     while (b) {
19         if (b & 1)
20             |     ans = ans * a;
21
22         b >= 1;
23         a = a * a;
24     }
25
26     return ans;
27 }
28
29 int qpow(int a, int b) {
30     int ans = 1;
31
32     while (b) {
33         if (b & 1)
34             |     ans = (long long)ans * a % p;
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;
}

```

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
24     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())))

```

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

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

$$\varphi * I = id$$

$$\mu * id = \varphi$$

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

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

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

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

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

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
8 → 一个副本
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
35         scanf("%d%d", &a[i].id, &a[i].v);
36     }
37 }
```

```

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(nlog^2n)
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 → 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
103     for (int i = r; i >= l; i--)
104         e[d][id[d][a[i].id]].w = a[i].v;
105 }
```

```

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
135         for (int i = r; i >= l; i--) {
136             e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边
137             ↪ 改回去
138             e[d][id[d][a[i].id]].vis = false;
139         }
140
141         m = cnt;
142
143     }
144
145 // Kruskal算法  $O(m \log n)$ 
146 // 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
147 // → 撤销即可
148 Long long Kruskal(int m, edge *e) {
149     int tmp = top;
150     long long ans = 0;
151
152     sort(e + 1, e + m + 1); // 比较函数在结构体中定义过
153     ↪ 了
154
155     for (int i = 1; i <= m; i++) {
156         if (findroot(e[i].u) != findroot(e[i].v)) {
157             e[i].vis = true;
158             ans += e[i].w;
159             mergeset(e[i].u, e[i].v);
160         }
161         else
162             e[i].vis = false;
163
164     for (int i = top; i > tmp; i--)
165         cut(stk[i]); // 撤销所有操作
166     top = tmp;
167
168     return ans;
169 }
```

```

168
169
170 // 以下是并查集相关函数
171 int findroot(int x) { // 因为需要撤销, 不写路径压缩
172     while (p[x] != x)
173         x = p[x];
174
175     return x;
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 }
```

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         memset(vis, 0, sizeof(int) * (n + 1));
22
23         mn[rt] = 0;
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     |     for (int i = 1; i <= n; i++) {
31     |         if (alive[i]) {
32     |             if (mn[i] >= inf)
33     |                 return -1; // 不存在最小树形图
34
35         |             ans += mn[i];
36     }
37
38     bool flag = false;
39
40     for (int i = 1; i <= n; i++) {
41         if (!alive[i])
42             continue;
43
44         int x = i;
45         while (x && !vis[x]) {
46             vis[x] = i;
47             x = pr[x];
48         }
49
50         if (x && vis[x] == i) {
51             flag = true;
52             for (int u = x; !ufs[u]; u = pr[u])
53                 ufs[u] = x;
54         }
55     }
56
57     for (int i = 1; i <= m; i++) {
58         e[i].w -= mn[e[i].v];
59
60         if (ufs[e[i].u])
61             e[i].u = ufs[e[i].u];
62         if (ufs[e[i].v])
63             e[i].v = ufs[e[i].v];
64     }
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        int ans = inf;
60        for (int i = 1; i <= n; i++)
61            ans = min(ans, dp[i][(1 << k) - 1]);
62
63        printf("%d\n", ans);
64    }
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     }
40
41     int o = 0;
42     long long ansv = inf; // vertex
43
44     for (int i = 1; i <= n; i++)
45         if (f[i][id[i][n]] * 2 < ansv) {
46             ansv = f[i][id[i][n]] * 2;
47             o = i;
48         }
49
50     int u = 0, v = 0;
51     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                         }
66                     }
67             }
68
69             u = x;
70             v = y;
71
72             disu = tmp / 2 - f[x][id[x]
73                             → [i]];
74             disv = w - disu;
75
76             j = i;
77
78             printf("%lld\n", min(ansv, anse) / 2); // 直径
79
80             memset(d, 63, sizeof(d));
81
82             if (ansv <= anse)
83                 d[o] = 0;
84             else {
85                 d[u] = disu;
86                 d[v] = disv;
87             }
88
89             for (int k = 1; k <= n; k++) { // Dijkstra
90                 int x = 0;
91                 for (int i = 1; i <= n; i++)
92                     if (!vis[i] && d[i] < d[x])
93                         x = i;
94
95                 vis[x] = true;
96                 for (int y = 1; y <= n; y++)
97                     if (g[x][y] && !vis[y]) {
98                         if (d[y] > d[x] + g[x][y]) {
99                             d[y] = d[x] + g[x][y];
100                            pr[y] = x;
101                        }
102                         else if (d[y] == d[x] + g[x][y] &&
103                                     → d[pr[y]] < d[x])
104                             pr[y] = x;
105                     }
106             }
107
108             if (ansv > anse)
109                 vec.emplace_back(u, v);
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     }
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

首先前提是图没有负环.

先任选一个起点 s , 跑一边SPFA, 计算每个点的势 $h_u = d_{s,u}$, 然后将每条边 $u \rightarrow v$ 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 // 结构体定义
8 struct A { // 用来求最短路
9     int x, d;
10
11    A(int x, int d) : x(x), d(d) {}
12
13    bool operator < (const A &a) const {
14        return d > a.d;
15    }
16};

17 struct node { // 左偏树结点
18     int w, i, d; // i: 最后一条边的编号 d: 左偏树附加信息
19     node *lc, *rc;
20
21     node() {}
22
23     node(int w, int i) : w(w), i(i), d(0) {}
24
25     void refresh(){
26         d = rc -> d + 1;
27     }
28 } null[maxn], *ptr = null, *root[maxn];
29
30 struct B { // 维护答案用
31     int x, w; // x是结点编号, w表示之前已经产生的权值
32     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
33
34     B(int x, node *rt, int w) : x(x), w(w), rt(rt) {}
35
36     bool operator < (const B &a) const {
37         return w + rt -> w > a.w + a.rt -> w;
38     }
39 };
40
41 // 全局变量和数组定义
42 vector<int> G[maxn], W[maxn], id[maxn]; // 最开始要存反
43   ↳ 向图, 然后把G清空作为儿子列表
44 bool vis[maxn], used[maxe]; // used表示边是否在最短路树
45   ↳ 上
46 int u[maxe], v[maxe], w[maxe]; // 存下每条边, 注意是有向边
47   ↳ 边
48 int d[maxn], p[maxn]; // p表示最短路树上每个点的父边
49 int n, m, k, s, t; // s, t分别表示起点和终点

```

```

49 // 以下是主函数中较关键的部分
50 for (int i = 0 ; i <= n; i++)
51 | root[i] = null; // 一定要加上!!!
52
53 // (读入& 建反向图)
54
55 Dijkstra();
56
57 // (清空G, w, id)
58
59 for (int i = 1; i <= n; i++)
60 | if (p[i]) {
61 | | used[p[i]] = true; // 在最短路树上
62 | | G[v[p[i]]].push_back(i);
63 | }
64
65 for (int i = 1; i <= m; i++) {
66 | w[i] = d[u[i]] - d[v[i]]; // 现在的w[i] 表示这条边
67 | → 能使路径长度增加多少
68 | if (!used[i])
69 | | root[u[i]] = merge(root[u[i]], newnode(w[i],
70 | | → i));
71 }
72
73 dfs(t);
74
75 priority_queue<B> heap;
76 heap.push(B(s, root[s], 0)); // 初始状态是找贡献最小的边
77 → 加进去
78
79 printf("%d\n", d[s]); // 第1短路需要特判
80 while (--k) { // 其余k-1短路径用二叉堆维护
81 | if (heap.empty())
82 | | printf("-1\n");
83 | else {
84 | | int x = heap.top().x, w = heap.top().w;
85 | | node *rt = heap.top().rt;
86 | | heap.pop();
87 |
88 | | printf("%d\n", d[s] + w + rt->w);
89 |
90 | | if (rt->lc != null || rt->rc != null)
91 | | | heap.push(B(x, merge(rt->lc, rt->rc),
92 | | | → w)); // pop掉当前边,换成另一条贡献大一
93 | | | → 点的边
94 | | | if (root[v[rt->i]] != null)
95 | | | | heap.push(B(v[rt->i], root[v[rt->i]], w
96 | | | | → + rt->w)); // 保留当前边,往后面再接上
97 | | | → 另一条边
98 | }
99 }
100
101 // 主函数到此结束
102
103
104 // Dijkstra预处理最短路 O(m\log n)
105 void Dijkstra() {
106     memset(d, 63, sizeof(d));
107     d[t] = 0;
108     priority_queue<A> heap;
109     heap.push(A(t, 0));
110
111     while (!heap.empty()) {
112         int x = heap.top().x;
113         heap.pop();
114
115         if(vis[x])
116             continue;
117
118         vis[x] = true;
119         for (int i = 0; i < (int)G[x].size(); i++)
120             if(G[x][i].w < d[G[x][i].v])
121                 d[G[x][i].v] = G[x][i].w;
122
123     }
124 }

```

```

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     |     |     heap.push(A(G[x][i], d[G[x][i]]));
118     |     |
119   }
120 }

// dfs求出每个点的堆 总计O(m\log n)
// 需要调用merge, 同时递归调用自身
122 void dfs(int x) {
123     root[x] = merge(root[x], root[v[p[x]]]);
124
125     for (int i = 0; i < (int)G[x].size(); i++)
126     |     dfs(G[x][i]);
127 }

130 // 包装过的new node() O(1)
131 node *newnode(int w, int i) {
132     ***ptr = node(w, i);
133     ptr -> lc = ptr -> rc = null;
134     return ptr;
135 }

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 }

```

```

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                 while (true) {
25                     auto pi = stk[stk_cnt--];
26
27                     if (bccid[pi.first] != bcc_cnt) {
28                         bcc[bcc_cnt].push_back(pi.first);
29                         bccid[pi.first] = bcc_cnt;
30                     }
31                     if (bccid[pi.second] != bcc_cnt) {
32                         bcc[bcc_cnt].push_back(pi.second);
33                         bccid[pi.second] = bcc_cnt;
34                     }
35
36                     if (pi.first == x && pi.second ==
37                         ↪ y)
38                         break;
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     }
51 }

```

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    }
}

```

```

48 }
49
50 void Tarjan(int n) {
51     for (int i = 1; i <= n; i++)
52         if (!dfn[i])
53             dfs(i, 0);
54 }
```

```

20     bad = false;
21
22     break;
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    }
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            if (!vis[C[x][i]]) {
17                vis[C[x][i]] = true;
18                vec.emplace_back(x, i);
19                x = G[x][i];
20            }
21    }
22 }
```

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], cnt;
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;
27
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     }
58 }
```

```

57 |     |     |     |     tmp += ww;
58 |
59 |     |     |     |     if (e[i ^ 1].to == y)
60 |     |     |     |         break;
61 |     |     |
62 |     |     |     addedge(y, cnt, 0);
63 |     |     |
64 |     |     |     sum[cnt] = tmp;
65 |     |     |
66 |     |     }
67 |
68 }
69
70 void dp(int x) {
71
72 |     for (auto o : v[x]) {
73 |         int y = o.first, w = o.second;
74 |         dp(y);
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 |     } else {
93 |         f[x] = sum[x];
94 |         for (auto o : v[x]) {
95 |             int y = o.first;
96 |
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 }

```

```
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 |
32     return ans;
33 }
```

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
6 bool vis[maxn]; // 右半的点是否已被访问
7 int dx[maxn], dy[maxn];
8 int q[maxn];
9
10 bool bfs(int n) {
11     memset(dx, -1, sizeof(int) * (n + 1));
12     memset(dy, -1, sizeof(int) * (n + 1));
13
14     int head = 0, tail = 0;
15     for (int i = 1; i <= n; i++) {
16         if (!girl[i]) {
17             q[tail++] = i;
18             dx[i] = 0;
19         }
20     }
21
22     bool flag = false;
23
24     while (head != tail) {
25         int x = q[head++];
26
27         for (auto y : G[x])
28             if (dy[y] == -1) {
29                 dy[y] = dx[x] + 1;
30
31                 if (boy[y]) {
32                     if (dx[boy[y]] == -1) {
33                         dx[boy[y]] = dy[y] + 1;
34                         q[tail++] = boy[y];
35                     }
36                 } else
37                     flag = true;
38             }
39     }
40
41     return flag;
42 }
43
44 bool dfs(int x) {
45     for (int y : G[x])
46         if (!vis[y] && dy[y] == dx[x] + 1) {
47             vis[y] = true;
48
49             if (dfs(y))
50                 return true;
51         }
52     }
53
54     return false;
55 }
```

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            if (!boy[y] || dfs(boy[y])) {
11                girl[x] = y;
12            }
13        }
14    }
15 }
```

```

47     |     if (boy[y] && !dfs(boy[y]))
48     |     | continue;
49
50     |     girl[x] = y;
51     |     boy[y] = x;
52     |     return true;
53
54 }
55
56     return false;
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 }
79
80     return ans;
81 }
```

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
// 如果要求最大权完美匹配就把不存在的边设为-INF, 否则所有
→边对0取max
5
6 bool visx[maxn], visy[maxn];
7
8 int boy[maxn], girl[maxn], p[maxn], q[maxn], head,
9     →tail; // p : pre
10
11 int n, m, N, e;
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 }
```

```

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
58             long long d = INF;
59             for (int i = 1; i <= N; i++)
60                 if (!visy[i])
61                     d = min(d, slack[i]);
62
63             for (int i = 1; i <= N; i++) {
64                 if (visx[i])
65                     lx[i] -= d;
66
67                 if (visy[i])
68                     ly[i] += d;
69                 else
70                     slack[i] -= d;
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
89         for (int i = 1; i <= N; i++) {
90             memset(slack, 0x3f, sizeof(slack));
91             memset(visx, 0, sizeof(visx));
92             memset(visy, 0, sizeof(visy));
93             bfs(i);
94         }
95
96         long long ans = 0;
97         for (int i = 1; i <= N; i++)
98             |     ans += w[i][girl[i]];
99         return ans;
100    }
```

```

102 // 为了方便贴上主函数
103 int main() {
104
105     scanf("%d%d%d", &n, &m, &e);
106     N = max(n, m);
107
108     while (e--) {
109         int x, y, c;
110         scanf("%d%d%d", &x, &y, &c);
111         w[x][y] = max(c, 0);
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 // 复杂度和常数所限, 1s之内500已经是这个算法的极限了
const int maxn = 505, p = 1000000007; // p可以是任
// 意10^9以内的质数
```

```

10 // 全局数组和变量定义
11 int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn],
12 // → id[maxn], a[maxn];
13 bool row[maxn] = {false}, col[maxn] = {false};
14 int n, m, girl[maxn]; // girl是匹配点, 用来输出方案
15
16 // 为了方便使用, 贴上主函数
17 // 需要调用高斯消元和eliminate
18 int main() {
19     srand(19260817);
20
21     scanf("%d%d", &n, &m); // 点数和边数
22     while (m--) {
23         int x, y;
24         scanf("%d%d", &x, &y);
25         A[x][y] = rand() % p;
26         A[y][x] = -A[x][y]; // Tutte矩阵是反对称矩阵
27     }
28
29     for (int i = 1; i <= n; i++)
30         id[i] = i; // 输出方案用的, 因为高斯消元的时候会
31         → 交换列
32     memcpy(t, A, sizeof(t));
33     Gauss(A, NULL, n);
34
35     m = n;
36     n = 0; // 这里变量复用纯属个人习惯
37
38     for (int i = 1; i <= m; i++)
39         if (A[id[i]][id[i]])
40             a[i+n] = i; // 找出一个极大满秩子矩阵
41
42     for (int i = 1; i <= n; i++)
43         for (int j = 1; j <= n; j++)
44             A[i][j] = t[a[i]][a[j]];
45
46     Gauss(A, B, n);
47
48     for (int i = 1; i <= n; i++)
49         if (!girl[a[i]])
50             for (int j = i + 1; j <= n; j++)
51                 if (!girl[a[j]] && t[a[i]][a[j]] &&
52                     → B[j][i]) {
53                     // 注意上面那句if的写法, 现在t是邻接
54                     // 矩阵的备份,
55                     // 逆矩阵j行i列不为0当且仅当这条边可
56                     // 行
57                     girl[a[i]] = a[j];
58                     girl[a[j]] = a[i];
59
59                     eliminate(i, j);
60                     eliminate(j, i);
61                     break;
62                 }
63
64     printf("%d\n", n / 2);
65     for (int i = 1; i <= m; i++)
66         printf("%d ", girl[i]);
67
68     return 0;
69 }
70
71 // 高斯消元 O(n^3)
72 // 在传入B时表示计算逆矩阵, 传入NULL则只需计算矩阵的秩
73 void Gauss(int A[][maxn], int B[][maxn], int n) {
74     if(B) {
75         memset(B, 0, sizeof(t));
76         for (int i = 1; i <= n; i++)
77             B[i][i] = 1;
78
79         for (int i = 1; i <= n; i++) {
```

```

77     if (!A[i][i]) {
78         for (int j = i + 1; j <= n; j++) {
79             if (A[j][i]) {
80                 swap(id[i], id[j]);
81                 for (int k = i; k <= n; k++)
82                     swap(A[i][k], A[j][k]);
83
84             if (B)
85                 for (int k = 1; k <= n; k++)
86                     swap(B[i][k], B[j][k]);
87             break;
88         }
89
90         if (!A[i][i])
91             continue;
92     }
93
94     int inv = qpow(A[i][i], p - 2);
95
96     for (int j = 1; j <= n; j++) {
97         if (i != j && A[j][i]) {
98             int t = (long long)A[j][i] * inv % p;
99
100            for (int k = i; k <= n; k++)
101                if (A[i][k])
102                    A[j][k] = (A[j][k] - (long
103                                     ↪ long)t * A[i][k]) % p;
104
105            if (B)
106                for (int k = 1; k <= n; k++)
107                    if (B[i][k])
108                        B[j][k] = (B[j][k] - (long
109                                     ↪ long)t * B[i][k]) % p;
110
111     if (B)
112         for (int i = 1; i <= n; i++) {
113             int inv = qpow(A[i][i], p - 2);
114
115             for (int j = 1; j <= n; j++) {
116                 if (B[i][j])
117                     B[i][j] = (long long)B[i][j] * inv
118                                     ↪ % p;
119     }
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     }

```

```

5   vector<int> G[maxn];
6   int girl[maxn], f[maxn], t[maxn], p[maxn], vis[maxn],
7       ↪ tim, q[maxn], head, tail;
8
9
10 // 封装好的主过程 O(nm)
11 int blossom() {
12     int ans = 0;
13
14     for (int i = 1; i <= n; i++)
15         if (!girl[i])
16             ans += bfs(i);
17
18     return ans;
19 }
20
21 // bfs找增广路 O(m)
22 bool bfs(int s) {
23     memset(t, 0, sizeof(t));
24     memset(p, 0, sizeof(p));
25
26     for (int i = 1; i <= n; i++)
27         f[i] = i; // 并查集
28
29     head = tail = 0;
30     q[tail++] = s;
31     t[s] = 1;
32
33     while (head != tail) {
34         int x = q[head++];
35         for (int y : G[x]) {
36             if (findroot(y) == findroot(x) || t[y] ==
37                                     ↪ 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                 }
51             }
52
53             t[girl[y]] = 1;
54             q[tail++] = girl[y];
55         }
56         else if (t[y] == 1) {
57             int z = LCA(x, y);
58
59             shrink(x, y, z);
60             shrink(y, x, z);
61         }
62     }
63
64     return false;
65 }
66
67 // 缩奇环 O(n)
68 void shrink(int x, int y, int z) {
69     while (findroot(x) != z) {
70         p[x] = y;
71         y = girl[x];
72
73         if (t[y] == 2) {

```

3.7.2 带花树

```

1 // 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
2 → 时候并没有高斯消元好写
3 // 当然输出方案要方便很多
4 // 全局数组与变量定义

```

```

75     |     t[y] = 1;
76     |     q[tail++] = y;
77   }
78
79   if (findroot(x) == x)
80     |     f[x] = z;
81   if (findroot(y) == y)
82     |     f[y] = z;
83
84   x = p[y];
85 }
86
87 //暴力找LCA O(n)
88 int LCA(int x, int y) {
89   tim++;
90   while (true) {
91     if (x) {
92       x = findroot(x);
93
94       if (vis[x] == tim)
95         |         return x;
96       else {
97         vis[x] = tim;
98         x = p[girl[x]];
99       }
100    }
101
102   swap(x, y);
103 }
104
105 //并查集的查找 O(1)
106 int findroot(int x) {
107   return x == f[x] ? x : (f[x] = findroot(f[x]));
108 }
109

```

```

25   slack[x] = u;
26 }
27 inline void set_slack(int x){
28   slack[x] = 0;
29   for(int u = 1; u <= n; ++u)
30     |     if(g[u][x].w > 0 && st[u] != x && S[st[u]] -->
31     |       update_slack(u, x);
32 }
33 void q_push(int x){
34   if(x <= n)q.push(x);
35   else for(size_t i = 0; i < flower[x].size(); i+
36     |       -->
37     |       q.push(flower[x][i]);
38 }
39 inline void set_st(int x, int b){
40   st[x]=b;
41   if(x > n) for(size_t i = 0; i <
42     |       flower[x].size(); ++i)
43     |       |       set_st(flower[x][i], b);
44 }
45 inline int get_pr(int b, int xr){
46   int pr = find(flower[b].begin(),
47     |       flower[b].end(), xr) - flower[b].begin();
48   if(pr % 2 == 1){
49     reverse(flower[b].begin() + 1,
50     |       flower[b].end());
51     return (int)flower[b].size() - pr;
52   } else return pr;
53 }
54 inline void set_match(int u, int v){
55   match[u]=g[u][v].v;
56   if(u > n){
57     edge e=g[u][v];
58     int xr = flower_from[u][e.u], pr=get_pr(u,
59     |       -->
60     |       for(int i = 0; i < pr; ++i)
61     |       set_match(flower[u][i], flower[u][i +
62     |           -->
63     |           set_match(xr, v);
64     |           rotate(flower[u].begin(),
65     |               flower[u].begin() + pr, flower[u].end());
66   }
67
68 inline void augment(int u, int v){
69   for(; ; ){
70     int xnv=st[match[u]];
71     set_match(u, v);
72     if(!xnv) return;
73     set_match(xnv, st[pa[xnv]]);
74     u=st[pa[xnv]], v=xnv;
75   }
76
77 inline int get_lca(int u, int v){
78   static int t=0;
79   for(++t; u || v; swap(u, v)){
80     if(u == 0) continue;
81     if(vis[u] == t) return u;
82     vis[u] = t;
83     u = st[match[u]];
84     if(u) u = st[pa[u]];
85   }
86   return 0;
87
88 inline void add_blossom(int u, int lca, int v){
89   int b = n + 1;
90   while(b <= n_x && st[b]) ++b;
91   if(b > n_x) ++n_x;
92   lab[b] = 0, S[b] = 0;
93   match[b] = match[lca];
94   flower[b].clear();
95   flower[b].push_back(lca);
96

```

3.7.3 带权带花树

Copy-and-pasted 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
11   struct edge{
12     int u, v, w;
13     edge() {}
14     edge(int u, int v, int w): u(u), v(v), w(w) {}
15   };
16   int n, n_x;
17   edge g[MAXN * 2 + 1][MAXN * 2 + 1];
18   int lab[MAXN * 2 + 1];
19   int match[MAXN * 2 + 1], slack[MAXN * 2 + 1],
20   |   → st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
21   int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 +
22   |   → 1], vis[MAXN * 2 + 1];
23   vector<int> flower[MAXN * 2 + 1];
24   queue<int> q;
25   inline int e_delta(const edge &e){ // does not work
26     |   → inside blossoms
27     |     return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
28   }
29   inline void update_slack(int u, int x){
30     |     if(!slack[x] || e_delta(g[u][x]) <
31     |       → e_delta(g[slack[x]][x]))
```

```

89     for(int x = u, y; x != lca; x = st[pa[y]]) {
90         flower[b].push_back(x),
91         flower[b].push_back(y = st[match[x]]),
92         q_push(y);
93     }
94     reverse(flower[b].begin() + 1,
95             flower[b].end());
95     for(int x = v, y; x != lca; x = st[pa[y]]) {
96         flower[b].push_back(x),
97         flower[b].push_back(y = st[match[x]]),
98         q_push(y);
99     }
100    set_st(b, b);
101    for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x]
102        → [b].w = 0;
103    for(int x = 1; x <= n; ++x) flower_from[b][x] =
104        → 0;
105    for(size_t i = 0; i < flower[b].size(); ++i){
106        int xs = flower[b][i];
107        for(int x = 1; x <= n_x; ++x)
108            if(g[b][x].w == 0 || e_delta(g[xs][x])
109                → < e_delta(g[b][x]))
110                g[b][x] = g[xs][x], g[x][b] = g[x]
111                    → [xs];
112        for(int x = 1; x <= n; ++x)
113            if(flower_from[xs][x]) flower_from[b]
114                → [x] = xs;
115    }
116    set_slack(b);
117 }
118 inline void expand_blossom(int b){ // S[b] == 1
119     for(size_t i = 0; i < flower[b].size(); ++i)
120         set_st(flower[b][i], flower[b][i]);
121     int xr = flower_from[b][g[b][pa[b]].u], pr =
122         → get_pr(b, xr);
123     for(int i = 0; i < pr; i += 2){
124         int xs = flower[b][i], xns = flower[b][i +
125             → 1];
126         pa[xs] = g[xns][xs].u;
127         S[xs] = 1, S[xns] = 0;
128         slack[xs] = 0, set_slack(xns);
129         q_push(xns);
130     }
131     S[xr] = 1, pa[xr] = pa[b];
132     for(size_t i = pr + 1; i < flower[b].size(); +
133         → i){
134         int xs = flower[b][i];
135         S[xs] = -1, set_slack(xs);
136     }
137     st[b] = 0;
138 }
139 inline bool on_found_edge(const edge &e){
140     int u = st[e.u], v = st[e.v];
141     if(S[v] == -1){
142         pa[v] = e.u, S[v] = 1;
143         int nu = st[match[v]];
144         slack[v] = slack[nu] = 0;
145         S[nu] = 0, q_push(nu);
146     }else if(S[v] == 0){
147         int lca = get_lca(u, v);
148         if(!lca) return augment(u, v), augment(v,
149             → u), true;
150         else add_blossom(u, lca, v);
151     }
152     return false;
153 }
154 inline bool matching(){
155     memset(S + 1, -1, sizeof(int) * n_x);
156     memset(slack + 1, 0, sizeof(int) * n_x);
157     q = queue<int>();
158     for(int x = 1; x <= n_x; ++x)
159         if(st[x] == x && !match[x]) pa[x] = 0,
160             → S[x] = 0, q.push(x);
161     if(q.empty()) return false;
162     for(; ;){
163         while(q.size()){
164             int u = q.front(); q.pop();
165             if(S[st[u]] == 1) continue;
166             for(int v = 1; v <= n; ++v)
167                 if(g[u][v].w > 0 && st[u] != st[v])
168                     → {
169                         if(e_delta(g[u][v]) == 0){
170                             if(on_found_edge(g[u]
171                                 → [v])) return true;
172                         }else update_slack(u, st[v]);
173                     }
174             int d = INF;
175             for(int b = n + 1; b <= n_x; ++b)
176                 if(st[b] == b && S[b] == 1) d = min(d,
177                     → lab[b]/2);
178             for(int x = 1; x <= n_x; ++x)
179                 if(st[x] == x && slack[x]){
180                     if(S[x] == -1) d = min(d,
181                         → e_delta(g[slack[x]][x]));
182                     else if(S[x] == 0) d = min(d,
183                         → e_delta(g[slack[x]][x])/2);
184                 }
185             for(int u = 1; u <= n; ++u){
186                 if(S[st[u]] == 0){
187                     if(lab[u] <= d) return 0;
188                     lab[u] -= d;
189                 }else if(S[st[u]] == 1) lab[u] += d;
190             }
191             for(int b = n + 1; b <= n_x; ++b)
192                 if(st[b] == b){
193                     if(S[st[b]] == 0) lab[b] += d * 2;
194                     else if(S[st[b]] == 1) lab[b] -= d
195                         → * 2;
196                 }
197             q = queue<int>();
198             for(int x = 1; x <= n_x; ++x)
199                 if(st[x] == x && slack[x] &&
200                     → st[slack[x]] != x &&
201                     → e_delta(g[slack[x]][x]) == 0)
202                     if(on_found_edge(g[slack[x]
203                         → [x])) return true;
204             for(int b = n + 1; b <= n_x; ++b)
205                 if(st[b] == b && S[b] == 1 && lab[b] ==
206                     → 0) expand_blossom(b);
207             }
208             return false;
209     }
210     inline pair<long long, int> solve(){
211         memset(match + 1, 0, sizeof(int) * n);
212         n_x = n;
213         int n_matches = 0;
214         long long tot_weight = 0;
215         for(int u = 0; u <= n; ++u) st[u] = u,
216             → flower[u].clear();
217         int w_max = 0;
218         for(int u = 1; u <= n; ++u)
219             for(int v = 1; v <= n; ++v){
220                 flower_from[u][v] = (u == v ? u : 0);
221                 w_max = max(w_max, g[u][v].w);
222             }
223         for(int u = 1; u <= n; ++u) lab[u] = w_max;
224         while(matching()) ++n_matches;
225         for(int u = 1; u <= n; ++u)
226             if(match[u] && match[u] < u)
227                 tot_weight += g[u][match[u]].w;
228         return make_pair(tot_weight, n_matches);
229     }

```

```

210 |     inline void init() {
211 |         for(int u = 1; u <= n; ++u)
212 |             for(int v = 1; v <= n; ++v)
213 |                 g[u][v]=edge(u, v, 0);
214 |
215 }

```

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

```

```

36    |     ufs[i] = anc[i] = i;
37
38    |     dfs(1);
39
40    |     for (int i = n; i > 1; i--) {
41    |         int x = id[i];
42
43    |         for (int y : R[x])
44    |             if (dfn[y]) {
45    |                 findufs(y);
46    |                 if (dfn[sdom[x]] > dfn[sdom[anc[y]]])
47    |                     sdom[x] = sdom[anc[y]];
48    |
49    |             son[sdom[x]].push_back(x);
50    |             ufs[x] = p[x];
51
52    |             for (int u : son[p[x]]) {
53    |                 findufs(u);
54    |                 idom[u] = (sdom[u] == sdom[anc[u]] ? p[x] :
55    |                               → anc[u]);
56    |
57    |             son[p[x]].clear();
58    |
59    }
60
61    |     for (int i = 2; i <= n; i++) {
62    |         int x = id[i];
63
64    |         if (idom[x] != sdom[x])
65    |             idom[x] = idom[idom[x]];
66
67    |         son[idom[x]].push_back(x);
68    |
69    }

```

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, 否则按拓扑序倒序尝试选择每个SCC即可.

由于Tarjan算法的特性, 找到SCC的顺序就是拓扑序倒序, 所以判断完是否有解之后, 每个变量只需要取SCC编号较小的那个.

```

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],
   ↳ i < n ? ' ' : '\n');
8 }

```

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     |     |     |
15     |     }     break;
16   }
17 }
18 // 最后stk中的所有元素就是选中的值
19
20 // dfs
21 bool dfs(int x) {
22   if (vis[x ^ 1])
23     return false;
24
25   if (vis[x])
26     return true;
27
28   vis[x] = true;
29   stk[++top] = x;
30
31   for (int i = 0; i < (int)G[x].size(); i++)
32     if (!dfs(G[x][i]))
33       return false;
34
35   return true;
36 }
```

```

42   return a;
43
44   int flow = 0, f;
45   for (int &i = cur[x]; ~i; i = e[i].prev)
46     if (e[i].cap > 0 && d[e[i].to] == d[x] + 1 &&
47       (f = dfs(e[i].to, min(e[i].cap, a)))) {
48
49     e[i].cap -= f;
50     e[i^1].cap += f;
51     flow += f;
52     a -= f;
53
54     if (!a)
55       break;
56
57   }
58 }
59
60 int Dinic() {
61   int flow = 0;
62   while (bfs(), ~d[t]) {
63     memcpy(cur, last, sizeof(int) * (t + 5));
64     flow += dfs(s, inf);
65   }
66   return flow;
67 }
```

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 int dfs(int x, int a) {
41   if (x == t || !a)
```

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况
2 // → 比较Dinic更优，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 }
```

```

38     for (int i = last[x]; ~i; i = e[i].prev)
39         if (e[i ^ 1].cap && d[e[i].to] == -1) {
40             d[e[i].to] = d[x] + 1;
41             q[tail++] = e[i].to;
42         }
43     }
44 }
45
46 // augment函数 O(n) 沿增广路增广一次，返回增广的流量
47 int augment() {
48     int a = (~0u) >> 1; // INT_MAX
49
50     for (int x = t; x != s; x = e[p[x] ^ 1].to)
51         a = min(a, e[p[x]].cap);
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
58     return a;
59 }
60
61 // 主过程 O(n^2 m)，返回最大流的流量
62 // 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
63 // → 变量记录下来并修改代码
64 int ISAP() {
65     bfs();
66
67     memcpy(cur, last, sizeof(cur));
68
69     int x = s, flow = 0;
70
71     while (d[s] < n) {
72         if (x == t) { // 如果走到了t就增广一次，并返
73             → 回s重新找增广路
74             flow += augment();
75             x = s;
76         }
77
78         bool ok = false;
79         for (int &i = cur[x]; ~i; i = e[i].prev)
80             if (e[i].cap && d[x] == d[e[i].to] + 1) {
81                 p[e[i].to] = i;
82                 x = e[i].to;
83
84                 ok = true;
85                 break;
86             }
87
88         if (!ok) { // 修改距离标号
89             int tmp = n - 1;
90             for (int i = last[x]; ~i; i = e[i].prev)
91                 if (e[i].cap)
92                     tmp = min(tmp, d[e[i].to] + 1);
93
94             if (!--c[d[x]])
95                 break; // gap优化，一定要加上
96
97             c[d[x]] = tmp++;
98             cur[x] = last[x];
99
100            if (x != s)
101                x = e[p[x] ^ 1].to;
102        }
103    }
104
105 // 重要！main函数最前面一定要加上如下初始化
106 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
19 priority_queue<int, vector<int>, cmp> heap;
20
21 void adde(int x, int y, int z) {
22     e[cnte].to = y;
23     e[cnte].cap = z;
24     e[cnte].prev = last[x];
25     last[x] = cnte++;
26 }
27
28 void addedge(int x, int y, int z) {
29     adde(x, y, z);
30     adde(y, x, 0);
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 }

```

```

67     |     |     if (!ex[x])
68     |     |     |     break;
69     |
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
107        while (!heap.empty()) {
108            int x = heap.top();
109            heap.pop();
110            inq[x] = false;
111
112            push(x);
113            if (ex[x]) {
114                if (!--gap[h[x]]) { // gap
115                    for (int i = 1; i <= n; i++)
116                        if (i != s && i != t && h[i] >
117                            ~h[x])
118                            h[i] = n + 1;
119
120                    relabel(x);
121                    ++gap[h[x]];
122                    heap.push(x);
123                    inq[x] = true;
124                }
125
126            return ex[t];
127        }
128
129 //记得初始化
130 memset(last, -1, sizeof(last));

```

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
8     →始化成-1, 不然会死循环
9 bool inq[maxn];
10
11 void spfa(int s) {
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
58     return ans;
59 }
60
61 void add(int x, int y, int c, int w) {
62     e[cnte].to = y;
63     e[cnte].cap = c;
64     e[cnte].w = w;
65
66     e[cnte].prev = last[x];
67     last[x] = cnte++;
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 }
```

```

51 pair<long long, long long> mcmf() {
52     /*
53     spfa();
54     for (int i = 1; i <= t; i++)
55         h[i] = d[i];
56     // 如果初始有负权就像这样跑一遍SPFA预处理
57     */
58
59     long long flow = 0, cost = 0;
60
61     while (dijkstra(), d[t] < 0x3f3f3f3f) {
62         for (int i = 1; i <= t; i++)
63             h[i] += d[i];
64
65         int a = 0x3f3f3f3f;
66
67         for (int x = t; x != s; x = e[p[x] ^ 1].to)
68             a = min(a, e[p[x]].cap);
69
70         flow += a;
71         cost += (long long)a * h[t];
72
73         for (int x = t; x != s; x = e[p[x] ^ 1].to) {
74             e[p[x]].cap -= a;
75             e[p[x] ^ 1].cap += a;
76         }
77
78     }
79
80     return make_pair(flow, cost);
81 }
82
83 // 记得初始化
84 memset(last, -1, sizeof(last));
85 }
```

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;
28     vector<unsigned int> isq, cur;
29     vector<flow_t> ex;
30     vector<cost_t> h;
31 }
```

```

28 mcsFlow(int _N, int _S, int _T): eps(0), N(_N),
29   → S(_S), T(_T), G(_N) {}
30
31 void add_edge(int a, int b, flow_t cap, cost_t
32   → cost) {
33   assert(cap >= 0);
34   assert(a >= 0 && a < N && b >= 0 && b < N);
35
36   if (a == b) {
37     assert(cost >= 0);
38     return;
39   }
40
41   cost *= N;
42   eps = max(eps, abs(cost));
43   G[a].emplace_back(b, cost, cap, G[b].size());
44   G[b].emplace_back(a, -cost, 0, G[a].size() -
45     → 1);
46
47 void add_flow(Edge &e, flow_t f) {
48   Edge &back = G[e.to][e.rev];
49
50   if (!ex[e.to] && f)
51     hs[h[e.to]].push_back(e.to);
52
53   e.f -= f;
54   ex[e.to] += f;
55   back.f += f;
56   ex[back.to] -= f;
57
58 vector<vector<int>> hs;
59 vector<int> co;
60
61 flow_t max_flow() {
62   ex.assign(N, 0);
63   h.assign(N, 0);
64   hs.resize(2 * N);
65   co.assign(2 * N, 0);
66   cur.assign(N, 0);
67   h[S] = N;
68   ex[T] = 1;
69   co[0] = N - 1;
70
71   for (auto &e : G[S])
72     add_flow(e, e.f);
73
74   if (hs[0].size())
75     for (int hi = 0; hi >= 0;) {
76       int u = hs[hi].back();
77       hs[hi].pop_back();
78
79       while (ex[u] > 0) { // discharge u
80         if (cur[u] == G[u].size())
81           h[u] = 1e9;
82
83         for (unsigned int i = 0; i <
84           → G[u].size(); ++i) {
85           auto &e = G[u][i];
86
87           if (e.f && h[u] > h[e.to] +
88             → 1) {
89             h[u] = h[e.to] + 1,
90             → cur[u] = i;
91           }
92
93         if (++co[h[u]], !--co[hi] && hi
94           → < N)
95           for (int i = 0; i < N; ++i)
96             if (hi < h[i] && h[i] <
97               → N) {
98               --co[h[i]];
99               h[i] = N + 1;
100
101             hi = h[u];
102           }
103
104         else if (G[u][cur[u]].f && h[u] ==
105           → h[G[u][cur[u]].to] + 1)
106           add_flow(G[u][cur[u]],
107             → min(ex[u], G[u]
108               → [cur[u]].f));
109
110       else
111         ++cur[u];
112
113     }
114
115   while (hi >= 0 && hs[hi].empty())
116     --hi;
117
118   return -ex[S];
119
120 void push(Edge &e, flow_t amt) {
121   if (e.f < amt)
122     amt = e.f;
123
124   e.f -= amt;
125   ex[e.to] += amt;
126   G[e.to][e.rev].f += amt;
127   ex[G[e.to][e.rev].to] -= amt;
128
129 void relabel(int vertex) {
130   cost_t newHeight = -INFCOST;
131
132   for (unsigned int i = 0; i < G[vertex].size();
133     → ++i) {
134     Edge const &e = G[vertex][i];
135
136     if (e.f && newHeight < h[e.to] - e.c) {
137       newHeight = h[e.to] - e.c;
138       cur[vertex] = i;
139     }
140
141   }
142
143   h[vertex] = newHeight - eps;
144
145 static constexpr int scale = 2;
146
147 pair<flow_t, cost_t> minCostMaxFlow() {
148   cost_t retCost = 0;
149
150   for (int i = 0; i < N; ++i)
151     for (Edge &e : G[i])
152       retCost += e.c * (e.f);
153
154   //find max-flow
155   flow_t retFlow = max_flow();
156   h.assign(N, 0);
157   ex.assign(N, 0);
158   isq.assign(N, 0);
159   cur.assign(N, 0);
160   queue<int> q;
161
162   for (; eps; eps >= scale) {
163     //refine
164     fill(cur.begin(), cur.end(), 0);
165
166     for (int i = 0; i < N; ++i)
167       for (int j = 0; j < N; ++j)
168         if (isq[j] && h[i] < h[j] && h[i] <
169           → N) {
170           --co[h[i]];
171           h[i] = N + 1;
172         }
173
174       if (hs[i].empty())
175         break;
176
177     if (hs[i].empty())
178       break;
179
180   }
181
182   return {retFlow, retCost};
183
184 }
```

```

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 (ex[u] == 0)
199     |                         break;
200
201     |                 }
202
203
204     |             if (eps > 1 && eps >> scale == 0)
205     |                 eps = 1 << scale;
206
207
208     |             for (int i = 0; i < N; ++i)
209     |                 for (Edge &e : G[i])
210     |                     retCost -= e.c * (e.f);
211
212
213     |             return make_pair(retFlow, retCost / 2 / N);
214
215
216     |     int n, m;
217     |     scanf("%d%d", &n, &m);
218
219     |     mcSFlow<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

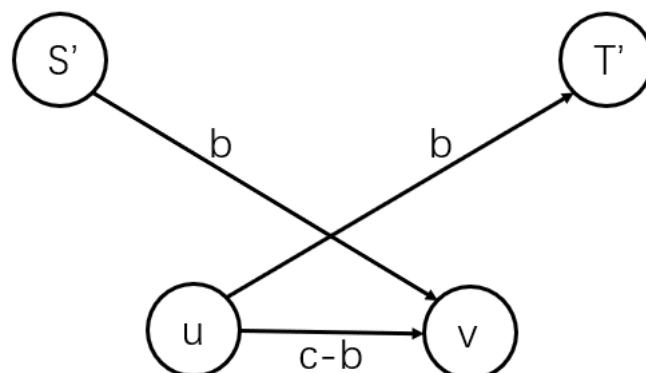
直接按字典序从小到大的顺序依次判断每条边能否在最小割中即可. 如果一条边是可行边, 我们就需要把它删掉, 同时进行退流, $u \rightarrow s$ 和 $t \rightarrow v$ 都退掉等同于这条边容量的流量.

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", &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", &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[ctne - 1].cap;
40        e[ctne - 1].cap = e[ctne - 2].cap = 0;
41        s = src;
42        t = sink;
43
44        printf("%d\n", flow + Dinic());
45    }
46
47    return 0;
}

```

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 Prufer序列

对一棵有 $n \geq 2$ 个结点的树, 它的Prufer编码是一个长为 $n - 2$, 且每个数都在 $[1, n]$ 内的序列.

构造方法: 每次选取编号最小的叶子结点, 记录它的父亲, 然后把它删掉, 直到只剩两个点为止. (并且最后剩的两个点一定有一个是 n 号点.)

相应的, 由Prufer编码重构树的方法: 按顺序读入序列, 每次选取编号最小的且度数为 1 的结点, 把这个点和序列当前点连上, 然后两个点剩余度数同时 -1.

Prufer编码的性质

- 每个至少 2 个结点的树都唯一对应一个Prufer编码. (当然也就可以做无根树哈希.)

- 每个点在Prufer序列中出现的次数恰好是度数 -1. 所以如果给定某些点的度数然后求方案数, 就可以用简单的组合数解决.

最后, 构造和重构直接写都是 $O(n \log n)$ 的, 想优化成线性需要一些技巧.

线性求Prufer序列代码:

```

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 return code;
37 }
```

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  edges.emplace_back(leaf, n - 1);
24  return edges;
25 }
```

```

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 |
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 |     }
61 |
62 |   }
63 |
64 |   int main() {
65 |     cin >> n >> m;
66 |
67 |     if (m < n - 1) {
68 |       cout << 0;
69 |       return 0;
70 |     }
71 |
72 |     for (int i = 1; i <= n; ++i)
73 |       fa[i] = i, siz[i] = 1;
74 |
75 |     for (int i = 1, u, v, w; i <= m; ++i) {
76 |       cin >> u >> v >> w;
77 |
78 |       int fu = find(u), fv = find(v);
79 |       if (fu != fv) {
80 |         if (siz[fu] > siz[fv]) swap(fu, fv);
81 |         fa[fu] = fv, siz[fv] += siz[fu];
82 |       }
83 |
84 |       edge[u][v] += w, edge[v][u] += w;
85 |     }
86 |   }
87 | }
```

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));
```

```
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
3     // →两边分别实际修改的区间长度
4     for (l += n - 1, r += n + 1; l ^ r ^ 1; l >>= 1, r
5     // →>>= 1, len <= 1) {
6         tree[l] += cntl * d, tree[r] += cntr * d;
7         if (~l & 1) tree[l ^ 1] += d * len, mark[l ^ 1]
8         // →+= d, cntl += len;
9         if (r & 1) tree[r ^ 1] += d * len, mark[r ^ 1]
10        // →+= d, cntr += len;
11    }
12
13    for (; l; l >>= 1, r >>= 1)
14        tree[l] += cntl * d, tree[r] += cntr * d;
15
16    int query(int l, int r) {
17        int ans = 0, len = 1, cntl = 0, cntr = 0;
18        for (l += n - 1, r += n + 1; l ^ r ^ 1; l >>= 1, r
19        // →>>= 1, len <= 1) {
20            ans += cntl * mark[l] + cntr * mark[r];
21            if (~l & 1) ans += tree[l ^ 1], cntl += len;
22            if (r & 1) ans += tree[r ^ 1], cntr += len;
23        }
24
25        for (; l; l >>= 1, r >>= 1)
26            ans += cntl * mark[l] + cntr * mark[r];
27
28        return ans;
29    }

```

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, r
3     // →>>= 1) {
4         if (l < N) {
5             tree[l] = max(tree[l << 1], tree[l << 1 |
6             // →1]) + mark[l];
7             tree[r] = max(tree[r << 1], tree[r << 1 |
8             // →1]) + mark[r];
9         }
10    }
11
12    if (~l & 1) tree[l ^ 1] += d;
13    if (r & 1) tree[r ^ 1] += d;
14}

```

```

6     }
7
8     if (~l & 1) {
9         tree[l ^ 1] += d;
10        mark[l ^ 1] += d;
11    }
12    if (r & 1) {
13        tree[r ^ 1] += d;
14        mark[r ^ 1] += d;
15    }
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
40     while (l) {
41         maxl += mark[l];
42         maxr += mark[r];
43
44         l >>= 1;
45         r >>= 1;
46     }
47
48     return max(maxl, maxr);
49}

```

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 int s, t, d;
6
7 void refresh(int l, int r, int o) {
8     if (cover[o])
9         sum[o] = r - l + 1;
10    else
11        sum[o] = sum[lc[o]] + sum[rc[o]];
12}
13
14 void modify(int l, int r, int &o) {
15    if (!o)
16        o = ++seg_cnt;
17
18    if (s <= l && t >= r) {
19        cover[o] += d;
20        refresh(l, r, o);
21    }
22}

```

```

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, rx, 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);
68
69     if (s <= mid)
70         modify(l, mid, o * 2);
71     if (t > mid)
72         modify(mid + 1, r, o * 2 + 1);
73
74     update(o);
75 }

```

```

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 }

```

```

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;

```

4.2 陈丹琦分治

4.2.1 动态图连通性

```

1 vector<pair<int, int>> seg[(1 << 22) + 5];
2
3 int s, t;
4 pair<int, int> d;

```

4.2.2 四维偏序

```

1 // 四维偏序
2
3 void CDQ1(int l, int r) {
4     if (l >= r)
5         return;
6
7     int mid = (l + r) / 2;
8
9     CDQ1(l, mid);
10    CDQ1(mid + 1, r);
11
12    int i = l, j = mid + 1, k = l;
13
14    while (i <= mid && j <= r) {
15        if (a[i].x < a[j].x) {
16            a[i].ins = true;
17            b[k++] = a[i++];
18        }
19        else {
20            a[j].ins = false;
21            b[k++] = a[j++];
22        }
23    }
24
25    while (i <= mid) {
26        a[i].ins = true;
27        b[k++] = a[i++];
28    }
29
30    while (j <= r) {
31        a[j].ins = false;
32        b[k++] = a[j++];
33    }
34
35    copy(b + l, b + r + 1, a + l); // 后面的分治会破坏排序, 所以要复制一份
36
37    CDQ2(l, r);
38}
39
40 void CDQ2(int l, int r) {
41     if (l >= r)
42         return;
43
44     int mid = (l + r) / 2;
45
46     CDQ2(l, mid);
47     CDQ2(mid + 1, r);
48
49     int i = l, j = mid + 1, k = l;
50
51     while (i <= mid && j <= r) {
52         if (b[i].y < b[j].y) {
53             if (b[i].ins)
54                 add(b[i].z, 1); // 树状数组
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);

```

```

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 }

```

4.3 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查询的 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, 查询区间 k 大数).

```

1 int op[maxn], ql[maxn], qr[maxn]; // 1: modify 2: query
2 long long qk[maxn]; // 修改和询问可以一起存
3
4 int ans[maxn];
5
6 void solve(int l, int r, vector<int> v) { // 如果想卡常
    → 可以用数组, 然后只需要传一个数组的l, r; 递归的时候类似归并反过来, 开两个辅助数组, 处理完再复制回去即可
7     if (v.empty())
8         return;
9
10    if (l == r) {
11        for (int i : v)
12            if (op[i] == 2)
13                ans[i] = l;
14
15        return;
16    }
17
18    int mid = (l + r) / 2;
19
20    vector<int> vl, vr;
21
22    for (int i : v) {
23        if (op[i] == 1) {
24            if (qk[i] <= mid)
25                vl.push_back(i);
26            else {
27                update(ql[i], qr[i], 1); // update是区间加
28                vr.push_back(i);
29            }
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
47     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;
}

```

```

27 // 插入键值 期望O(log n)
28 // 需要调用旋转
29 void insert(int x, node *&rt) { // rt为当前结点, 建议调
30     // 用时传入root, 下同
31     if (rt == null) {
32         rt = newnode(x);
33         return;
34     }
35
36     int d = x > rt->key;
37     insert(x, rt->ch[d]);
38     rt->refresh();
39
40     if (rt->ch[d]->p < rt->p)
41         rot(rt, d ^ 1);
42 }
43
44 // 删除一个键值 期望O(log n)
45 // 要求键值必须存在至少一个, 否则会导致RE
46 // 需要调用旋转
47 void erase(int x, node *&rt) {
48     if (x == rt->key) {
49         if (rt->ch[0] != null && rt->ch[1] != null)
50             // 
51             int d = rt->ch[0]->p < rt->ch[1]->p;
52             rot(rt, d);
53             erase(x, rt->ch[d]);
54         else
55             rt = rt->ch[rt->ch[0] == null];
56     }
57     else
58         erase(x, rt->ch[x > rt->key]);
59
60     if (rt != null)
61         rt->refresh();
62 }

```

// 求元素的排名 (严格小于键值的个数 + 1) 期望O(log n)
// 非递归

```

63 int rank(int x, node *rt) {
64     int ans = 1, d;
65     while (rt != null) {
66         if ((d = x > rt->key))
67             ans += rt->ch[0]->size + 1;
68
69         rt = rt->ch[d];
70     }
71
72     return ans;
73 }

```

// 返回排名第k(从1开始)的键值对应的指针 期望O(log n)
// 非递归

```

74 node *kth(int x, node *rt) {
75     int d;
76     while (rt != null) {
77         if (x == rt->ch[0]->size + 1)
78             return rt;
79
80         if ((d = x > rt->ch[0]->size))
81             x -= rt->ch[0]->size + 1;
82
83         rt = rt->ch[d];
84     }
85
86     return rt;
87 }

```

4.4 平衡树

pb_ds 平衡树参见 8.11.Public Based DataStructure (PB_DS) (第 92 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
18 // 重要! 在主函数最开始一定要加上以下预处理:
19 null->ch[0] = null->ch[1] = null;
20 null->size = 0;
21
22 // 伪构造函数 O(1)
23 // 为了方便, 在结点类外面再定义一个伪构造函数
24 node *newnode(int x) { // 键值为x
25     *++ptr = node(x);
26     ptr->ch[0] = ptr->ch[1] = null;
27     return ptr;
}

```

```

95 // 返回前驱 (最大的比给定键值小的键值) 对应的指针 期
96 // → 望 O(log n)
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
112 // → 望 O(log n)
113 // 非递归
114 node *succ(int x, node *rt) {
115     node *y = null;
116     int d;
117
118    while (rt != null) {
119        if ((d = x < rt -> key))
120            y = rt;
121
122        rt = rt -> ch[d ^ 1];
123    }
124
125    return y;
126}
127
128 // 旋转 (Treap版本) O(1)
129 // 平衡树基础操作
130 // 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
131 void rot(node *&x, int d) { // x为被转下去的结点, 会被修
132     // 改以维护树结构
133     node *y = x -> ch[d ^ 1];
134
135     x -> ch[d ^ 1] = y -> ch[d];
136     y -> ch[d] = x;
137
138     x -> refresh();
139     (x = y) -> refresh();
}

```

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     return pi;
64 }
65
66 // 记得初始化null
67 int main() {
68     for (int i = 0; i <= n; i++)
69         null[i].ch[0] = null[i].ch[1] = null;
70     null -> size = 0;
71
72     // do something
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 |     |
13 |         return;
14 |
15 |         ch[0] -> rev ^= true;
16 |         ch[1] -> rev ^= true;
17 |         swap(ch[0], ch[1]);
18 |
19 |     rev=false;
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 |     rot(x -> p, dir(x) ^ 1);
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 |
83 |     null -> ch[0] = null -> ch[1] = null -> p = null;
84 |
85 |     scanf("%d%d", &n, &m);
86 |     root = null + n + 1;
87 |     root -> ch[0] = root -> ch[1] = root -> p = null;
88 |
89 |     for (int i = 1; i <= n; i++) {
90 |         null[i].ch[1] = null[i].p = null;
91 |         null[i].ch[0] = root;
92 |         root -> p = null + i;
93 |         (root = null + i) -> refresh();
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() {}
12
13     void push(int x) {
14         q.push(x);
15     }
16
17     void erase(int x) {
18         t.push(x);
19     }
20
21     int top() {
22         while (!t.empty() && q.top() == t.top()) {
23             q.pop();
24             t.pop();
25         }
26
27         return q.top();
28     }
29 } heap;
30
31 int pool[maxm][2][2], (*pt)[2][2] = pool;
32
33 void merge(int a[2][2], int b[2][2]) {
34     static int c[2][2];
35     memset(c, 0, sizeof(c));
36
37     for (int i = 0; i < 2; i++)
38         for (int j = 0; j < 2; j++)
39             for (int k = 0; k < 2; k++)
40                 if (!(j && k))
41                     for (int t = 0; t < 2; t++)
42                         c[i][j] += a[i][t] * b[t][k];
43
44     for (int i = 0; i < 2; i++)
45         for (int j = 0; j < 2; j++)
46             pt[i][j] = c[i][j];
47 }
```

```

41 |     |     |     |     |     c[i][t] = max(c[i][t], a[i][j]
42 |     |     |     |     ↪ + b[k][t]);
43 |     memcpy(a, c, sizeof(c));
44 |
45 vector<pair<int, int>> tw;
46
47 struct seg_tree {
48     int (*tr)[2][2], n;
49
50     int s, d[2];
51
52     seg_tree() {}
53
54     void update(int o) {
55         memcpy(tr[o], tr[o * 2], sizeof(int) * 4);
56         merge(tr[o], tr[o * 2 + 1]);
57     }
58
59     void build(int l, int r, int o) {
60         if (l == r) {
61             tr[o][0][0] = tw[l - 1].first;
62             tr[o][0][1] = tr[o][1][0] = -inf;
63             tr[o][1][1] = tw[l - 1].second;
64
65             return;
66         }
67
68         int mid = (l + r) / 2;
69
70         build(l, mid, o * 2);
71         build(mid + 1, r, o * 2 + 1);
72
73         update(o);
74     }
75
76     void modify(int l, int r, int o) {
77         if (l == r) {
78             tr[o][0][0] = d[0];
79             tr[o][0][1] = tr[o][1][0] = -inf;
80             tr[o][1][1] = d[1];
81
82             return;
83         }
84
85         int mid = (l + r) / 2;
86
87         if (s <= mid)
88             modify(l, mid, o * 2);
89         else
90             modify(mid + 1, r, o * 2 + 1);
91
92         update(o);
93     }
94
95     int getval() {
96         int ans = 0;
97         for (int i = 0; i < 2; i++)
98             for (int j = 0; j < 2; j++)
99                 ans = max(ans, tr[1][i][j]);
100
101         return ans;
102     }
103
104     pair<int, int> getpair() {
105         int ans[2] = {0};
106         for (int i = 0; i < 2; i++)
107             for (int j = 0; j < 2; j++)
108                 ans[i] = max(ans[i], tr[1][i][j]);
109
110         return make_pair(ans[0], ans[1]);
111     }
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
155 void dfs2(int x) {
156     if (x == son[p[x]])
157         top[x] = top[p[x]];
158     else
159         top[x] = x;
160
161     for (int y : G[x])
162         if (y != p[x])
163             dfs2(y);
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         heap.push(seg[x].getval());
184     }
185 }
```

```

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     scanf("%d%d", &n, &m);
217
218     for (int i = 1; i <= n; i++)
219         scanf("%d", &w[i]);
220
221     for (int i = 1; i < n; i++) {
222         int x, y;
223         scanf("%d%d", &x, &y);
224
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
236         modify(x, dat);
237
238         printf("%d\n", heap.top());
239     }
240
241     return 0;
242 }
```

4.6 树分治

4.6.1 动态树分治

// 为了减小常数，这里采用bfs写法，实测预处理比dfs快将近一半
// 以下以维护一个点到每个黑点的距离之和为例

```

4 // 全局数组定义
5 vector<int> G[maxn], W[maxn];
6 int size[maxn], son[maxn], q[maxn];
7 int p[maxn], depth[maxn], id[maxn][20], d[maxn][20]; // ← id是对应层所在子树的根
8 int a[maxn], ca[maxn], b[maxn][20], cb[maxn][20]; // 维护距离和用的
9 bool vis[maxn], col[maxn];
10
11 // 建树 总计 O(n log n)
12 // 需要调用找重心和预处理距离，同时递归调用自身
13 void build(int x, int k, int s, int pr) { // 结点，深度，← 连通块大小，点分树上的父亲
14     x = getcenter(x, s);
15     vis[x] = true;
16     depth[x] = k;
17     p[x] = pr;
18
19     for (int i = 0; i < (int)G[x].size(); i++)
20         if (!vis[G[x][i]]) {
21             d[G[x][i]][k] = W[x][i];
22             p[G[x][i]] = x;
23
24             getdis(G[x][i], k, G[x][i]); // bfs每个子树，→ 预处理距离
25         }
26
27     for (int i = 0; i < (int)G[x].size(); i++)
28         if (!vis[G[x][i]])
29             build(G[x][i], k + 1, size[G[x][i]], x); // → 递归建树
30     }
31
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 → 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 }
```

```

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
86             // 到子问题大小
87     }
88
89 // 修改 O(\log n)
90 void modify(int x) {
91     if (col[x])
92         ca[x]--;
93     else
94         ca[x]++;
95
96     for (int u = p[x], k = depth[x] - 1; u; u = p[u],
97          ~k--) {
98         if (col[x]) {
99             a[u] -= d[x][k];
100            ca[u]--;
101
102             b[id[x][k]][k] -= d[x][k];
103             cb[id[x][k]][k]--;
104         } else {
105             a[u] += d[x][k];
106             ca[u]++;
107
108             b[id[x][k]][k] += d[x][k];
109             cb[id[x][k]][k]++;
110         }
111     }
112     col[x] ^= true;
113 }
114
115 // 询问 O(\log n)
116 int query(int x) {
117     int ans = a[x]; // 特判自己是重心的那层
118
119     for (int u = p[x], k = depth[x] - 1; u; u = p[u],
120          ~k--)
121         ans += a[u] - b[id[x][k]][k] + d[x][k] * (ca[u]
122             ~ - cb[id[x][k]][k]);
123
124     return ans;
125 }
```

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 vector<node*> pool;
23
24 node *newnode(int val) {
25     node *x;
26
27     if (!pool.empty()) {
28         x = pool.back();
29         pool.pop_back();
30     } else
31         x = ++pt;
32
33     *x = node(val);
34     x -> ch[0] = x -> ch[1] = null;
35
36     return x;
37 }
38
39
40 void rot(node *&x, int d) {
41     node *y = x -> ch[d ^ 1];
42     x -> ch[d ^ 1] = y -> ch[d];
43     y -> ch[d] = x;
44
45     x -> update();
46     (x == y) -> update();
47 }
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 void destroy(node *x) {
80     if (x == null)
81         return;
82
83     pool.push_back(x);
84     destroy(x -> ch[0]);
85     destroy(x -> ch[1]);
86 }
87
88 struct my_tree { // 封装了一下，如果不卡内存直接换成PBDS就好了
89     node *rt;
90
91     my_tree() : rt(null) {}
92
93     void clear() {
94         ::destroy(rt);
95         rt = null;
96     }
97
98     void insert(int x) {
99         ::insert(rt, x);
100    }
101
102    int order_of_key(int x) { // less than x
103        return ::get_order(rt, x);
104    }
105 } tr[maxn], tre[maxn][maxk];
106
107 vector<pair<int, int>> G[maxn];
108
109 int p[maxn], depth[maxn], d[maxn][maxk], rid[maxn]
110     ↪ [maxk];
111 int sz[maxn], siz[maxn][maxk], q[maxn];
112 bool vis[maxn];
113
114 int w[maxn];
115
116 void destroy(int o) {
117     int head = 0, tail = 0;
118     q[tail++] = o;
119     vis[o] = false;
120
121     while (head != tail) {
122         int x = q[head++];
123         tr[x].clear();
124
125         for (int i = depth[o]; i <= depth[x]; i++) {
126             tre[x][i].clear();
127             d[x][i] = rid[x][i] = siz[x][i] = 0;
128         }
129
130         for (auto pi : G[x]) {
131             int y = pi.first;
132
133             if (vis[y] && depth[y] >= depth[o]) {
134                 vis[y] = false;
135                 q[tail++] = y;
136             }
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             if (!vis[y] && y != p[x] && sz[y] * 2 > s)
169                 ↪ {
170                     x = y;
171                     ok = true;
172                     break;
173                 }
174
175         if (!ok)
176             break;
177     }
178
179     return x;
180 }
181
182 void getdis(int st, int o, int k) {
183     int head = 0, tail = 0;
184     q[tail++] = st;
185
186     while (head != tail) {
187         int x = q[head++];
188         sz[x] = 1;
189         rid[x][k] = st;
190
191         tr[o].insert(d[x][k] - w[x]);
192         tre[st][k].insert(d[x][k] - w[x]);
193
194         for (auto pi : G[x]) {
195             int y = pi.first, val = pi.second;
196
197             if (!vis[y] && y != p[x]) {
198                 p[y] = x;
199                 d[y][k] = d[x][k] + val;
200                 q[tail++] = y;
201             }
202         }
203     }
204
205     for (int i = tail - 1; i; i--)
206         sz[p[q[i]]] += sz[q[i]];
207
208     siz[st][k] = sz[st];
209 }
210
211 void rebuild(int x, int s, int pr) {
212     x = getcenter(x, s);
213     vis[x] = true;
214     p[x] = pr;
215     depth[x] = depth[pr] + 1;
216     sz[x] = s;
217
218     tr[x].insert(-w[x]);
219 }
```

```

220 for (auto pi : G[x]) {
221     int y = pi.first, val = pi.second;
222
223     if (!vis[y]) {
224         p[y] = x;
225         d[y][depth[x]] = val;
226         getdis(y, x, depth[x]);
227     }
228 }
229
230 for (auto pi : G[x]) {
231     int y = pi.first;
232
233     if (!vis[y])
234         rebuild(y, sz[y], x);
235 }
236 }
237
238 long long add_node(int x, int nw) { // nw是边权
239     depth[x] = depth[p[x]] + 1;
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 }
```

```

47 }
48
49 // Cut 均摊 $O(\log n)$ 
50 // 把x与其父亲的连边断掉
51 // x可以是所在树的根节点, 这时此操作没有任何实质效果
52 // 需要调用access和splay
53 void cut(node *x) {
54     access(x);
55     splay(x);
56
57     x->ch[0] -> p = null;
58     x->ch[0] = null;
59
60     x->refresh();
61 }
62
63 // 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 }
97
98
99 // Cut 均摊 $O(\log n)$ 
100 // 把x与其父亲的连边断掉
101 // x可以是所在树的根节点, 这时此操作没有任何实质效果
102 // 需要调用access和splay
103 void cut(node *x) {
104     access(x);
105     splay(x);
106
107     x->ch[0] -> p = null;
108     x->ch[0] = null;
109
110     x->refresh();
111 }
112
113 // Splay 均摊 $O(\log n)$ 
114 // 需要调用旋转
115 void splay(node *x) {
116     while (!isroot(x)) {
117         if (pos != -1)
118             rev ^= true;
119
120         ch[0] -> rev ^= true;
121         ch[1] -> rev ^= true;
122         swap(ch[0], ch[1]);
123
124         if (pos != -1)
125             pos ^= 1;
126
127         rev = false;
128     }
129
130     void refresh() {
131         mx = key;
132         pos = -1;
133         if (ch[0] -> mx > mx) {
134             mx = ch[0] -> mx;
135             pos = 0;
136         }
137         if (ch[1] -> mx > mx) {
138             mx = ch[1] -> mx;
139             pos = 1;
140         }
141     }
142
143     void init(node *x, int k) {
144         x->ch[0] = x->ch[1] = x->p = null;
145         x->key = x->mx = k;
146     }
147
148     void rot(node *x, int d) {
149         node *y = x->ch[d ^ 1];
150         if ((x->ch[d ^ 1] = y->ch[d]) != null)
151             y->ch[d]->p = x;
152
153         y->p = x->p;
154         if (!isroot(x))
155             x->p->ch[dir(x)] = y;
156
157         (y->ch[d] = x)->p = y;
158
159         x->refresh();
160         y->refresh();
161     }
162
163     void splay(node *x) {
164         x->pushdown();
165
166         while (!isroot(x)) {
167             if (!isroot(x->p))
168                 x->p->p->pushdown();
169             x->p->pushdown();
170             x->pushdown();
171
172             if (isroot(x->p)) {
173                 rot(x->p, dir(x) ^ 1);
174                 break;
175             }
176
177             if (dir(x) == dir(x->p))
178                 rot(x->p->p, dir(x->p) ^ 1);
179             else
180                 rot(x->p, dir(x) ^ 1);
181             rot(x->p, dir(x) ^ 1);
182         }
183
184         node *access(node *x) {
185             node *y = null;
186
187             if (!rev)
188
189             if (rev)
190
191             if (rev)
192
193             if (rev)
194
195             if (rev)
196
197             if (rev)
198
199             if (rev)
200
201             if (rev)
202
203             if (rev)
204
205             if (rev)
206
207             if (rev)
208
209             if (rev)
210
211             if (rev)
212
213             if (rev)
214
215             if (rev)
216
217             if (rev)
218
219             if (rev)
220
221             if (rev)
222
223             if (rev)
224
225             if (rev)
226
227             if (rev)
228
229             if (rev)
230
231             if (rev)
232
233             if (rev)
234
235             if (rev)
236
237             if (rev)
238
239             if (rev)
240
241             if (rev)
242
243             if (rev)
244
245             if (rev)
246
247             if (rev)
248
249             if (rev)
250
251             if (rev)
252
253             if (rev)
254
255             if (rev)
256
257             if (rev)
258
259             if (rev)
260
261             if (rev)
262
263             if (rev)
264
265             if (rev)
266
267             if (rev)
268
269             if (rev)
270
271             if (rev)
272
273             if (rev)
274
275             if (rev)
276
277             if (rev)
278
279             if (rev)
280
281             if (rev)
282
283             if (rev)
284
285             if (rev)
286
287             if (rev)
288
289             if (rev)
290
291             if (rev)
292
293             if (rev)
294
295             if (rev)
296
297             if (rev)
298
299             if (rev)
300
301             if (rev)
302
303             if (rev)
304
305             if (rev)
306
307             if (rev)
308
309             if (rev)
310
311             if (rev)
312
313             if (rev)
314
315             if (rev)
316
317             if (rev)
318
319             if (rev)
320
321             if (rev)
322
323             if (rev)
324
325             if (rev)
326
327             if (rev)
328
329             if (rev)
330
331             if (rev)
332
333             if (rev)
334
335             if (rev)
336
337             if (rev)
338
339             if (rev)
340
341             if (rev)
342
343             if (rev)
344
345             if (rev)
346
347             if (rev)
348
349             if (rev)
350
351             if (rev)
352
353             if (rev)
354
355             if (rev)
356
357             if (rev)
358
359             if (rev)
360
361             if (rev)
362
363             if (rev)
364
365             if (rev)
366
367             if (rev)
368
369             if (rev)
370
371             if (rev)
372
373             if (rev)
374
375             if (rev)
376
377             if (rev)
378
379             if (rev)
380
381             if (rev)
382
383             if (rev)
384
385             if (rev)
386
387             if (rev)
388
389             if (rev)
390
391             if (rev)
392
393             if (rev)
394
395             if (rev)
396
397             if (rev)
398
399             if (rev)
400
401             if (rev)
402
403             if (rev)
404
405             if (rev)
406
407             if (rev)
408
409             if (rev)
410
411             if (rev)
412
413             if (rev)
414
415             if (rev)
416
417             if (rev)
418
419             if (rev)
420
421             if (rev)
422
423             if (rev)
424
425             if (rev)
426
427             if (rev)
428
429             if (rev)
430
431             if (rev)
432
433             if (rev)
434
435             if (rev)
436
437             if (rev)
438
439             if (rev)
440
441             if (rev)
442
443             if (rev)
444
445             if (rev)
446
447             if (rev)
448
449             if (rev)
450
451             if (rev)
452
453             if (rev)
454
455             if (rev)
456
457             if (rev)
458
459             if (rev)
460
461             if (rev)
462
463             if (rev)
464
465             if (rev)
466
467             if (rev)
468
469             if (rev)
470
471             if (rev)
472
473             if (rev)
474
475             if (rev)
476
477             if (rev)
478
479             if (rev)
480
481             if (rev)
482
483             if (rev)
484
485             if (rev)
486
487             if (rev)
488
489             if (rev)
490
491             if (rev)
492
493             if (rev)
494
495             if (rev)
496
497             if (rev)
498
499             if (rev)
500
501             if (rev)
502
503             if (rev)
504
505             if (rev)
506
507             if (rev)
508
509             if (rev)
510
511             if (rev)
512
513             if (rev)
514
515             if (rev)
516
517             if (rev)
518
519             if (rev)
520
521             if (rev)
522
523             if (rev)
524
525             if (rev)
526
527             if (rev)
528
529             if (rev)
530
531             if (rev)
532
533             if (rev)
534
535             if (rev)
536
537             if (rev)
538
539             if (rev)
540
541             if (rev)
542
543             if (rev)
544
545             if (rev)
546
547             if (rev)
548
549             if (rev)
550
551             if (rev)
552
553             if (rev)
554
555             if (rev)
556
557             if (rev)
558
559             if (rev)
560
561             if (rev)
562
563             if (rev)
564
565             if (rev)
566
567             if (rev)
568
569             if (rev)
570
571             if (rev)
572
573             if (rev)
574
575             if (rev)
576
577             if (rev)
578
579             if (rev)
580
581             if (rev)
582
583             if (rev)
584
585             if (rev)
586
587             if (rev)
588
589             if (rev)
590
591             if (rev)
592
593             if (rev)
594
595             if (rev)
596
597             if (rev)
598
599             if (rev)
600
601             if (rev)
602
603             if (rev)
604
605             if (rev)
606
607             if (rev)
608
609             if (rev)
610
611             if (rev)
612
613             if (rev)
614
615             if (rev)
616
617             if (rev)
618
619             if (rev)
620
621             if (rev)
622
623             if (rev)
624
625             if (rev)
626
627             if (rev)
628
629             if (rev)
630
631             if (rev)
632
633             if (rev)
634
635             if (rev)
636
637             if (rev)
638
639             if (rev)
640
641             if (rev)
642
643             if (rev)
644
645             if (rev)
646
647             if (rev)
648
649             if (rev)
650
651             if (rev)
652
653             if (rev)
654
655             if (rev)
656
657             if (rev)
658
659             if (rev)
660
661             if (rev)
662
663             if (rev)
664
665             if (rev)
666
667             if (rev)
668
669             if (rev)
670
671             if (rev)
672
673             if (rev)
674
675             if (rev)
676
677             if (rev)
678
679             if (rev)
680
681             if (rev)
682
683             if (rev)
684
685             if (rev)
686
687             if (rev)
688
689             if (rev)
690
691             if (rev)
692
693             if (rev)
694
695             if (rev)
696
697             if (rev)
698
699             if (rev)
700
701             if (rev)
702
703             if (rev)
704
705             if (rev)
706
707             if (rev)
708
709             if (rev)
710
711             if (rev)
712
713             if (rev)
714
715             if (rev)
716
717             if (rev)
718
719             if (rev)
720
721             if (rev)
722
723             if (rev)
724
725             if (rev)
726
727             if (rev)
728
729             if (rev)
730
731             if (rev)
732
733             if (rev)
734
735             if (rev)
736
737             if (rev)
738
739             if (rev)
740
741             if (rev)
742
743             if (rev)
744
745             if (rev)
746
747             if (rev)
748
749             if (rev)
750
751             if (rev)
752
753             if (rev)
754
755             if (rev)
756
757             if (rev)
758
759             if (rev)
760
761             if (rev)
762
763             if (rev)
764
765             if (rev)
766
767             if (rev)
768
769             if (rev)
770
771             if (rev)
772
773             if (rev)
774
775             if (rev)
776
777             if (rev)
778
779             if (rev)
780
781             if (rev)
782
783             if (rev)
784
785             if (rev)
786
787             if (rev)
788
789             if (rev)
790
791             if (rev)
792
793             if (rev)
794
795             if (rev)
796
797             if (rev)
798
799             if (rev)
800
801             if (rev)
802
803             if (rev)
804
805             if (rev)
806
807             if (rev)
808
809             if (rev)
810
811             if (rev)
812
813             if (rev)
814
815             if (rev)
816
817             if (rev)
818
819             if (rev)
820
821             if (rev)
822
823             if (rev)
824
825             if (rev)
826
827             if (rev)
828
829             if (rev)
830
831             if (rev)
832
833             if (rev)
834
835             if (rev)
836
837             if (rev)
838
839             if (rev)
840
841             if (rev)
842
843             if (rev)
844
845             if (rev)
846
847             if (rev)
848
849             if (rev)
850
851             if (rev)
852
853             if (rev)
854
855             if (rev)
856
857             if (rev)
858
859             if (rev)
860
861             if (rev)
862
863             if (rev)
864
865             if (rev)
866
867             if (rev)
868
869             if (rev)
870
871             if (rev)
872
873             if (rev)
874
875             if (rev)
876
877             if (rev)
878
879             if (rev)
880
881             if (rev)
882
883             if (rev)
884
885             if (rev)
886
887             if (rev)
888
889             if (rev)
890
891             if (rev)
892
893             if (rev)
894
895             if (rev)
896
897             if (rev)
898
899             if (rev)
900
901             if (rev)
902
903             if (rev)
904
905             if (rev)
906
907             if (rev)
908
909             if (rev)
910
911             if (rev)
912
913             if (rev)
914
915             if (rev)
916
917             if (rev)
918
919             if (rev)
920
921             if (rev)
922
923             if (rev)
924
925             if (rev)
926
927             if (rev)
928
929             if (rev)
930
931             if (rev)
932
933             if (rev)
934
935             if (rev)
936
937             if (rev)
938
939             if (rev)
940
941             if (rev)
942
943             if (rev)
944
945             if (rev)
946
947             if (rev)
948
949             if (rev)
950
951             if (rev)
952
953             if (rev)
954
955             if (rev)
956
957             if (rev)
958
959             if (rev)
960
961             if (rev)
962
963             if (rev)
964
965             if (rev)
966
967             if (rev)
968
969             if (rev)
970
971             if (rev)
972
973             if (rev)
974
975             if (rev)
976
977             if (rev)
978
979             if (rev)
980
981             if (rev)
982
983             if (rev)
984
985             if (rev)
986
987             if (rev)
988
989             if (rev)
990
991             if (rev)
992
993             if (rev)
994
995             if (rev)
996
997             if (rev)
998
999             if (rev)
1000
1001             if (rev)
1002
1003             if (rev)
1004
1005             if (rev)
1006
1007             if (rev)
1008
1009             if (rev)
1010
1011             if (rev)
1012
1013             if (rev)
1014
1015             if (rev)
1016
1017             if (rev)
1018
1019             if (rev)
1020
1021             if (rev)
1022
1023             if (rev)
1024
1025             if (rev)
1026
1027             if (rev)
1028
1029             if (rev)
1030
1031             if (rev)
1032
1033             if (rev)
1034
1035             if (rev)
1036
1037             if (rev)
1038
1039             if (rev)
1040
1041             if (rev)
1042
1043             if (rev)
1044
1045             if (rev)
1046
1047             if (rev)
1048
1049             if (rev)
1050
1051             if (rev)
1052
1053             if (rev)
1054
1055             if (rev)
1056
1057             if (rev)
1058
1059             if (rev)
1060
1061             if (rev)
1062
1063             if (rev)
1064
1065             if (rev)
1066
1067             if (rev)
1068
1069             if (rev)
1070
1071             if (rev)
1072
1073             if (rev)
1074
1075             if (rev)
1076
1077             if (rev)
1078
1079             if (rev)
1080
1081             if (rev)
1082
1083             if (rev)
1084
1085             if (rev)
1086
1087             if (rev)
1088
1089             if (rev)
1090
1091             if (rev)
1092
1093             if (rev)
1094
1095             if (rev)
1096
1097             if (rev)
1098
1099             if (rev)
1100
1101             if (rev)
1102
1103             if (rev)
1104
1105             if (rev)
1106
1107             if (rev)
1108
1109             if (rev)
1110
1111             if (rev)
1112
1113             if (rev)
1114
1115             if (rev)
1116
1117             if (rev)
1118
1119             if (rev)
1120
1121             if (rev)
1122
1123             if (rev)
1124
1125             if (rev)
1126
1127             if (rev)
1128
1129             if (rev)
1130
1131             if (rev)
1132
1133             if (rev)
1134
1135             if (rev)
1136
1137             if (rev)
1138
1139             if (rev)
1140
1141             if (rev)
1142
1143             if (rev)
1144
1145             if (rev)
1146
1147             if (rev)
1148
1149             if (rev)
1150
1151             if (rev)
1152
1153             if (rev)
1154
1155             if (rev)
1156
1157             if (rev)
1158
1159             if (rev)
1160
1161             if (rev)
1162
1163             if (rev)
1164
1165             if (rev)
1166
1167             if (rev)
1168
1169             if (rev)
1170
1171             if (rev)
1172
1173             if (rev)
1174
1175             if (rev)
1176
1177             if (rev)
1178
1179             if (rev)
1180
1181             if (rev)
1182
1183             if (rev)
1184
1185             if (rev)
1186
1187             if (rev)
1188
1189             if (rev)
1190
1191             if (rev)
1192
1193             if (rev)
1194
1195             if (rev)
1196
1197             if (rev)
1198
1199             if (rev)
1200
1201             if (rev)
1202
1203             if (rev)
1204
1205             if (rev)
1206
1207             if (rev)
1208
1209             if (rev)
1210
1211             if (rev)
1212
1213             if (rev)
1214
1215             if (rev)
1216
1217             if (rev)
1218
1219             if (rev)
1220
1221             if (rev)
1222
1223             if (rev)
1224
1225             if (rev)
1226
1227             if (rev)
1228
1229             if (rev)
1230
1231             if (rev)
1232
1233             if (rev)
1234
1235             if (rev)
1236
1237             if (rev)
1238
1239             if (rev)
1240
1241             if (rev)
1242
1243             if (rev)
1244
1245             if (rev)
1246
1247             if (rev)
1248
1249             if (rev)
1250
1251             if (rev)
1252
1253             if (rev)
1254
1255             if (rev)
1256
1257             if (rev)
1258
1259             if (rev)
1260
1261             if (rev)
1262
1263             if (rev)
1264
1265             if (rev)
1266
1267             if (rev)
1268
1269             if (rev)
1270
1271             if (rev)
1272
1273             if (rev)
1274
1275             if (rev)
1276
1277             if (rev)
1278
1279             if (rev)
1280
1281             if (rev)
1282
1283             if (rev)
1284
1285             if (rev)
1286
1287             if (rev)
1288
1289             if (rev)
1290
1291             if (rev)
1292
1293             if (rev)
1294
1295             if (rev)
1296
1297             if (rev)
1298
1299             if (rev)
1300
1301             if (rev)
1302
1303             if (rev)
1304
1305             if (rev)
1306
1307             if (rev)
1308
1309             if (rev)
1310
1311             if (rev)
1312
1313             if (rev)
1314
1315             if (rev)
1316
1317             if (rev)
1318
1319             if (rev)
1320
1321             if (rev)
1322
1323             if (rev)
1324
1325             if (rev)
1326
1327             if (rev)
1328
1329             if (rev)
1330
1331             if (rev)
1332
1333             if (rev)
1334
1335             if (rev)
1336
1337             if (rev)
1338
1339             if (rev)
1340
1341             if (rev)
1342
1343             if (rev)
1344
1345             if (rev)
1346
1347             if (rev)
1348
1349             if (rev)
1350
1351             if (rev)
1352
1353             if (rev)
1354
1355             if (rev)
1356
1357             if (rev)
1358
1359             if (rev)
1360
1361             if (rev)
1362
1363             if (rev)
1364
1365             if (rev)
1366
1367             if (rev)
1368
1369             if (rev)
1370
1371             if (rev)
1372
1373             if (rev)
1374
1375             if (rev)
1376
1377             if (rev)
1378
1379             if (rev)
1380
1381             if (rev)
1382
1383             if (rev)
1384
1385             if (rev)
1386
1387             if (rev)
1388
1389             if (rev)
1390
1391             if (rev)
1392
1393             if (rev)
1394
1395             if (rev)
1396
1397             if (rev)
1398
1399             if (rev)
1400
1401             if (rev)
1402
1403             if (rev)
1404
1405             if (rev)
1406
1407             if (rev)
1408
1409             if (rev)
1410
1411             if (rev)
1412
1413             if (rev)
1414
1415             if (rev)
1416
1417             if (rev)
1418
1419             if (rev)
1420
1421             if (rev)
1422
1423             if (rev)
1424
1425             if (rev)
1426
1427             if (rev)
1428
1429             if (rev)
1430
1431             if (rev)
1432
1433             if (rev)
1434
1435             if (rev)
1436
1437             if (rev)
1438
1439             if (rev)
1440
1441             if (rev)
1442
1443             if (rev)
1444
1445             if (rev)
1446
1447             if (rev)
1448
1449             if (rev)
1450
1451             if (rev)
1452
1453             if (rev)
1454
1455             if (rev)
1456
1457             if (rev)
1458
1459             if (rev)
1460
1461             if (rev)
1462
1463             if (rev)
1464
1465             if (rev)
1466
1467             if (rev)
1468
1469             if (rev)
1470
1471             if (rev)
1472
1473             if (rev)
1474
1475             if (rev)
1476
1477             if (rev)
1478
1479             if (rev)
1480
1481             if (rev)
1482
1483             if (rev)
1484
1485             if (rev)
1486
1487             if (rev)
1488
1489             if (rev)
1490
1491             if (rev)
1492
1493             if (rev)
1494
1495             if (rev)
1496
1497             if (rev)
1498
1499             if (rev)
1500
1501             if (rev)
1502
1503             if (rev)
1504
1505             if (rev)
1506
1507             if (rev)
1508
1509             if (rev)
1510
1511             if (rev)
1512
1513             if (rev)
1514
1515             if (rev)
1516
1517             if (rev)
1518
1519             if (rev)
1520
1521             if (rev)
1522
1523             if (rev)
1524
1525             if (rev)
1526
1527             if (rev)
1528
1529             if (rev)
1530
1531             if (rev)
1532
1533             if (rev)
1534
1535             if (rev)
1536
1537             if (rev)
1538
1539             if (rev)
1540
1541             if (rev)
1542
1543             if (rev)
1544
1545             if (rev)
1546
1547             if (rev)
1548
1549             if (rev)
1550
1551             if (rev)
1552
1553             if (rev)
1554
1555             if (rev)
1556
1557             if (rev)
1558
1559             if (rev)
1560
1561             if (rev)
1562
1563             if (rev)
1564
1565             if (rev)
1566
1567             if (rev)
1568
1569             if (rev)
1570
1571             if (rev)
1572
1573             if (rev)
1574
1575             if (rev)
1576
1577             if (rev)
1578
1579             if (rev)
1580
1581             if (rev)
1582
1583             if (rev)
1584
1585             if (rev)
1586
1587             if (rev)
1588
1589             if (rev)
1590
1591             if (rev)
1592
1593             if (rev)
1594
1595             if (rev)
1596
1597             if (rev)
1598
1599             if (rev)
1600
1601             if (rev)
1602
1603             if (rev)
1604
1605             if (rev)
1606
1607             if (rev)
1608
1609             if (rev)
1610
1611             if (rev)
1612
1613             if (rev)
1614
1615             if (rev)
1616
1617             if (rev)
1618
1619             if (rev)
1620
1621             if (rev)
1622
1623             if (rev)
1624
1625             if (rev)
1626
1627             if (rev)
1628
1629             if (rev)
1630
1631             if (rev)
1632
1633             if (rev)
1634
1635             if (rev)
1636
1637             if (rev)
1638
1639             if (rev)
1640
1641             if (rev)
1642
1643             if (rev)
1644
1645             if (rev)
1646
1647             if (rev)
1648
1649             if (rev)
1650
1651             if (rev)
1652
1653             if (rev)
1654
1655             if (rev)
1656
1657             if (rev)
1658
1659             if (rev)
1660
1661             if (rev)
1662
1663             if (rev)
1664
1665             if (rev)
1666
1667             if (
```

```

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 void makeroot(node *x) {
101   access(x);
102   splay(x);
103   x -> rev ^= true;
104 }
105

106 void link(node *x, node *y) {
107   makeroot(x);
108   x -> p = y;
109 }
110

111 void cut(node *x, node *y) {
112   makeroot(x);
113   access(y);
114   splay(y);
115
116   y -> ch[0] -> p = null;
117   y -> ch[0] = null;
118   y -> refresh();
119 }
120

121 node *getroot(node *x) {
122   x = access(x);
123   while (x -> pushdown(), x -> ch[0] != null)
124   |   x = x -> ch[0];
125   splay(x);
126   return x;
127 }
128

129 node *getmax(node *x, node *y) {
130   makeroot(x);
131   x = access(y);
132
133   while (x -> pushdown(), x -> pos != -1)
134   |   x = x -> ch[x -> pos];
135   splay(x);
136
137   return x;
138 }
139

140 // 以下为主函数示例
141 for (int i = 1; i <= m; i++) {
142   init(null + n + i, w[i]);
143   if (getroot(null + u[i]) != getroot(null + v[i])) {
144     ans[q + 1] -= k;
145     ans[q + 1] += w[i];
146
147     link(null + u[i], null + n + i);
148     link(null + v[i], null + n + i);
149     vis[i] = true;
150   }
151   else {
152     int ii = getmax(null + u[i], null + v[i]) -
153       | null - n;
154     if (w[i] >= w[ii])
155       continue;
156
157     cut(null + u[ii], null + n + ii);
158     cut(null + v[ii], null + n + ii);
159   }
}

```

```

160      link(null + u[i], null + n + i);
161      link(null + v[i], null + n + i);
162
163      ans[q + 1] -= w[ii];
164      ans[q + 1] += w[i];
165    }
166 }

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] -> sumr : 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
53 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
54
55 // 封装构造函数
56 node *newnode(int w) {
57   node *x = nodes.front(); // 因为有删边加边，可以用一
58   // → 个队列维护可用结点
59   nodes.pop();
60   initialize(x);
61   x -> w = w;
62   x -> refresh();
63 }
```

```

48 |     return x;
49 |
50 |
51 // 封装初始化函数
52 // 记得在进行操作之前对所有结点调用一遍
53 inline void initialize(node *x) {
54     *x = node();
55     x -> ch[0] = x -> ch[1] = x -> p = null;
56 }
57
58 // 注意一下在Access的同时更新子树信息的方法
59 node *access(node *x) {
60     node *y = null;
61
62     while (x != null) {
63         splay(x);
64
65         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
66         ↪ chain_cnt;
67         x -> tree_sum += (x -> ch[1] -> rev ? x ->
68         ↪ ch[1] -> sumr : x -> ch[1] -> suml) - y ->
69         ↪ suml;
70         x -> ch[1] = y;
71
72         (y = x) -> refresh();
73         x = x -> p;
74     }
75
76     return y;
77 }
78
79 // 找到一个点所在连通块的根
80 // 对比原版没有变化
81 node *getroot(node *x) {
82     x = access(x);
83
84     while (x -> pushdown(), x -> ch[0] != null)
85         x = x -> ch[0];
86     splay(x);
87
88     return x;
89 }
90
91 // 换根, 同样没有变化
92 void makeroot(node *x) {
93     access(x);
94     splay(x);
95     x -> rev ^= true;
96     x -> pushdown();
97 }
98
99 // 连接两个点
100 // !!! 注意这里必须把两者都变成根, 因为只能修改根结点
101 void link(node *x, node *y) {
102     makeroot(x);
103     makeroot(y);
104
105     x -> p = y;
106     y -> tree_cnt += x -> chain_cnt;
107     y -> tree_sum += x -> suml;
108     y -> refresh();
109
110 // 删一条边
111 // 对比原版没有变化
112 void cut(node *x, node *y) {
113     makeroot(x);
114     access(y);
115     splay(y);
116
117     y -> ch[0] -> p = null;
118     y -> ch[0] = null;
119 }
120
121 // 修改/询问一个点, 这里以询问为例
122 // 如果是修改就在换根之后搞一些操作
123 long long query(node *x) {
124     makeroot(x);
125     return x -> suml;
126 }
127
128 // Splay函数
129 // 对比原版没有变化
130 void splay(node *x) {
131     x -> pushdown();
132
133     while (!isroot(x)) {
134         if (!isroot(x -> p))
135             x -> p -> p -> pushdown();
136         x -> p -> pushdown();
137         x -> pushdown();
138
139         if (isroot(x -> p)) {
140             rot(x -> p, dir(x) ^ 1);
141             break;
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 }

```

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() {}
19
20 void push(long long x) {
21   if (x > (-INF) >> 2)
22     q1.push(x);
23 }
24
25 void erase(long long x) {
26   if (x > (-INF) >> 2)
27     q2.push(x);
28 }
29
30 long long top() {
31   if (empty())
32     return -INF;
33
34   while (!q2.empty() && q1.top() == q2.top()) {
35     q1.pop();
36     q2.pop();
37   }
38
39   return q1.top();
40 }
41
42 long long top2() {
43   if (size() < 2)
44     return -INF;
45
46   long long a = top();
47   erase(a);
48   long long b = top();
49   push(a);
50   return a + b;
51 }
52
53 int size() {
54   return q1.size() - q2.size();
55 }
56
57 bool empty() {
58   return q1.size() == q2.size();
59 }
60 } heap; // 全局堆维护每条链的最大子段和
61
62 struct node {
63   long long sum, maxsum, prefix, suffix;
64   int key;
65   binary_heap heap; // 每个点的堆存的是它的子树中到它的
66   → 最远距离, 如果它是黑点的话还会包括自己
67   node *ch[2], *p;
68   bool rev;
69   node(int k = 0): sum(k), maxsum(-INF),
70     → prefix(-INF),
71     → suffix(-INF), key(k), rev(false) {}
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   inline void refresh() {
83     pushdown();
84     ch[0] → pushdown();
85     ch[1] → pushdown();
86     sum = ch[0] → sum + ch[1] → sum + key;
87     prefix = max(ch[0] → prefix,
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
     ch[1] → sum + key + ch[0] →
     → prefix);
     suffix = max(ch[1] → suffix,
     ch[1] → sum + key + ch[0] →
     → suffix);
     maxsum = max(max(ch[0] → maxsum, ch[1] →
     → maxsum),
     ch[0] → suffix + key + ch[1] →
     → prefix);
     if (!heap.empty()) {
       prefix = max(prefix,
       ch[0] → sum + key +
       → heap.top());
       suffix = max(suffix,
       ch[1] → sum + key +
       → heap.top());
       maxsum = max(maxsum, max(ch[0] → suffix,
       ch[1] → prefix) +
       → key +
       → heap.top());
     }
     if (heap.size() > 1) {
       maxsum = max(maxsum, heap.top() +
       → key);
     }
   }
   } null[maxn << 1], *ptr = null;
void addedge(int, int, int);
void deledge(int, int);
void modify(int, int, int);
void modify_color(int);
node *newnode(int);
node *access(node *);
void makeroot(node *);
void link(node *, node *);
void cut(node *, node *);
void splay(node *);
void rot(node *, int);
queue<node *> freenodes;
tree<pair<int, int>, node *> mp;
bool col[maxn] = {false};
char c;
int n, m, k, x, y, z;
int main() {
  null → ch[0] = null → ch[1] = null → p = null;
  scanf("%d%d%d", &n, &m, &k);
  for (int i = 1; i <= n; i++)
    newnode(0);
  heap.push(0);
  while (k--) {
    scanf("%d", &x);
    col[x] = true;
    null[x].heap.push(0);
  }
  for (int i = 1; i < n; i++) {
    scanf("%d%d%d", &x, &y, &z);
    if (x > y)
      swap(x, y);
    addedge(x, y, z);
  }
}

```

```

148
149     while (m--) {
150         scanf(" %c%d", &c, &x);
151
152         if (c == 'A') {
153             scanf("%d", &y);
154
155             if (x > y)
156                 swap(x, y);
157             deledge(x, y);
158         }
159         else if (c == 'B') {
160             scanf("%d%d", &y, &z);
161
162             if (x > y)
163                 swap(x, y);
164             addedge(x, y, z);
165         }
166         else if (c == 'C') {
167             scanf("%d%d", &y, &z);
168
169             if (x > y)
170                 swap(x, y);
171             modify(x, y, z);
172         }
173         else
174             modify_color(x);
175
176         printf("%lld\n", (heap.top() > 0 ? heap.top() : -1));
177     }
178
179     return 0;
180 }

181 void addedge(int x, int y, int z) {
182     node *tmp;
183     if (freenodes.empty())
184         tmp = newnode(z);
185     else {
186         tmp = freenodes.front();
187         freenodes.pop();
188         *tmp = node(z);
189     }
190
191     tmp->ch[0] = tmp->ch[1] = tmp->p = null;
192
193     heap.push(tmp->maxsum);
194     link(tmp, null + x);
195     link(tmp, null + y);
196     mp[make_pair(x, y)] = tmp;
197 }

198 void deledge(int x, int y) {
199     node *tmp = mp[make_pair(x, y)];
200
201     cut(tmp, null + x);
202     cut(tmp, null + y);
203
204     freenodes.push(tmp);
205     heap.erase(tmp->maxsum);
206     mp.erase(make_pair(x, y));
207 }

208 void modify(int x, int y, int z) {
209     node *tmp = mp[make_pair(x, y)];
210     makeroot(tmp);
211     tmp->pushdown();
212
213     heap.erase(tmp->maxsum);
214     tmp->key = z;
215     tmp->refresh();
216
217     heap.push(tmp->maxsum);
218 }

219     heap.push(tmp->maxsum);
220 }
221
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     x->pushdown();
289 }

```

```

290     x -> p = y;
291     heap.erase(y -> maxsum);
292     y -> heap.push(x -> prefix);
293     y -> refresh();
294     heap.push(y -> maxsum);
295 }
296
297 void cut(node *x, node *y) { // 断开一条实边, 一条链变成
298     // 两条链, 需要维护全局堆
299     makeroot(x);
300     access(y);
301     splay(y);
302
303     heap.erase(y -> maxsum);
304     heap.push(y -> ch[0] -> maxsum);
305     y -> ch[0] -> p = null;
306     y -> ch[0] = null;
307     y -> refresh();
308     heap.push(y -> maxsum);
309 }
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         x -> pushdown();
320
321         if (isroot(x -> p)) {
322             rot(x -> p, dir(x) ^ 1);
323             break;
324         }
325
326         if (dir(x) == dir(x -> p))
327             rot(x -> p -> p, dir(x -> p) ^ 1);
328         else
329             rot(x -> p, dir(x) ^ 1);
330
331         rot(x -> p, dir(x) ^ 1);
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] -> l[0]));
16         l[1] = min(x[1], min(ch[0] -> l[1], ch[1] -> l[1]));
17         r[0] = max(x[0], max(ch[0] -> r[0], ch[1] -> r[0]));
18         r[1] = max(x[1], max(ch[0] -> r[1], ch[1] -> r[1]));
19     }
20 } null[maxn], *root = null;
21
22 void build(int l, int r, int k, node *&rt) {
23     if (l > r) {
24         rt = null;
25         return;
26     }
27     int mid = (l + r) / 2;
28
29     d = k;
30     nth_element(null + l, null + mid, null + r + 1);
31
32     rt = null + mid;
33     build(l, mid - 1, k ^ 1, rt -> ch[0]);
34     build(mid + 1, r, k ^ 1, rt -> ch[1]);
35
36     rt -> refresh();
37 }
38
39
40 void query(node *rt) {
41     if (l[0] <= rt -> l[0] && l[1] <= rt -> l[1] && rt -> r[0] <= r[0] && rt -> r[1] <= r[1])
42         ans += rt -> sum;
43     return;
44 }
45 else if (l[0] > rt -> r[0] || l[1] > rt -> r[1] || r[0] < rt -> l[0] || r[1] < rt -> l[1])
46     return;
47
48 if (l[0] <= rt -> x[0] && l[1] <= rt -> x[1] && rt -> x[0] <= r[0] && rt -> x[1] <= r[1])
49     ans += rt -> w;
50
51 query(rt -> ch[0]);
52 query(rt -> ch[1]);
53 }
54
55 int main() {
56
57     null -> l[0] = null -> l[1] = 100000000;
58     null -> r[0] = null -> r[1] = -100000000;
59     null -> sum = 0;
60     null -> ch[0] = null -> ch[1] = null;
61     scanf("%*d");
62
63     while (scanf("%d", &op) == 1 && op != 3) {
64         if (op == 1) {
65             tmp++;
66             scanf("%d%d%d", &x[tmp][0], &x[tmp][1],
67                   &w[tmp]);
68             x[tmp][0] ^= ans;
69             x[tmp][1] ^= ans;
70             w[tmp] ^= ans;
71         }
72     }
73 }

```

```

70
71     if (tmp == B) {
72         for (int i = 1; i <= tmp; i++) {
73             null[cnt + i].x[0] = x[i][0];
74             null[cnt + i].x[1] = x[i][1];
75             null[cnt + i].w = w[i];
76         }
77
78         build(1, cnt += tmp, 0, root);
79         tmp = 0;
80     }
81 }
82 else {
83     scanf("%d%d%d%d", &l[0], &l[1], &r[0],
84           → &r[1]);
85     l[0] ^= ans;
86     l[1] ^= ans;
87     r[0] ^= ans;
88     r[1] ^= ans;
89     ans = 0;
90
91     for (int i = 1; i <= tmp; i++) {
92         if (l[0] <= x[i][0] && l[1] <= x[i][1]
93             → && x[i][0] <= r[0] && x[i][1] <=
94                 → r[1])
95             ans += w[i];
96
97         query(root);
98         printf("%d\n", ans);
99     }
100 }
```

```

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    void clear(int x) {
33      G[x].clear();
34      W[x].clear();
35      col[x] = false;
36    }
37
38    void solve(int rt) {
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136

```

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    ufs[x] = p[x];
21}
22
23 } virtree;
24
25 void dfs(int);
26 int LCA(int, int);
27
28 vector<int> G[maxn];
29 int f[maxn][20], d[maxn], dfn[maxn], tim = 0;
30
31 bool cmp(int x, int y) {
32     return dfn[x] < dfn[y];
33 }
34
35 int n, m, lgn = 0, a[maxn], s[maxn], v[maxn];
36
37 int main() {
38     scanf("%d", &n);
39
40     for (int i = 1, x, y; i < n; i++) {
41         scanf("%d%d", &x, &y);
42         G[x].push_back(y);
43         G[y].push_back(x);
44     }
45 }

```

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];
5     bool col[maxn];
6     long long ans_sum;
7
8     void dfs(int v, int p) {
9         for (int i = 0; i < G[v].size(); i++) {
10            if (G[v][i] == p) continue;
11            dfs(G[v][i], v);
12            size[v] += size[G[v][i]];
13            mn[v] = min(mn[v], mn[G[v][i]]);
14            mx[v] = max(mx[v], mx[G[v][i]]);
15        }
16        if (mn[v] > mx[v]) {
17            cout << "NO" << endl;
18            exit(0);
19        }
20        if (col[v]) {
21            cout << "YES" << endl;
22            cout << ans_sum << endl;
23            exit(0);
24        }
25        if (mx[v] > 1) {
26            cout << "NO" << endl;
27            exit(0);
28        }
29        if (size[v] > 1) {
30            cout << "NO" << endl;
31            exit(0);
32        }
33    }
34
35    void solve() {
36        int n, m;
37        cin << n << m;
38        for (int i = 0; i < n; i++) {
39            G[i].clear();
40            W[i].clear();
41            p[i] = -1;
42            d[i] = 0;
43            size[i] = 1;
44            mn[i] = mx[i] = 1;
45            col[i] = false;
46        }
47        for (int i = 0; i < m; i++) {
48            int u, v;
49            cin << u << v;
50            G[u].push_back(v);
51            G[v].push_back(u);
52            W[u].push_back(i);
53            W[v].push_back(i);
54        }
55        dfs(0, -1);
56    }
57
58    int main() {
59        solve();
60        return 0;
61    }
62 }
```

```

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                 s[++top] = a[i];
112             }
113
114             for (int i = top - 1; i; i--)
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         }
129
130         return 0;
131     }
132
133     void dfs(int x) {
134         dfn[x] = ++tim;
135         d[x] = d[f[x][0]] + 1;
136
137         while ((1 << lgn) < d[x])
138             lgn++;
139
140         for (int i = 0; i < (int)G[x].size(); i++)
141             if (G[x][i] != f[x][0]) {
142                 f[G[x][i]][0] = x;
143                 dfs(G[x][i]);
144             }
145     }
146
147 }
```

```

142     int LCA(int x, int y) {
143         if (d[x] != d[y]) {
144             if (d[x] < d[y])
145                 swap(x, y);
146
147             for (int i = lgn; i >= 0; i--)
148                 if (((d[x] - d[y]) >> i) & 1)
149                     x = f[x][i];
150
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     }
164
165 }
```

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++) {
42                     v[x][h[x] - j - 1] += v[y][h[y] - j];
43
44                     int t = v[x][h[x] - j - 1];
45                 }
46             }
47
48         }
49
50 }
```

```

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 }
```

```

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

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
35 // 伸一倍）数组
36 void dfs2(int x) {
37     top[x] = (x == son[f[0][x]] ? top[f[0][x]] : x);
38
39     for (int y : G[x])
40         if (y != f[0][x])
41             dfs2(y);
42
43     if (top[x] == x) {
44         int u = x;
45         while (top[son[u]] == x)
46             u = son[u];
47
48         len[x] = d[u] - d[x];
49         for (int i = 0; i < len[x]; i++, u = f[0][u])
50             v[x].push_back(u);
51
52         u = x;
53         for (int i = 0; i < len[x] && u; i++, u = f[0]
54             & [u])
55             v[x].push_back(u);
56 }
```

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    int & top() {
19        return a[1];
20    }
21
22    int pop() {
23        int res = a[1];
24        a[1] = a[size--];
25
26        for (int x = 1, son; ; x = son) {
27            if (x * 2 == size)
28                son = x * 2;
29            else if (x * 2 > size)
30                break;
31            else if (a[x * 2] < a[x * 2 + 1])
32                son = x * 2;
33            else
34                son = x * 2 + 1;
35
36            if (a[son] < a[x])
37                swap(a[x], a[son]);
38            else
39                break;
40        }
41
42        return res;
43    }
44};
```

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
22    q[maxn]; // 结构体可以复用, d既可以作为左端点块编号, 也
23    → 可以作为二次离线处理的倍数
24
25    int global_n, bid[maxn], L[maxn], R[maxn], cntb;
26
27    int sa[maxn], sb[maxn];
28
29    void addp(int x) { // sqrt(n) 修改 O(1) 查询
30        for (int k = bid[x]; k <= cntb; k++)
31            sb[k]++;
32
33        for (int i = x; i <= R[bid[x]]; i++)
34            sa[i]++;
35    }
36
37    int queryp(int x) {
38        if (!x)
39            return 0;
40
41        return sa[x] + sb[bid[x] - 1];
42    }
43
44    void adds(int x) {
45        for (int k = 1; k <= bid[x]; k++)
46            sb[k]++;
47
48        for (int i = L[bid[x]]; i <= x; i++)
49            sa[i]++;
50
51    }
52
53    int querys(int x) {
54        if (x > global_n)
55            return 0; // 为了防止越界就判一下
56        return sa[x] + sb[bid[x] + 1];
57    }
58
59    vector<Q> vp[maxn], vs[maxn]; // prefix, suffix
60
61    long long fp[maxn], fs[maxn]; // prefix, suffix
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));
108
109           l = s;
110
111           if (t > r)
112               vp[l - 1].push_back(Q(r + 1, t, 1, i));
113           else if (t < r)
114               vp[l - 1].push_back(Q(t + 1, r, -1, i));
115
116           r = t;
117       }
118
119    }
120
121    cout << ans[n];
122
123    return 0;
124}
```

```

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
122         adds(a[i]);
123
124         for (auto q : vp[i]) {
125             long long tmp = 0;
126             for (int k = q.l; k <= q.r; k++)
127                 tmp += querys(a[k] + 1);
128
129             ta[q.id] -= q.d * tmp;
130         }
131
132     memset(sa, 0, sizeof(sa));
133     memset(sb, 0, sizeof(sb));
134
135     for (int i = n; 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
139         addp(a[i]);
140
141         for (auto q : vs[i]) {
142             long long tmp = 0;
143             for (int k = q.l; k <= q.r; k++)
144                 tmp += queryp(a[k] - 1);
145
146             ta[q.id] -= q.d * tmp;
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     for (int i = 1; i <= m; i++)
167         printf("%lld\n", ans[i]);
168
169     return 0;
}

```

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[maxn], R[maxn], cntb;
5
6 struct DS { // O(sqrt(n)) 修改 O(1) 查询
7     int total;
8     int sa[maxn], sb[maxn];
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     return ans;
51 }
52
53 long long fp[maxn], fs[maxn];
54
55 int rkn[maxn], val[maxn][B + 5];
56
57 long long dat[maxb][maxb];

```

```

58 int a[maxn];
59
60 int main() {
61     int m;
62     cin >> n >> m;
63
64     for (int i = 1; i <= n; i++) {
65         cin >> a[i];
66
67         bid[i] = (i - 1) / B + 1;
68         if (!L[bid[i]])
69             L[bid[i]] = i;
70         R[bid[i]] = i;
71         cntb = bid[i];
72
73         rnk[i] = i;
74     }
75
76     for (int k = 1; k <= cntb; k++)
77         sort(rnk + L[k], rnk + R[k] + 1, [](int x, int y) {return a[x] < a[y];}); // 每个块排序
78
79     for (int i = n; i; i--)
80         for (int j = 2; i + j - 1 <= R[bid[i]]; j++) {
81             val[i][j] = val[i + 1][j - 1] + val[i][j - 2];
82             if (a[i] > a[i + j - 1])
83                 val[i][j]++;
84         }
85
86     for (int k = 1; k <= cntb; k++)
87         for (int i = L[k]; i <= R[k]; i++) {
88             dat[k][k] += queryc(a[i] + 1); // 单块内的逆
89             // 序对直接用树状数组预处理
90             addc(a[i], 1);
91         }
92
93     for (int i = L[k]; i <= R[k]; i++)
94         addc(a[i], -1);
95     }
96
97     for (int i = 1; i <= n; i++) {
98         if (i > 1 && i == L[bid[i]])
99             pr[bid[i]].init(pr[bid[i] - 1]);
100
101         fp[i] = fp[i - 1] + pr[bid[i]].query(a[i] + 1);
102
103         pr[bid[i]].add(a[i]);
104     }
105
106     for (int i = n; i; i--) {
107         fs[i] = fs[i + 1] + (n - i - queryc(a[i] + 1));
108         addc(a[i], 1);
109     }
110
111     for (int s = 1; s <= cntb; s++)
112         for (int t = s + 1; t <= cntb; t++) {
113             dat[s][t] = dat[s][t - 1] + dat[t][t];
114
115             for (int i = L[t]; i <= R[t]; i++) // 块间的
116                 // 逆序对用刚才处理的分块求出
117                 dat[s][t] += pr[t - 1].query(a[i] + 1);
118                 // - pr[s - 1].query(a[i] + 1);
119             }
120
121     long long ans = 0;
122
123     while (m--) {
124         long long s, t;

```

```

124         cin >> s >> t;
125
126         int l = s ^ ans, r = t ^ ans;
127
128         if (bid[l] == bid[r])
129             ans = val[l][r - l + 1];
130         else {
131             ans = dat[bid[l] + 1][bid[r] - 1];
132
133             ans += fp[r] - fp[L[bid[r]] - 1];
134             for (int i = L[bid[r]]; i <= r; i++)
135                 ans -= pr[bid[l]].query(a[i] + 1);
136
137             ans += fs[l] - fs[R[bid[l]] + 1];
138             for (int i = l; i <= R[bid[l]]; i++)
139                 ans -= (a[i] - 1) - pr[bid[r] - 1].query(a[i] - 1);
140
141             int i = L[bid[l]], j = L[bid[r]], w = 0; // 手写归并
142
143             while (true) {
144                 while (i <= R[bid[l]] && rnk[i] < l)
145                     i++;
146                 while (j <= R[bid[r]] && rnk[j] > r)
147                     j++;
148
149                 if (i > R[bid[l]] && j > R[bid[r]])
150                     break;
151
152                 int x = (i <= R[bid[l]] ? a[rnk[i]] : (int)1e9), y = (j <= R[bid[r]] ? a[rnk[j]] : (int)1e9);
153
154                 if (x < y) {
155                     ans += w;
156                     i++;
157                 } else {
158                     j++;
159                     w++;
160                 }
161             }
162
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, 但是一般空间需要 $n\sqrt{n}$
 - * 例题: 询问区间中有几种数出现次数恰好为 k , 强制在线
 - 如果带修改可以试着想一想带修莫队, 但是复杂度高达 $n^{\frac{5}{3}}$
- 线段树可以解决的问题也可以用分块来做到 $O(1)$ 询问或是 $O(1)$ 修改, 具体要看哪种操作更多

3. 树

- 与序列类似, 树上也有树分块和树上莫队
 - 树上带修莫队很麻烦, 常数也大, 最好不要先考虑
 - 树分块不要想当然
- 树分治也可以套根号分治, 道理是一样的

4. 字符串

- 循环节长度大于 \sqrt{n} 的子串最多只有 $O(n)$ 个, 如果是极长子串则只有 $O(\sqrt{n})$ 个

5 字符串

5.1 KMP

```

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) {
25         x = q[head++];
26 }
```

```

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 fill(f[0], f[0] + 26, cnt + 1);
39 }
```

5.3 后缀数组 SA

5.3.1 倍增

```

1 constexpr int MAXN = 1000005;
2
3 void get_sa(char *s, int n, int *sa, int *rnk, int
4     → *height) { // 1-based
5     static int buc[MAXN], id[MAXN], p[MAXN], t[MAXN];
6
7     int m = 300;
8
9     for (int i = 1; i <= n; i++)
10        buc[rnk[i]] = s[i]++;
11    for (int i = 1; i <= m; i++)
12        buc[i] += buc[i - 1];
13    for (int i = n; i; i--)
14        sa[buc[rnk[i]]--] = i;
15
16    memset(buc, 0, sizeof(int) * (m + 1));
17
18    for (int k = 1, cnt = 0; cnt != n; k *= 2, m = cnt)
19    → {
20        cnt = 0;
21
22        for (int i = n; i > n - k; i--)
23            id[++cnt] = i;
24        for (int i = 1; i <= n; i++)
25            if (sa[i] > k)
26                id[++cnt] = sa[i] - k;
27
28        for (int i = 1; i <= n; i++)
29            buc[p[i]] = rnk[id[i]]++;
30        for (int i = 1; i <= m; i++)
31            buc[i] += buc[i - 1];
32        for (int i = n; i; i--)
33            sa[buc[p[i]]--] = id[i];
34
35        memset(buc, 0, sizeof(int) * (m + 1));
36
37        memcpy(t, rnk, sizeof(int) * (n + 1));
38        t[n + 1] = 0; // 记得清空 n + 1
39
40        cnt = 0;
41        for (int i = 1; i <= n; i++) {
42            if (t[sa[i]] != t[sa[i - 1]] || t[sa[i] +
43                → k] != t[sa[i - 1] + k])
44                cnt++;
45
46            rnk[sa[i]] = cnt;
47        }
48
49        for (int i = 1; i <= n; i++)
50            rnk[sa[i]] = i;
51    }
52 }
```

```

48     sa[rnk[i]] = i;
49
50     for (int i = 1, k = 0; i <= n; i++) {
51         if (k)
52             k--;
53
54         if (rnk[i] > 1) // 两个都要判, 否则会左/右越界
55             while (sa[rnk[i] - 1] + k <= n && s[i + k]
56                 → == s[sa[rnk[i] - 1] + k])
57                 k++;
58
59         height[rnk[i]] = k;
60     }

```

5.3.2 SA-IS

```

1 constexpr int l_type = 0, s_type = 1;
2
3 // 判断一个字符是否为LMS字符
4 bool is_lms(vector<int>& tp, int x) {
5     return x > 0 && tp[x] == s_type && tp[x - 1] ==
6         → l_type;
7 }
8
9 // 判断两个LMS子串是否相同
10 bool equal_substr(int* s, int x, int y, vector<int>&
11     → 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 // s 是输入字符串, len 是字符串的长度, m 是字符集的大小
23 vector<int> sais(int *s, int len, int m) {
24     int n = len - 1;
25
26     vector<int> tp(n + 1), pos(n + 1), name(n + 1, -1),
27         → sa(n + 1, -1);
28     vector<int> buc(m + 1), lbuc(m + 1), sbuc(m + 1);
29
30     // 诱导排序 (从 * 型诱导到 L 型, 从 L 型诱导到 S 型)
31     // 调用之前应将 * 型按要求放入 SA 中
32     auto induced_sort = [&]() {
33         for (int i = 0; i <= n; i++)
34             if (sa[i] > 0 && tp[sa[i] - 1] == l_type)
35                 sa[lbuc[s[sa[i] - 1]]++] = sa[i] - 1;
36
37         for (int i = 1; i <= m; i++)
38             sbuc[i] = buc[i] - 1;
39
40         for (int i = n; ~i; i--)
41             if (sa[i] > 0 && tp[sa[i] - 1] == s_type)
42                 sa[sbuc[s[sa[i] - 1]]--] = sa[i] - 1;
43     };
44
45     for (int i = 0; i <= n; i++)
46         buc[s[i]]++;
47
48     for (int i = 1; i <= m; i++) {
49         buc[i] += buc[i - 1];
50
51         lbuc[i] = buc[i - 1];
52         sbuc[i] = buc[i] - 1;
53     }

```

```

52     tp[n] = s_type;
53     for (int i = n - 1; ~i; i--) {
54         if (s[i] < s[i + 1])
55             tp[i] = s_type;
56         else if (s[i] > s[i + 1])
57             tp[i] = l_type;
58         else
59             tp[i] = tp[i + 1];
60     }
61
62     int cnt = 0;
63     for (int i = 1; i <= n; i++) {
64         if (tp[i] == s_type && tp[i - 1] == l_type)
65             pos[cnt++] = i;
66
67     for (int i = 0; i < cnt; i++)
68         sa[sbuc[s[pos[i]]]--] = pos[i];
69     induced_sort();
70
71     int lastx = -1, namecnt = 1;
72     bool flag = false;
73
74     for (int i = 1; i <= n; i++) {
75         int x = sa[i];
76
77         if (is_lms(tp, x)) {
78             if (lastx >= 0 && !equal_substr(s, x,
79                 → lastx, tp))
80                 namecnt++;
81
82             if (lastx >= 0 && namecnt == name[lastx])
83                 flag = true;
84
85             name[x] = namecnt;
86             lastx = x;
87         }
88     name[n] = 0;
89
90     int* t = new int[cnt];
91     int p = 0;
92     for (int i = 0; i <= n; i++)
93         if (name[i] >= 0)
94             t[p++] = name[i];
95
96     vector<int> tsa;
97     if (!flag) {
98         tsa.resize(cnt);
99
100        for (int i = 0; i < cnt; i++)
101            tsa[t[i]] = i;
102    }
103    else
104        tsa = move(sais(t, cnt, namecnt));
105
106    lbuc[0] = sbuc[0] = 0;
107    for (int i = 1; i <= m; i++) {
108        lbuc[i] = buc[i - 1];
109        sbuc[i] = buc[i] - 1;
110    }
111
112    sa.assign(n + 1, -1);
113    for (int i = cnt - 1; ~i; i--)
114        sa[sbuc[s[pos[tsa[i]]]]--] = pos[tsa[i]];
115    induced_sort();
116
117    return sa;
118 }
119
120 // 封装好的函数, 1-based
121 void get_sa(char *s, int n, int *sa, int *rnk, int
122     → *height) {

```

```

122 static int a[MAXN];
123
124 a[n] = '$';
125 for (int i = 1; i <= n; i++)
126 | a[i - 1] = s[i];
127
128 vector<int> t = sais(a, n + 1, 256);
129 copy(t.begin(), t.end(), sa);
130
131 sa[0] = 0;
132 for (int i = 1; i <= n; i++)
133 | rnk[++sa[i]] = i;
134
135 for (int i = 1, k = 0; i <= n; i++) {
136 | if (k)
137 | | k--;
138
139 | while (s[i + k] == s[sa[rnk[i] - 1] + k])
140 | | k++;
141
142 | height[rnk[i]] = k; // height[i] = lcp(sa[i],
143 | | → sa[i - 1])
144 }

```

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],
4 | → tim = 0;
5
6 void dfs(int x) {
7 | if (id[x]) {
8 | | height[tim++] = val[last];
9 | | sa[tim] = id[x];
10
11 | | last = x;
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
20 int 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 =
35 | | | → par[x]) {
36 | | | vis[x] = true;
37 | | | pos -= val[x] - val[par[x]];
38 | | | ch[par[x]][s[pos + 1] - 'a'] = x;
39
40 dfs(1);
41 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 }

```

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 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 SAM

```

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 | → 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 | | if (val[q] == val[p] + 1)
35 | | | par[np] = q;
36 | | else {
37 | | | int nq = ++sam_cnt;
38 | | | val[nq] = val[p] + 1;
39 | | | memcpy(go[nq], go[q], sizeof(go[q]));
40 }

```

```

41     |     par[nq] = par[q];
42     |     par[np] = par[q] = nq;
43
44     |     while (p && go[p][c] == q) {
45     |         go[p][c] = nq;
46     |         p = par[p];
47     |
48     }
49
50
51     last = np;
52

```

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}
// 调用的时候直接last = 1然后一路调用last = extend(last,
// → c) 就行了

```

5.5.2 区间本质不同子串计数 (后缀自动机 + LCT + 线段树)

问题 给定一个字符串 s ，多次询问 $[l, r]$ 区间的本质不同的子串个数，可能强制在线。

做法 考虑建出后缀自动机，然后枚举右端点，用线段树维护每个左端点的答案。

显然只有 $right$ 集合在 $[l, r]$ 中的串才有可能有贡献，所以我们可以只考虑每个串最大的 $right$ 。

每次右端点 $+ 1$ 时找到它对应的结点 u ，则 u 到根节点路径上的每个点，它的 $right$ 集合都会被 r 更新。

对于某个特定的左端点 l ，我们需要保证本质不同的子串左端点不能越过它；因此对于一个结点 p ，我们知道它对应的子串长度 (val_{par_p}, val_p) 之后，在 p 的 $right$ 集合最大值减去对应长度，这样对应的 l 内全部 $+ 1$ 即可。这样询问时就只需要查询 r 对应的线段树中 $[l, r]$ 的区间和了。(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 }

```

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;
15
16     mx[o][0] = max(mx[u][0], t);
17     mx[o][1] = max(mx[u][1], d);
18
19     if (l == r)
20         return;
21
22     lc[o] = lc[u];
23     rc[o] = rc[u];
24
25     int mid = (l + r) / 2;
26     if (s <= mid)
27         modify_seg(l, mid, lc[o]);
28     else
29         modify_seg(mid + 1, r, rc[o]);
30 }
31
32 int query_seg(int l, int r, int o, int k) {
33     if (s <= l && t >= r)
34         return mx[o][k];
35
36     int mid = (l + r) / 2, ans = 0;
37
38     if (s <= mid)
39         ans = max(ans, query_seg(l, mid, lc[o], k));
40     if (t > mid)
41         ans = max(ans, query_seg(mid + 1, r, rc[o],
42             → k));
43
44     return ans;
45 }
46
47 int N;
48
49 void modify(int pos, int u, int v, int &rt) {
50     s = pos;
51     t = u;
52     d = v;
53
54     modify_seg(1, N, rt);
55 }
56
57 int query(int l, int r, int rt) {
58     s = l;
59     t = r;
60     int ans = query_seg(1, N, rt, 0);
61
62     s = 1;
63     t = l;
64     return max(ans, query_seg(1, N, rt, 1) - l);
65 }
66
67 struct node {
68     int size, l, r, id, tim;
69     node *ch[2], *p;
70     bool tag;
71
72     node() = default;
73
74     void apply(int v) {
75         tim = v;
76         tag = true;
77     }

```

```

78     void pushdown() {
79         if (tag) {
80             ch[0] → tim = ch[1] → tim = tim;
81             ch[0] → tag = ch[1] → tag = true;
82
83             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 → ch[1];
96 }
97
98 inline bool dir(node *x) {
99     return x == x → p → ch[1];
100 }
101
102 void init(node *x, int i) {
103     *x = node();
104     x → ch[0] = x → ch[1] = x → p = null;
105     x → size = 1;
106     x → id = x → l = x → r = i;
107 }
108
109 void rot(node *x, int d) {
110     node *y = x → ch[d ^ 1];
111
112     y → p = x → p;
113     if (!isroot(x))
114         x → p → ch[dir(x)] = y;
115
116     if ((x → ch[d ^ 1] = y → ch[d]) != null)
117         y → ch[d] → p = x;
118     (y → ch[d] = x) → p = y;
119
120     x → update();
121     y → update();
122 }
123
124 void splay(node *x) {
125     x → pushdown();
126
127     while (!isroot(x)) {
128         if (!isroot(x → p))
129             x → p → p → pushdown();
130         x → p → pushdown();
131         x → pushdown();
132
133         if (isroot(x → p)) {
134             rot(x → p, dir(x) ^ 1);
135             break;
136         }
137
138         if (dir(x) == dir(x → p))
139             rot(x → p → p, dir(x → p) ^ 1);
140         else
141             rot(x → p, dir(x) ^ 1);
142
143         rot(x → p, dir(x) ^ 1);
144     }
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         x->p->pushdown();
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 int val[maxn], par[maxn], go[maxn][26], sam_cnt,
170   ↪ sam_last;
171
172 node *access(node *x, int r) {
173     root[r] = root[r - 1];
174
175     node *y = null;
176
177     while (x != null) {
178         splay(x);
179
180         x->pushdown();
181
182         x->ch[1] = null;
183         x->update();
184
185         if (x->tim && val[x->r]) { // last time
186             ↪ visited
187             int right = x->tim, left = right - val[x->r] + 1;
188             modify(left, val[x->r], right + 1,
189                   ↪ root[r]);
190
191             x->apply(r);
192             x->pushdown();
193
194             x->ch[1] = y;
195             (y = x)->update();
196
197             x = x->p;
198         }
199
200     return y;
201 }
202 void new_leaf(node *x, node *par) {
203     x->p = par;
204 }
205
206 void new_node(node *x, node *y, node *par) {
207     splay(y);
208
209     if (isroot(y) && y->p == par) {
210         assert(y->ch[0] == null);
211
212         y->ch[0] = x;
213         x->p = y;
214         y->update();
215     }
216     else {
217         splay(par, y);
218
219         assert(y->ch[0] == par);
220         assert(par->ch[1] == null);
221         par->ch[1] = x;
222         x->p = par;
223
224         par->update();
225         y->update();
226     }
227
228     x->tim = y->tim;
229 }
230
231 void extend(int c) {
232     int p = sam_last, np = ++sam_cnt;
233     val[np] = val[p] + 1;
234
235     init(null + np, np);
236
237     while (p && !go[p][c]) {
238         go[p][c] = np;
239         p = par[p];
240     }
241
242     if (!p) {
243         par[np] = 1;
244         new_leaf(null + np, null + par[np]);
245     }
246     else {
247         int q = go[p][c];
248
249         if (val[q] == val[p] + 1) {
250             par[np] = q;
251             new_leaf(null + np, null + par[np]);
252         }
253         else {
254             int nq = ++sam_cnt;
255             val[nq] = val[p] + 1;
256             memcpy(go[nq], go[q], sizeof(go[q]));
257
258             init(null + nq, nq);
259
260             new_node(null + nq, null + q, null +
261                     ↪ par[q]);
262             new_leaf(null + np, null + nq);
263
264             par[nq] = par[q];
265             par[np] = par[q] = nq;
266
267             while (p && go[p][c] == q) {
268                 go[p][c] = nq;
269                 p = par[p];
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     n = strlen(str + 1);
287

```

```

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 |
321 |     return 0;
}

```

```

33 |     last = go[p][c];
34 |
35 |     // a[last]++;
36 }

```

5.6.1 广义回文树

(代码是梯子部分的版本, 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             dfs1(y);
44
45             mxd[x] = max(mxd[x], mxd[y]);
46             if (mxd[y] > mxd[son[x]])
47                 son[x] = y;
48
49 }
50
51 void dfs2(int x) {
52     if (x == son[f[0][x]])
53         top[x] = top[f[0][x]];
54     else
55         top[x] = x;
56
57     for (int i = 0; i < 26; i++)
58         if (trie[x][i])
59             int y = trie[x][i];
60             dfs2(y);
61
62 }

```

5.6 回文树 PAM

```

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         par[q] = go[p][c];
32         last = go[now][c] = q;
33     }
34 }

```

```

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     ↑
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     int new_last;
112
113     if (!go[p][c]) {
114         int q = ++pam_cnt, now = p;
115         val[q] = val[p] + 2;
116
117         p = getfail(x, par[p]);
118
119         par[q] = go[p][c];
120         new_last = go[now][c] = q;
121
122         for (int i = 0; i < 26; i++)
123             fail[q][i] = fail[par[q]][i];
124
125         if (get_char(x, val[par[q]]) >= 'a')
126             fail[q][get_char(x, val[par[q]]) - 'a'] =
127                 → par[q];
128
129         if (val[q] <= n)
130
131             for (int i = 0; i < len[x]; i++)
132                 v[x].push_back(q);
133
134             u = f[0][q];
135
136             for (int i = 0; i < len[x]; i++) {
137                 v[x].push_back(u);
138                 u = f[0][u];
139             }
140
141             u = x;
142             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
143                 // 要延长一倍
144                 v[x].push_back(u);
145                 u = f[0][u];
146             }
147
148             u = x;
149             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
150                 // 要延长一倍
151                 v[x].push_back(u);
152                 u = f[0][u];
153             }
154
155             u = x;
156             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
157                 // 要延长一倍
158                 v[x].push_back(u);
159             }
160
161             u = x;
162
163             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
164                 // 要延长一倍
165                 v[x].push_back(u);
166             }
167
168             u = x;
169             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
170                 // 要延长一倍
171                 v[x].push_back(u);
172             }
173
174             u = x;
175
176             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
177                 // 要延长一倍
178                 v[x].push_back(u);
179             }
180
181             u = x;
182             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
183                 // 要延长一倍
184                 v[x].push_back(u);
185             }
186
187             u = x;
188             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
189                 // 要延长一倍
190                 v[x].push_back(u);
191             }
192
193             u = x;
194
195             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
196                 // 要延长一倍
197                 v[x].push_back(u);
198             }
199
200             u = x;
201             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
202                 // 要延长一倍
203                 v[x].push_back(u);
204             }
205
206             u = x;
207             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
208                 // 要延长一倍
209                 v[x].push_back(u);
210             }
211
212             u = x;
213             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
214                 // 要延长一倍
215                 v[x].push_back(u);
216             }
217
218             u = x;
219             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
220                 // 要延长一倍
221                 v[x].push_back(u);
222             }
223
224             u = x;
225             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
226                 // 要延长一倍
227                 v[x].push_back(u);
228             }
229
230             u = x;
231             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
232                 // 要延长一倍
233                 v[x].push_back(u);
234             }
235
236             u = x;
237             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
238                 // 要延长一倍
239                 v[x].push_back(u);
240             }
241
242             u = x;
243             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
244                 // 要延长一倍
245                 v[x].push_back(u);
246             }
247
248             u = x;
249             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
250                 // 要延长一倍
251                 v[x].push_back(u);
252             }
253
254             u = x;
255             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
256                 // 要延长一倍
257                 v[x].push_back(u);
258             }
259
260             u = x;
261             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
262                 // 要延长一倍
263                 v[x].push_back(u);
264             }
265
266             u = x;
267             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
268                 // 要延长一倍
269                 v[x].push_back(u);
270             }
271
272             u = x;
273             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
274                 // 要延长一倍
275                 v[x].push_back(u);
276             }
277
278             u = x;
279             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
280                 // 要延长一倍
281                 v[x].push_back(u);
282             }
283
284             u = x;
285             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
286                 // 要延长一倍
287                 v[x].push_back(u);
288             }
289
290             u = x;
291             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
292                 // 要延长一倍
293                 v[x].push_back(u);
294             }
295
296             u = x;
297             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
298                 // 要延长一倍
299                 v[x].push_back(u);
300             }
301
302             u = x;
303             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
304                 // 要延长一倍
305                 v[x].push_back(u);
306             }
307
308             u = x;
309             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
310                 // 要延长一倍
311                 v[x].push_back(u);
312             }
313
314             u = x;
315             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
316                 // 要延长一倍
317                 v[x].push_back(u);
318             }
319
320             u = x;
321             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
322                 // 要延长一倍
323                 v[x].push_back(u);
324             }
325
326             u = x;
327             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
328                 // 要延长一倍
329                 v[x].push_back(u);
330             }
331
332             u = x;
333             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
334                 // 要延长一倍
335                 v[x].push_back(u);
336             }
337
338             u = x;
339             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
340                 // 要延长一倍
341                 v[x].push_back(u);
342             }
343
344             u = x;
345             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
346                 // 要延长一倍
347                 v[x].push_back(u);
348             }
349
350             u = x;
351             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
352                 // 要延长一倍
353                 v[x].push_back(u);
354             }
355
356             u = x;
357             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
358                 // 要延长一倍
359                 v[x].push_back(u);
360             }
361
362             u = x;
363             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
364                 // 要延长一倍
365                 v[x].push_back(u);
366             }
367
368             u = x;
369             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
370                 // 要延长一倍
371                 v[x].push_back(u);
372             }
373
374             u = x;
375             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
376                 // 要延长一倍
377                 v[x].push_back(u);
378             }
379
380             u = x;
381             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
382                 // 要延长一倍
383                 v[x].push_back(u);
384             }
385
386             u = x;
387             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
388                 // 要延长一倍
389                 v[x].push_back(u);
390             }
391
392             u = x;
393             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
394                 // 要延长一倍
395                 v[x].push_back(u);
396             }
397
398             u = x;
399             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
400                 // 要延长一倍
401                 v[x].push_back(u);
402             }
403
404             u = x;
405             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
406                 // 要延长一倍
407                 v[x].push_back(u);
408             }
409
410             u = x;
411             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
412                 // 要延长一倍
413                 v[x].push_back(u);
414             }
415
416             u = x;
417             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
418                 // 要延长一倍
419                 v[x].push_back(u);
420             }
421
422             u = x;
423             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
424                 // 要延长一倍
425                 v[x].push_back(u);
426             }
427
428             u = x;
429             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
430                 // 要延长一倍
431                 v[x].push_back(u);
432             }
433
434             u = x;
435             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
436                 // 要延长一倍
437                 v[x].push_back(u);
438             }
439
440             u = x;
441             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
442                 // 要延长一倍
443                 v[x].push_back(u);
444             }
445
446             u = x;
447             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
448                 // 要延长一倍
449                 v[x].push_back(u);
450             }
451
452             u = x;
453             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
454                 // 要延长一倍
455                 v[x].push_back(u);
456             }
457
458             u = x;
459             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
460                 // 要延长一倍
461                 v[x].push_back(u);
462             }
463
464             u = x;
465             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
466                 // 要延长一倍
467                 v[x].push_back(u);
468             }
469
470             u = x;
471             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
472                 // 要延长一倍
473                 v[x].push_back(u);
474             }
475
476             u = x;
477             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
478                 // 要延长一倍
479                 v[x].push_back(u);
480             }
481
482             u = x;
483             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
484                 // 要延长一倍
485                 v[x].push_back(u);
486             }
487
488             u = x;
489             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
490                 // 要延长一倍
491                 v[x].push_back(u);
492             }
493
494             u = x;
495             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
496                 // 要延长一倍
497                 v[x].push_back(u);
498             }
499
500             u = x;
501             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
502                 // 要延长一倍
503                 v[x].push_back(u);
504             }
505
506             u = x;
507             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
508                 // 要延长一倍
509                 v[x].push_back(u);
510             }
511
512             u = x;
513             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
514                 // 要延长一倍
515                 v[x].push_back(u);
516             }
517
518             u = x;
519             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
520                 // 要延长一倍
521                 v[x].push_back(u);
522             }
523
524             u = x;
525             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
526                 // 要延长一倍
527                 v[x].push_back(u);
528             }
529
530             u = x;
531             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
532                 // 要延长一倍
533                 v[x].push_back(u);
534             }
535
536             u = x;
537             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
538                 // 要延长一倍
539                 v[x].push_back(u);
540             }
541
542             u = x;
543             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
544                 // 要延长一倍
545                 v[x].push_back(u);
546             }
547
548             u = x;
549             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
550                 // 要延长一倍
551                 v[x].push_back(u);
552             }
553
554             u = x;
555             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
556                 // 要延长一倍
557                 v[x].push_back(u);
558             }
559
560             u = x;
561             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
562                 // 要延长一倍
563                 v[x].push_back(u);
564             }
565
566             u = x;
567             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
568                 // 要延长一倍
569                 v[x].push_back(u);
570             }
571
572             u = x;
573             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
574                 // 要延长一倍
575                 v[x].push_back(u);
576             }
577
578             u = x;
579             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
580                 // 要延长一倍
581                 v[x].push_back(u);
582             }
583
584             u = x;
585             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
586                 // 要延长一倍
587                 v[x].push_back(u);
588             }
589
590             u = x;
591             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
592                 // 要延长一倍
593                 v[x].push_back(u);
594             }
595
596             u = x;
597             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
598                 // 要延长一倍
599                 v[x].push_back(u);
600             }
601
602             u = x;
603             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
604                 // 要延长一倍
605                 v[x].push_back(u);
606             }
607
608             u = x;
609             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
610                 // 要延长一倍
611                 v[x].push_back(u);
612             }
613
614             u = x;
615             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
616                 // 要延长一倍
617                 v[x].push_back(u);
618             }
619
620             u = x;
621             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
622                 // 要延长一倍
623                 v[x].push_back(u);
624             }
625
626             u = x;
627             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
628                 // 要延长一倍
629                 v[x].push_back(u);
630             }
631
632             u = x;
633             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
634                 // 要延长一倍
635                 v[x].push_back(u);
636             }
637
638             u = x;
639             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
640                 // 要延长一倍
641                 v[x].push_back(u);
642             }
643
644             u = x;
645             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
646                 // 要延长一倍
647                 v[x].push_back(u);
648             }
649
650             u = x;
651             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
652                 // 要延长一倍
653                 v[x].push_back(u);
654             }
655
656             u = x;
657             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
658                 // 要延长一倍
659                 v[x].push_back(u);
660             }
661
662             u = x;
663             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
664                 // 要延长一倍
665                 v[x].push_back(u);
666             }
667
668             u = x;
669             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
670                 // 要延长一倍
671                 v[x].push_back(u);
672             }
673
674             u = x;
675             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
676                 // 要延长一倍
677                 v[x].push_back(u);
678             }
679
680             u = x;
681             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
682                 // 要延长一倍
683                 v[x].push_back(u);
684             }
685
686             u = x;
687             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
688                 // 要延长一倍
689                 v[x].push_back(u);
690             }
691
692             u = x;
693             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
694                 // 要延长一倍
695                 v[x].push_back(u);
696             }
697
698             u = x;
699             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
700                 // 要延长一倍
701                 v[x].push_back(u);
702             }
703
704             u = x;
705             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
706                 // 要延长一倍
707                 v[x].push_back(u);
708             }
709
710             u = x;
711             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
712                 // 要延长一倍
713                 v[x].push_back(u);
714             }
715
716             u = x;
717             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
718                 // 要延长一倍
719                 v[x].push_back(u);
720             }
721
722             u = x;
723             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
724                 // 要延长一倍
725                 v[x].push_back(u);
726             }
727
728             u = x;
729             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
730                 // 要延长一倍
731                 v[x].push_back(u);
732             }
733
734             u = x;
735             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
736                 // 要延长一倍
737                 v[x].push_back(u);
738             }
739
740             u = x;
741             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
742                 // 要延长一倍
743                 v[x].push_back(u);
744             }
745
746             u = x;
747             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
748                 // 要延长一倍
749                 v[x].push_back(u);
750             }
751
752             u = x;
753             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
754                 // 要延长一倍
755                 v[x].push_back(u);
756             }
757
758             u = x;
759             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
760                 // 要延长一倍
761                 v[x].push_back(u);
762             }
763
764             u = x;
765             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
766                 // 要延长一倍
767                 v[x].push_back(u);
768             }
769
770             u = x;
771             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
772                 // 要延长一倍
773                 v[x].push_back(u);
774             }
775
776             u = x;
777             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
778                 // 要延长一倍
779                 v[x].push_back(u);
780             }
781
782             u = x;
783             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
784                 // 要延长一倍
785                 v[x].push_back(u);
786             }
787
788             u = x;
789             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
790                 // 要延长一倍
791                 v[x].push_back(u);
792             }
793
794             u = x;
795             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
796                 // 要延长一倍
797                 v[x].push_back(u);
798             }
799
800             u = x;
801             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
802                 // 要延长一倍
803                 v[x].push_back(u);
804             }
805
806             u = x;
807             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
808                 // 要延长一倍
809                 v[x].push_back(u);
810             }
811
812             u = x;
813             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
814                 // 要延长一倍
815                 v[x].push_back(u);
816             }
817
818             u = x;
819             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
820                 // 要延长一倍
821                 v[x].push_back(u);
822             }
823
824             u = x;
825             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
826                 // 要延长一倍
827                 v[x].push_back(u);
828             }
829
830             u = x;
831             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
832                 // 要延长一倍
833                 v[x].push_back(u);
834             }
835
836             u = x;
837             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
838                 // 要延长一倍
839                 v[x].push_back(u);
840             }
841
842             u = x;
843             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
844                 // 要延长一倍
845                 v[x].push_back(u);
846             }
847
848             u = x;
849             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
850                 // 要延长一倍
851                 v[x].push_back(u);
852             }
853
854             u = x;
855             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
856                 // 要延长一倍
857                 v[x].push_back(u);
858             }
859
860             u = x;
861             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
862                 // 要延长一倍
863                 v[x].push_back(u);
864             }
865
866             u = x;
867             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
868                 // 要延长一倍
869                 v[x].push_back(u);
870             }
871
872             u = x;
873             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
874                 // 要延长一倍
875                 v[x].push_back(u);
876             }
877
878             u = x;
879             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
880                 // 要延长一倍
881                 v[x].push_back(u);
882             }
883
884             u = x;
885             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
886                 // 要延长一倍
887                 v[x].push_back(u);
888             }
889
890             u = x;
891             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
892                 // 要延长一倍
893                 v[x].push_back(u);
894             }
895
896             u = x;
897             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
898                 // 要延长一倍
899                 v[x].push_back(u);
900             }
901
902             u = x;
903             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
904                 // 要延长一倍
905                 v[x].push_back(u);
906             }
907
908             u = x;
909             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
910                 // 要延长一倍
911                 v[x].push_back(u);
912             }
913
914             u = x;
915             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
916                 // 要延长一倍
917                 v[x].push_back(u);
918             }
919
920             u = x;
921             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
922                 // 要延长一倍
923                 v[x].push_back(u);
924             }
925
926             u = x;
927             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
928                 // 要延长一倍
929                 v[x].push_back(u);
930             }
931
932             u = x;
933             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
934                 // 要延长一倍
935                 v[x].push_back(u);
936             }
937
938             u = x;
939             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
940                 // 要延长一倍
941                 v[x].push_back(u);
942             }
943
944             u = x;
945             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
946                 // 要延长一倍
947                 v[x].push_back(u);
948             }
949
950             u = x;
951             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
952                 // 要延长一倍
953                 v[x].push_back(u);
954             }
955
956             u = x;
957             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
958                 // 要延长一倍
959                 v[x].push_back(u);
960             }
961
962             u = x;
963             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
964                 // 要延长一倍
965                 v[x].push_back(u);
966             }
967
968             u = x;
969             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
970                 // 要延长一倍
971                 v[x].push_back(u);
972             }
973
974             u = x;
975             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
976                 // 要延长一倍
977                 v[x].push_back(u);
978             }
979
980             u = x;
981             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
982                 // 要延长一倍
983                 v[x].push_back(u);
984             }
985
986             u = x;
987             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
988                 // 要延长一倍
989                 v[x].push_back(u);
990             }
991
992             u = x;
993             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
994                 // 要延长一倍
995                 v[x].push_back(u);
996             }
997
998             u = x;
999             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
1000                // 要延长一倍
1001                v[x].push_back(u);
1002            }
1003
1004             u = x;
1005             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
1006                // 要延长一倍
1007                v[x].push_back(u);
1008            }
1009
1010             u = x;
1011             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
1012                // 要延长一倍
1013                v[x].push_back(u);
1014            }
1015
1016             u = x;
1017             for (int i = 0; i < len[x]; i++) { // 梯子剖分,
1018                // 要延长一倍

```

```

197         }
198     else
199         last = del(last);
200
201     queries[last].push_back(i);
202 }
203
204 dfs1(1);
205 dfs2(1);
206
207 for (int j = 1; j <= log_tbl[trie_cnt]; j++)
208     for (int i = 1; i <= trie_cnt; i++)
209         f[j][i] = f[j - 1][f[j - 1][i]];
210
211 par[0] = pam_cnt = 1;
212
213
214 for (int i = 0; i < 26; i++)
215     fail[0][i] = fail[1][i] = 1;
216
217 val[1] = -1;
218 pam_last[1] = 1;
219
220 bfs();
221
222 for (int i = 0; i <= m; i++)
223     printf("%d\n", ans[i]);
224
225 for (int j = 0; j <= log_tbl[trie_cnt]; j++)
226     memset(f[j], 0, sizeof(f[j]));
227
228 for (int i = 1; i <= trie_cnt; i++) {
229     chr[i] = 0;
230     d[i] = mxd[i] = son[i] = top[i] = len[i] =
231         → pam_last[i] = sum[i] = 0;
232     v[i].clear();
233     queries[i].clear();
234
235     memset(trie[i], 0, sizeof(trie[i]));
236 }
237 trie_cnt = 0;
238
239 for (int i = 0; i <= pam_cnt; i++) {
240     val[i] = par[i] = weight[i];
241
242     memset(go[i], 0, sizeof(go[i]));
243     memset(fail[i], 0, sizeof(fail[i]));
244 }
245 pam_cnt = 0;
246
247 }
248
249 return 0;
}

```

```

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。当然它不能做最长公共后缀，不过可以用一个树链的并来做一个串问题。

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 也就是他们在后缀树上的 LCA。

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; 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

```

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 - 1]);
17
18 int solve(long long tmp) {
19     memset(f, 63, sizeof(f));
20     f[0] = 0;
21
22     int head = 1, tail = 0;
23
24     for (int i = 1; i <= n; i++) {
25         f[i] = calc(1, i);
26         g[i] = 1;
27
28         while (head < tail && p[head + 1] <= i)
29             head++;
30         if (head <= tail) {
31             if (f[q[head]] + calc(q[head] + 1, i) <
32                 f[i]) {
33                 f[i] = f[q[head]] + calc(q[head] + 1,
34                                           i);
35                 g[i] = g[q[head]] + 1;
36             }
37             while (head < tail && p[head + 1] <= i + 1)
38                 head++;
39             if (head <= tail)
40                 p[head] = i + 1;
41         }
42         f[i] += tmp;
43
44         int r = n;
45
46         while (head <= tail) {
47             if (f[q[tail]] + calc(q[tail] + 1, p[tail]) <
48                 f[i] + calc(i + 1, p[tail])) {
49                 r = p[tail] - 1;
50                 tail--;
51             }
52             else if (f[q[tail]] + calc(q[tail] + 1, r) <
53                 f[i] + calc(i + 1, r)) {
54                 if (r < n) {
55                     q[++tail] = i;
56                     p[tail] = r + 1;
57                 }
58                 break;
59             }
60             else {
61                 int L = p[tail], R = r;
62                 while (L < R) {
63                     int M = (L + R) / 2;
64                     if (f[q[tail]] + calc(q[tail] + 1, M) <=
65                         f[i] + calc(i + 1, M))
66                         L = M + 1;
67                     else
68                         R = M;
69                 }
70                 q[++tail] = i;
71                 p[tail] = L;
72             }
73             if (head > tail) {
74                 q[++tail] = i;
75                 p[tail] = i + 1;
76             }
77         }
78
79         return g[n];
80     }
81
82 }

```

```

61             if (f[q[tail]] + calc(q[tail] + 1,
62                                     M) <= f[i] + calc(i + 1, M))
63                 L = M + 1;
64             else
65                 R = M;
66         }
67
68         q[++tail] = i;
69         p[tail] = L;
70
71         break;
72     }
73     if (head > tail) {
74         q[++tail] = i;
75         p[tail] = i + 1;
76     }
77 }
78
79 return g[n];
80 }
```

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    for (int i = 0; i < n; i++) {
29        for (int j = i + 1; j <= n; j++) {
30            if (a[i] < a[j])
31                ans += f[i][j];
32            else
33                ans -= f[i][j];
34        }
35    }
36
37    cout << ans;
38 }
```

```
23 | }
24 |
25 |     for (int i = n - 1; i; i--)
26 |         a[i] = min(a[i], a[i + 1]);
27 |
28 | // b_i >= b_{i - 1} && b_i <= a_i
29 | // 我们可以假设 b_i <= a_i 是必定被满足的, 然后对 bi
30 |     ↪ 非严格递增的条件进行容斥, 枚举某一段是严格递减的
31 |     // 如果 [j, i] 严格递减, 显然它们都 <= a_j, 所以这个
32 |     ↪ 区间的方案数是 {a_j \choose i - j + 1}
33 |     // 如果 i 是合法的, 直接一个个转移即可, 因为这一部分的
34 |     ↪ 转移和区间长度没有关系
35 |
36 |     for (int i = 1; i <= n; i++) {
37 |         f[i][0] = 1;
38 |
39 |         for (int j = 1; j <= n - i + 1 && j <= a[i]; j+
40 |             ↪ +)
41 |             f[i][j] = (long long)f[i][j - 1] * (a[i] -
42 |                 ↪ j + 1) % p * inv[j] % p;
43 |
44 |     dp[0] = 1;
45 |
46 |     for (int i = 1; i <= n; i++) {
47 |         dp[i] = (long long)dp[i - 1] * a[i] % p;
48 |
49 |         for (int j = 1; j < i; j++) {
50 |             int tmp = (long long)dp[j - 1] * f[j][i - j
51 |                 ↪ + 1] % p;
52 |
53 |             if ((i - j) % 2)
54 |                 tmp = p - tmp;
55 |             dp[i] = (dp[i] + tmp) % p;
56 |
57 |     }
58 |
59 |     printf("%d\n", dp[n]);
60 |
61 |     return 0;
62 }
```

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);
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133

```

```

struct quad_edge {
    point origin;
    quad_edge *rot = nullptr;
    quad_edge *onext = nullptr;
    bool used = false;
}
quad_edge *rev() const {
    return rot -> rot;
}
quad_edge *lnext() const {
    return rot -> rev() -> onext -> rot;
}
quad_edge *oprev() const {
    return rot -> onext -> rot;
}
point dest() const {
    return rev() -> origin;
};
};

quad_edge *make_edge(point from, point to) {
    quad_edge *e1 = new quad_edge;
    quad_edge *e2 = new quad_edge;
    quad_edge *e3 = new quad_edge;
    quad_edge *e4 = new quad_edge;
    e1 -> origin = from;
    e2 -> origin = to;
    e3 -> origin = e4 -> origin = inf_point;
    e1 -> rot = e3;
    e2 -> rot = e4;
    e3 -> rot = e2;
    e4 -> rot = e1;
    e1 -> onext = e1;
    e2 -> onext = e2;
    e3 -> onext = e4;
    e4 -> onext = e3;
    return e1;
}

void splice(quad_edge *a, quad_edge *b) { // 拼接
    swap(a -> onext -> rot -> onext, b -> onext -> rot
        -> onext);
    swap(a -> onext, b -> onext);
}

void delete_edge(quad_edge *e) {
    splice(e, e -> oprev());
    splice(e -> rev(), e -> rev() -> oprev());
    delete e -> rev() -> rot;
    delete e -> rev();
    delete e -> rot;
    delete e;
}

quad_edge *connect(quad_edge *a, quad_edge *b) {
    quad_edge *e = make_edge(a -> dest(), b -> origin);
    splice(e, a -> lnext());
    splice(e -> rev(), b);
    return e;
}

bool left_of(point p, quad_edge *e) {
    return p.cross(e -> origin, e -> dest()) > 0;
}

```



```

256
257     while (e -> onext -> dest().cross(e -> dest(), e ->
258         -> origin) < 0)
259     |     e = e -> onext;
260
261     auto add = [&p, &e, &edges] () { // 修改 p, e, edges
262     |     quad_edge *cur = e;
263     |     do {
264     |     |     cur -> used = true;
265     |     |     p.push_back(cur -> origin);
266     |     |     edges.push_back(cur -> rev());
267
268     |     |     cur = cur -> lnext();
269     } while (cur != e);
270
271     add();
272     p.clear();
273
274     int kek = 0;
275     while (kek < (int)edges.size())
276     |     if (!(*e = edges[kek++]) -> used)
277     |     |     add();
278
279     vector<tuple<point, point, point>> ans;
280     for (int i = 0; i < (int)p.size(); i += 3)
281     |     ans.push_back(make_tuple(p[i], p[i + 1], p[i +
282         -> 2]));
283
284     return ans; // 返回每个三角形
285 }
286
287 #define sq(x) ((x) * (ll)(x)) // 平方
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     return 0;
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.

42 6 条评论

收起 ▲

8 杂项

8.1 $O(1)$ 快速乘

```

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 // →用
5 long long mul(long long a, long long b, long long p) {
6     a %= p;
7     b %= p;
8     return ((a * b - p * (long long)((long double)a / p
9         → * b + 0.5)) % p + p) % p;
10 }
11
12 // 指令集快速乘
13 // 试机记得测试能不能过编译
14 inline long long mul(const long long a, const long long
15     → b, const long long p) {
16     long long ans;
17     __asm__ __volatile__ ("\\tmulq %%rbx\\n\\tdivq
18     → %%rcx\\n" : "=d"(ans) : "a"(a), "b"(b), "c"(p));
19     return ans;
20 }
21
22 // int乘法取模，大概比直接做快一倍
23 inline int mul_mod(int a, int b, int p) {
24     int ans;
25     __asm__ __volatile__ ("\\tmull %%ebx\\n\\tdivl
26     → %%ecx\\n" : "=d"(ans) : "a"(a), "b"(b), "c"(p));
27     return ans;
28 }
```

8.2 Kahan 求和算法

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
7     float本身就不准确
8
9 x = decimal.Decimal('50.5679'). \
10    quantize(decimal.Decimal('0.00')) # 保留两位小数,
11    → 50.57
12 x = decimal.Decimal('50.5679'). \
13    quantize(decimal.Decimal('0.00'),
14    → decimal.ROUND_HALF_UP) # 四舍五入
15
16 # 第二个参数可选如下:
17 # ROUND_HALF_UP 四舍五入
18 # ROUND_HALF_DOWN 五舍六入
19 # ROUND_HALF_EVEN 银行家舍入法，舍入到最近的偶数
20 # 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 # 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     friend big_decimal operator >> (const big_decimal
121         &u, int k) {
122         big_decimal o = u;
123         return o >>= k;
124     }
125
126     friend int cmp(const big_decimal &u, const
127         &big_decimal &v) {
128         if (u.negative || v.negative) {
129             if (u.negative && v.negative)
130                 return -cmp(-u, -v);
131
132             if (u.negative)
133                 return -1;
134
135             if (v.negative)
136                 return 1;
137         }
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 big_decimal decimal_plus(const big_decimal
170         &u, const big_decimal &v) { // 保证u, v均为正数
171         // 的话可以直接调用
172         big_decimal o;
173
174         o.a[0] = max(u.a[0], v.a[0]);
175
176         for (int i = 1; i <= u.a[0] || i <= v.a[0]; i+
177             &+) {
178             o.a[i] += u.a[i] + v.a[i];
179
180             if (o.a[i] >= 10) {
181                 o.a[i + 1]++;
182                 o.a[i] -= 10;
183             }
184         }
185     }

```



```

316     |     if (o.a[i] >= 10) {
317     |     o.a[i + 1] += o.a[i] / 10;
318     |     o.a[i] %= 10;
319     }
320
321     |     if (o.a[a[0] + 1])
322     |     o.a[0]++;
323
324     return o;
325 }
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 friend big_decimal operator / (const big_decimal
341     → &u, const big_decimal &v) { // v不能是0
342     if (u.negative || v.negative) {
343     |     big_decimal o = abs(u) / abs(v);
344
345     if (u.negative ^ v.negative)
346     |     return -o;
347     return o;
348 }
349
350     return decimal_divide(u, v).first;
351 }
352
353 friend big_decimal operator % (const big_decimal
354     → &u, const big_decimal &v) {
355     if (u.negative || v.negative) {
356     |     big_decimal o = abs(u) % abs(v);
357
358     if (u.negative ^ v.negative)
359     |     return -o;
360     return o;
361 }
362
363     return decimal_divide(u, v).second;
364 }
365 };

```

```

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
1945555039024054273	27	56	5
4179340454199820289	29	57	3

*注: 1005060097 有点危险, 在变化长度大于 $524288 = 2^{19}$ 时不可用。

8.8 xorshift

```

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 }

```

8.5 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];

```

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 n, const T &value)`: 创建一个 `vector`, 元素个数为 `n`, 且值均为 `value`。
- `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 $\Theta(\log n)$; 果只有 `increase_key`, 那么 `modify` 为均摊 $O(1)$
- * “不支持” 不是不能用, 而是用起来很慢 csdn.net/TRiddle

常用操作:

- `push()`: 向堆中压入一个元素, 返回迭代器。
- `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
8 // rb_tree_tag 红黑树 (还有 splay_tree_tag 和 ov_tree_tag,
9 // → 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。
- `join(other)`: 将 `other` 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。`other` 树会被删除。

- `split(x, other)`: 分裂成两部分，小于等于 `x` 的属于当前树，其余的属于 `other`。

(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
   ↪ 毒
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)`: 向前循环移动，移动后 `mid` 位置的元素会跑到 `begin` 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::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::tuple(a, b, c)`。

8.13.4 complex

- `complex<double> imaginary = 1i, x = 2 + 3i`: 语法糖，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)` 就行。)
- `polar(len, theta)`: 用模长和辐角构造复数。

8.14 一些游戏

8.14.1 德州扑克

一般来说德州扑克 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州扑克区别不大。

```

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 = -1; // error
21
22         char b = s[1];
23         suit = b; // Club, Diamond, Heart, Spade
24     }
25
26     friend bool operator < (const Card &a, const Card
27       &b) {
28         return a.value < b.value;
29     }
30
31     friend bool operator == (const Card &a, const Card
32       &b) {
33         return a.value == b.value;
34     }
35
36     constexpr int Highcard = 1, Pair = 2, TwoPairs = 3,
37       ↪ ThreeofaKind = 4, Straight = 5,
38     Flush = 6, FullHouse = 7, FourofaKind = 8,
39       ↪ StraightFlush = 9;
40
41     struct Hand {
42         vector<Card> v;
43         int type;
44
45         Hand() : type(0) {}
46
47         Hand(const Hand &o) : v(o.v), type(o.type) {}
48
49         Hand(const vector<Card> &v) : v(v), type(0) {}
50     };
51
52     enum class Suit { Club, Diamond, Heart, Spade };
53
54     enum class Rank { Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, Ace };
55
56     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
57     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
58
59     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
60       ↪ ThreeofaKind = 4, Straight = 5,
61     Flush = 6, FullHouse = 7, FourofaKind = 8,
62       ↪ StraightFlush = 9;
63
64     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
65     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
66
67     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
68       ↪ ThreeofaKind = 4, Straight = 5,
69     Flush = 6, FullHouse = 7, FourofaKind = 8,
70       ↪ StraightFlush = 9;
71
72     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
73     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
74
75     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
76       ↪ ThreeofaKind = 4, Straight = 5,
77     Flush = 6, FullHouse = 7, FourofaKind = 8,
78       ↪ StraightFlush = 9;
79
80     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
81     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
82
83     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
84       ↪ ThreeofaKind = 4, Straight = 5,
85     Flush = 6, FullHouse = 7, FourofaKind = 8,
86       ↪ StraightFlush = 9;
87
88     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
89     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
90
91     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
92       ↪ ThreeofaKind = 4, Straight = 5,
93     Flush = 6, FullHouse = 7, FourofaKind = 8,
94       ↪ StraightFlush = 9;
95
96     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
97     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
98
99     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
100      ↪ ThreeofaKind = 4, Straight = 5,
101     Flush = 6, FullHouse = 7, FourofaKind = 8,
102       ↪ StraightFlush = 9;
103
104     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
105     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
106
107     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
108       ↪ ThreeofaKind = 4, Straight = 5,
109     Flush = 6, FullHouse = 7, FourofaKind = 8,
110       ↪ StraightFlush = 9;
111
112     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
113     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
114
115     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
116       ↪ ThreeofaKind = 4, Straight = 5,
117     Flush = 6, FullHouse = 7, FourofaKind = 8,
118       ↪ StraightFlush = 9;
119
120     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
121     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
122
123     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
124       ↪ ThreeofaKind = 4, Straight = 5,
125     Flush = 6, FullHouse = 7, FourofaKind = 8,
126       ↪ StraightFlush = 9;
127
128     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
129     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
130
131     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
132       ↪ ThreeofaKind = 4, Straight = 5,
133     Flush = 6, FullHouse = 7, FourofaKind = 8,
134       ↪ StraightFlush = 9;
135
136     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
137     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
138
139     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
140       ↪ ThreeofaKind = 4, Straight = 5,
141     Flush = 6, FullHouse = 7, FourofaKind = 8,
142       ↪ StraightFlush = 9;
143
144     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
145     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
146
147     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
148       ↪ ThreeofaKind = 4, Straight = 5,
149     Flush = 6, FullHouse = 7, FourofaKind = 8,
150       ↪ StraightFlush = 9;
151
152     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
153     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
154
155     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
156       ↪ ThreeofaKind = 4, Straight = 5,
157     Flush = 6, FullHouse = 7, FourofaKind = 8,
158       ↪ StraightFlush = 9;
159
160     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
161     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
162
163     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
164       ↪ ThreeofaKind = 4, Straight = 5,
165     Flush = 6, FullHouse = 7, FourofaKind = 8,
166       ↪ StraightFlush = 9;
167
168     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
169     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
170
171     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
172       ↪ ThreeofaKind = 4, Straight = 5,
173     Flush = 6, FullHouse = 7, FourofaKind = 8,
174       ↪ StraightFlush = 9;
175
176     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
177     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
178
179     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
180       ↪ ThreeofaKind = 4, Straight = 5,
181     Flush = 6, FullHouse = 7, FourofaKind = 8,
182       ↪ StraightFlush = 9;
183
184     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
185     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
186
187     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
188       ↪ ThreeofaKind = 4, Straight = 5,
189     Flush = 6, FullHouse = 7, FourofaKind = 8,
190       ↪ StraightFlush = 9;
191
192     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
193     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
194
195     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
196       ↪ ThreeofaKind = 4, Straight = 5,
197     Flush = 6, FullHouse = 7, FourofaKind = 8,
198       ↪ StraightFlush = 9;
199
200     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
201     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
202
203     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
204       ↪ ThreeofaKind = 4, Straight = 5,
205     Flush = 6, FullHouse = 7, FourofaKind = 8,
206       ↪ StraightFlush = 9;
207
208     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
209     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
210
211     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
212       ↪ ThreeofaKind = 4, Straight = 5,
213     Flush = 6, FullHouse = 7, FourofaKind = 8,
214       ↪ StraightFlush = 9;
215
216     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
217     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
218
219     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
220       ↪ ThreeofaKind = 4, Straight = 5,
221     Flush = 6, FullHouse = 7, FourofaKind = 8,
222       ↪ StraightFlush = 9;
223
224     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
225     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
226
227     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
228       ↪ ThreeofaKind = 4, Straight = 5,
229     Flush = 6, FullHouse = 7, FourofaKind = 8,
230       ↪ StraightFlush = 9;
231
232     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
233     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
234
235     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
236       ↪ ThreeofaKind = 4, Straight = 5,
237     Flush = 6, FullHouse = 7, FourofaKind = 8,
238       ↪ StraightFlush = 9;
239
240     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
241     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
242
243     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
244       ↪ ThreeofaKind = 4, Straight = 5,
245     Flush = 6, FullHouse = 7, FourofaKind = 8,
246       ↪ StraightFlush = 9;
247
248     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
249     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
250
251     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
252       ↪ ThreeofaKind = 4, Straight = 5,
253     Flush = 6, FullHouse = 7, FourofaKind = 8,
254       ↪ StraightFlush = 9;
255
256     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
257     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
258
259     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
260       ↪ ThreeofaKind = 4, Straight = 5,
261     Flush = 6, FullHouse = 7, FourofaKind = 8,
262       ↪ StraightFlush = 9;
263
264     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
265     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
266
267     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
268       ↪ ThreeofaKind = 4, Straight = 5,
269     Flush = 6, FullHouse = 7, FourofaKind = 8,
270       ↪ StraightFlush = 9;
271
272     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
273     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
274
275     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
276       ↪ ThreeofaKind = 4, Straight = 5,
277     Flush = 6, FullHouse = 7, FourofaKind = 8,
278       ↪ StraightFlush = 9;
279
280     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
281     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
282
283     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
284       ↪ ThreeofaKind = 4, Straight = 5,
285     Flush = 6, FullHouse = 7, FourofaKind = 8,
286       ↪ StraightFlush = 9;
287
288     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
289     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
290
291     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
292       ↪ ThreeofaKind = 4, Straight = 5,
293     Flush = 6, FullHouse = 7, FourofaKind = 8,
294       ↪ StraightFlush = 9;
295
296     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
297     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
298
299     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
300       ↪ ThreeofaKind = 4, Straight = 5,
301     Flush = 6, FullHouse = 7, FourofaKind = 8,
302       ↪ StraightFlush = 9;
303
304     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
305     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
306
307     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
308       ↪ ThreeofaKind = 4, Straight = 5,
309     Flush = 6, FullHouse = 7, FourofaKind = 8,
310       ↪ StraightFlush = 9;
311
312     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
313     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
314
315     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
316       ↪ ThreeofaKind = 4, Straight = 5,
317     Flush = 6, FullHouse = 7, FourofaKind = 8,
318       ↪ StraightFlush = 9;
319
320     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
321     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
322
323     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
324       ↪ ThreeofaKind = 4, Straight = 5,
325     Flush = 6, FullHouse = 7, FourofaKind = 8,
326       ↪ StraightFlush = 9;
327
328     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
329     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
330
331     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
332       ↪ ThreeofaKind = 4, Straight = 5,
333     Flush = 6, FullHouse = 7, FourofaKind = 8,
334       ↪ StraightFlush = 9;
335
336     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
337     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
338
339     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
340       ↪ ThreeofaKind = 4, Straight = 5,
341     Flush = 6, FullHouse = 7, FourofaKind = 8,
342       ↪ StraightFlush = 9;
343
344     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
345     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
346
347     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
348       ↪ ThreeofaKind = 4, Straight = 5,
349     Flush = 6, FullHouse = 7, FourofaKind = 8,
350       ↪ StraightFlush = 9;
351
352     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
353     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
354
355     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
356       ↪ ThreeofaKind = 4, Straight = 5,
357     Flush = 6, FullHouse = 7, FourofaKind = 8,
358       ↪ StraightFlush = 9;
359
360     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
361     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
362
363     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
364       ↪ ThreeofaKind = 4, Straight = 5,
365     Flush = 6, FullHouse = 7, FourofaKind = 8,
366       ↪ StraightFlush = 9;
367
368     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
369     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
370
371     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
372       ↪ ThreeofaKind = 4, Straight = 5,
373     Flush = 6, FullHouse = 7, FourofaKind = 8,
374       ↪ StraightFlush = 9;
375
376     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
377     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
378
379     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
380       ↪ ThreeofaKind = 4, Straight = 5,
381     Flush = 6, FullHouse = 7, FourofaKind = 8,
382       ↪ StraightFlush = 9;
383
384     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
385     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
386
387     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
388       ↪ ThreeofaKind = 4, Straight = 5,
389     Flush = 6, FullHouse = 7, FourofaKind = 8,
390       ↪ StraightFlush = 9;
391
392     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
393     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
394
395     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
396       ↪ ThreeofaKind = 4, Straight = 5,
397     Flush = 6, FullHouse = 7, FourofaKind = 8,
398       ↪ StraightFlush = 9;
399
400     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
401     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
402
403     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
404       ↪ ThreeofaKind = 4, Straight = 5,
405     Flush = 6, FullHouse = 7, FourofaKind = 8,
406       ↪ StraightFlush = 9;
407
408     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
409     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
410
411     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
412       ↪ ThreeofaKind = 4, Straight = 5,
413     Flush = 6, FullHouse = 7, FourofaKind = 8,
414       ↪ StraightFlush = 9;
415
416     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
417     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
418
419     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
420       ↪ ThreeofaKind = 4, Straight = 5,
421     Flush = 6, FullHouse = 7, FourofaKind = 8,
422       ↪ StraightFlush = 9;
423
424     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
425     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
426
427     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
428       ↪ ThreeofaKind = 4, Straight = 5,
429     Flush = 6, FullHouse = 7, FourofaKind = 8,
430       ↪ StraightFlush = 9;
431
432     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
433     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
434
435     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
436       ↪ ThreeofaKind = 4, Straight = 5,
437     Flush = 6, FullHouse = 7, FourofaKind = 8,
438       ↪ StraightFlush = 9;
439
440     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
441     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
442
443     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
444       ↪ ThreeofaKind = 4, Straight = 5,
445     Flush = 6, FullHouse = 7, FourofaKind = 8,
446       ↪ StraightFlush = 9;
447
448     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
449     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
450
451     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
452       ↪ ThreeofaKind = 4, Straight = 5,
453     Flush = 6, FullHouse = 7, FourofaKind = 8,
454       ↪ StraightFlush = 9;
455
456     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
457     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
458
459     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
460       ↪ ThreeofaKind = 4, Straight = 5,
461     Flush = 6, FullHouse = 7, FourofaKind = 8,
462       ↪ StraightFlush = 9;
463
464     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
465     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
466
467     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
468       ↪ ThreeofaKind = 4, Straight = 5,
469     Flush = 6, FullHouse = 7, FourofaKind = 8,
470       ↪ StraightFlush = 9;
471
472     static const string Suits[] = {"Club", "Diamond", "Heart", "Spade"};
473     static const string Ranks[] = {"Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"};
474
475     static const int Highcard = 1, Pair = 2, TwoPairs = 3,
476       ↪ ThreeofaKind = 4, Straight = 5,
477     Flush = 6, FullHouse = 7, FourofaKind = 8,
478       ↪ StraightFlush = 9;
4
```

```

47 void init_high() {
48     sort(v.begin(), v.end()); // 升序排序
49
50     bool straight = false;
51     if (v.back().value == 14) {
52         if (v[0].value == 2 && v[1].value == 3 &&
53             v[2].value == 4 && v[3].value == 5) {
54             straight = true;
55             rotate(v.begin(), v.begin() + 1,
56                     v.end());
57         }
58
59     if (!straight) {
60         bool ok = true;
61         for (int i = 1; i < 5; i++)
62             ok &= (v[i].value == v[i - 1].value +
63                     1);
64
65         if (ok)
66             straight = true;
67     }
68
69     bool flush = all_of(v.begin(), v.end(), [&]
70                         (const Card &a) {return a.suit ==
71                         v.front().suit;});
72
73     if (flush && straight) { // 同花顺
74         type = StraightFlush;
75         reverse(v.begin(), v.end());
76         return;
77     }
78
79     vector<int> c;
80     c.assign(15, 0);
81
82     for (auto &o : v)
83         c[o.value]++;
84
85     vector<int> kind[5];
86
87     for (int i = 2; i <= 14; i++)
88         if (c[i] > 1)
89             kind[c[i]].push_back(i);
90
91     if (!kind[4].empty()) { // 四条
92         type = FourofaKind;
93
94         for (int i = 0; i < 4; i++)
95             if (v[i].value != kind[4].front())
96                 swap(v[i], v.back());
97             break;
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    }
112
113    if (flush) {
114        type = Flush;
115        sort(v.begin(), v.end());
116        reverse(v.begin(), v.end());
117
118    }
119
120    if (straight) {
121        type = Straight;
122        reverse(v.begin(), v.end());
123
124    }
125
126    if (!kind[3].empty()) {
127        type = ThreeofaKind;
128
129        sort(v.begin(), v.end(), [&] (const Card
130                                     &a, const Card &b) {
131            bool ta = (a.value == kind[3].front()),
132                    tb = (b.value == kind[3].front());
133
134            return ta > tb;
135        });
136
137        if (v[3] < v[4])
138            swap(v[3], v[4]);
139
140    }
141
142    if ((int)kind[2].size() == 2) {
143        type = TwoPairs;
144
145        sort(v.begin(), v.end(), [&] (const Card
146                                     &a, const Card &b) {
147            bool ta = (c[a.value] == 2), tb =
148                    (c[b.value] == 2);
149
150            if (ta != tb)
151                return ta > tb;
152
153        });
154
155    }
156
157    if ((int)kind[2].size() == 1) {
158        type = Pair;
159
160        sort(v.begin(), v.end(), [&] (const Card
161                                     &a, const Card &b) {
162            bool ta = (c[a.value] == 2), tb =
163                    (c[b.value] == 2);
164
165            if (ta != tb)
166                return ta > tb;
167
168        });
169
170    }
171
172    type = Highcard;
173
174    sort(v.begin(), v.end());
175    reverse(v.begin(), v.end());
176
177    void init_low() {

```

```

178     |     for (auto &o : v)
179     |     if (o.value == 14)
180     |         o.value = 1;
181
182     |     sort(v.begin(), v.end());
183     |     reverse(v.begin(), v.end());
184 }
185
186 friend int cmp_high(const Hand &a, const Hand &b) {
187     if (a.type != b.type)
188         return a.type < b.type ? -1 : 1;
189
190     if (a.v != b.v)
191         return a.v < b.v ? -1 : 1;
192
193     return 0;
194 }
195
196 friend bool small_high(const Hand &a, const Hand
197     &b) {
198     return cmp_high(a, b) < 0;
199 }
200
201 friend int cmp_low(const Hand &a, const Hand &b) {
202     for (int i = 0; i < 5; i++)
203         if (a.v[i].value != b.v[i].value)
204             return a.v[i] < b.v[i] ? 1 : -1;
205
206     return 0;
207 }
208
209 friend bool small_low(const Hand &a, const Hand &b)
210     &{
211     return cmp_low(a, b) < 0;
212 }
213
214 bool operator ! () const {
215     return v.empty();
216 }
217
218 string str() const {
219     stringstream ss;
220
221     for (auto &o : v)
222         ss << o.value << ' ';
223
224     return ss.str();
225 }
226
227 Hand get_max_high(vector<Card> u, vector<Card> v) { // 
228     // private, public
229     Hand ans;
230
231     for (int i = 0; i < 4; i++)
232         for (int j = i + 1; j < 4; j++)
233             for (int k = 0; k < 5; k++)
234                 for (int p = k + 1; p < 5; p++)
235                     for (int q = p + 1; q < 5; q++) {
236                         Hand tmp({u[i], u[j], v[k],
237                             v[p], v[q]});
238
239                         if (!ans || cmp_high(tmp, ans)
240                             > 0)
241                             ans = tmp;
242
243     return ans;
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
268                         bool bad = false;
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                         if (bad)
277                             continue;
278
279                         Hand tmp(vec);
280                         if (!ans || cmp_low(tmp, ans) >
281                             0)
282                             ans = tmp;
283
284     return ans;
285 }
286
287 int main() {
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         else
341             ans[dh == -1] += p - p / 2;
342     }
343     else {
344         int dl = cmp_low(alice_low, bob_low);
345
346         if (!dl) {
347             ans[0] += p / 2 - p / 2 / 2;
348             ans[1] += p / 2 / 2;
349         }
350         else
351             ans[dl == -1] += p / 2;
352
353         if (!dh) {
354             ans[0] += p - p / 2 - (p - p / 2) / 2;
355             ans[1] += (p - p / 2) / 2;
356         }
357         else
358             ans[dh == -1] += p - p / 2;
359     }
360
361     cout << ans[0] << ' ' << ans[1] << '\n';
362
363     return 0;
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)

$$a_n = \frac{\binom{3n}{n}}{2^{n+1}}$$

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)

$$a_n = 2^n s_{n-2} \quad (n > 2), s_n \text{ 是上面的小施罗德数.}$$

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, 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(15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就是1次+1, 2次+2, 3次+4, 4次+8...的规律.

3. 乌拉姆数 (Ulam numbers, A002858)

1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, 57, 62, 69, 72, 77, 82, 87, 97, 99, 102, 106, 114, 126, 131, 138, 145, 148, 155, 175, 177, 180, 182, 189, 197, 206, 209, 219, 221, 236, 238, 241, 243, 253, 258, 260, 273, 282, 309, 316, 319, 324, 339, ...

$a_1 = 1, a_2 = 2, a_n$ 表示在所有 $> a_{n-1}$ 的数中, 最小的, 能被表示成 (前面的两个不同的元素的和) 的数.

8.16 编译选项

- `-O2 -g -std=c++20`: 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。
- `#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)

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/MonoWindows 可用)

8.17.3 快捷键

- `Ctrl + `` (键盘左上角的倒引号): 显示终端
- `Ctrl + 1/2/3`: 切到对应栏
- `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 + [or]`: 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 + \`: 水平分栏, 最多 3 栏

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，尤其是那一种怨怒的感情。想骂人的时候而不骂，时常在身体上弄出毛病，所以想骂人时，骂骂何妨？

但是，骂人是一种高深的学问，不是人人都可以随便试的。有因为骂人挨嘴巴的，有因为骂人吃官司的，有因为骂人反被人骂的，这都是不会骂人的原故。今以研究所得，公诸同好，或可为骂人时之一助乎？

1. 知己知彼

骂人是和动手打架一样的，你如其敢打人一拳，你先要自己忖度下，你吃得起别人的一拳否。这叫做知己知彼。骂人也是一样。譬如你骂他是“屈死”，你先要反省，自己和“屈死”有无分别。你骂别人荒唐，你自己想想曾否吃喝嫖赌。否则别人回敬你一二句，你就受不了。所以别人有着某种短处，而足下也正有同病，那么你在骂他的时候只得割爱。

2. 无骂不如己者

要骂人须要挑比你大一点的人物，比你漂亮一点的或者比你坏得万倍而比你得势的人物，总之，你要骂人，那人无论在好的一方面或坏的一方面都要能胜过你，你才不吃亏。你骂大人物，就怕他不理你，他一回骂，你就算骂着了。因为身份相同的人才肯对骂。在坏的一方面胜过你的，你骂他就如教训一般，他既便回骂，一般人仍不会理会他的。假如你骂一个无关痛痒的人，你越骂他他越得意，时常可以把一个无名小卒骂出名了，你看冤与不冤？

3. 适可而止

骂大人物骂到他回骂的时候，便不可再骂；再骂则一般人对你必无同情，以为你是无理取闹。骂小人物骂到他不能回骂的时候，便不可再骂；再骂下去则一般人对你也必无同情，以为你是欺负弱者。

4. 旁敲侧击

他偷东西，你骂他是贼；他抢东西，你骂他是盗，这是笨伯。骂人必须先明虚实掩映之法，须要烘托旁衬，旁敲侧击，于要紧处只一语便得，所谓杀人于咽喉处着刀。越要骂他你越要原谅他，即便说些恭维话亦不为过，这样的骂法才能显得你所骂的句句是真实确凿，让旁人看起来也可见得你的度量。

5. 态度镇定

骂人最忌浮躁。一语不合，面红筋跳，暴躁如雷，此灌夫骂座，泼妇骂街之术，不足以言骂人。善骂者必须态度镇静，行若无事。普通一般骂人，谁的声音高便算谁占理，谁的来势猛便算谁骂赢，惟真善骂人者，乃能避其锋而击其懈。你等他骂得疲倦的时候，你只消轻轻的回敬他一句，让他再狂吼一阵。在他暴躁不堪的时候，你不妨对他冷笑几声，包管你不费力气，把他气得死去活来，骂得他针针见血。

6. 出言典雅

骂人要骂得微妙含蓄，你骂他一句要使他不甚觉得是骂，等到想过一遍才慢慢觉悟这句话不是好话，让他笑着的面孔由白而红，由红而紫，由紫而灰，这才是骂人的上乘。欲达到此种目的，深刻之用意固不可少，而典雅之言词则尤为重要。言词典雅可使听者不致刺耳。如要骂人骂得典雅，则首先要在骂时万万别提起女人身上的某一部分，万万不要涉及生理学范围。骂人一骂到生理学范围以内，底下再有什么话都不好说了。譬如你骂某甲，千万别提起他的令堂令妹。因为那样一来，便无是非可言，并且你自己也不免有令堂令妹，他若回敬起来，岂非势均力敌，半斤八两？再者骂人的时候，最好不要加入以种种难堪的名词，称呼起来总要客气，即使他是极卑鄙的小人，你也不妨称他先生，越客气，越骂得有力量。骂得时节最好引用他自己的词句，这不但可以使他难堪，还可以减轻他对你骂的力量。俗话少用，因为俗话一览无遗，不若典雅古文曲折含蓄。

7. 以退为进

两人对骂，而自己亦有理屈之处，则处于开骂伊始，特宜注意，最好是毅然将自己理屈之处完全承认下来，即使道歉认错均不妨事。先把自己理屈之处轻轻遮掩过去，然后你再重整旗鼓，着着逼人，方可无后顾之忧。即使自己没有理屈的地方，也绝不可自行夸张，务必要谦逊不逞，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

把你这句话骂过去，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，那时节你便可用小题大做的方法，来扩大题目。先用诚恳而怀疑的态度引申对方的意思，由不紧要之点引到大题目上去，处处用严谨的逻辑逼他说出不逻辑的话来，或是逼他说出合于逻辑但不合乎理的话来，然后你再大举骂他，骂到体无完肤为止，而原来惹动你的小题目，轻轻一提便了。

10. 远交近攻

一个时候，只能骂一个人，或一种人，或一派人。决不宜多树敌。所以骂人的时候，万勿连累旁人，即使必须牵涉多人，你也要表示好意，否则回骂之声纷至沓来，使你无从应付。

骂人的艺术，一时所能想起来的有上面十条，信手拈来，并无条理。我做此文的用意，是助人骂人。同时也是想把骂人的技术揭破一点，供爱骂人者参考。挨骂的人看看，骂人的心理原来是这样的，也算是揭开一张黑幕给你瞧瞧！

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. $\sum_{k=0}^n \left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle = \frac{(2n)^n}{2^n},$

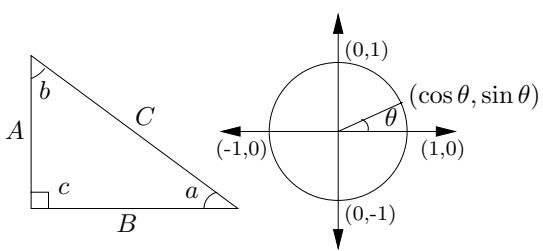
Theoretical Computer Science Cheat Sheet		
Identities Cont.		Trees
38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \sum_{k=0}^n \begin{bmatrix} k \\ m \end{bmatrix} n^{n-k} = n! \sum_{k=0}^n \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix}$,	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. $(n-m)! \begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{Bmatrix} \binom{k}{m} (-1)^{m-k}$, 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:

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

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

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

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

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

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

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

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

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

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

$$\theta \quad \sin \theta \quad \cos \theta \quad \tan \theta$$

$$0 \quad 0 \quad 1 \quad 0$$

$$\frac{\pi}{6} \quad \frac{1}{2} \quad \frac{\sqrt{3}}{2} \quad \frac{\sqrt{3}}{3}$$

$$\frac{\pi}{4} \quad \frac{\sqrt{2}}{2} \quad \frac{\sqrt{2}}{2} \quad 1$$

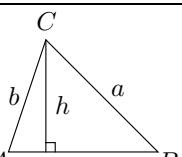
$$\frac{\pi}{3} \quad \frac{\sqrt{3}}{2} \quad \frac{1}{2} \quad \sqrt{3}$$

$$\frac{\pi}{2} \quad 1 \quad 0 \quad \infty$$

... in mathematics you don't understand things, you just get used to them.

– J. von Neumann

More Trig.



Law of cosines:
 $c^2 = a^2 + b^2 - 2ab \cos C.$
 Area:

$$A = \frac{1}{2}hc,$$

$$= \frac{1}{2}ab \sin C,$$

$$= \frac{c^2 \sin A \sin B}{2 \sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$

$$s = \frac{1}{2}(a+b+c),$$

$$s_a = s - a,$$

$$s_b = s - b,$$

$$s_c = s - c.$$

More identities:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\tanh ix = \frac{\sinh ix}{\cosh ix}.$$

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> $C \equiv r_1 \pmod{m_1}$ $\vdots \quad \vdots \quad \vdots$ $C \equiv r_n \pmod{m_n}$ <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>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> $\cos \theta = \frac{(x_1 - x_0)(x_2 - x_0) + (y_1 - y_0)(y_2 - y_0)}{\ell_1 \ell_2}.$ <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: $\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$</p> <p>Brouncker's continued fraction expansion: $\frac{\pi}{4} = 1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{\ddots}}}}$</p> <p>Gregory's series: $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$</p> <p>Newton's series: $\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$</p> <p>Sharp's series: $\frac{\pi}{6} = \frac{1}{\sqrt{3}} \left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots \right)$</p> <p>Euler's series: $\begin{aligned}\frac{\pi^2}{6} &= \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots \\ \frac{\pi^2}{8} &= \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots \\ \frac{\pi^2}{12} &= \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots\end{aligned}$</p>	<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.

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

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:

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

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

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

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

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

Falling Factorial Powers:

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

$$x^0 = 1,$$

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

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

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:

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

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

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

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

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

$x^1 =$	x^1	$=$	$x^{\overline{1}}$
$x^2 =$	$x^2 + x^1$	$=$	$x^{\overline{2}} - x^{\overline{1}}$
$x^3 =$	$x^3 + 3x^2 + x^1$	$=$	$x^{\overline{3}} - 3x^{\overline{2}} + x^{\overline{1}}$
$x^4 =$	$x^4 + 6x^3 + 7x^2 + x^1$	$=$	$x^{\overline{4}} - 6x^{\overline{3}} + 7x^{\overline{2}} - x^{\overline{1}}$
$x^5 =$	$x^5 + 15x^4 + 25x^3 + 10x^2 + x^1$	$=$	$x^{\overline{5}} - 15x^{\overline{4}} + 25x^{\overline{3}} - 10x^{\overline{2}} + x^{\overline{1}}$
$x^{\overline{1}} =$	x^1	$=$	x^1
$x^{\overline{2}} =$	$x^2 + x^1$	$=$	$x^2 - x^1$
$x^{\overline{3}} =$	$x^3 + 3x^2 + 2x^1$	$=$	$x^3 - 3x^2 + 2x^1$
$x^{\overline{4}} =$	$x^4 + 6x^3 + 11x^2 + 6x^1$	$=$	$x^4 - 6x^3 + 11x^2 - 6x^1$
$x^{\overline{5}} =$	$x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1$	$=$	$x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1$

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

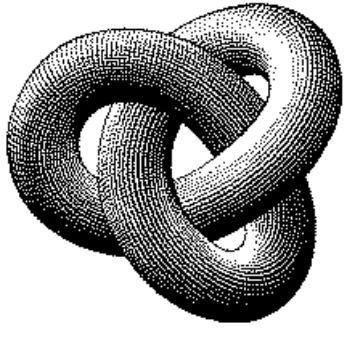
$$B(x) = \frac{1}{1-x} A(x).$$

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.

